**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačních technologií**


**Diplomová práce**

# Architektonický návrh aplikace rozpoznání plevelů v počátečním stádiu růstu ČZU

**Bc. Markéta Palmová**

**© 2018 ČZU v Praze**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Markéta Palmová

Informatika

Název práce

**Architektonický návrh aplikace rozpoznání baby plevelů ČZU**

Název anglicky

**Baby weeds recognition application architecture design for CULS purposes**

---

**Cíle práce**

Cílem práce je architektonický návrh aplikace určené k rozpoznaní rostlinek plevelů (baby plevelů) studovaných na půdě ČZU. Aplikace musí umožnit zaměřit snímací zařízení na plochu o rozměru zhruba od max. 100x100cm do min. 10x10 cm. Zařízení musí umožnit uložit snímek pořízený aplikací do libovolné složky, označit nalezené plevele a případně doporučit vhodný zásah.

**Metodika**

Prostudujte možnosti rozpoznání obrazu a obrazových dat s využitím konvolučních neuronových sítí. Navrhněte vhodnou neuronovou síť (na základě analýzy existujících sítí a jejich grafů). Načrtněte komponentový diagram, navrhněte diagram spolupráce jednotlivých komponent, pro vybranou komponentu vytvořte diagram stavů. Práci zhodnoťte a navrhněte případná vylepšení.

**Doporučený rozsah práce**

70str

**Klíčová slova**

neuronová síť, konvoluční neuronová síť, rozpoznání obrazu, segmentace obrazu, mobilní aplikace, UML diagram

---

**Doporučené zdroje informací**

Dokumentace https://www.tensorflow.org/

Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems,Andrea Tettamanzi, Marco Tomassini,Springer Science & Business Media, 7. 9. 2001

---

**Předběžný termín obhajoby**

2018/19 ZS – PEF (únor 2019)

**Vedoucí práce**

Ing. Josef Pavlíček, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 23. 11. 2018

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2018

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 23. 11. 2018

**Čestné prohlášení**

Prohlašuji, že svou diplomovou práci "Architektonický návrh aplikace rozpoznání plevelů v počátečním stádiu růstu ČZU" jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne _____

**Poděkování**

Ráda bych touto cestou poděkovala vedoucímu své diplomové práce panu Ing. Josefu Pavlíčkovi Ph.D., za vstřícnou spolupráci, cenné rady a trpělivost při jejím vypracování. Dále bych ráda poděkovala paní Mgr. Renatě Čadské, která mi dala možnost dokončit studium a svým rodičům za podporu během celého studia.

# Architektonický návrh aplikace rozpoznání plevelů v počátečním stádiu růstu na ČZU

**Abstrakt**

Diplomová práce je věnována tématu architektonického návrhu aplikace rozpoznávání plevelů v počátečním stádiu růstu na ČZU. Teoretická část poskytuje komplexní pohled na problematiku a rozdělení jednotlivých druhů neuronových sítí. Další problematika, která je v této práci řešena, je rozdělení jednotlivých konvolučních neuronových sítí a jejich využití, výhody či nevýhody. Práce se dále zabývá rozpoznáním obrazu na základě konvolučních neuronových sítí. Je řešena problematika segmentace obrazu, již použité a navrhované metody nebo snímání obrazu.

První část praktické části se zabývá návrhem a možnou implementací konvolučních neuronových sítí. Jsou zde poskytnuty tři navržené architektury. Další část analyzuje možné frameworky a knihovny vhodné pro implementaci konvoluční neuronové sítě v jazyce Java. V poslední části práce jsou navrženy dva vývojové diagramy na učení konvoluční neuronové sítě a na vyhledávání obrazů v databázi. Dále je zde navrhnut Use Case diagram aplikace a počáteční návrh class diagramu. Díky nasbíraným datům z teoretické části jsou v závěru zhodnoceny výsledky a doporučeny další kroky či vylepšení.

**Klíčová slova:** neuronová síť, konvoluční neuronová síť, rozpoznání obrazu, segmentace obrazu, mobilní aplikace, UML diagram

# Baby weeds recognition application architecture design for CULS purposes

**Abstract**

The diploma thesis is devoted to the Baby weeds recognition application architecture design for CULS purposes. The theoretical part provides a extensive view of the problematics and the distribution of individual types of neural networks. Another issue that is dealt with in this thesis is the division of individual convolutional neural networks and their use, advantages or disadvantages. The thesis also deals with image recognition based on convolutional neural networks. The problem of image segmentation, applied and proposed methods or image acquisition is solved.

The first part of the practical part deals with the design and possible implementation of convolutional neural networks. Three designed architectures are provided here. The next part analyzes the possible frameworks and libraries suitable for implementation of Java convolutional neural network. In the last part of the thesis two flowcharts are designed to learn the convolutional neural network and look for images in the database. In addition, the Use Case diagram application and the initial design of the class diagram are designed. Thanks to the collected data from the theoretical part, the results are evaluated in the end and further steps or improvements are recommended.

**Key words:** neural network, convolutional neural network, image recognition, image segmentation, mobile applications, UML diagram

# Obsah

# Seznam obrázků

# Seznam tabulek

# 1 Úvod

Identifikace objektů z obrazu je v dnešní době velmi rozšířená, ať už se jedná o detekci a rozpoznání obličeje, která se nyní běžně používá u chytrých telefonů při odemykání obrazovky, nebo identifikaci objektů z obrazu. Pro realizaci těchto identifikací se v praxi velmi často používá implementace pomocí různých typů neuronových sítí, které jsou vhodné nejen díky jejich architektuře, ale i díky schopnosti abstrakce a učení se.

Na začátku této práce je řešena problematika neuronových sítí, jejich architektury a principu fungování. Následně se práce zaměřuje na různé modely a typy neuronových sítí. V této části jsou popsané vlastnosti jednotlivých modelů, jako je architektura, princip fungování a případně jejich použití.

V další části práce jsou řešeny konvoluční neuronové sítě, které jsou v současnosti jedny z nejvhodnějších modelů pro identifikaci objektů z obrazu, jelikož před vstupem obrazu do sítě není nutná žádná úprava. V této části práce jsou také pospané různé příklady použití tohoto typu neuronové sítě a dále je zde popsáno zpracování obrazu.

Praktická část se věnuje návrhu konvoluční neuronové sítě, která by měla být nejvhodnější pro mobilní aplikaci na rozpoznávání baby plevelů, jejíž uživatelské rozhraní bylo řešeno v bakalářské práci. Cílem praktické části je nejen navrhnout vhodnou konvoluční neuronovou síť, ale také vyhotovit diagramy. Nejen Use Case diagram, ale také vývojové diagramy učení sítí, vyhledávání obrazů v databázi apod..

V závěru práce jsou shrnuté dosažené výsledky a návrhy konvolučních sítí a diagramů. Dále je zde uvedeno možné vylepšení a zhodnocení dosavadních návrhů, ke kterým se během práce došlo.

# 2 Cíl práce a metodika.

## 2.1 Cíl práce

Hlavním cílem práce je architektonický návrh aplikace určené k rozpoznání rostlinek plevelů (baby plevelů) studovaných na půdě ČZU. Dalším cílem je návrh diagramů, pro lepší pochopení a orientaci v aplikace. Jedná se kupříkladu o vývojový diagram, class diagram nebo use case diagram. Aplikace musí umožnit zaměřit snímací zařízení na plochu o rozměru zhruba od max. 100x100 cm do min. 10x10 cm. Zařízení musí umožnit uložit snímek pořízený aplikací do libovolné složky, označit nalezené plevele a případně doporučit vhodný zásah.

Prvním dílčím cílem je rozebrání různých konvolučních neuronových sítí a zvolení té nejvhodnější pro danou aplikaci. Dále je zde řešeno zpracování UML diagramů a také zpracování obrazu.

## 2.2 Metodika

Řešení problematiky diplomové práce je založeno na analýze vědeckých a odborných zdrojů. V teoretické části práce je všeobecně popsána problematika neuronových sítí, případně konvolučních neuronových sítí, u nichž se jedná právě o zpracování objektů z obrazu. Dále je v teoretické části rozebrána technika zpracování obrazu, jako segmentace či snímání obrazu.

Praktická část je realizována formou návrhu konvoluční neuronové sítě. První část se zaobírá návrhem samotné sítě. V druhé části práce je zpracována problematika UML diagramů a na základě toho jsou diagramy navrženy. Jedná se o vývojové diagramy, které umožňují pochopit různé procesy systému, v tomto případě učení konvoluční neuronové sítě a vyhledávání obrazů v databázi. Také je zde navržen use case diagram celé aplikace a class diagram.

Pomocí syntézy z teoretické části i vlastního řešení je zhotoven souhrn výsledků a návrh vylepšení. Na základě nashromážděných informací a poznatkům získaným z praktické části bude formulován závěr diplomové práce.

# 3 Teoretická část

Teoretická část práce pojednává o problematice neuronových sítí jako takových, konvolučních neuronových sítí a problematice rozpoznávání obrazu. Kostrou teoretické části je představení toho, jak neuronové sítě fungují a jaké přináší rizika i výhody. Také je představeno několik příkladů konvolučních neuronových sítí. Dalším důležitým tématem je rozpoznávání obrazu, na jakém principu funguje a jakou roli hraje v dnešní době.

## 3.1 Neuronová síť

Teorie neuronových sítí vychází z neurofyzikálních poznatků a snaží se vysvětlit chování se na principu zpracování informací v nervových buňkách. Někdy se umělé neuronové sítě označují také jako modely mozku bez mysli (angl.. *brain without mind*), protože se snaží pochopit nervový systém, ale nezabírají se psychikou. Za začátek vzniku oboru neuronových sítí se považuje práce Warrena McCullocha a Waltera Pittsa z roku 1943, kteří vytvořili jednoduchý matematický model neuronu [1].

Neuronová síť je algoritmus, který se inspiruje činností lidského mozku. Je to metoda, která spadá do tzv. data miningu, což je analytická metoda, která získává skryté informace z dat. Již dříve bylo zjištěno, že mozek je tvořen mnoha buňkami, které jsou vzájemně propojené. Říká se jim neurony, ty spolu komunikují pomocí elektrických impulzů. Již u prvních počítačů se programátoři snaží vytvořit algoritmus, jenž by napodoboval činnost lidského mozku. Tady se objevuje pojem umělá inteligence. V dnešní době jsou neuronové sítě implementované v řadě dostupných rozhodovacích a analytických softwarů a v různých oborech lidské činnosti. Tyto sítě se ve srovnání s jinými rozhodovacími algoritmy podávaly velmi dobré výsledky[2].

### 3.1.1 Princip sítě

Předností neuronových sítí je schopnost učit se. Mohou si tedy zapamatovat různé kombinace, které vedly k požadovanému výstupu a následně u nových vstupů se podívat do paměti. Na základě těchto zkušeností poté odvozují nový výsledek. V tomto případě se jedná o generalizaci (zevšeobecňování). Schopnost učit se bývá považována za definici umělé inteligence.

Další vlastností neuronových sítí je schopnost řešit nelineární úlohy. Využitím neuronové sítě v analýze dat můžeme vyřešit i úlohy, kde selže např. regrese. Tyto sítě jsou také do jisté míry schopné pracovat s nepřesnými daty a šumy. Lepší a přesnější výsledky se ovšem dostanou, pokud je provedena příprava a čištění dat.

### 3.1.2 Model neuronu

Model umělého neuronu představuje většinou abstrakci mechanizmu, jako nervové buňky zpracovávají informace. Není ovšem možné vytvořit přesnou kopii modelu skutečného biologického neuronu, protože to jsou složité objekty a doposud není přesně známý jejich mechanizmus zpracování informací. Proto modely umělých neuronů, které se v současnosti nejčastěji používají, opisují pouze základní funkci neuronu.

Základní funkce umělého neuronu spočívá ve sběru elementárních informací, vyhodnocení jejich souhrnného stavu a zodpovězení reakcí na výstupu.

Matematicky je neuron chápán jako vztah[4]:

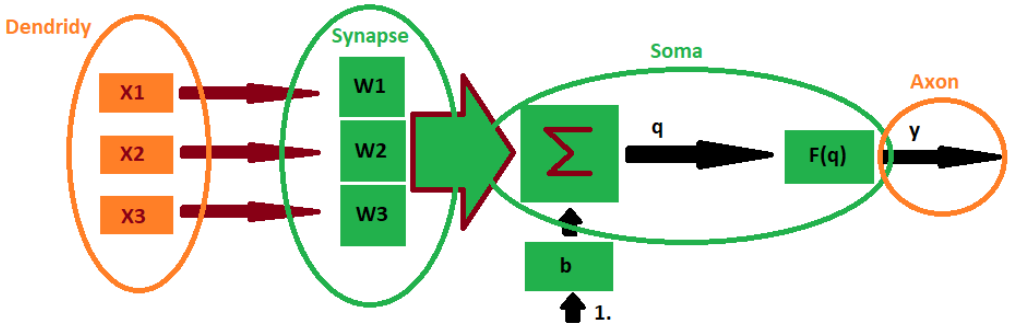
$$in = f(q) = f\left(\sum_{i=1}^{n} x_i\, w_i\right)$$

Kde

- n – představuje počet vstupů;
- $x_i$ – představuje i-ty vstupní signál neuronu;
- $w_i$ – představuje i-tou vstupní váhu;
- f – představuje aktivační funkci neuronu;
- in – představuje vstupní signál neuronu;
- q – představuje potencionál neuronu.

Častěji používaný model umělého neuronu obsahuje kromě jiného i práh (bias) *b*. Tento typ umělého neuronu je vidět na obrázku 1 a je popsaný vztahem[4]:

$$in = f\left(\sum_{i=1}^{n} w_i\, x_i + b\right)$$




*Obrázek 1 - Neuron s vyznačenými částmi biologického neuronu*[1]

[1] Zdroj: Vlastní zpracování

Jak již bylo řečeno, neuronové sítě se skládají z neuronů propojených vazbami. Samotný model neuronu je složen tedy ze tří částí: vstupní, výstupní a funkční. Podle určitých vah, je možné jednotlivé vstupy potlačit či zvýhodnit. Samostatný neuron není schopen vykonávat žádné složité procesy, jedná se spíše o vykonávání funkcí na úrovni regresní analýzy. V neuronových sítích jde o princip propojení jednotlivých neuronů do větších struktur, a především do různých vrstev a tím získávají na síle. Důležitým aspektem je, že algoritmus se učí sám, a tudíž i všechny váhy si volí sám.

Skvěle se k modelu neuronu vyjádřili Miloš Uldrich a Tomáš Jurczyk ve svém článku o využití neuronových sítí. „*Pokud si představíme, jak takovouto sítí proudí příchozí data, je nám jasné, že právě díky složitosti spojení dokáže neuronová síť najít i složitější a nelineární vztahy, na druhou stranu je ale pravda, že ze získané neuronové sítě nikdy nebudeme schopni získat interpretaci, proč to u konkrétního pozorování dopadlo, jak to dopadlo. Je to tedy metoda typu „black box" – nejsme schopni jednoduše interpretovat výsledky či získat jednoduchý předpis závislosti mezi závislou a nezávislými proměnnými* [2]." Pokud jde pouze o předpověď veličiny našeho zájmu, je to velmi účinná metoda, ovšem pokud by šlo o důvody výsledku, pak by tato metoda příliš nepomohla.

Nejpoužívanější typy neuronových sítí jsou: vícevrstvá, perceptronová, radial, basis, function, Kohonenova, lineární a Bayesovské sítě. Každá síť má rozdílné vlastnosti a výběr závisí čistě na povaze úlohy a charakteru dat [3].

### 3.1.3 Umělá neuronová síť

Každá umělá neuronová síť se skládá ze vzájemně propojených neuronů tak, že výstup jednoho neuronu je vstupem do jiných neuronů. Počet neuronů a jejich vzájemné propojení určuje topologie umělé neuronové sítě, která je určená ohodnoceným orientovaným grafem a ten představuje propojení mezi jednotlivými neurony. V tomto grafu jednotlivé neurony představují vrcholy a propojení mezi nimi hrany.

Neuronová síť je charakterizovaná několika vlastnostmi. Mezi hlavní vlastnosti neuronových sítí patří například topologie sítě, způsob učení nebo model sítě. U topologie neuronové sítě se rozlišují dva základní typy:

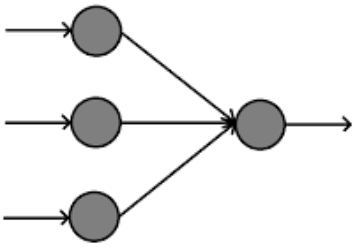
- Neuronové sítě s dopředným sířením signálu
- Neuronové sítě se zpětnovazebným šířením signálu

U neuronových sítí s dopředným šířením signálu (angl. feedforward neural network) postupují všechny signály směrem ze vstupní vrstvy do vrstvy výstupní bez zpětných vazeb. U sítí se zpětnou vazbou (angl. feedback neural network) je vstup každého neuronu závislý na hodnotě výstupu z předchozího cyklu. Příklad neuronů s dopřednou a zpětnovazebnou topologií je uvedený na obrázku 2 a 3.

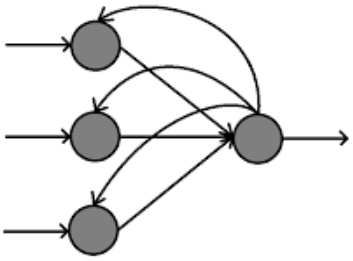
Umělé neurony jsou z hlediska struktury organizované do vrstev, přičemž ve vrstvě se nacházejí neurony se společnými charakteristikami. Umělá neuronová síť se skládá minimálně ze dvou vrstev, a to vstupní a výstupní, případně dalších vrstev, které se nazývají skryté.

Cílem vstupní vrstvy je zabezpečit distribuci vstupních signálů sítě do ostatních vrstev. Neurony v ní mají jediný vstup, který je zároveň i vstupem do neuronové sítě a jejich přenosová funkce reprezentuje lineární funkci, to znamená, že neurony vstupní vrstvy posílají vstupní signál na svůj výstup beze změny.

Výstupní vrstva určuje výstup neuronové sítě. Vrstvy, které se nacházejí mezi vstupní a výstupní vrstvou se nazývají skryté, protože vstupy ani výstupy, které se v nich nacházejí, nezasahují přímo do vnějšího prostředí[5].



*Obrázek 2 - Síť s dopředným šířením signálu*[2]



*Obrázek 3 - Síť se zpětnovazebným šířením signálu*[1]

[2] Zdroj: https://dspace.cvut.cz/bitstream/handle/10467/70080/F3-DP-2017-Hobza-Martin-Dp_2017_hobza_martin.pdf

Činnost umělé neuronové sítě se mohou chápat jako transformace vstupního signálu *in* na výstupní signál $out \rightarrow in = T(out)$ a jejich chování je možné definovat třemi základními prvky:

- Aktivační funkce neuronů
- Množina spojení mezi neurony
- Učícím pravidlem

Aktivační funkce je vybírána tak, aby umělá neuronová síť byla schopná mapovat, co nejširší rozsah lineárních, nelineárních, vstupních a výstupních vztahů. O umělé neuronové síti se říká, že je stabilní, právě tedy, když veškeré její váhy konvergují do rovnovážného vztahu, kde neuronový model aproximuje aktuální chování procesu. V případě, že neuronová síť diverguje, je vedena jako nestabilní.

## 3.2 Příklady modelů umělých neuronových sítí

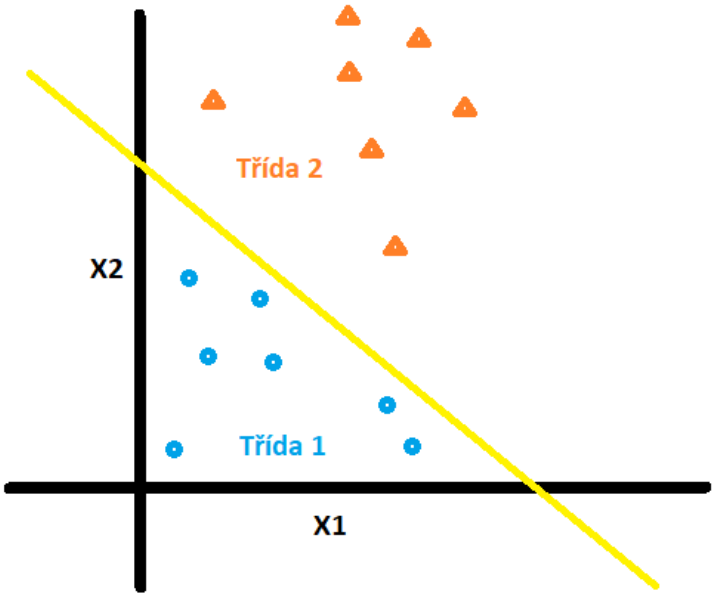
### 3.2.1 Perceptron

Perceptron je základním stavebním prvkem neuronových sítí, jehož zakladatelem byl roku 1957 Frank Rosenblatt. Název pochází z matematického modelu biologického neuronu. Původně byl navržený jako model zrakové soustavy. Typický perceptron je jednoduchá síť, která obsahuje n vstupů $(x_1, x_2, \ldots, x_3)$ a jeden neuron. Přenosová funkce neuronu může být binární (tj. s hodnotou 0 nebo 1), nebo bipolární (tj. s hodnotou -1,0 nebo 1). Každé spojení mezi vstupním a výstupním neuronem, má přidanou hodnotu váhy $(w_1, w_2, \ldots, w_n)$[6].

Při modelování perceptronu se používá metoda učení s učitelem a algoritmus je v následujícím tvaru:

- Nastavení vah náhodnými hodnotami a nastavení prahu
- Přivedení vstupního vektoru do Perceptronu
- Výpočet skutečných hodnot na výstupu
- Porovnání výstupu z perceptronu s požadovaným výstupem
- Výpočet chybové funkce *e*, jako rozdíl zjištěné a očekávané hodnoty
- Pokud není žádná chyba, není potřeba úprava vah a může se pokračovat k dalšímu vstupnímu vektoru
- Při zjištění chyby se k prahu a k váhám přičítá hodnota *e*

Využití perceptonu je především pro klasifikaci, kupříkladu u přiřazení objektu do konkrétní množiny. Učící algoritmus je konečný jen v případě, že řešení existuje a učící množina je lineárně separovaná (viz. obrázek 4). Složitější funkce jako je XOR, nelze řešit pomocí perceptronu. Můžeme je vyřešit rozšířením základního modelu perceptronu do vícevrstvé dopředné neuronové sítě[5].

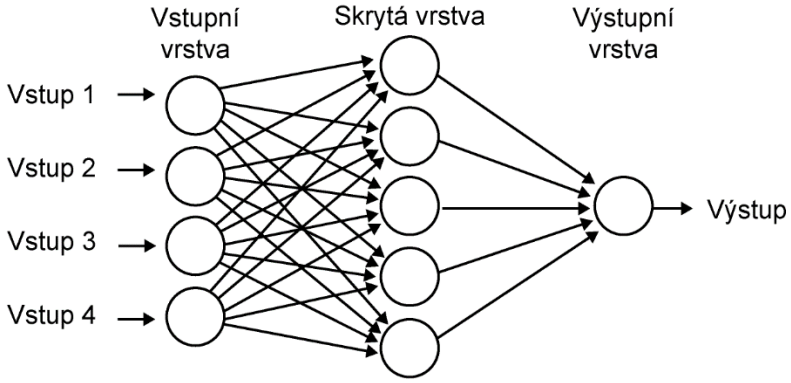



*Obrázek 4 - Lineárné spárovatelná množina*[3]

### 3.2.2 Vícevrstvá dopředná neuronová síť

Vícevrstvá dopředná neuronová síť (angl. multilayer feedforward neural network) byla uvedená v roce 1986 a jejími autory jsou G.E. Hilton, R.J. Williams a E. Rumelhart. Cílem tohoto projektu bylo vyřešit omezené schopnosti perceptronu. Tyto sítě se skládají alespoň ze tří vrstev a to vstupní, skryté a výstupní, jak již bylo zmíněno výše. Ukázka této sítě je na obrázku číslo 5, ovšem někdy se vstupní vrstva nepočítá jako součást modelu a pak je model považován za dvouvrstvý[7].

[3] Zdroj: Vlastní zpracování




*Obrázek 5 - Vícevrstvá dopředná neuronová síť*[4]

Spojení mezi jednotlivými neurony jsou realizované tak, že každý neuron i-té vrstvy je spojený s každým neuronem *i+1-ty vrstvy*. Spojení mezi neurony stejné vrstvě neexistuje.

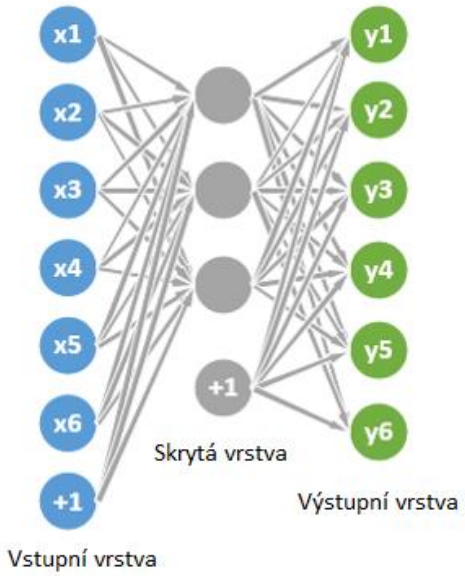
### 3.2.3 Autoenkodér

Autoenkóder je typ neuronové sítě, ve kterém je učení neuronové sítě založené na metodě učení bez učitele. Tato metoda je založena na principu *kodéru* a *dekodéru*. Na začátku, tedy vstup je typicky kódovaný. Tento kódovaný výstup je následně opět dekódovaný aby reprezentoval původní vstup[8].

První autoenkodéry byly uvedeny v osmdesátých letech 20. století jako sítě, které místo učitele používají na učení vlastní vstup. Větší využití ovšem našli až v hlubších architekturách po roce 2006. Příklad autoenkodéru je vidět na obrázku 6[9].

V současnosti se enkodéry uplatňují zejména ve složitých neuronových sítí, které jsou tvořené shluky enkodérů. V těchto situacích tvoří vstupní vrstva prvního enkodéru vstup celé sítě. Následující vrstva, která tvoří skrytou vrstvu vstupního autoenkodéru, je současně vstupní vrstvou dalšího autoenkodéru. Tímto způsobem je síť složená z libovolného množství autoenkodérů.

[4] Zdroj: http://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site




*Obrázek 6 – Autoenkodér*[5]

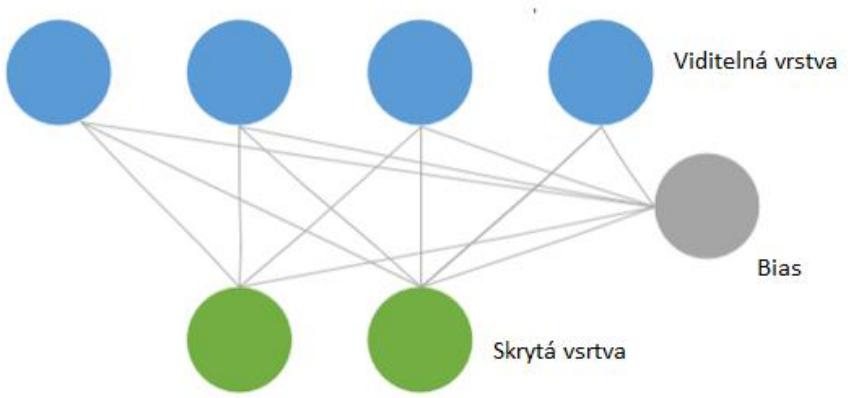
### 3.2.4 Omezené Bolzmannové stroje

Omezené Bolzmannové stroje (původně pojmenované jako Harmonium) patří mezi stochastické typy neuronových sítí, kde binární aktivační funkce neuronu je závislá na sousedech, se kterými je neuron spojený. Jsou součástí modelů zvaných „*Modely založené na energii*“. Původně byly vytvořené v roce 1986 Paulom Smolenskym, ale populárnější začali být až po roce 2000, kdy našli uplatnění například při dimenzionální redukci, klasifikací či spolupracujícím filtrováním. V současnosti je široké použití omezených Bolzmmanových strojů zejména v hlubokém učení sítí a ve složitých neuronových sítí[10].

Omezené Bolzmannové stroje jsou tvořené jednou viditelnou neuronovou vrstvou, jednou skrytou a biasem (chybou), jehož hodnota je vždy rovna jedné. Každý neuron z viditelné vrstvy je propojený s každým neuronem z vrstvy skryté. Výjimečnost těchto spojení je, že jsou neorientované, tedy i každý neuron ze skryté vrstvy je propojený s každým neuronem z vrstvy viditelné. Bias je propojený s každým neuronem ze skryté i viditelné vrstvy. Neurony ze stejných vrstev nesmí být propojeny mezi sebou[11]. Příklad omezeného Bolzmannova stroje je vidět na obrázku 7.

[5] Zdroj: https://lazyprogrammer.me/tag/pca/

Používá se nejčastěji algoritmus, který se označuje jako *konkrétní divergence*. Tento algoritmus byl vyvinut G. Hintonem a původně byl určený pro trénování modelů speciálních produktů[10].



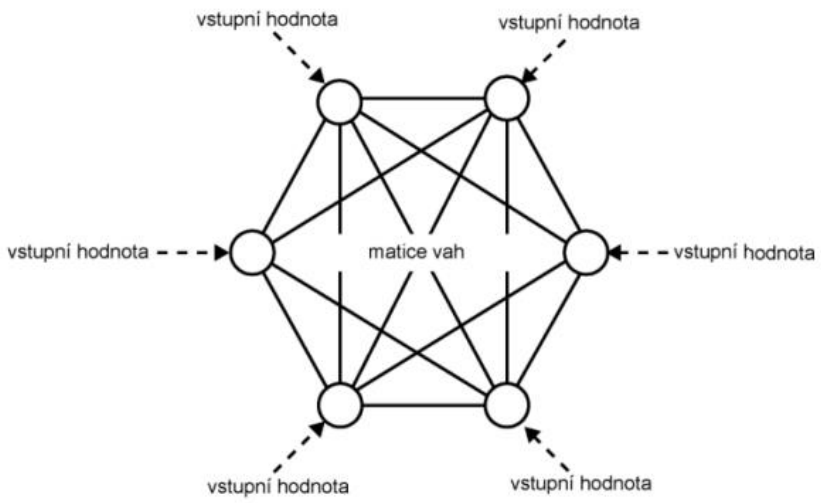

*Obrázek 7 - Omezený Bolzmannův stroj*

### 3.2.5 Hopfieldova neuronová síť

Hopfieldova neuronová síť byla vytvořená v roce 1982 Johnem Hopfieldem. Tato síť je tvořená neurony, jejichž výstup je tvořený hodnotami +1 a -1, kde je hodnota +1 nastavená na výstup neuronu v případě, kdy součet vah je hodnota větší než 0. V případě, že je hodnota součtu vah menší než 0, je výstup neuronu nastavený na -1.

Příklad Hopfieldovi neuronové sítě je možné vidět na obrázku 8. Je tvořena neurony, které jsou plně propojené. Hodnoty všech vah spojení mezi neurony jsou uložené ve váhové matici W, která reprezentuje stav sítě. Pro jednotlivé váhy sítě platí, že diagonální váhové koeficienty $w_{ii}$ jsou rovny 0 a nediagonální váhové koeficienty jsou symetrické, tedy platí $w_{ij} = w_{ji}$[12].

U Hopfieldovi sítě je možné se setkat s několika omezeními, kdy síť funguje podle očekávání. Jedním z omezení sítě je schopnost uchování pouze omezeného počtu vzorů. Dalším z problémů je, že může dojít k situaci, kdy se v síti nachází dva stavy se stejnou hodnotou energie, což zabraňuje konvergenci. Tento jev bývá nazýván jako oscilace sítě[12].




*Obrázek 8 - Hopfieldova neuronová síť*[6]

#### *3.2.5.1 Hebbovo učení*

Jedním ze způsobů učení Hopfieldovi neuronové sítě je tzv. *„Hebbovo učení"*, které je založené na pravidle, které bylo vytvořeno roku 1949 psychologem Donaldem Hebbem. Velmi dobře toto vysvětlila paní I. Mrázové ve své prezentaci: *„Pokud jsou dva neurony současně aktivní, měli by mít vyšší stupeň vzájemné interakce než neurony, jejichž aktivita korelaci nevykazuje. V takovém případě by jejich vzájemná interakce měla být buď nulová, nebo velmi malá* [13]". Doplnit tuto definici můžeme přímo z knihy D.O. Hebba z roku 1949 „*V praxi to znamená, že synapse mezi neurony je posílena, pokud aktivita vstupního neuronu často vede k aktivitě neuronu na výstupní straně synapse* [14]".

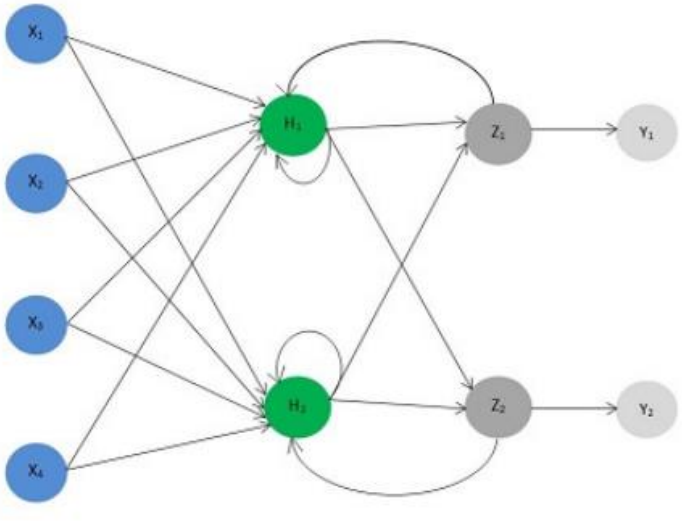
### 3.2.6 Rekurentní neuronové sítě

Na rozdíl od dopředných neuronových sítí, jejichž spojení mezi neurony je jednosměrné, mají rekurentní neuronové sítě minimálně jedno cyklické spojení. Nejčastěji se používají při modelování biologických procesů nebo implementaci dynamických systémů.

Rekurentní neuronové sítě jsou podobné dopředným neuronovým sítím s tím rozdílem, že spojení mezi neurony nejsou vedeny jedním směrem, ale výstupní vrstva

[6] Zdroj: http://portal.matematickabiologie.cz/res/f/neuronove-site-site-se-vzajemnymi-vazbami.pdf

neuronů může obsahovat spojení nejen s neurony ze skryté vrstvy ale i ze vstupní vrstvy, případně i mezi sebou ve vrstvě výstupní.




*Obrázek 9 - Rekurentní neuronová síť*[7]

Stejně i tento typ má jisté nevýhody. V tomto případě může nastat stav, který se nazývá *„overfitting“*, kdyby bylo sítí dáno velké množství dat na učení, tak nedokáže spolehlivě generalizovat data ze vstupu[15].

V České republice se rekurentní neuronové sítě používají především v lingvistice. Jako příklad je možné použít automatický převod textu na mluvenou řeč[16].

# 4 Konvoluční neuronové sítě

Tato kapitola je zaměřena na speciální typ neuronových sítí, a to konvoluční neuronové sítě. Jsou zde popsány určité výhody oproti normálním vícevrstvým neuronovým sítím. Dále je zde popsána struktura, jejich učení a příklady reálného použití.

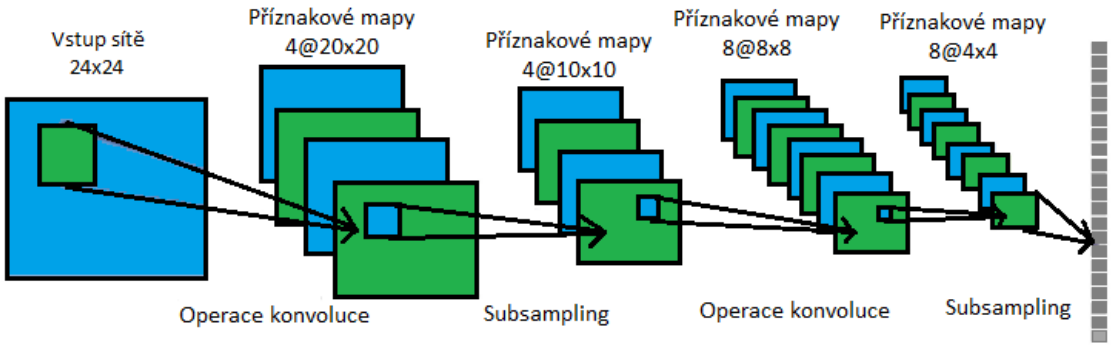
## 4.1 Základní charakteristika

Konvoluční neuronové sítě jsou speciálním druhem vícevrstvých neuronových sítí, které se používají pro rozpoznávání obrazových vzorů a přímo z pixelů obrazu s použitím minimálního předzpracování. Vycházejí z principů neocognitronové sítě, která byla uvedená roku 1987 panem K. Fukushimem. Neocognitron se skládá z velkého množství neuronových

[7] Zdroj: https://www.czechency.org/slovnik/UM%C4%9AL%C3%81%20NEURONOV%C3%81%20S%C3%8D%C5%A4

vrstev a obsahuje variabilní spojení mezi buňkami sousedních vrstev. Cílem této sítě je identifikace objektů na základě podobností vzorů bez ohledu na deformaci, změnu velikosti nebo posunutí pozice. Toto se uplatnilo například při implementaci rozpoznávání ručně psaných znaků[17].

Typická konvoluční neuronová síť je ukázána na obrázku 10. Zde je vidět že se skládá z několika vrstev. K jejím vlastnostem patří získávání příznaku z polí, technologie sdílených vah a prostorové vzorkování.




*Obrázek 10 - Konvoluční neuronová síť*[8]

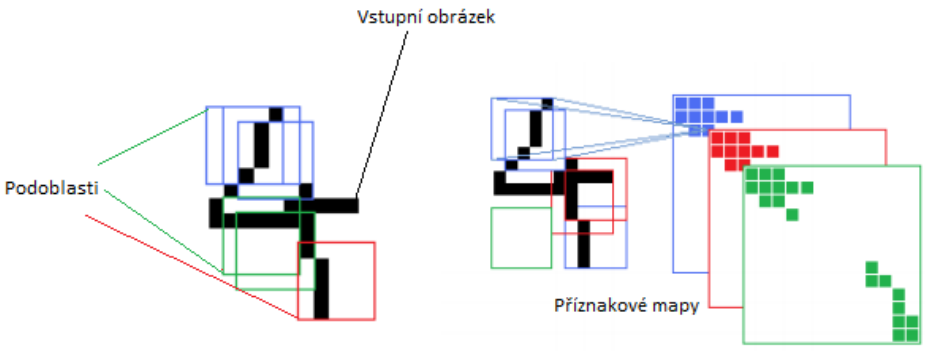
První vrstvou je *konvoluční vrstva*. Na vstup konvoluční neuronové sítě je nejprve přiveden obraz, kde je aplikován filtr, jehož cílem je získávání příznaků. Tento filtr je složen z neuronů, které mezi sebou sdílejí stejnou množinu vah vektorů, což snižuje počet parametrů sítě, a tedy i velikost kapacity. Pomocí těchto lokálních recepčních polí dokáže síť z obrazů získat příznaky jako kupříkladu hrany nebo rohy. Další výhodou konvolučních neuronových sítí je sdílení vah, což urychluje učení sítě tím, že se omezí celkový počet výpočtů. Tento filtr se označuje jako *recepční pole*. Recepční pole je postupně aplikované na vstupní obraz, tedy počítá jednu operaci v různých částech obrazu. Ve většině případů se na jeden vstupní obraz aplikuje několik recepčních polí s tím, že jednotlivé recepční pole se liší hodnotami sdílených vah vektorů. Výstupem jednoho recepčního pole, které bylo aplikované na celý vstupní obraz se nazývá *příznaková mapa* a počet příznakových map odpovídá počtu recepčních polí aplikovaných na obraz. V případě obrázku 10 je vidět, že první konvoluční vrstva je složena ze čtyř příznakových map a každá mapa obsahuje 20x20 recepčních polí.

[8] Zdroj: Vlastní zpracování

V případě, že je příznak pomocí recepčního pole identifikovatelný, stává se lokací, v které byl příznak identifikovatelný méně významný. Z toho důvodu se za konvoluční vrstvou nachází další vrstva, které se říká *subsampligová*. Cílem této vrstvy je zredukovat velikost vstupu. V praxi je během této metody na všechny příznakové mapy, které jsou výstupem konvoluční vrstvy aplikovaná funkce, jehož cílem je tyto mapy zmenšit.

### 4.1.1 Příznaková mapa

Tato podkapitola je zaměřená blíže a funkce příznakových map. Základní funkcí příznakové mapy je získání lokálních charakteristik obrazu. V případě obrázku číslo 11 s rozměry 20x20 zvolíme podoblast například 5x5. Každý pixel podoblasti bude vstupem jednoho neuronu, který bude rozpoznávat vlastnosti. Ať už se jedná o horizontální či svislé čáry, nějakou křivku nebo prázdnou oblast. Tento neuron bude mít 26 vstupů (25 + práh). Tento neuron je kopírován nad celým obrazem, aby se jednotlivé podoblasti překrývali. Těchto podoblastí a jejich neuronů bude následně 256 (16x16). Proces je *konvolucí nad obrazem* a neuron, který zpracovává pixely z podoblasti *konvolučním jádrem*. Z toho vyplívá, že všechny neurony v jedné příznakové mapě sdílejí práh i váhy. Celá mapa má pouze 26 vah, i když obsahuje 256 neuronů. Tím, že je pokrytý celý obraz důsledně příznakovou mapou je možná detekce lokálních charakteristik i při jejich posunu.




*Obrázek 11 - Příznakové mapy*

### 4.1.2 Subsampling

Subsamplingová vrstva má za úkol redukci velikosti příznakových map a vždy se váže na konvoluční vrstvu. Existuje několik typů funkcí, které je možné použít v subsampligové vrstvě, například průměrování, výběr maxima (angl. Max Pooling) nebo lineární kombinace

neuronů dané příznakové mapy. Po spojení několika konvolučních a subsamplingových vrstev je poslední výstup transformován do jednorozměrného vektoru. Tento vektor je převeden na vstup do fungující neuronové sítě a ta vypočítá konečný výstup. Tento proces je znázorněn na obrázku 10.

## 4.2 Učení konvoluční neuronové sítě

Na základě výše uvedených poznatků se všechny vlastnosti z předchozích vrstev sítě spojují do jednoho vektoru, který je následně poslán na vstup sítě. Tato metoda učení odpovídá metodě zpětného šíření chyby, jenže u konvolučních neuronových sítí je tato metoda upravená pro jednotlivé vrstvy sítě.

Pokud se bude předpokládat, že konvoluční vrstvu *l* následuje subsamplingová vrstva *l + 1*, v tomto případě jeden pixel ze subsampligové vrstvy představuje blok pixelů v konvoluční vrstvě, což znamená, že několik jednotek z vrstvy *l* je spojených do jedné vrstvy *l + 1*. Pro počítání chyb v konvoluční vrstvě *l* je proto vhodné zvětšit mapu chyb ze subsampligové vrstvy *l + 1* na velikost mapy z konvoluční vrstvy *l*. Následně bude vynásobena zvětšená mapa ze subsamplingové vrstvy *l + 1* derivací aktivační funkce příslušnou mapou z konvoluční vrstvy *l*[18].

### 4.2.1 Plně propojená konvoluční neuronová síť

Plně propojená konvoluční síť se od předchozí odlišuje všemi vrstvami, které neobsahují kompletně propojené vrstvy a vše jsou pouze filtry.

Plně konvoluční neuronová síť se učí a rozhoduje na základě lokálních charakteristik. Plně propojená síť se dokáže učit i za pomoci globální informace ovšem nevýhodou je, že plně propojené vrstvy očekávají vstup určité velikosti. Pokud tyto vrstvy chybí, je konvoluční síť schopna zpracovat obraz libovolné velikosti.

Další nevýhodou je zachování informace o umístění. Plně propojené vrstvy u Konvolučních neuronových sítí většinou informaci o umístění ztrácejí, protože jsou právě „*plně propojeny*“. Přesto mohou tyto sítě provádět segmentaci obrazu, je ale zapotřebí aby se zredukovali na omezené vizuální variace a na co nejmenší kategorizaci[19][20].

### 4.2.2 Implementace učení

Implementací konvolučních neuronových sítí je v dnešní době nepřeberné množství. Především se jedná o toolboxy a knihovny s řadou funkcí s nimiž pak pracuje programátor. Pro tyto implementace je možné použít i grafický procesor (dále jen GPU). S GPU je možné využít dva druhy programování. První je OpenCL a druhý CUDA.

OpenCL (angl. Open Computing Language) je otevřený standard pro multiplatformové paralelní programování. Jak funguje OpenCL velmi dobře popsal Michal Kwolek, ve svém článku z roku 2009 OpenCL. „*Programátor napíše výpočetní jádro, které se velmi podobá běžné funkci v C. Funkce může mít lokální proměnné, přístup ke globálním datům a datům společným pro všechny instance jádra, k dispozici jsou běžné matematické operace, goniometrické funkce, operace nad vektory...*[21]“ I zde jsou určitá omezení například u rekurze, větvení a nebo cyklů. „*Výpočetní jádro se převede překladačem do mezijazyka, něco jako bytecode pro Javu. Takto vytvořené výpočetní jádro se následně poštve nad vektor, matici nebo trojrozměrnou matici. OpenCL ovladač zařízení se postará o celý zbytek- převede si výpočetní jádro do podoby, které bude rozumět dané zařízení, nakrmí ho daty a provede výpočet*[21]“. To vše probíhá za běhu, a vzhledem k jednoduchosti jazyka se jedná o velmi rychlí proces.

CUDA (angl. Computer Unifield Device Architecture) je platforma vytvořená společností nVidia. Je implementována za pomoci grafických jednotek a umožňuje jednoduchý a přímí přístup k virtuálním instrukcím a paměti. Je tvořen za pomoci C/C++. Je možné implementace i za pomoci jazyka Java, v tom případě jsou k dispozici knihovny JCUDA. Hlavní nevýhodou je, že funguje pouze za pomoci grafických jednotek od společnosti nVidia.

#### *4.2.2.1 MatConvNet*

MatConvNet je toolbox pro prostředí MATLAB. Za vznikem stojí Oxford Visual Geometry Group. Veškerý jeho kód je psán v MATLABu a na rozdíl od jiných toolboxů neschovává nic jako kompilovaný kód. Pokud se spojí platforma CUDA a Parallel Computing toolbox lze používání urychlit s využitím výpočetního výkonu grafické karty.

#### *4.2.2.2 Caffe*

Caffe je Framework, který byl vyvinut Barkley AI Research Laboratory. Celý je napsán v C++ a dá se spustit v MATLABu, stejně jako předchozí, ale i v Pythonu. I Caffe podporuje výpočet na GPU za pomoci CUDA, stejně jako MatConvNet.

#### *4.2.2.3 TensorFlow*

TensorFlow je vyvinut společností Google a jedná se o softwarovou knihovna pro numerické výpočty za pomocí metody data flow graph. Tato platforma je psána v jazyce C++ a Pythonu. Stejně jako u předchozích Platforem i tato podporuje výpočet na GPU za pomoci CUDA.

### 4.2.3 Deep Learning

Deep learning je metoda strojového učení, kde pomocí vícevrstvých nelineárních výpočetních modulů je získávána informace přímo z dat. Tyto modely dosahují vysoké přesnosti, a to nejčastěji v klasifikačních úlohách. Jejich architektura je postavena na *hlubokých neuronových sítích* (angl. deep networks)[22]. Deep learning se využívá především v následujících úlohách:

- Klasifikace obrazových dat (rozpoznávání objektů na snímcích)
- Určení objektů na snímcích
- Segmentace snímků
- Predikce a klasifikace časových řad a signálů

Přesnost modelu je závislá na množství vstupních dat. Pro co nejpřesnější modely je zapotřebí tisíce i milióny dat. Učení takovýchto modelů je na dlouhou dobu, poté je možné model nasadit do provozu v reálném čase. Jednou z nejběžnějších aplikací je detekce chodců.

Konvoluční neuronové sítě jsou časté v oblasti deep learningu. Síť je uspořádána do vrstev, které obsahují navzájem propojené uzle. Konvoluční neuronová síť využívá dvourozměrné vrstvy, a proto je vhodná pro zpracování a rozpoznání 2-D dat (např. obrazy)[22].

## 4.3 Využití konvoluční neuronové sítě

Praktické využití konvolučních neuronových sítí v posledních letech silně narůstá. V této části bude uvedeno několik příkladů použití konvolučních neuronových sítí.

### 4.3.1 LeCunův model

Za úplný počátek konvolučních neuronových sítí se považuje síť, kterou navrhl Yann LeCun v roce 1998. Tato síť sloužila pro rozpoznávání rukou psaných číslic a skládala se ze sedmi vrstev. První vrstva se skládala z šesti příznakových map, kde každá mapa byla o rozměru 28x28 pixelů. Za první vrstvou se nacházela subsamplingová vrstva, která zmenšovala příznakové mapy z předchozí vrstvy na rozměr 14x14 pixelů. Každá jednotka příznakové mapy ze subsomplingové vrstvy byla spojená s jednotkami příslušné příznakové mapy z konvoluční vrstvy o velikosti 2x2 pixelů. Druhá konvoluční vrstva byla složena z 16 příznakových map. Druhá subsamplingová vrstva zmenšovala všech 16 příznakových map na velikost 5x5. Třetí konvoluční vrstva se skládala ze 120 příznakových map o rozměru 1x1 pixel. Za touto vrstvou se nacházela poslední plně propojená vrstva a ta se skládala z 84 jednotek. Následující vrstva se skládala z 10 neuronů, kde každý neuron identifikoval jedno z 10 čísel[23].

Výstupní vrstva se počítala na základě Euklidovské radiální funkce ve tvaru:

$$y_i = \sum_j (x_j - w_{ij})^2$$

Pro zkoušení této neuronové sítě byl použit dataset MNIST. Po zkoušce sítě tímto datasetem dosahovala chybovost sítě hodnoty přibližně 0,95%. Následně se podařilo hodnotu chybovosti sítě snížit na menší než 0,3% pomocí úpravy obrazu z datasetu prostřednictvím různých lineárních kombinací a elastické deformace[23].

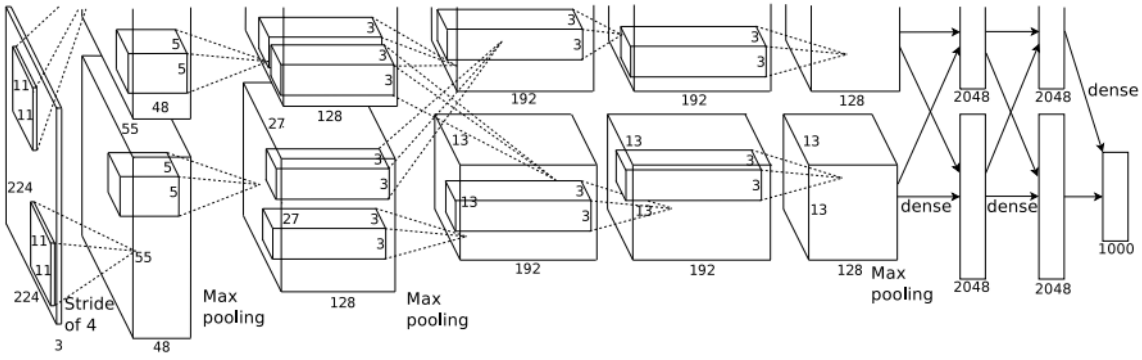
### 4.3.2 ImageNet a Implementace konvoluční neuronové sítě pomocí GPU

Kromě datasetu MNIST jsou v dnešní době i větší datasety, jako například *LabelMe*, který obsahuje statisíce plně segmentovaných obrazů. Dalším typem datasetu je *ImageNet*, který obsahuje 15 miliónů označených obrazů ve vysokém rozlišení. Jednotlivé obrazy z datasetu ImageNet jsou posbírané z internetu a označené pomocí nástroje *Amazon's Mechanical Turk*. ImageNet vznikl roku 2010 jako součást soutěže *ImageNet Large-Scale*

*Visual Recognition Challenge*, nebo také *ILSVRC,* v které byla využita podmnožina z ImageNetu a přibližně 1,2 milióny testovacích obrazů[24].

Podle článku na toronto.edu, kde autoři vytvořili neuronovou síť, získala nejlepší výsledky v soutěži *ILSVRC-2010* a *ILSVRC-2012*.

Vytvořená konvoluční neuronová síť se skládá z osmi vrstev, kde pět z nich je konvoluční a tři jsou plně propojené. Výstup poslední, tedy plně propojené vrstvy je spojený s více směrnou normalizovanou exponenciální funkcí, která poskytuje dělení do více jak 1000 tříd. Příznakové mapy druhé, čtvrté a páté konvoluční vrstvy jsou spojené pouze s těmi mapami předchozí vrstvy, které se nacházejí na stejné úrovni GPU. Příznakové mapy na třetí konvoluční vrstvě jsou spojené se všemi příznakovými mapami druhé konvoluční vrstvy. Vstupem neuronové sítě jsou obrazy o rozměrech 224x224x3. První konvoluční vrstva je tvořená 96 příznakovými mapami s velikostí 11x11x3 s posunem o 4 pixely. Následně obraz postupuje jednotlivými konvolučními vrstvami s jistou úpravou. Pátá, a tedy poslední konvoluční vrstva se skládá z 256 příznakových map a velikostí 3x3x192. U plně propojených vrstev obsahuje každá vrstva 4096 neuronů. Schéma uvedené konvoluční neuronové sítě je zobrazeno na obrázku 12[24].




*Obrázek 12 - Architektura konvoluční neuronové sítě s pomocí GPU*[9]

Uvedená neuronová síť obsahuje 60 miliónů parametrů. Při takto velkém množství parametrů nastává situace, kdy síť ztrácí schopnost identifikaci objektu na základě generalizace. Tato situace se nazývá „*overfitting*“. Řešením této situace je zvětšení dat v tréninkové množině. V tomto případě jsou použité dvě metody pro zvětšení počtu obrazů. První způsob je vytvoření nových obrazů pomocí transformace původních a druhý způsob

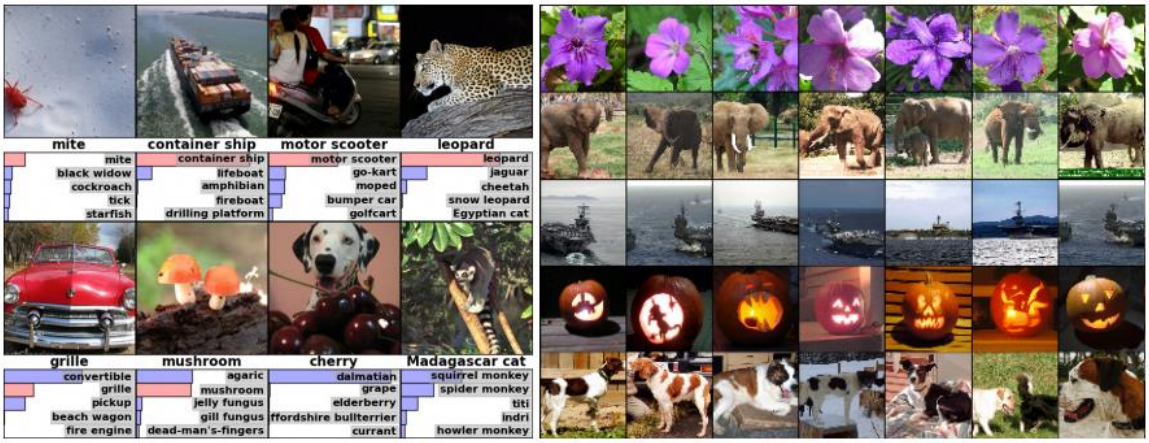
[9] Zdroj: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

je vytvoření nových obrazů pomocí změny intenzity barev RGB spektra v původních obrazcích[24].

Z důvodu času, kdy trénování velkých neuronových sítí trvalo i několik dní, byla při implementaci této konvoluční neuronové sítě uvedena metoda „*dropout*“, která nastavuje nulovou hodnotu pro každý neuron ze skryté vrstvy s pravděpodobností 0,5. Je to z toho důvodu, že tyto typy neuronů nemají vliv na výpočty při průchodu neuronů sítí a stejně tak nemají vliv na učení sítě pomocí zpětného šíření chyby. *Dropout* je použitý v prvních dvou plně propojených vrstvách sítě a jeho použití má velký vliv při potlačení *overfittingu[24]*.

V kategorii, kdy síť vybrala jednu možnost pro označení obrazu (Top-1 error) byla hodnota chybové funkce 37,5%. Jen pro informaci, druhý nejlepší výsledek v soutěži byl 45,7%. V kategorii, kde síť vybírala pět možností pro označení obrazu, dosáhla chybová hodnota 17%, kde druhý nejlepší výsledek byl 25,7%[24].

Na obrázku 13 je možné vidět soubor obrazů a výsledků při ILSVRC-2010. K tomuto obrázku je připojen popis a vysvětlení přímo ze zdroje.




*Obrázek 13 - Snímky testi ILSVRC-2010*[10]

*„(Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5).*
*(Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.*[24]*“*

[10] Zdroj: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

Zkráceně řečeno, vlevo můžete vidět osm testovacích snímků a pět popisků, které model považuje na nejpravděpodobnější. Správný popisek je zobrazen pod každým snímkem a pravděpodobnost přiřazená správnému popisku je zobrazena červeným pruhem. Vpravo můžete vidět 5 snímků v prvním sloupci jako základní a zbývajících šest snímků je tréninkových.

### 4.3.3 Využití konvoluční neuronové sítě při detekci příznaků v hudbě

V současnosti se konvoluční neuronové sítě nepoužívají jen při detekci obrazových vzorů, ale aktuálně i při detekci příznaků v hubě, které patří k základním úlohám v hudební analýze. Na základě identifikace těchto příznaků jsou následně analyzovány další vlastnosti jako například odhad tempa. Identifikace těchto věcí v hudbě se dá přirovnat k identifikaci hran v obrazech.

Na rozdíl od obrazových dat, kde se používají čtvercové filtry, u spektogramů se víc osvědčily filtry obdélníkové. Tento tvar vychází ze skutečnosti, že síť hledá změny v čase, proto delší část obdélníku představuje časovou jednotku a kratší frekvenci.

Učení tohoto typu konvoluční neuronové sítě probíhá pomocí shluku spektogramů, které mají různou velikost ale také rychlost v počtu snímků za sekundu a jsou zmenšené na stejnou velikost frekvenčního pásma. Na základě těchto vlastností, každý neuron může obsahovat informace o časové a frekvenční přesnosti pro danou lokaci. Při testování sítě byl testovací signál nejdříve převedený do podoby spektogramu, který byl přivedený na vstup sítě vcelku, na rozdíl od učení sítě, kde byly na vstup přivedeny jen části spektogramu[25].

K učení sítě byl použit dataset, který se skládal ze 102 minut hudby, která obsahovala 25927 příznaků. Tato nahrávka se skládala z různých typů nahrávek. Byly použity monofonně, polyfonně instrumentální části nebo moderní populární hudba. Z tohoto záznamu, byl vytvořen spektrograf se skokovou velikostí 10ms a okny s velikostmi 23ms, 46ms a 93ms. Výsledný vstup sítě určení k učení sítě se skládal z třech spektogramů založených 15 snímky na 80 pásmech. Na vstupní vrstvu byla aplikovaná konvoluční vrstva, která se skládala z filtrů založených ze 7 snímků na 3 pásma. Výstupem konvoluční vrstvy bylo 10 příznakových map složených z 9 snímků na 78 pásem. Další vrstvou byla vrstva *„max-pooling“*, jehož cílem bylo zredukovat velikost map na 26 pásem. Dále následovala další konvoluční a max-poolingová vrstva. Příznakové mapy, které z nich vycházeli, byly zpracované do jedné plně propojené vrstvy a končili v poslední plně propojené vrstvě, jejím výstupem byla jediná jednotka a ta určovala hudební příznak. V síti bylo použito několik

aktivačních funkcí. U konvolučních vrstev byl jako aktivační funkce použit tangens a u plně propojených vrstev aktivační funkce logický sigmoid[25].

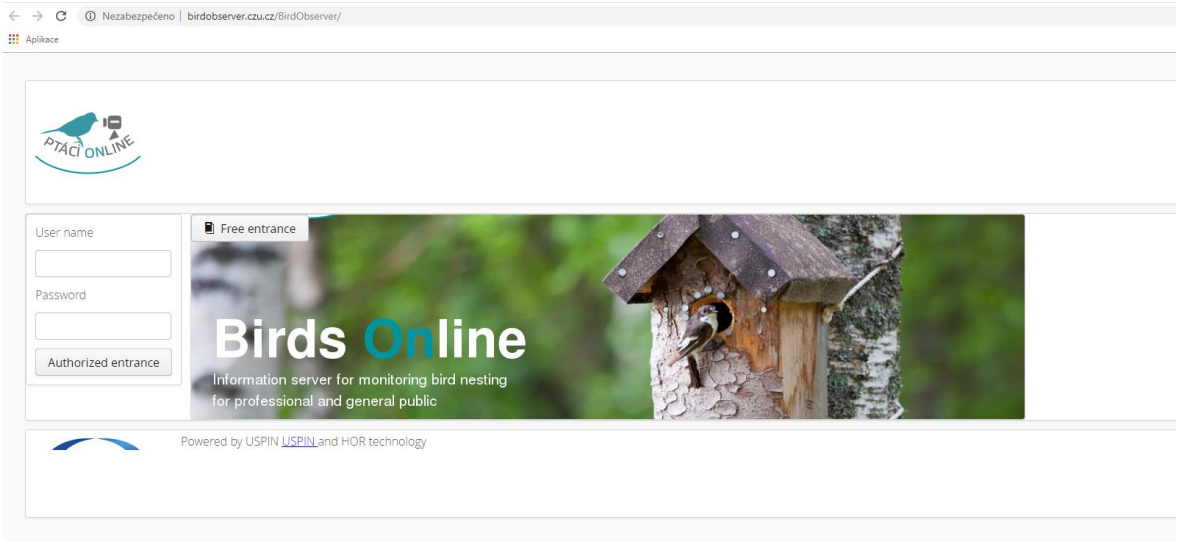
V závěrečném testování dosahovala konvoluční neuronová síť úspěšnosti 88,5%. Pro zvýšení výkonnosti bylo na konvoluční neuronovou síť aplikovaných několik dodatečných metod. První metodou byl *Dropaout* a s jeho použitím dosáhla síť úspěšnosti 89%. Další metodou použitou pro zlepšení bylo na základě učení nepřesně definovanými vzory. Například se při učení sítě používalo několik blízkých spektografických snímků spojených dohromady. Výsledkem tohoto způsobu učení dosáhla přesnost hodnoty 89,9%. Na závěr byla pro zvýšení výkonu konvoluční neuronové sítě nahrazená aktivační funkce funkcí lineární y(x) = max(0,x), která zdvihla úspěšnost sítě na 90,3%[25].

### 4.3.4 Bird‘s Online

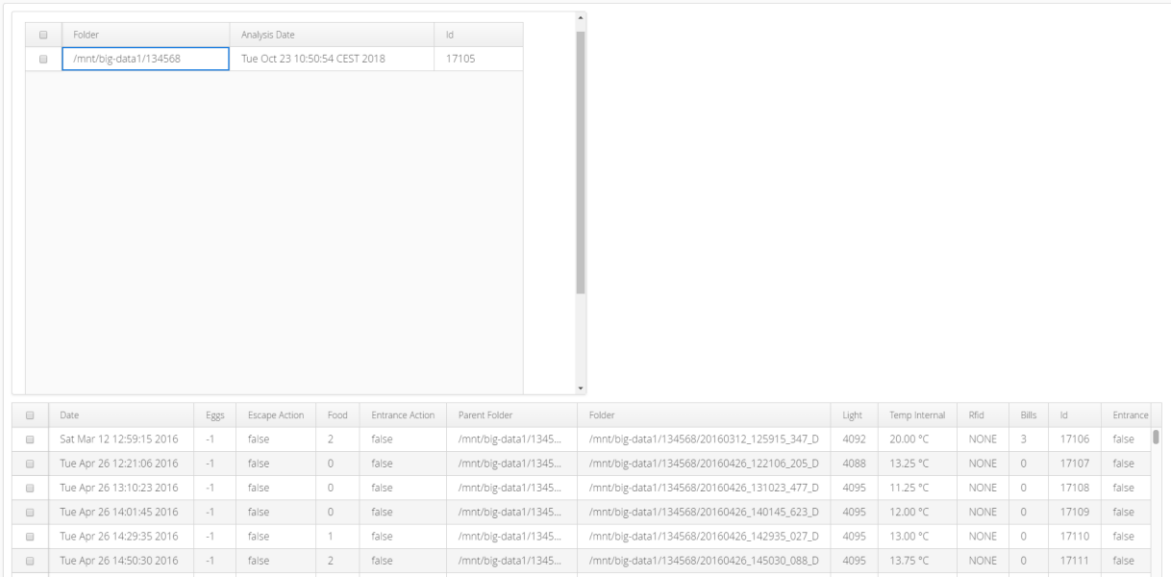
Tento projekt je typickým využitím konvolučních neuronových sítí v praxi. Bird‘s Online byl založen Fakultou věd a životního prostředí České vědecké fakulty Tomáše Bati ve Zlíně roku 2014. Nyní je hlavním účastníkem projektu Fakulta životního prostředí a Provozně ekonomická fakulta na České zemědělské univerzitě v Praze. Projekt má být v konečné fázi veden pro sledování ptáků v budkách, pro sledování jejich potravy, vývoje i možnosti nebezpečí. Následně se mohou záznamy využít jako učební pomůcka nebo pro vědecké výzkumy či ochranu ohrožených druhů. V inteligentním hnízdě je umístěný počítač, snímač pohybu na vstupním otvoru, jedna či dvě kamery s nočním osvětlením, mikrofon, vnitřní i venkovní snímač teploty a osvětlení. Všechna zařízení jsou vestavěná[36].

Cílem projektu je navržení právě takové konvoluční neuronové sítě, která by zaznamenávala a sama ukládala a rozpoznávala jednotlivé ptáky, počet mláďat v hnízdě, potravu přinesenou do hnízda, z čeho je hnízdo postaveno nebo časové sekvence návštěvy hnízda. Přenos dat i elektronický výkon je zajištěn pomocí kabelu Ethernet[36].

Na následujícím obrázku je ukázka systému, kde probíhá ukládání a vyhledávání nahraných videí a obrázků z chytrých budek, kde na této databázi pracuje právě i vedoucí této práce pan Ing. Josef Pavlíček, Ph.D., který se zároveň podílí na projektu se zaměřením automatizovaného rozpoznání divoké zvěře.




*Obrázek 14 - Uvodní strana projektu BirsOnline k přístupu do databáze*[11]




*Obrázek 15 - Databáze vyhledávání v projektu BirdsOnline*[10]

## 4.4 Zpracování obrazu

Při zpracování obrazu (image processing) rozpoznávaného objektu je potřeba nejdříve provést digitalizaci a snímání obrazu. Po provedení těchto dvou kroků se přejde na předzpracování obrazu (image preprocessing), což zajistí vylepšení obrazu a je zaměřen na převod stupně šedi, nastavení jasu a kontrastu, na ošetření obrazu, ekvalizaci histogramu a na různé metody filtrace[26]. Velmi dobře se k této problematice vyjádřil i vedoucí této

[11] Zdroj: Vlastní zpracování

práce pan Ing. Josef Pavlíček, Ph.D., který se zmínil o zpracování obrazu ve své práci o automatizovaném rozpoznání divoké zvěře.

„*At this stage, cleaning the photo from "normal noises", such as changes in brightness, takes place. These are due to light conditions changing during recording (change in light intensity). By balancing the brightness and colour layers, it is ensured that the algorithm processing the photograph has the necessary properties to perform segmentation.*[37]“ Neboli záleží hlavně na změně jasu. To může být způsobeno změnou světelných podmínek a při vyvážení barevných vrstev musí být zajištěno, že algoritmus, který zpracovává fotografie, má potřebné vlastnosti pro segmentaci.

Následujícím krokem je použití segmentačních metod, které rozlišují rozpoznávaný objekt od pozadí. Jedná se především o segmentaci prahováním, metody pro detekci a spojování hran, algoritmy barvení obrazu a různé další algoritmy pro vyplňování objektů. Další fází zpracovávání obrazu je popis daného objektu. Jedny z nejznámějších metod popisu objektu jsou Fourierovy deskriptory, momentová metoda nebo řetězové kódy, které můžeme použít i pro *„strukturální popis objektů“* [27][28].

Poslední fází procesu je samotné zpracování a rozpoznání objektu. Rozpoznání objektu má za úkol zařadit objekt nalezený v obraze do skupiny tříd, které jsou předem známé. Metody rozpoznání objektu se dělí do dvou skupin, které souvisí se způsobem popisu objektu. První skupinou je příznaková klasifikace a druhou skupinou je strukturální klasifikace[26][28].

### 4.4.1 Segmentace obrazu

Segmentace obrazu je soubor různých objektů, kde dochází k analýze obsahu zpracovaných dat. Jeden z nejdůležitějších faktorů pro správnou segmentaci obrazu je charakter vstupního obrazu[28].

Hlavním úkolem segmentace obrazu je rozdělení vstupního obrazu na části, které mají souvislost s realitou. Důležitý faktor pro tuto množinu oblastí je vzájemná disjunktnost. Segmentace se dělí na kompletní a částečnou posle míry shody mezi nalezenými a skutečnými objekty. U segmentace kompletní je míra shody větší než u segmentace částečné[28]. I k této problematice se vyjádřil pan Ing. Josef Pavlíček, Ph.D. ve své práci.
„*Image segmentation is a basic task of image processing. It separates unnecessary "noise" objects from a photograph and creates a favourite image to be processed. Assuming the task*

*is to search for a herd of animals in the open air, the following knowledge base serves as a starting point:*

*• animals are in a free space, without growing vegetation*

*• the target are hooved animals (deer)*

*• the target is a herd - not an individual*

*• animals are scanned from a flying autonomous machine from a height of about 50 m*

*The task is to localize a herd on a pasture based on the data stream of images*

*To be able to locate the object, it was first needed:*

*• Select a favourite area (based on the knowledge base)*

*• Search the favourite area to for objects that resemble the animals (or herd)*[37]"

Kde tedy říká, že segmentace obrazu je základem pro zpracování obrazu. Odděluje šum a vytváří obraz, který je teprve poté zpracován. Jako příklad uvedl stádo zvířat na volném prostranství, kde se řeší pro rozpoznání tato problematika: zvířata ve volném prostoru bez vegetace, cílem jsou kopytnatá zvířata, cíl je stádo, zvířata jsou skenována z 50 metrů apod..

Šum je náhodné nebo závislé narušení obrazu, kde následkem tohoto narušení obrazu není možné správné rozpoznání. K náhodnému šumu se vyjádřil ve své disertační práci Ing. Petr Hanzlík, který se zaměřil na odstranění tohoto šumu z obrazu. „*Mediánový filtr vyhlazuje obrazovou funkci na základě výpočtu hodnoty mediánu ve stanoveném okolí (obvykle 3x3 či 5x5). Výsledný medián je pak přiřazen centrálnímu bodu okolí. Na rozdíl od filtru Gaussovského zachovává významné hrany a odstraňuje pouze drobné lokální změny v obrazové funkci. Tento filtr je tedy vhodný k odstranění náhodného šumu*[38]."

Pro kompletní segmentaci je nutná dostatečné množství znalostí o daném problému a je nutné použít vyšší úroveň zpracování. Je možné nalézt úlohy u kterých není nutné použít vyšší úroveň zpracování, poté se používá nižší úroveň zpracování obrazu. Pro použití s nižší úrovní mohou být objekty, které mají na pozadí konstantní jas[27].

U segmentace částečné je obraz rozdělen na několik částí. Hlavní výhodou je schopnost zpracovávat složité scény. Výsledkem pak bývá konečný soubor s homogenními oblastmi s určenými vlastnostmi barvy, jasu apod. Tyto oblasti se mohou vzájemně překrývat. Bohužel je nutné použít další postupy pro získání relevantních výsledků. Ovšem výhodou je rychlé snížení objemu zpracovaných dat[26].

Segmentace obrazu se dělí do tří základních metod dle použitých vlastností. První skupinu tvoří metody založené na globální znalosti obrazu či jeho částí. Tyto metody jsou

zastoupeny histogramem. Druhou skupinu tvoří algoritmy, které určují hranice mezi oblastmi a třetí skupinu tvoří algoritmy, které tyto oblasti vytvářejí[28].

Použité metody společného předzpracování a segmentace obrazu zahrnují[38]:

- Převod do tónů šedi na základě rozkladu světla
- Redukce šumu pomocí mediánového filtru
- Segmentace obrazu pomocí algoritmu globálního prahování

Tato procedura je vidět na obrázku 16.

*Obrázek 16 - Předzpracování obrazu a segmentace*[12]

Ve výsledném obrazu jsou odhaleny souvislé oblasti, které přesahují minimální stanovenou velikost. Předpokládá se, že objekt, který má být rozpoznán je co nejvíce na středu obrazu. Tento objekt je následně vybrán pro další zpracování. Vzhledem k poměrně jednoduché segmentaci obrazu se občas stává, že dochází k rozpadu objektu do několika částí kuli nevýrazné rostlině vůči pozadí. Na základě této problematiky Ing. Petr Hanzlík ve své disertační práci navrhl řešení[38]:

- Každá nálezná oblast O je popsána pomocí svého těžiště, maximální vzdálenosti mezi těžištěm a hranou objektu (d) a průměrem kruhu s plochou stejně velkou jako je plocha nalezené oblasti (ed).
- Výpočet poměru vzdáleností (r); r = d/ed.
- Je-li r> 0.7 (široký list) vzdálenost (d) je použita jako kritická vzdálenost (θ), v opačném případě je použit poloměr kruhu o stejném obsahu (ed).

[12] Zdroj: Disertační práce, Ing. Petr Hanzlík, Metody umělé inteligence v rozpoznávání rostlin

- Je-li Euklidovská vzdálenost mezi těžišti dvou oblastí menší než součet jejich kritických vzdáleností θ, budou oblasti sloučeny

Cílem je rozlišení rostliny od pozadí a následné smazání veškerých objektů z obrazu, které nejsou identifikovány jako rostlina.

### 4.4.2 Snímání obrazu

Zpracování dvourozměrného obrazu je matematicky zpracováno pomocí funkce dvou proměnných f(x,y), kde tyto hodnoty odpovídají intenzitě zachyceného signálu. Snímání obrazu je zaznamenání určitého pásma elektromagnetického zařízení za pomoci optického zařízení např. kamera nebo fotografický aparát. Pokud zařízení funguje na bázi analogového snímání, je nutné signál převést do digitální podoby[38].

Metody strojového vidění zpracovávající digitální obraz využívá spousta aplikací. Tyto systémy využívají metody snímkování. Ty dokáží zaznamenat především viditelné světlo, infračervené záření, ultrafialové záření a další elektromagnetické záření[38].

Důležité je, aby systémy pro pořizování obrazů fungovaly i za nepříznivých podmínek osvětlení. Klasické digitální fotoaparáty, které zaznamenávají obraz pomocí spektra RGB, jsou velmi citlivé na změny světelných podmínek. Je tedy nutné u navrhované aplikace pro rozpoznávání přidat speciální prvky, které jsou schopny danou oblast osvětlit nebo ztmavit, aby byly podmínky pro rozpoznání adekvátní.

# 5 Praktická část

Cílem práce je najít a vybrat vhodnou metodu pro implementaci konvolučních neuronových sítí a následně vytvořit UML diagram. V této kapitole budou postupně rozebrány návrhy konvolučních neuronových sítí, které by bylo vhodné implementovat. Kapitola bude taktéž obsahovat různé druhy UML diagramů a následně bude sestaven a popsán diagram pro mobilní aplikaci na rozpoznávání baby plevelů. Závěrem bude popis a shrnutí obou návrhů.

## 5.1 Postup snímaní rostlin

Konvoluční neuronové sítě využívají princip učení s učitelem, a tedy před samotným učením, je potřeba připravit vzorové příklady, které budou následně použity pro tvorbu modelu. Některé architektury neuronových sítí vyžadují i klasifikaci hledaných objektů nebo ruční označení objektů v obraze. Je nutné snímky nasnímat ze stabilní vzdálenosti, která bude následně nastavena i pro budoucí aplikaci. Objekt, na který se bude zaměřovat by měl být uprostřed obrazu. V případě dodržení postupu je možné strojové předzpracování obrazu.

## 5.2 Návrh konvoluční neuronové sítě

V této části kapitoly bude zhotoven návrh konvoluční neuronové sítě, který by mohl být následně implementován pro danou aplikaci.

### 5.2.1 Vstupní data

Jak již bylo řečeno, konvoluční neuronové sítě jsou učeny metodou s učitelem, tedy je nutné nejprve této síti předložit data na základě kterých bude síť učena. V rámci této práce, budou použity již existující data pro učení sítě, porovnání a zhodnocení před tím, než by byla vytvořena vlastní databáze dat, která by byla implementována přímo do aplikace. Konkrétně jde o dataset MNIST.

MNIST je databáze, která vznikla úpravou databáze složené ze vzorů ručně psaných symbolů, které byly vytvořené firmou *National Institute of Standards and Technology* – NIST. Dataset MNIST se skládá z dvou oddělených setů, kde první slouží pro učení neuronových sítí a je složen z 60000 černobílých obrazů ručně psaných číslic. Druhý set

slouží pro testování neuronové sítě a je složený z 10000 černobílých obrazů ručně psaných číslic.

Data set MNIST je volně k dispozici na oficiálních stránkách [30]. Ke stáhnutí jsou k dispozici čtyři soubory. Prvním souborem je „train-images-idx3-ubyte.gz“, který obsahuje vzory pro trénování neuronové sítě. Dalším souborem je „tran-labels-idx1-ubyte.gz“, které odpovídají jednotlivým tréninkovým vzorům a určují číselnou hodnotu, kterou daný tréninkový vzor reprezentuje. Třetím souborem je „t10k-images-idx3-ubyte.gz“, ten je složený z testovacích vzorů. Čtvrtým souborem je „t10k-labeles-idx1-ubyte.gz“, který umožňuje určit, zda výstupy neuronové sítě odpovídají očekávaným výsledkům.

### 5.2.2 Návrh architektury konvoluční neuronové sítě

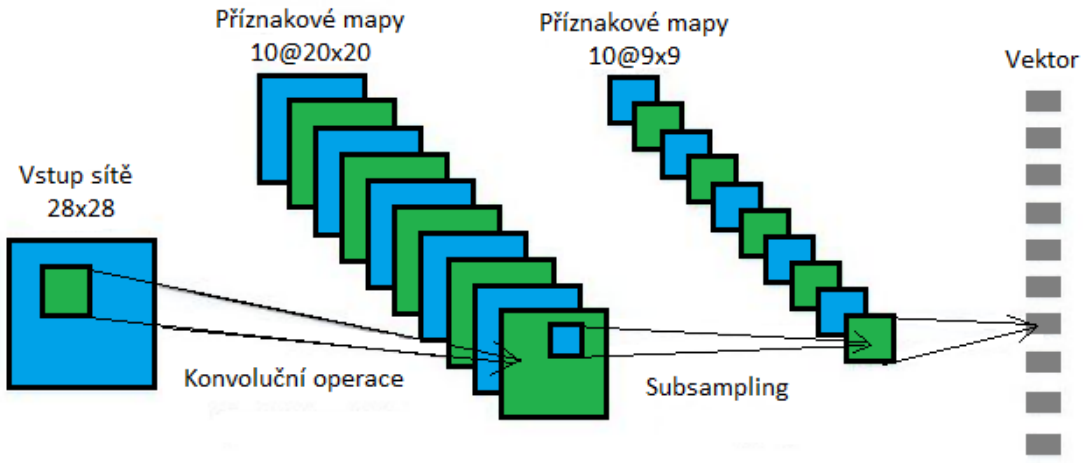
V rámci této práce budou vytvořeny tři různé architektury konvolučních neuronových sítí, na které budou aplikovány uvedená vstupní data. Výsledky výstupů jednotlivých sítí budou uvedené a porovnané mezi sebou.

Jako první navržená konvoluční neuronová síť je jednoduchý model sítě. Tento model je vidět na ilustračním obrázku 14. Je vidět, že vstupem sítě je obraz z MNIST databáze o rozměrech 28x28, který představuje 784 neuronů. Následující vrstvou je konvoluční vrstva, která je složena z deseti recepčních polí s rozměrem 20x20. Výstupem této konvoluční vrstvy je deset příznakových map o rozměrech 9x9. Tato vrstva je složena z 9x9x10 = 810 neuronů, (20x20+1)x10 = 4010 sdílených vah, kde hodnota +1 představuje bias a 810x(20x20+1) = 324810 spojení s předchozí vstupní vrstvou. Další vrstvou je subsamplingová vrstva, která počítá operace „max-pooling“ z příznakových map, které jsou výstupem předcházející konvoluční vrstvy. V tomto případě je na příznakové mapy konvoluční vrstvy aplikovaná subsamplingová vrstva o velikosti 9x9, která zmenší tyto příznakové mapy na velikost 1x1. Výsledkem aplikace subsamplingové vrstvy je znovu deset příznakových map, jen tentokrát s rozměrem 1x1 a obsahem 10 neuronů. Následně je na síť přiveden jednorozměrný vektor představující poslední plně propojenou vrstvu o velikosti 10 a obsahu 10x10 = 100 spojení. Tato vrstva představuje výstup konvoluční neuronové sítě tím, že určuje jednu z možných hodnot sítě, kterou může nabývat.

Přehled jednotlivých vrstev konvoluční neuronové sítě je vidět v tabulce 1.

| Typ vrstvy sítě | Počet příznakových map / neuronů | Velikost recepčního pole | Velikost příznakové mapy |
|---|---|---|---|
| Vstupní vrstva | 1 / 784 | 0 | 28 x 28 |
| Konvoluční vrstva | 10 / 810 | 20 x 20 | 9 x 9 |
| Subsamplingová vrstva | 10 / 10 | 9 x 9 | 1 x 1 |
| Výstupní vrstva | 10 neuronů | 0 | 0 |

*Tabulka 1 - Přehled rozměrů jednotlivých vrstev první konvoluční neuronové sítě*[13]




*Obrázek 17 - Ilustrační model první konvoluční neuronové sítě*[13]

Návrh druhé konvoluční neuronové sítě je možné vidět na obrázku 15. Jako druhý model konvoluční neuronové sítě je model, který je složen ze třech konvolučních a dvou subsamplingových vrstev. Vstupní vrstva je stejně jako u prvního modelu tvořena obrazem o velikosti 28x28. Za touto vstupní vrstvou se nachází první konvoluční vrstva, která je tentokrát tvořena šesti recepčními poli o rozměrech 5x5. Výstupem první konvoluční vrstvy je šest příznakových map s rozměrem 24x24. Z hlediska velikosti je první konvoluční vrstva složena z 24x24x6 = 3456 neuronů, obsahuje (5x5+1)x6 = 156 sdílených vah a 3456x(5x5+1) = 89856 spojení se vstupní vrstvou.

Za první konvoluční vrstvou se nachází první subsamplingová vrstva, která vykonává operaci „max-pooling“ na příznakových mapách z první konvoluční vrstvy. Velikost subsamplingové mapy je 2x2 a jejím výsledkem je šest zmenšených příznakových map s rozměrem 12x12.

Po první subsamplingové vrstvě následuje další konvoluční vrstva. Druhá konvoluční vrstva je složena z šestnácti recepčních polí o velikosti 5x5. Výstupem druhé konvoluční

[13] Zdroj: Vlastní zpracování

vrstvy šestnáct příznakových map, které mají velikost 8x8, spolu obsahují 8x8x16 = 1024 neuronů a (5x5+1)x6 = 2496 sdílených vah a 1024x(5x5x+1) = 26624 spojení s předešlou vrstvou.

Za druhou konvoluční vrstvou následuje poslední subsamplingová vrstva, která stejně jako první zmenšuje příznakové mapy na polovinu pomocí filtru s rozměrem 2x2 a operací „max-pooling“. Výsledkem je tedy šestnáct příznakových s rozměrem 4x4.

Na těchto šestnáct příznakových map je aplikována poslední konvoluční vrstva, která je složena ze 120 recepčních polí o rozměru 4x4. Výstupem této vrstvy je 120 příznakových map s rozměrem 1x1, která je složena ze 120 neuronů sa obsahuje (4x4+1)x6x16x120 = 195840 sdílených vah a 120x(4x4+1) = 2040 spojení s předešlou vrstvou.

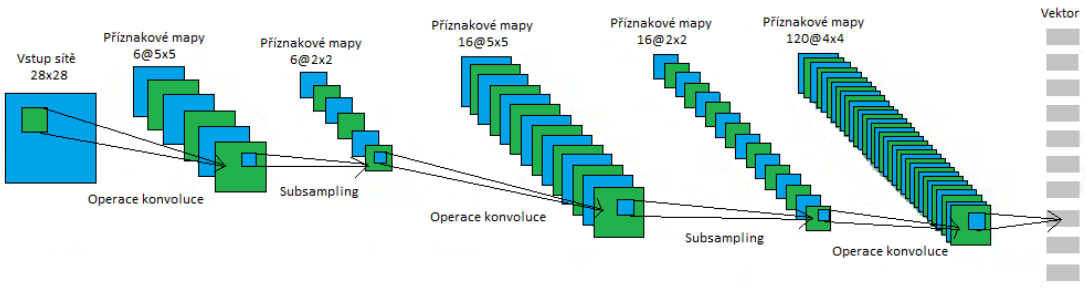
Poslední vrstvou je plně propojená vrstva složená z 10 neuronů. V této plně propojené vrstvě jsou všechny neurony propojeny se všemi neurony z předchozí vrstvy a tato vrstva obsahuje (120+1)x10 = 1210 spojení.

Přehled jednotlivých vrstev sítě je možné vidět v tabulce 2.

| Typ vrstvy sítě | Počet příznakových map / neuronů | Velikost recepčního pole | Velikost příznakové mapy |
|---|---|---|---|
| **Vstupní** | 1 / 784 | 0 | 28 x 28 |
| **První konvoluční** | 6 / 3456 | 5 x 5 | 24 x 24 |
| **První Subsamplingová** | 6 / 864 | 2 x 2 | 12 x 12 |
| **Druhá konvoluční** | 16 / 1024 | 5 x 5 | 8 x 8 |
| **Druhá subsamplingová** | 16 / 256 | 2 x 2 | 4 x 4 |
| **Třetí konvoluční** | 120 / 120 | 4 x 4 | 1 x 1 |
| **Výstupní** | 10 neuronů | 0 | 0 |

*Tabulka 2 - Přehled rozměrů jednotlivých vrstev druhé konvoluční neuronové sítě*[14]

[14] Zdroj: Vlastní zpracování




*Obrázek 18 - Ilustrační model druhé konvoluční neuronové sítě*[15]

Třetí typ architektury konvoluční neuronové sítě je vytvořena konvoluční neuronová síť, která se skládá ze čtyřech konvolučních a třech subsamplingových vrstev. Ilustrační model této konvoluční neuronové sítě je možné vidět na obrázku 16. Vstupem sítě je obraz s velikostí 28x28. Za touto vstupní vrstvou je první konvoluční vrstva, která je složena ze šesti recepčních polí o velikosti 2x2. Výstupem první konvoluční vrstvy je šest příznakových map, jejichž rozměry jsou 27x27. Počet neuronů v této vrstvě je 27x27x6 = 4374, počet sdílených vah je (2x2+1)x6 = 30 a počet spojení této vrstvy se vstupní vrstvou je 729x(2x2+1) = 3645.

Za první konvoluční vrstvou následuje první subsamplingová vrstva, která znovu, pomocí „max-pooling“, zmenšuje příznakové mapy, jenž jsou výstupem předchozí první konvoluční vrstvy. V této vrstvě je použitý filtr o velikosti 2x2, který zmenšuje příznakové mapy na velikost 13x13.

Za první subsamplingovou vrstvou se nachází druhá konvoluční vrstva. Ta je složena z šestnácti recepčních polí o velikosti 2x2. Výstupem této vrstvy je šestnáct příznakových map s rozměrem 12x12. Tato vrstva je složena z 12x12x16 = 2304 neuronů, obsahuje (2x2+1)x16x6 = 480 sdílených vah a 2304x(2x2+1) = 11520 spojení s předchozí vrstvou.

Následující vrstvou je druhá subsamplingová vrstva, která opět pomocí filtru 2x2 zmenšuje příznakové mapy z druhé konvoluční vrstvy na velikost 6x6.

Za druhou subsamplingovou vrstvou se nachází třetí konvoluční vrstva složená z recepčních polí o velikosti 2x2. Tentokrát je počet recepčních polí rovný hodnotě 120. Výstupem této konvoluční vrstvy je tedy 120 příznakových map s velikostí 5x5. V tomto případě je počet neuronů v této vrstvě rovný 5x5x120 = 3000, počet sdílených vah dosahuje

[15] Zdroj: Vlastní zpracování

hodnoty (2x2+1)x6x16x120 = 57600 a počet spojení s předchozí vrstvou je 3000x(2x2+1) = 15000.

Za třetí konvoluční vrstvou se nachází poslední subsamplingová vrstva, která zmenšuje příznakové mapy z předchozí vrstvy filtrem s velikostí 2x2 na příznakové mapy s velikostí 2x2.

Poslední konvoluční vrstva je složena ze 150 recepčních polí o velikosti 2x2. Výstupem této vrstvy je 150 příznakových map s velikostí 1x1. Tato vrstva je složená z 1x1x150 = 150 neuronů a obsahuje (2x2+1)x6x16x120x150 = 8640000 sdílených vah a 150x(2x2+1) = 750 spojení s předchozí vrstvou.

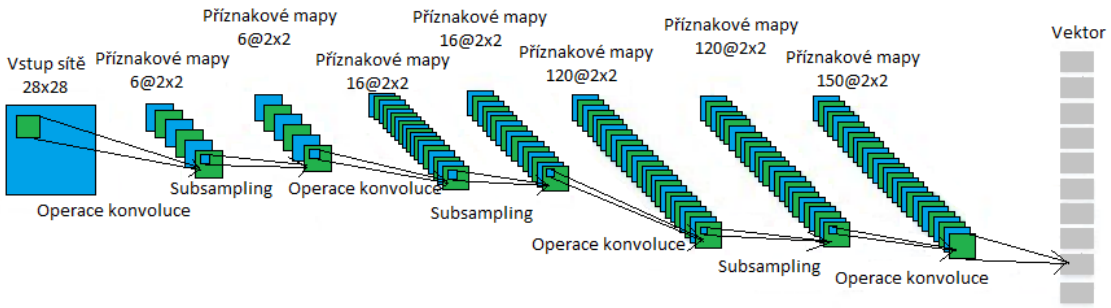
Poslední vrstvou modelu je plně propojená vrstva, která se skládá z 10 neuronů, kde stejně jako v předchozích případech každý neuron z této vrstvy zastupuje jedno číslo od 0 do 9. V této vrstvě je každý neuron propojený s každým neuronem z vrstvy předchozí, proto počet spojení dosahuje hodnoty 10x(150+1) = 1510.

Přehled jednotlivých vrstev tohoto modelu konvoluční neuronové sítě je uvedený v tabulce 3.

| Typ vrstvy sítě | Počet příznakových map / neuronů | Velikost recepčního pole | Velikost příznakové mapy |
|---|---|---|---|
| **Vstupní** | 1 / 784 | 0 | 28 x 28 |
| **První konvoluční** | 6 / 4374 | 2 x 2 | 27 x 27 |
| **První subsamplingová** | 6 / 1014 | 2 x 2 | 13 x 13 |
| **Druhá konvoluční** | 16 / 2304 | 2 x 2 | 12 x 12 |
| **Druhá subsamplingová** | 16 / 576 | 2 x 2 | 6 x 6 |
| **Třetí konvoluční** | 120 / 3000 | 2 x 2 | 5 x 5 |
| **Třetí subsamplingová** | 120 / 720 | 2 x 2 | 2 x 2 |
| **Čtvrtá konvoluční** | 150 / 150 | 2 x 2 | 1 x 1 |
| **Výstupní** | 10 neuronů | 0 | 0 |

*Tabulka 3 - Přehled rozměrů jednotlivých vrstev konvoluční neuronové sítě*[16]

[16] Zdroj: Vlastní zpracování




*Obrázek 19 - Ilustrační model třetí konvoluční neuronové sítě*[17]

### 5.2.3 Dostupné frameworky pro implementaci neuronových sítí v jazyku Java

Před samotnou implementací je potřeba vybrat vhodný Framework, který bude při implementaci použit. Pro jazyk Java bylo vytvořeno hned několik frameworků, které jsou určené pro implementaci neuronových sítí. Tato část práce je zaměřena na několik z nich a na závěr bude vybrán jeden, jako nejvhodnější pro implementaci do aplikace.

#### *5.2.3.1 JOONE*

JOONE neboli Java Object Oriented Neural Engine je open source Framework určený pro implementaci neuronových sítí, ale může být použit i na další věci. Je k dispozici od roku 2005. Jeho architektura je založena na komponentách, které se mohou znovu použít, popřípadě spojovat je s jinými komponentami. U JOONE je k dispozici grafický workbench pro tvorbu jednoduchých modelů a jejich trénování. Nevýhodou JOONE je větší komplexnost a v současné době již není tolik podporovaný, proto je nevýhodné ho použít v moderních technologiích[31].

#### *5.2.3.2 Deeplearning4J*

Eclipse Deeplearning4J je open-source, vytvořený pro hluboké učení v Javě a Scale lidmi ve firmě Skymind, která se zabývá business inteligence a podnikovým softwarem v San Francisku Adamem Gibsonem. Deeplearning4J zahrnuje implementace omezeného Boltzmannova stroje, autoencodéru, rekurzivní sítě (word2vec, doc2vec a GloVe) apod.. Tyto algoritmy zahrnují distribuované paralelní verze, které se integrují s Apache Hadoop a

[17] Zdroj: Vlastní zpracování

Spark. Je to software s otevřeným zdrojovým kódem, který se vydává pod Apache License 2.0[32].

#### *5.2.3.3 Encog*

Encog je open source Framework, který je mimo Javy implementovaný i pro jazyk C# a je vyvíjen od roku 2008. Je zaměřený na implementaci strojového učení a mimo podpory implementace různých druhů neuronových sítí, také podporuje algoritmy pro Support Vector Machine, generické programování nebo Skrytý Markovův model. Encog umožňuje použití GPU při implementaci pro zrychlení jednotlivých operací a procesového času. K dispozici je i grafický workbench, pro pomoc při tvorbě jednotlivých modelů a jejich trénování[33].

#### *5.2.3.4 Neuroph*

Neuroph vznikl roku 2008 jako diplomová práce na univerzitě v Bělehradě a jeho autorem je Zoran Sevarac. Skládá se z knihoven, které odpovídají základním konceptům neuronových sítí. Příkladem podporovaných architektur neuronových sítí v Neurophu jsou Perceptron, Vícevrstvý perceptron s učením zpětného šíření, Hopfieldova, Kohenenova popřípadě Hebbianova síť. Jeho součástí je také grafický nástroj, ve kterém je možné tyto sítě vytvořit, trénovat i testovat. Aktuální verze je Neuroph 2.9 a podporuje několik funkcí pro práci s konvolučními neuronovými sítěmi, zatím pouze v rámci grafického nástroje, ale předpokládá se tuto funkci rozšířit[34].

### 5.2.4 Grafické jednotky při implementaci v jazyku Java

V současnosti hodně programovacích jazyků nabízí mimo běžného programování, kdy program běží na procesoru (CPU), také možnost spouštění programů nebo jejich částí pomocí grafické jednotky (GPU). Samotná GPU se typicky skládá z množství procesorů, které sami o sobě nejsou extrémně výkonné, ale uplatňuje se tu mutitasking, kde několik různých operací běží ve stejném čase na různých procesorech. Využití GPU má tedy smysl v případě, kdy program stráví velké množství času výpočtem složitých operací, kdy použití GPU umožňuje zrychlení tohoto procesu.

Programovatelné grafické jednotky se začali používat až koncem 90 let 20. století, před tím byl GPU využíván jako grafický akcelerátor, který podporoval výpočty jen specifických funkcí. V současnosti je možné použít dva způsoby programování s využitím GPU a to OpenCL a CUDA.

#### *5.2.4.1 OpenCL*

OpenCL neboli Open Computing Language je první způsob využití GPU při implementaci. Jedná se o první standart pro multi-platformové paralelní programování. OpenCL je vytvořen společností Khronos Group v roce 2008. Program napsaný v OpenCL může být spuštěn na CPU, GPU i programovatelných hradlech (FPGA), případně jiném hradwaru. OpenCL poskytuje paralelní programování a v současnosti je dostupný pro produkty firem jako je AMD, Apple, IBM, Intel, Nvidia nebo Samsung.

OpenCL se skládá z několika sdílených objektů, které umožňují spuštění výpočtů, přičemž instalační driver musí být nainstalovaný na každé platformě, aby mohl tuto platformu identifikovat. Výhodou je možnost spustit program na podobných platformách, které nemusí být hardwarově úplně stejné[35].

OpenCL je složen z knihovny implementované v jazyce C, C++ a překladače, který umožňuje spouštění výpočtů na jednotlivých platformách. Pro využití v jiných programovacích jazycích než C++ kupříkladu Python nebo Java existují různé API knihovny třetích stran. Konkrétně pro Javu jsou knihovny uvedeny níže.

1) **Aparapi** je knihovna, která umožňuje vývojářům využívat výhody GPU a akcelerovaných procesorových jednotek (APU) namísto CPU při spuštění paralelních operací v programu. Aparapi pracuje tak, že za běhu programu konvertuje javovský bytekód do OpenCL a následně ho spouští pomocí GPU. V případě, že Aparapi není schopná spustit operace na GPU, jsou tyto operace spouštěné pomocí pracovních vláken Java Thread Pool (JTP). Výhodou je, že není závislá na výrobcích grafických karet, protože pracuje s OpenCL. Původně byla vyvinutá firmou AMD, ale v současnosti je licencovaná jako open source.
2) **JOCL** je knihovna, která pracuje s OpenCL tak, že umožňuje aplikacím běžících na javovském virtuálním stroji (JVM) využívat funkce paralelně, tedy poskytuje vyšší výkon a spouštění na různých platformách. Knihovna JOCL je složena ze

statistických metod, které sémanticky odpovídají funkcím z OpenCL, jen jsou syntakticky přizpůsobeny jazyku java.

3) **CUDA** neboli Parallel Computing Platform je platforma a programovací model vyrobený firmou NVIDIA. Je implementován pomcí GPU a umožňuje vývojářům přímí přístup k výpočetním jednotkám jako jsou virtuální instrukce nebo paměť.
   Výhodou CUDA je sdílení paměti mezi jednotlivými vlákny nebo možnost čtení dat z různých částí paměti. Nevýhodou je, že program funguje pouze pro grafické karty, které tuto platformu obsahují.
   CUDA je dostupná jako rozšíření jazyku C, C++ nebo Fortan, kde je kompilována pomocí příkazu „*nvcc*“. V případě jazyka Java je možné implementovat CUDA grafické jednotky pomocí několika knihoven. Tyto knihovny jsou uvedeny níže.
4) **Rootbeer** knihovna umožňuje uživateli spustit program implementovaný v jazyce Java na grafické jednotce CUDA společnosti NVIDIA. Rootbeer funguje tak, že javovský bytekód je převeden do CUDA programovacího jazyka, z kterého je spuštěný na grafické jednotce. Autorem této knihovny je Phil Szeliga ze Syrakúzské univerzity. Jeho cílem bylo vytvořit knihovnu, jenž umožňuje využití vysokovýkonných grafických jednotek pro implementaci v jazyce java s minimálním zásahem do procesů.
5) **JCUDA** je primárně určená pro interakci s jinými CUDA knihovnami. Umožňuje uživateli volat v jazyce java CUDA funkce napsané v jazycích C nebo C++. JCUDA tedy uživateli neumožňuje vytvořit vlastní jádra a následně ho zpracovat, v tomto případě je vhodnější použít knihovnu Rootbeer.
6) **Encog** je famework, který byl zmíněný už v předchozí kapitole jako Framework určený k implementaci neuronových sítí. V současnosti se Encog snaží rozšířit svoji funkčnost tím, že začíná umožňovat spouštění programů pomocí grafických jednotek modelu CUDA. V současnosti podporuje spouštění jen některých operací pomocí GPU a je součástí nástroje „*Encog for C“*, který primárně slouží pro implementaci neuronových sítí v jazyce C.

Na základě dostupných frameworků pro implementaci konvolučních neuronových sítí v programovacím jazyce Java vyšel pro mobilní aplikaci jako nejvhodnější framework Deeplearning4J a Neuropf. Použití Neurophu je nejvýhodnější z toho důvodu, že Framework umožňuje použití vestavěných funkcí pro implementaci konvolučních neuronových sítí.

Nevýhodou tohoto frameworku je, že v aktuální verzi, kterou je Neuroph 2.94, není možné pro práci s konvolučními neuronovými sítěmi použít příslušný grafický nástroj, ale jednotlivé konvoluční neuronové sítě je nutné implementovat manuálně.

## 5.3 Implementace vstupních dat

Před vytvořením samotné konvoluční sítě je nutné implementovat dataset, který slouží pro učení a následné testování konvoluční neuronové sítě. Jak už bylo uvedeno výše, pro učení a testování sítě bych zvolila nejprve použít dataset MNIST pro odladění celé aplikace, který před samotným vložením do neuronové sítě musí být upravený na potřebný tvar.

U frameworku Neuroph se pro vytváření datasetu používá třída DataSet. Každý DataSet je složený z řádků, kde každý řádek reprezentuje jeden vzor, který je aplikovaný do konvoluční neuronové sítě. Tento řádek bývá definován prostřednictvím třídy DataSetRow a skládá se ze vstupních dat a požadovaného výstupu. V případě datasetu MNIST je tento dataset uložen ve čtyřech souborech, kde obrázky a k nim příslušné labely se nachází v různých souborech, jejichž struktura je uvedena na webových stránkách datasetu MNIST. Je nutné tyto soubory upravit před samotným vytvořením datasetu.

Pro implementaci vytvoření datasetu z MNIST souborů je potřeba vytvořit třídu a jejím výstupem bude upravený dataset, který může být následně předložen konvoluční neuronové síti. Tato třída se skládá z několika metod, jenž umožňují vytvoření datasetu. Jedna z metod obsahuje cesty k souborům s obrazy a k nim příslušné datasety jako vstupní parametry. Tato metoda pomocí další metody může načítat data ze souborů, odstraňovat přebytečná data a vracet seznam, který se skládá z bytových dat obrazů a k nim příslušných labelů.

Další z metod má za cíl vytvořit vhodně upravený dataset, který bude předložen konvoluční neurnové síti. Tato metoda je definovaná dvěma parametry, kde první parametr je seznam MNSIT obrazů a druhým je uživatelem zvolený počet obrazů, které budou ze seznamu použité. Jelikož jednotlivé řádky definovaného datasetu mají vstupní i očekávaná data v datovém typu duble, je nutné překonvertovat již upravené obrazy, jejichž data jsou v datovém typu byte. V rámci této metody je tedy brán uživatelen zvolený počet obrazů ze seznamu, následně jsou bytové data obrazu převedeny na hodnotu double a spolu s labelem jsou uloženy jako řádek datasetu do proměnné DataSet.

Poslední metoda řídí předešlé dvě metody. Parametry této metody jsou uživatelem zadané cesty k souborům s obrazy a labely, dalším parametrem je uživatelem definovaný počet obrazů a labelů z datasetu MNIST, které budou použity pro konvoluční neuronovou síť.

## 5.4 Vytvoření konvoluční neuronové sítě

Po načtení potřebných datasetů pro učení a testování sítě, je možné pokračovat v programu samotnou implementací konvoluční neuronové sítě. V prostředí Neuroph je konvoluční síť definovaná prostřednictvím třídy ConvolutionalNetwork. Po vytvoření objektu, který představuje konvoluční neuronovou síť se postupně přidávají jednotlivé vrstvy sítě. Pro každý typ vrstvy obsahuje Neuroph specifické třídy.

Konvoluční neuronová síť je definovaná prostřednictvím třídy Convoluitonal Layer. Při vytváření objektu pro konvoluční vrstvu, je nutné definovat několik atributů. Prvním atributem je vrstva, na kterou bude konvoluční vrstva aplikovaná, dalším atributem je recepční pole, které je definované v rámci třídy Kernel a je určený svými rozměry. Posledním atributem je počet recepčních polí použitých v této vrstvě.

Příklad vytvoření konvoluční vrstvy:

```
ConvolutionLayer konvolucniVrstva1 = new ConvolutionLayer
(vstupniVrstva, konvolucniJadro, 6);
```

Další vrstvou, která je použita v konvolučních neuronových sítích je vrstva subsamplingová. Tato vrstva je v prostředí frameworku Neuroph je definovaná prostřednictvím třídy PoolLayer. Stejně jako u konvoluční vrstvy i při vytváření objektu pro subsamplingovou vrstvu je nutné definovat několik atributů. První atribut definuje vrstvu konvoluční neuronové sítě, na kterou bude aplikovaná subsamplingová. Druhým atributem při vytváření subsamplingové vrstvy je jádro, které bude aplikované na definovanou vrstvu. Toto jádro je definované pomocí třídy Kernel a je určené svými rozměry.

Příklad vytvoření subsampligové vrstvy:

```
PoolingLayer subsamplingovaVrstva1 = new PoolingLayer
(konvolucniVrstva1, subsamplingoveJadro);
```

Poslední vrstvou, která je použitá v navrhnutých konvolučních neuronových sítí, je plně propojená vrstva, která slouží jako výstup konvoluční neuronové sítě. Tato vrstva je v prostředí frameworku Neuroph definovaná v třídě Layer a při vytváření příslušného objektu, uživatel definuje jako atributy počet neuronů v této vrstvě a vlastnosti těchto neuronů.

Příklad vytvoření plně propojené vrstvy:

```
Layer vystupniVrstva = new Layer (10, neuron);
```

Jak už bylo řečeno, před vytvořením samotné plně propojené vrstvy, je nutné vytvořit a definovat neurony použité v této vrstvě. Neurony jsou v prostředí Neuroph definované prostřednictvím třídy NeuronProperties.

Po vytvoření jednotlivých vrstev je nutné tyto vrstvy přidat do již vytvořené konvoluční neuronové sítě. Přidávání vrstev je umožněno pomocí určité metody, která bývá součástí třídy ConvolutionalNetwork.

Po přidání jednotlivých vrstev do konvoluční neuronové sítě je nutné sousední vrstvy mezi sebou propojit. Toto propojení se vytvoří metodou, která bývá součástí třídy ConvolutionalUtils a jako parametry metody jsou dosazené sousední vrstvy konvoluční neuronové sítě, které mají být spojeny.

Příklad použití spojení dvou vrstev konvoluční neuronové sítě:

```
ConvolutionalUtils.plnePropojeneDveVrstvy(vstupniVrstva, konvolucniVrstva1);
```

Před zahájením samotného učení konvoluční neuronové sítě je nutné nastavit vstupní a výstupní neurony sítě. Parametry metod použitých pro toto nastavení jsou neurony vstupní a výstupní vrstvy sítě.

Po nastavení struktury konvoluční neuronové sítě je v programu nastavený typ učícího algoritmu, který bude v síti použitý. V případě implementovaných konvolučních neuronových sítí je pro učení použit algoritmus BackPropagation upravený pro konvoluční neuronové sítě, který v prostředí frameworku Neuroph zabezpečuje třída ConvolutionalBackpropagarion.

Příklad nastavení učení:

```
konvolucniSit.nastaveniUceniSite (new ConvolitionalBackpropagation());
```

Po nastavení typu učení konvoluční neuronové sítě následuje samotné učení sítě. Toto učení je realizované pomocí metody učení, která je součástí třídy ConvolutionalNetwork.

```
konvolucniSit.learn(trenovaciData);
```

Výstupem implementovaných konvolučních neuronových sítí je výsledná chybovost konvoluční neuronové sítě a čas, který síť strávila učením. Na základě těchto výsledků bude vybrána správná konvoluční neuronová síť, která bude nejvhodnější pro navrhovanou aplikaci.

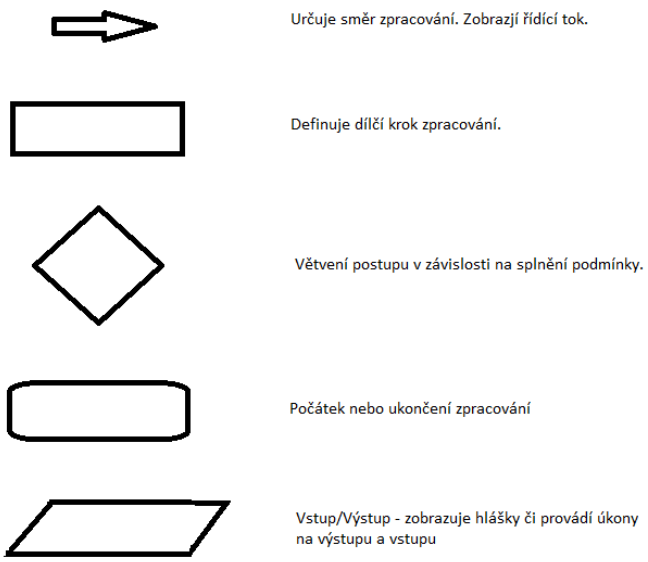
## 5.5 Rozšíření modelu na další rostliny

Jak bylo řečeno výše, důležité je nejdříve vybrat správnou architekturu, která by byla vhodná pro aplikaci a následně na to aplikovat vhodnou databázi rostlin. Při konstrukci modelu, který dokáže rozpoznat rostliny v rané fázi růstu je naučená síť schopná rozpoznat specifické rostliny na základě určitých příznaků v jedné z částí architektury sítě. Tato část je následně vybrána za pomoci vícevrstvých perceptronových sítí. Vzhledem k tomu, že rostliny v raném stádiu jsou si velmi podobné lze předpokládat, že naučené příznakové mapy v konvolučních vrstvách popisují rozpoznávací příznaky a je tedy možné tuto část konvoluční sítě znovu využít pro rozpoznání dalších rostlin, které nebyly zahrnuty do učení sítě.

Pro znovupoužití sítě zůstává část pro vybrání příznaků zachována, ale je nutné změnit konstrukci výstupních vrstev sítě, aby bylo možné zařazovat do většího množství kategorií. Ing. Petr Hanzlík uvádí ve své disertační práci metodu, jak co nejrychleji síť upravit. „*Aby toto bylo realizováno, jsou odebrány neurony v klasifikační části sítě a váhy které z nich vycházejí. Po přidání neuronů potřebných pro klasifikaci do odpovídajícího počtu kategorií jsou váhy v konvoluční části sítě zmraženy a v celé síti se pak pouze učí váhy dopředných neuronových sítí v klasifikačních vrstvách. Díky této metodě je možné rychle a s relativně nízkými náklady rozšiřovat aplikační doménu, ve které je síť použitelná.*[38]“

## 5.6 Vývojové diagramy

Vývojový diagram (Flow chart) je grafické znázornění určitého procesu, postupu nebo algoritmu. Cílem vývojového diagramu je znázornění toku kroků procesu od začátku do konce grafickým způsobem. Grafický způsob se dá mnohem lépe pochopit než slovní popis. Vývojový digram používá jednoduché geometrické symboly, které představují různé elementy popisovaného procesu. Klíčové prvky jsou popsány níže[39].




*Obrázek 20 - Klíčové prvky vývojového diagramu*[18]

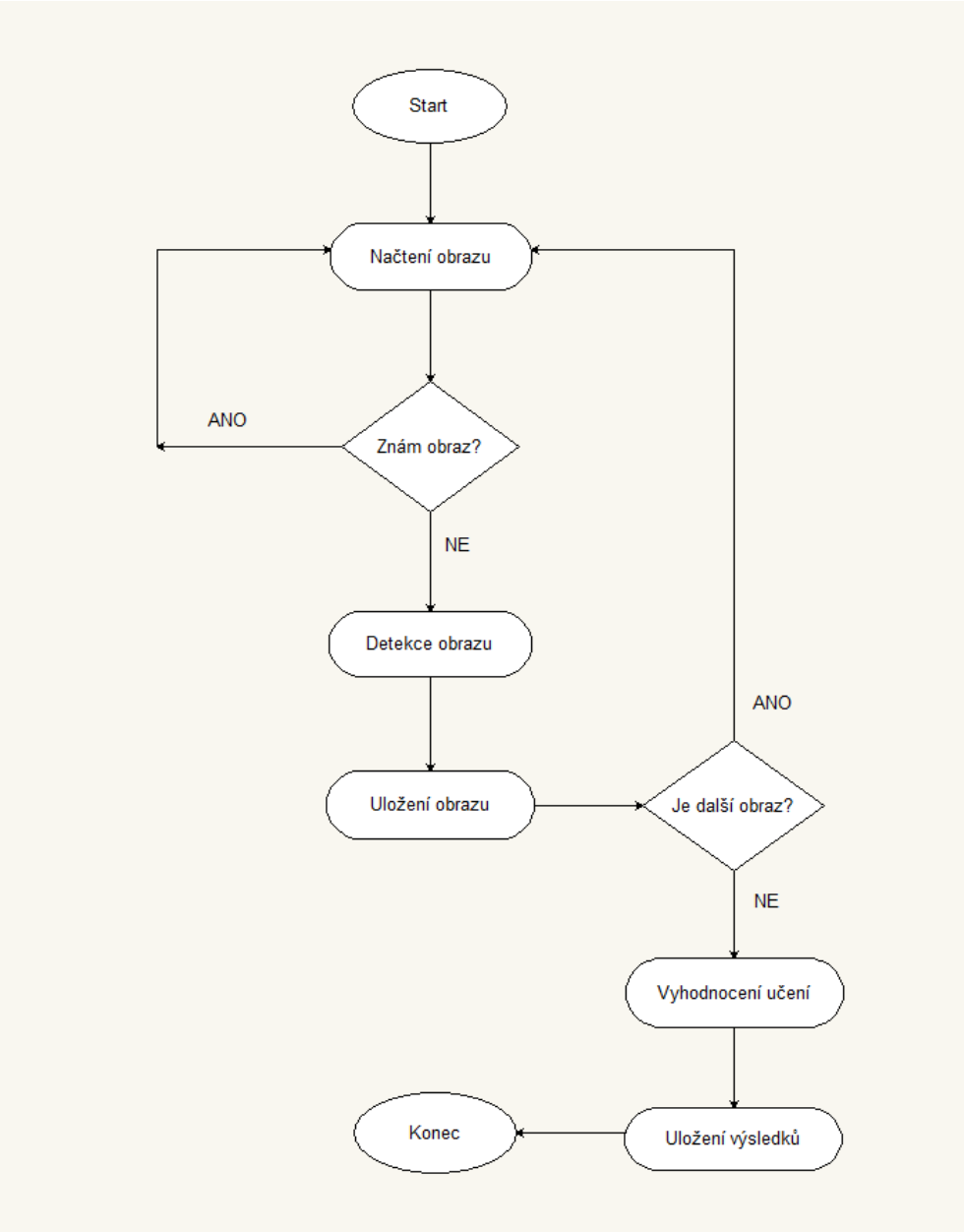
První strukturovaná metoda pracovního toku byla vytvořena Frankem Gilbrethem v roce 1921. Používá se převážně k popisu procesu, pracovního postupu, výrobního postupu nebo různých algoritmů.

V této práci jsou vytvořeny dva vývojové diagramy. První se týká učení sítě, kde vstupuje právě jeden obraz. Tento obraz je porovnán s databází již naučených obrazů, vyhodnotí se, zda se shoduje s některým z nich, pokud ano načte se další obraz k učení. Jestliže žádný další obraz k učení není proces učení vyhodnotí a uloží výsledky učení a následně se ukončí. Pokud další obraz je, cyklus načtení a rozpoznání obrazu se opakuje. Jestliže obraz není znám, poté proces obraz detekuje a uloží do databáze. Následně se

[18] Zdroj: Vlastní zpracování

přechází na cyklus, zda je další obraz k dispozici či nikoliv a po vyčerpání veškerých možných obrazů k učení se proces vyhodnotí, uloží a ukončí.

Jak vypadá vývojový diagram tohoto procesu učení je možné vidět na obrázku 21.



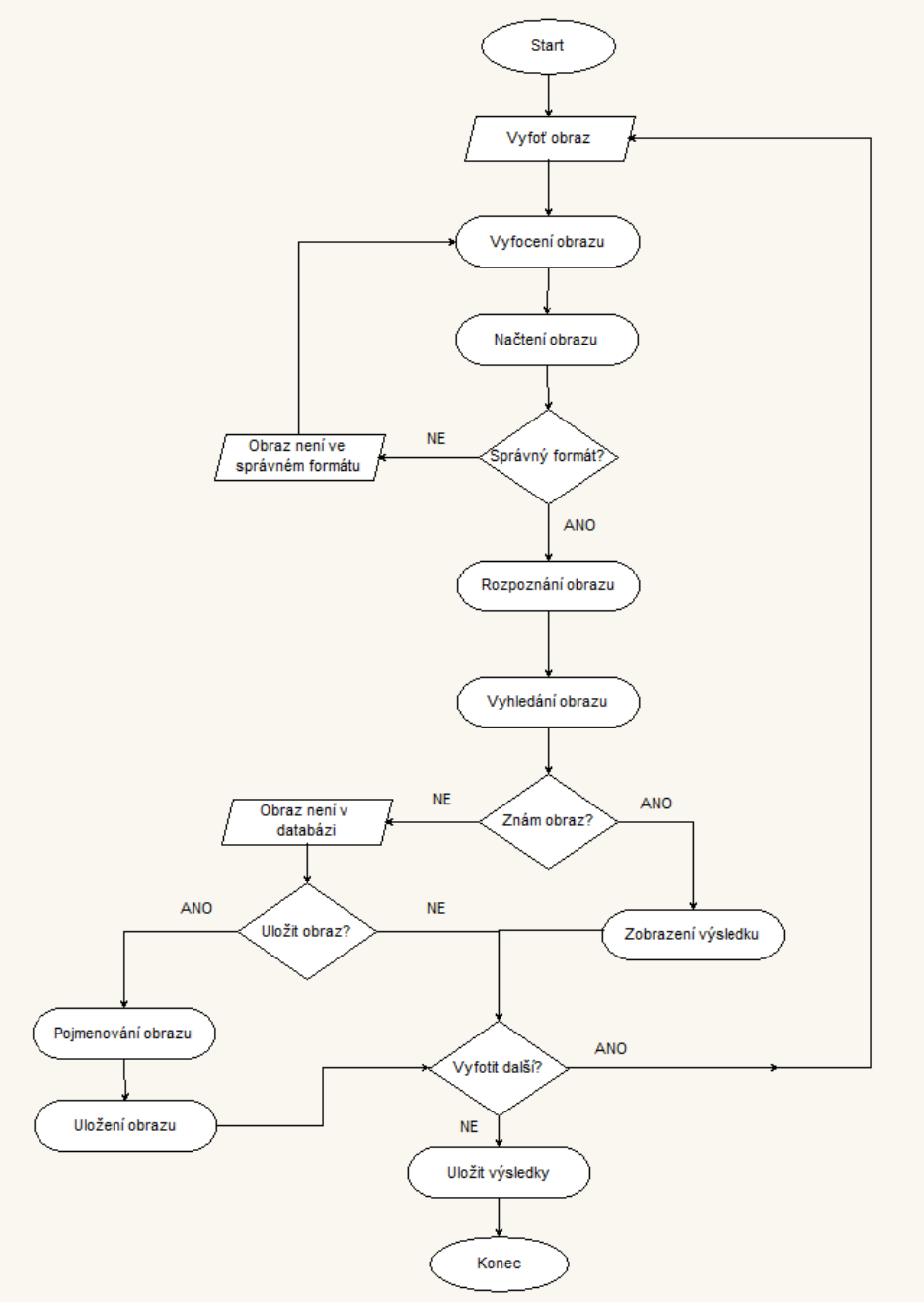

*Obrázek 21 - Vývojový diagram učení sítě*[19]

Druhý vývojový diagram se týká vyhledávání v databázi na základě vyfocení obrazu a porovnání s databází. Jako vstup je výzva uživatele k vyfocení obrazu, který má být

[19] Zdroj: Vlastní zpracování

rozpoznán. Po vyfocení obrazu uživatelem (tudíž provedení aktivity) se obraz načte do aplikace a pomocí algoritmů se zhodnotí správnost formátu (zda je objekt na středu, správně osvětlen apod. (jak bylo řešeno výše). Po zhodnocení správného formátu je vyhodnocen výsledek. Pokud formát nevyhovuje je to oznámeno výpisem hlášky uživateli a dochází k zacyklení kde se musí obraz vyfotit znovu dokud nebude ideální k porovnání. Pokud bude ve správném formátu dochází k rozpoznání obrazu a následném vyhledání v databázi naučených obrazů. Pokud obraz v databázi je, uživateli je zobrazen výsledek a následně může fotit další obraz. Pokud obraz není v databázi, je uživateli tato informace poskytnuta a nabídnuta možnost obraz uložit do databáze jako nový k příštímu rozpoznání. Pokud uživatel nechce obraz uložit může fotit další obraz. Pokud obraz uložit chce, dochází následně k pojmenování, rozřazení a uložení obrazu. Následně bude uživateli nabídnuta možnost fotit další obraz. Pokud uživatel chce fotit další obraz, algoritmus se zacyklí a začíná celý proces od začátku. Pokud nechce fotit další, veškeré informace se uloží a proces se ukončí.

Vývojový diagram tohoto procesu vyhledávání je možné vidět na obrázku 22.




*Obrázek 22 - Vývojový diagram vyhledání a porovnání*[20]

## 5.7 Use Case diagram

Use Case diagram (neboli diagram případu užití) ukazuje chování systému přesně tak, jak ho vidí uživatel. Jeho účelem je zjednodušeně popsat co je od systému očekáváno. Digram popisuje, co má systém umět, ale už neukazuje, jak to bude dělat. Celkově se

[20] Zdroj: Vlastní zpracování

diagram skládá pouze ze třech komponent. První jsou případy užití, druhý aktéři a třetí vztahy mezi nimi.

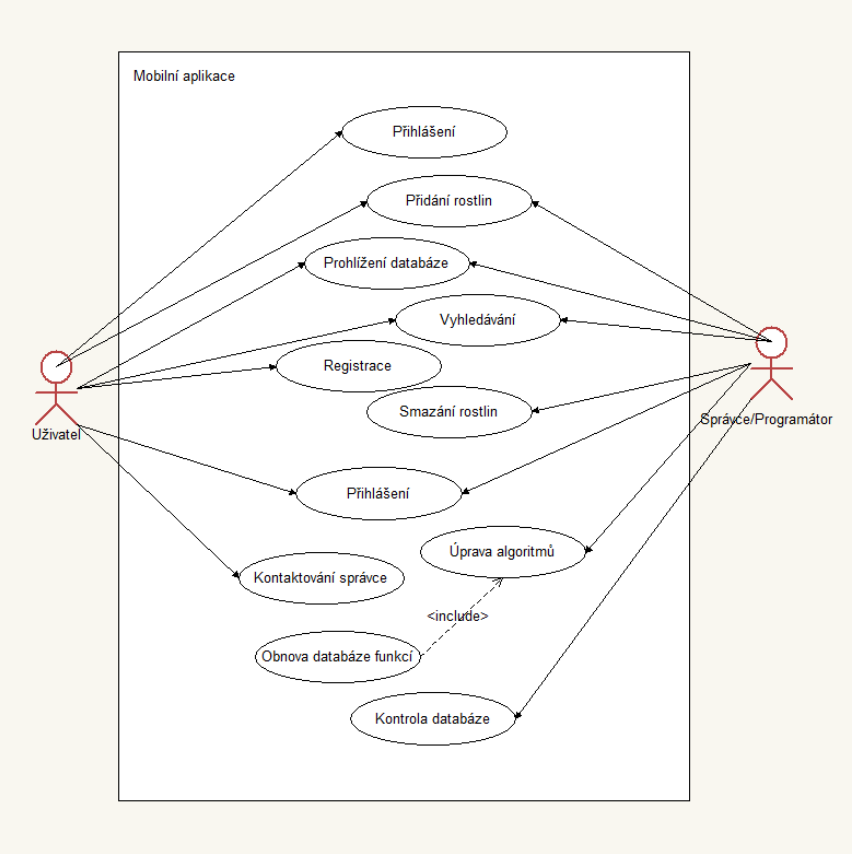
Případ užití je několik akcí, které vedou k dosažení cíle. Může se jednat o registraci nového uživatele, přihlášení stávajícího uživatele, přidání obrázků či jiných činností. Případy užití jsou zakreslovány elipsou.

Aktér je role, která zastává buď uživatele, externí systémy nebo třeba čas a komunikuje s případy užití. Aktéři se vyznačují postavou a většinou se v diagramu vyznačují mimo systém.

V Use Case diagramu se mohou objevit vazby <include> a <extend>. Tyto vazby se objevují mezi případy užití, nikoli mezi aktérem a případem užití. Vazba <include> se spouští vždy, pokud je spuštěn případ, na který je napojena. Vazba <extend> se používá velmi vzácně ve specifických případech, jedná se o obrácený směr závislosti.

Vzhledem k náročnosti navrhované aplikace byl vytvořen základní a velmi zjednodušený use case diagram. Zde jsou zobrazeny hlavní dva aktéři a základní případy užití. Od tohoto diagramu by se mohl odvíjet doplňující use case a následně odvíjet kompletní architektura aplikace.

Navržený use case diagram je možné vidět na obrázku 23.




*Obrázek 23 - Use Case diagram mobilní aplikace*[21]

## 5.8 Class Diagram

Class Diagram (neboli diagram tříd) je diagram implementace. Class diagram je poslední štace před samotným psaním kódu a z toho vyplívá že musí být úplný a plně funkční. Měl by obsahovat všechny třídy co bude obsahovat aplikace, třídy by měli mít všechny atributy a metody. Zároveň v diagramu tříd jsou všechny atributy a metody napsané již v jazyu, ve kterém se následně bude aplikace programovat.

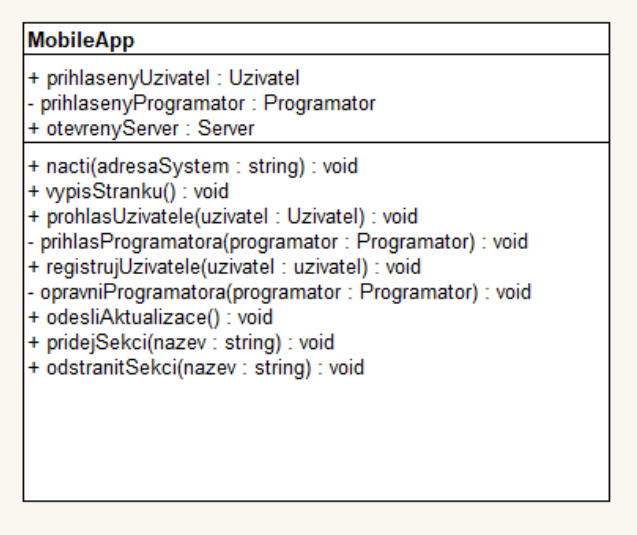
Class diagram je psaný v rámci obdélníků a vazeb mezi nimi. V první části obdélníku je název třídy. V druhé části jsou vypsány atributy s datovými typy a před každým atributem je uveden modifikátor přístupu kde platí:

- - (mínus) - Privátní atribut
- + (plus) – Veřejný atribut
- # (hash kříž) – Protected atribut
- ~ (tilda) – Atribut viditelný v rámci balíku

[21] Zdroj: Vlastní zpracování

Mezi název atributu a datový typ se píše dvojtečka. V třetí části jsou zobrazeny metody. Ty se píší obdobně jako atributy.

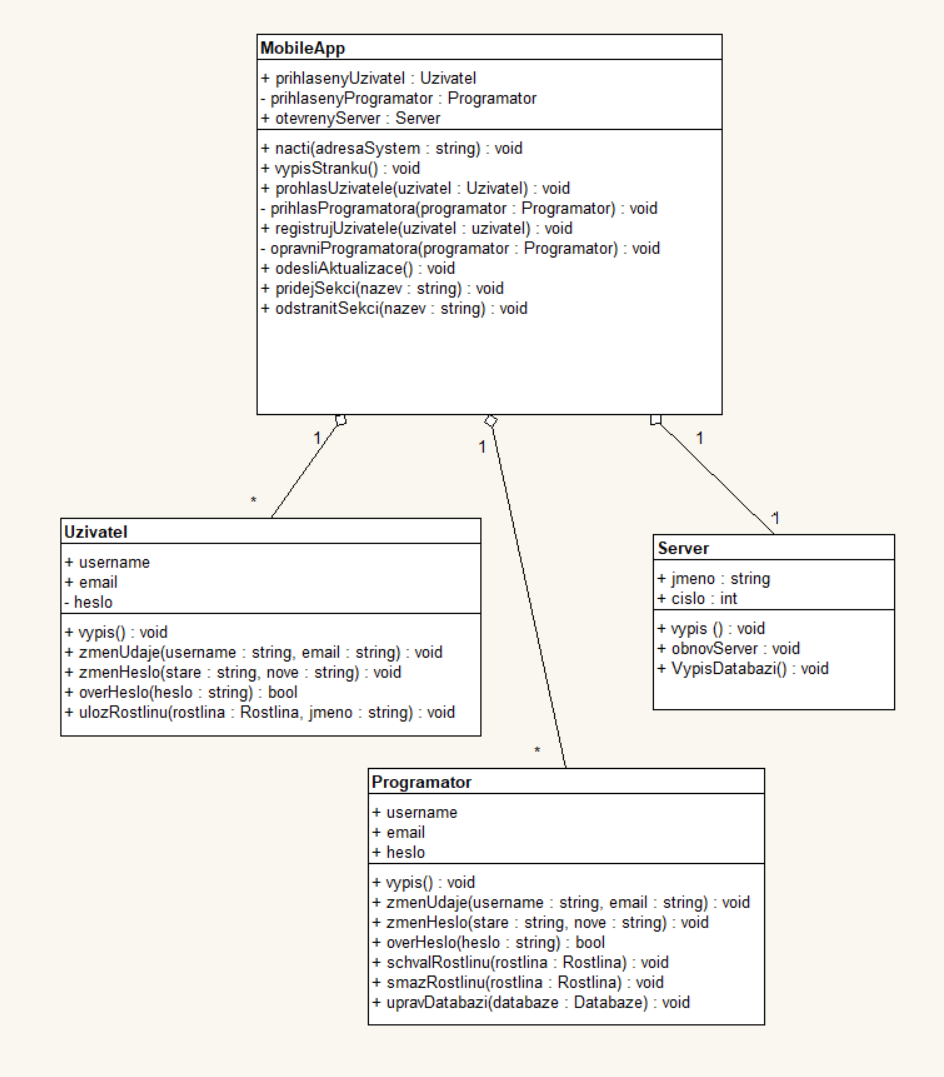
U mobilní aplikace na rozpoznávání rostlin by figurovala hlavní třída MobileApp, která by obsahovala příslušné atributy a metody, a na kterou by byly připojeny třídy Uživatel, Programátor a Server. Ukázku, jak by mohla taková třída vypadat je vidět na obrázku 24.




*Obrázek 24 - Třída MobileApp v Class Diagramu*[22]

Třída MobileApp může obsahovat několik uživatelů i několik programátorů ovšem pouze jeden server, na kterém bude uložena databáze rostlin a kterou bude programátor spravovat. Ukázka, jak by mohli v Class Diagramu vypadat další tři třídy a jejich vazby je možné vidět na obrázku 25.

[22] Zdroj: Vlastní zpracování




*Obrázek 25 - Class Diagram - hlavní třídy a jejich vazby*[23]

Na následující třídy by navazovali další třídy. Na třídu Server by následně navazovala třída Databáze, na ní třída Sekce a na ní třída Rostliny, kde by byly uloženy jednotlivé rostliny. Aplikace by tedy pro rychlejší rozpoznávání mohla hledat nejdříve v sekcích (kde by uživatel před vyhledáváním zadal o jakou sekci rostliny se jedná) a následně teprve v jednotlivých rostlinách a tím by se zkrátila doba zpracování požadavku.

Dále by zde mohla být třída <enum> - StavRostliny, která by ukazovala, zda programátor schválil či smazal přidané rostliny od uživatelů. Dále by zde mohla být třída Vzkaz, kde by uživatel přidával připomínky k aplikaci a Programátor by se mohl podívat na možnou iniciativu k vylepšení aplikace nebo třída View, která by dovolovala uživateli nahlédnout do kompletní databáze rostlin.

[23] Zdroj: Vlastní zpracování

# 6 Závěr

Tato diplomová práce je zaměřena na neuronové sítě, konvoluční neuronové sítě, rozpoznání obrazu, návrh konvolučních neuronových sítí a návrh počáteční architektury aplikace pomocí UML diagramů.

Na začátku této práce byla řešena problematika neuronových sítí, jejich architektury a principu fungování. Následně se práce zaměřuje na různé modely a typy neuronových sítí které jsou dopodrobna prozkoumány.

V další části práce jsou řešeny konvoluční neuronové sítě, které jsou v současnosti jedny z nejvhodnějších modelů pro identifikaci objektů z obrazu, což v případě návrhu mobilní aplikace pro rozpoznání plevelů v počátečním stádiu růstu vyhovovalo. V této části práce jsou také pospané různé příklady použití tohoto typu neuronové sítě.

Další část se zabývá samotným rozpoznáním obrazu. Je velice důležité vědět jakým způsobem bude aplikace fungovat. Na základě znalostí o zpracování, segmentaci a snímání obrazu bylo možné vytvořit další část diplomové práce. S částí o zpracování obrazu mi výrazně pomohli články od vedoucího této diplomové práce pana Ing. Josefa Pavlíčka, Ph.D., který se touto problematikou hluboce zabývá.

V praktické části byly navrženy tři architektury konvolučních neuronových sítí, kde k učení těchto sítí bylo navrhnuto použití již dostupné a ověřené databáze obrazů MNIST se kterou by se testovali jednotlivé návrhy. Po vybrání nejefektivnějšího a nejvhodnějšího návrhu pro budoucí aplikaci by se následně vyhotovila již „ostrá“ databáze s rostlinami, která by byla předložena aplikaci k naučení.

Další část diplomové práce se zabývá správným výběrem frameworku a knihovny pro testování architektury navržených konvolučních neuronových sítí. V této části jsou rozebrány výhody i nevýhody dostupných prostředků s ohledem na programování v jazyce Java. Na základě dostupných frameworků pro implementaci konvolučních neuronových sítí v programovacím jazyce Java vyšel pro mobilní aplikaci jako nejvhodnější framework Deeplearning4J a Neuropf. Pro použití Neurophu jsem se rozhodla z toho důvodu, že Framework umožňuje použití vestavěných funkcí pro implementaci konvolučních neuronových sítí. Deeplearning4J by byl nejspíš lepší pro zkušeného programátora, jinak by bylo programování pomalejší. V ideálním případě bych navrhovala otestovat oba frameworky a zvolit pro aplikaci vhodnější.

Poslední částí diplomové práce jsou návrhy UML diagramů. Nejdříve jsem navrhla vývojové diagramy pro učení konvoluční neuronové sítě již s databází obrazů rostlin, které

by se v aplikaci skutečně vyskytovali, tudíž až po otestování a zvolení vhodné konvoluční neuronové sítě. Druhý vývojový diagram znázorňuje vyhledávání v databázi na základě vyfotografovaného obrazu uživatelem. Tento vývojový diagram by se dal pojmout podrobněji.

Dále je zde navrhnut zjednodušený Use Case diagram, kde jsou navrženy základní funkce aplikace pro uživatele a programátora či správce. Na základě tohoto modelu bylo možné navrhnout základní architekturu Class Diagramu, podle které by při rozšíření mohla být dále aplikace naprogramována.

Všechny cíle, které byly na začátku práce zadány se podařilo úspěšně splnit. Vzhledem k tomu že je aplikace rozsáhlá, plná možností a velmi složitá, dosažené výsledky jsou nad rámec očekávání.

# 7 Seznam použitých zdrojů


[1] Warren McCulloch and Walter Pitts: A Logical Calculus of the Ideas Immanent in Nervous Activity, https://link.springer.com, [online], [cit. 2018-12-27], < https://link.springer.com/chapter/10.1007/978-3-642-70911-1_14>

[2] Neuronové sítě a jejich využití https://www.systemonline.cz, [online], [cit. 2018-01-10], <https://www.systemonline.cz/clanky/neuronove-site-a-jejich-vyuziti-1.htm >

[3] Úvod do problematiky umělých neuronových sítí http://www.elektrorevue.cz, [online], [cit.2018-01-11], <http://www.elektrorevue.cz/clanky/00013/index.html#typy >

[4] Matematický model a aktivní dynamika neuron http://portal.matematickabiologie.cz, [online], [cit. 2018-02-09], <http://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--jednotlivy-neuron--matematicky-model-a-aktivni-dynamika-neuronu>

[5] CELEBI, O.C. Celebi Tutorial: Neural Networks and Pattern Recognition Using MATLAB https://www.byclb.com, [online], [cit. 2018-02-20], <https://www.byclb.com/TR/Tutorials/neural_networks/ch7_1.htm>

[6] Rosenblatt contributions http://csis.pace.edu, [online], [cit. 2018-02-27], < http://csis.pace.edu/~ctappert/srd2011/rosenblatt-contributions.htm>

[7] Learning Internal Representation by Error Propagation http://www.cs.toronto.edu, [online], [cit. 2018-03-09], < http://www.cs.toronto.edu/~fritz/absps/pdp8.pdf>

[8] HINTON G.E., ZEMEL R.S. Autoencoders, Minimum Destription LEngh and Helmholtz Free Energy http://www.cs.toronto.edu, [online], [cit. 2018-03-10], <http://www.cs.toronto.edu/~fritz/absps/cvq.pdf>

[9] BALDI P. Autoencoders, Unsupervised Learning, and Deep Architectures, http://jmlr.org, [online], [cit. 2018-03-13], <http://jmlr.org/proceedings/papers/v27/baldi12a/baldi12a.pdf>

[10] HINTON G.E. Baltzmann machine, www.scholarpedia.org, [online], [cit. 2018-03-13], <http://www.scholarpedia.org/article/Boltzmann_machine>

[11] CHO K., Raiko T., ILIN A. Enhanced Gradient and Adaptive Learning Rate for Training Restricted Bolzmann Machines www.icml-2011.org, [online], [cit. 2018-03-15], <http://www.icml-2011.org/papers/98_icmlpaper.pdf>

[12] HOPEFIELD J.J. Hopefield Network, www.scholarpedia.org, [online], [cit. 2018-03-15], <http://www.scholarpedia.org/article/Hopfield_network>

[13] MRÁZOVÁ I. Prezentace k přednášce k předmětu Neuronové sítě (NAIL002), http://ksvi.mff.cuni.cz [online], [cit. 2018-03-17], < http://ksvi.mff.cuni.cz/~mraz/nn/Neuronove_Site_Prednaska_AM.pdf >

[14] D.O. HEBB, The Organization of Behavior 1994. ISBN 978-0805843002

[15] ELMAN J. L., Distributed Representations, Simple Recurrent Networks and Grammatical Structure. Machine Learning 7, 195-225 (1991), [cit. 2018-03-17], < https://link.springer.com/content/pdf/10.1007%2FBF00114844.pdf>

[16] ČERMÁK J., BALÍK M., Přístupy k modelování prozodie v TTS systémech, http://www.elektrorevue.cz, [online], [cit. 2018-03-19], < http://www.elektrorevue.cz/clanky/05050/index.html>

[17] LE Q.L., NGIAM J., CHEN Z., CHIA D., KOH P.W., NG A.Y., Tiled convolutional neural networks searchsecurity.techtarget.com, http://ai.stanford.edu [online], [cit. 2018-03-20], <http://ai.stanford.edu/~ang/papers/nips10-TiledConvolutional NeuralNetworks.pdf>

[18] BOUVRIE J., Notes on Convolutional Neural Networks, http://cogprints.org, [online] 2006, [cit. 2018-03-25], <http://cogprints.org/5869/1/cnn_tutorial.pdf>

[19] KULKARNI T.D., WHITNEY W.F., KOHLI P., TENENBAUM J.B., Deep Convolutional Inverse Graphics Network, http://papers.nips.cc, [online], [cit.2018-03-25], <http://papers.nips.cc/paper/5851-deepconvolutional-inverse-graphics-network.pdf>

[20] DOSOVITSKIY A., SPRINGENBERG J.T., TATARCHENKO M., BROX T., Learning to Generate Chairs, Tables and Car with Convolutional Networks, [online], [cit.2018-04-01], <https://pdfs.semanticscholar.org/e47e/988e6d96b876bcab8ca8e2275a6d73a3f7e8.pdf >

[21] OpenCL, www.abclinuxu.cz, [online], [cit. 2018-04-07], < http://www.abclinuxu.cz/blog/Mihovy_sochory/2009/9/opencl>

[22] JIRKOVSKÝ J., Deep Learning a prostředí MATTLAB, Microsoft, www.humusoft.cz, [online], [cit. 2018-04-08], < https://www.humusoft.cz/blog/20180831-deep-learning/>

[23] LECUN Y., BOTTOU L., BENGIO Y., HAFFNER P., Gradient-Based Learning Applied to Document Recogniting http://yann.lecun.com, [online], [cit. 2018-04-17], <http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf>

[24] KRIZHEVSKY A., SUTSKEVER I., HINTON G.E., ImageNet Classification with Deep Convolutional Neural Networks www.cs.toronto.edu, [online], [cit. 2018-04-18], <http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>

[25] SCHLUTER J., BOCK S., Improved Musical Oneset Detection with Convolutional Neural Networks, [online], [cit. 2018-04-18], <http://www.ofai.at/~jan.schlueter/pubs/2014_icassp.pdf>

[26] Šonka M., Hlaváč V., Boyle R., Image processing, analysis, and machine vision. Thomson, Toronto, 2008, ISBN 80–010–3110–1

[27] Hlaváč V., Sedláček M., Zpracování signálů a obrazů. ČVUT, Praha, 2005

[28] Parker J. R., Algorithms for Image Processing and Computer Vision. Wiley Publishing, Indianapolis, 2011, ISBN 978–0–470–64385–3

[29] Hlaváč V., Šonka M., Počítačové vidění. Grada, Praha, 1993, ISBN 80–85424

[30] Lecun Y., Cortes C., Burges C.J.C., The MNIST Database of handwritten digits [online], [cit. 2018-05-10], <http://yann.lecun.com/exdb/mnist/>

[31] Marrone P., An Object Oriented Neural Engine [online], [cit. 2018-05-11], <http://sourceforge.net/projects/joone/>

[32] Kriesel D., Deeplearning for Java [online], [cit. 2018-05-28], < https://deeplearning4j.org/>

[33] Heaton J., Encog Machine Learning Framework [online], [cit. 2018-05-29], <http://www.heatonresearch.com/encog>

[34] Severac Z., Neuroph [online], [cit. 2018-05-29], <http://neuroph.sourceforge.net/>

[35] The open standard for parallel programing of heterogeneous system [online], [cit. 2018-06-03], <https://khronos.org/opencl>

[36] About the project, www.birdsonline.cz, [online], [cit. 2018-06-10], < https://www.birdsonline.cz/en/about-the-project>

[37] Pavlíček J., Jarolímek J., Jarolímek J., Pavlíčková P., Dvořák S., Pavlík J., Hanzlík P., Agris on-line Papers in Economics and Informatics, Automated Wildlife Recognition, Volume IX, Number 1, 2018 [cit. 2018-06-12]

[38] Hanzlík P., Metody umělé inteligence v rozpoznávání rostlin, Disertační práce, [cit. 2018-06-15]

[39] Vývojový diagram (Flow chart), www.managementmania.com, [online], [cit. 2018-06-15], <https://managementmania.com/cs/vyvojovy-diagram-flow-chart >

[40] Tettamanzi A., Tomassini M., Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems, Science & Business Media, 2001, ISBN 978-3-662-04335-6 [cit. 2018-06-15]