Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Rozpoznání obrazu a jeho aplikace ve Smart Agriculture

Jan Bakule

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jan Bakule

Systémové inženýrství a informatika
Informatika

Název práce

Rozpoznání obrazu a jeho aplikace ve Smart Agriculture

Název anglicky

Image recognition and its application in Smart Agriculture

Cíle práce

Bakalářská práce je tematicky zaměřena na analýzu využití konvolučních neuronových sítí pro rozpoznání obrazu a možnosti aplikace v rámci Smart Agriculture. Hlavním cílem je charakterizovat základní aspekty rozpoznávání obrazu pomocí strojového učení. Dílčí cíle jsou:
- analyzovat problematiku rozpoznávání obrazu
- charakterizovat rozdílné architektury neuronových sítí
- porovnat různé architektury konvolučních neuronových sítí pro rozpoznání obrazu

Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní řešení je realizováno formou porovnání často používaných architektur neuronových sítí využívaných pro rozpoznávání obrazu. Na základě syntézy teoretických poznatků a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Doporučený rozsah práce

30 – 40 stran

Klíčová slova

Umělá inteligence, počítačové učení, rozpoznání obrazu, konvoluční neuronové sítě, Smart Agriculture

---

Doporučené zdroje informací

GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, Massachusetts: The MIT Press, [2016]. ISBN 978-0262035613.

CHITYALA, Ravishankar a Sridevi PUDIPEDDI. Image processing and acquisition using Python. Boca Raton: CRC Press, Taylor & Francis Group, [2014]. ISBN 978-1466583757.

CHOLLET, François. Deep learning with Python. Shelter Island, New York: Manning Publications Co., [2018]. ISBN 978-1617294433.

RASHID, Tariq. Make your own neural network. CreateSpace Independent Publishing Platform,[2016]. ISBN 978-1530826605.

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jan Jarolímek, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 11. 10. 2019

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 14. 10. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 01. 03. 2020

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci Rozpoznání obrazu a jeho aplikace ve Smart Agriculture jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 23.3.2020 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Janu Jarolímkovi, Ph.D. za vedení, trpělivost a rady při zpracování této bakalářské práce.

Rozpoznání obrazu a jeho aplikace ve Smart Agriculture

Abstrakt

Tato bakalářská práce se zabývá využitím neuronových sítí pro rozpoznání obrazu. Jejím cílem je charakterizovat základní aspekty rozpoznávání obrazu pomocí neuronových sítí. V první, teoretické, části se zaměřuje na analýzu principů, z nichž vychází neuronové sítě, jejich architektury a využití pro rozpoznání, hlavně klasifikaci, obrazu. V této části také obsahuje možnosti aplikace rozpoznání obrazu pomocí neuronových sítí v rámci Smart Agriculture. V druhé, praktické, části je na základě poznatků z teoretické části navržena, implementována a zlepšována jednoduchá hluboká konvoluční neuronová síť pro klasifikaci datasetu CIFAR-10, dosahující přesnosti klasifikace 91,53 %, která je následně porovnána s implementacemi složitějších architektur pro stejný dataset.

Klíčová slova: umělá inteligence, strojové učení, hluboké učení, neuronová síť, konvoluční neuronová síť, rozpoznání obrazu, klasifikace obrazu, Smart Agriculture

Image recognition and its application in Smart Agriculture

Abstract

This work is about the use of neural networks for image recognition. It aims to characterize the basic aspects of image recognition using neural networks. In the first, theoretical, part, it analyzes principles upon which are based neural networks, their architecture and use for image recognition, mainly classification. In this part it also contains some possibilities for application of image recognition using neural networks in Smart Agriculture. In the second, practical, part, a simple deep convolutional neural network for CIFAR10 dataset is designed, implemented and upgraded to reach classification accuracy of 91.53 %. The network is then compared with implementations of other, more complex, networks for the same dataset.

Keywords: artificial intelligence, machine learning, deep learning, neural network, convolutional neural network, image recognition, image classification, Smart Agriculture

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Neuronové sítě a deep learning jsou v posledních letech na vzestupu. V různých oblastech vznikají programy vykazující umělou inteligenci schopnou vyrovnat se člověku, a v některých případech ho i překonat jako například program AlphaGo od společnosti Deepmind Technologies, který dokázal ve hře Go (pro počítač složitější než šachy) nejen porazit profesionální lidské hráče, ale ještě se s dalšími verzemi dále zlepšovat.[43] Zájem o deep learning mimo jiné projevila společnost Google, která dokonce pro svůj framework TensorFlow, využívaný pro deep learning, vyvinula specializovaný hardware, tzv. TPU – tensor processing unit.[42] Jako poslední příklad uveďme technologii založenou na deep learningu, schopnou vyměnit člověka ve videu jen na základě fotografie s takovou věrohodností, že musel vzniknout nový termín označující taková videa – deepfake.

Jde o velmi široké téma. Protože v oblasti rozpoznání obrazu již neuronové sítě dosáhly značných úspěchů, například DeepFace od Facebooku údajně dokáže identifikovat lidské obličeje na digitálních obrazech s přesností 97,35 %[43], zaměřuje se tato práce právě na tuto oblast, konkrétně klasifikaci obrazu a její možné aplikace v rámci Smart Agriculture, kde, jak bude ukázáno, vykazuje tato technologie potenciál usnadnění automatizace v zemědělství a díky tomu úsporu lidské práce.

Cílem práce je charakterizovat základní aspekty rozpoznávání obrazu pomocí neuronových sítí a analyzovat možnosti aplikace v zemědělství, čehož je dosaženo v teoretické části studiem a analýzou odborných zdrojů a v praktické části vytvořením a zlepšováním jednoduché konvoluční neuronové sítě. Součástí praktické části je také porovnání vytvořeného modelu se složitějšími modely.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je tematicky zaměřena na analýzu využití konvolučních neuronových sítí pro rozpoznání obrazu a možnosti aplikace v rámci Smart Agriculture. Hlavním cílem je charakterizovat základní aspekty rozpoznávání obrazu pomocí neuronových sítí. Dílčí cíle jsou:

- analyzovat problematiku rozpoznávání obrazu
- charakterizovat rozdílné architektury neuronových sítí
- porovnat různé architektury konvolučních neuronových sítí pro rozpoznání obrazu

2.2 Metodika

Metodika řešené problematiky bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Vlastní řešení je realizováno formou porovnání často používaných architektur neuronových sítí využívaných pro rozpoznávání obrazu. Na základě syntézy teoretických poznatků a výsledků vlastního řešení budou formulovány závěry bakalářské práce.

Teoretická část je zaměřena na analýzu principů, z nichž vychází neuronové sítě, jejich architektury a využití pro rozpoznání, respektive klasifikaci obrazu. V další části obsahuje možnosti aplikace rozpoznání obrazu pomocí neuronových sítí v rámci Smart Agriculture a přenositelnosti vytrénovaných hodnot pomocí tzv. transfer learningu.

Praktická část spočívá v návrhu a implementaci jednoduché konvoluční neuronové sítě pro klasifikaci CIFAR-10 datasetu a jejím porovnání s implementacemi složitějších architektur pro stejný dataset. Porovnání není založeno pouze na přesnosti klasifikace, ale také na dalších faktorech, kterými jsou například počet parametrů (a s ním spojená výpočetní náročnost), nebo doba trénování.

3 Teoretická východiska

3.1 Machine learning

Machine learning (strojové učení) spadá pod oblast umělé inteligence, zabývá se vytvářením programů schopných naučit se z dat dělat činnost, např. na jejich základě provádět i nelineární regresi, nebo odhadovat budoucí vývoj nějaké veličiny; která není explicitně naprogramována, změnou vlastních vnitřních charakteristik. Podmnožinou machine learningu je pak deep learning (hluboké učení), ve kterém je naučení dosáhnuto pomocí algoritmů hlubokých neuronových sítí. [2]

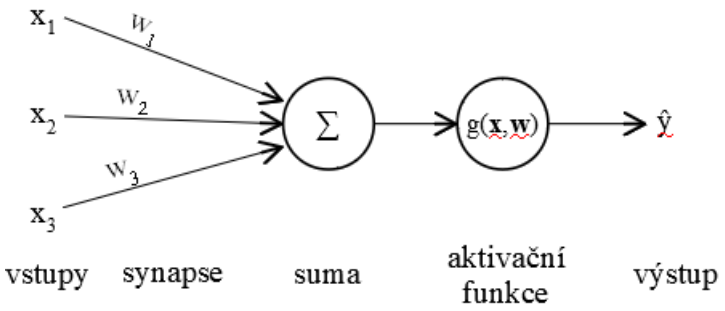
3.2 Stručná historie

Počátky neuronových sítí lze datovat až k přelomu devatenáctého a dvacátého století, kdy se vědci snažili zjistit, jak obecně u lidského mozku probíhá učení či vidění. Moderní pohled na neuronové sítě vznikl ve čtyřicátých letech dvacátého století, kdy bylo dokázáno, že sítě umělých neuronů jsou schopny provádět aritmetické a logické funkce. První praktická aplikace neuronových sítí se objevila v letech padesátých s vytvořením tzv. perceptronu, který zvládal rozeznávat určitý typ vzorů, jeho schopnosti však byly velmi omezené. Nový život do výzkumu v osmdesátých letech dvacátého století vdechlo mj. vytvoření tzv. backpropagation algoritmu. Oblast se pak dále rozvíjela s novými koncepty, jako například inovativními architekturami sítí, či novými přístupy k učení. Nemalý podíl na rozvoji machine learningu má také technologický rozvoj, který umožňuje nejen vytvářet stále složitější modely, ale také je relativně rychle trénovat. [1] [2]

3.3 Neuronová síť

3.3.1 Neuron

Neuronová síť je, v tomto kontextu, matematický výpočetní model. Jejími základními stavebními kameny jsou tzv. neurony a spoje mezi nimi – synapse, skrze které proudí informace v podobě čísel. Synapse mají váhy, určují tedy pro neuron, jak důležitá je pro něj informace přicházející z předešlého neuronu. [3]



Obr. 1: Schéma neuronu

Účelem neuronu zde je jednoduše řečeno vzít informace ze svých vstupů, vyhodnotit je a poslat je dál. Princip lze vyjádřit matematicky jako: [3]

$$\hat{y} = g(w_0 + \sum_{i=1}^{m} w_i * x_i)$$

kde:

- x_i je i-tý vstup neuronu (vstupy značíme $x_1 \ldots x_m$),
- w_i je váha i-tého vstupu,
- w_0 je tzv. bias neuronu,
- g je aktivační funkce neuronu
- a $\hat{y}$ pak je funkční hodnota g v bodě odpovídajícímu váženým hodnotám vstupů neuronu.

Informace nemusejí vždy proudit pouze jedním směrem, tak je tomu například pro rekurentní neuronové sítě, pro naše účely však budeme používat síť, ve které informace proudí pouze jedním směrem a podle toho je její typ také označován anglicky feed-forward (přeloženo doslovně – krmená dopředně). [4]

3.3.1.1 Aktivační funkce

Neuron pro nás tedy funguje tak, že sečte vážené vstupy, přidá svůj bias a výsledek pak prožene svou aktivační funkcí, bez které by jeho výstupem byla prostá lineární kombinace vstupů. Ona funkce musí mít nějaké náležitosti.

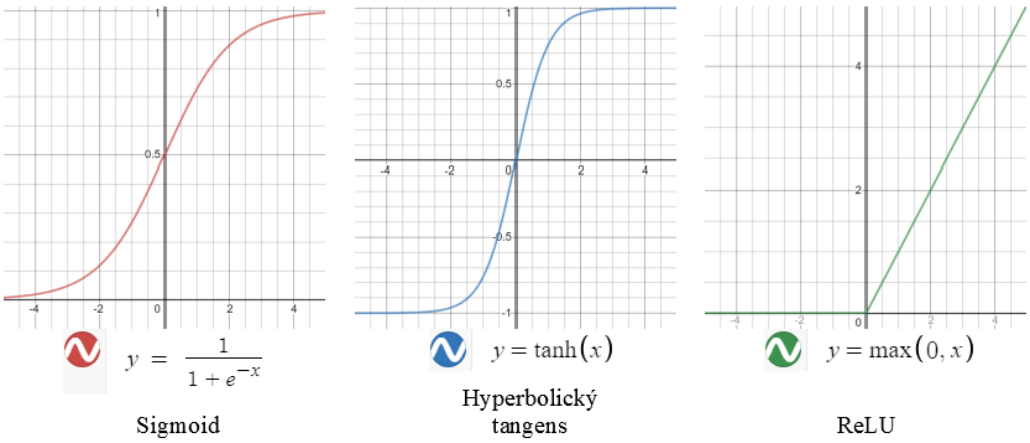
Za prvé se samozřejmě nesmí její definiční obor a obor hodnot vymykat danému modelu. Musí tedy být schopna zobrazit jakýkoli vstup, který může přijít, na výstup, který dokáže zpracovat následující neuron.

Vzhledem k tomu, že strojové učení, a tedy i neuronové sítě, se aplikuje na problémy z reálného světa, kde informace a vztahy mezi nimi zpravidla nebývají lineární, musejí tomu

být přizpůsobeny i použité aktivační funkce. Právě nelineární aktivační funkce umožňují řešení netriviálních problémů. [1]

Proto se v neuronových sítích převážně používají jako aktivační funkce nelineární matematické funkce, s ohledem na algoritmus gradient descent (popsaný ve vlastní kapitole) pak také nejlépe diferencovatelné, například:

- Sigmoid – funkce definovaná pro všechna reálná čísla, omezená, monotónní a diferencovatelná. Její obor hodnot (otevřený interval (0;1)) umožňuje ji využít i pro práci s pravděpodobnostními hodnotami. [2]
- Hyperbolický tangens – je funkce podobná funkci sigmoid s tím rozdílem, že jejím oborem hodnot je otevřený interval (-1;1) a je symetrická okolo nuly.
- Rectified linear unit (ReLU) – je často používaná funkce. Její nevýhodou může být to, že zobrazuje všechny nekladné hodnoty na nulu. Proto může být někdy modifikována na tzv. „propustnou“ (anglicky Leaky ReLU) [7], kde pro nekladné hodnoty nabývá formy $f(x) = a * x$, s nějakou relativně malou hodnotou parametru a (například 0,01). [2] [5]
- Exponential Linear Unit (ELU) – je méně linearní alternativou ReLU. Je velmi podobná Leaky ReLU s tím rozdílem, že je hladká. [12]

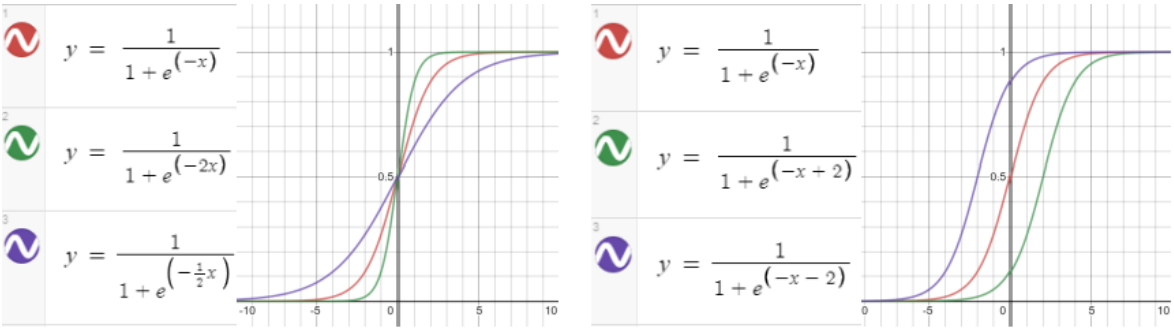


Obr. 2: Ukázka průběhu příkladových aktivačních funkcí

3.3.1.2 Bias

Bias neuronu nemusí mít na první pohled úplně zřejmou funkci. Koukneme-li se na „před-aktivaci“ neuronu, tedy na jeho činnost před použitím aktivační funkce, vidíme, že bez biasu je jediným argumentem ve zjednodušené podobě pouze člen $w * x$, kde, bereme-li x jako proměnnou a w jako parametr, můžeme ovlivnit pouze sklon aktivační funkce (měněním parametru w) a „citlivá“ část jejího průběhu zůstává na stejném intervalu (okolo stejného bodu). Právě pro případ, kdy neuron potřebuje posunout „citlivost“ své aktivační

funkce, má jeden parametr navíc – vlastní bias. Ten může nabýt podoby jednoho neuronu, vysílajícího do každého neuronu s biasem neustále stejný signál, jehož hodnota je upravována vahou synapse mezi oním neuronem, který slouží jako zdroj biasu, a neurony, které jsou na něj napojeny. [4]



Obr. 3: Vliv vah a biasu na aktivační funkci sigmoid

3.3.2 Single layer neural network

Pro vytvoření organizovaného modelu z neuronů musí být nějak uspořádány. U neuronových sítí je toho dosaženo pomocí vrstev (z angličtiny – layer). Rozlišujeme 3 základní typy vrstev z hlediska pozice v síti: Input, Output a Hidden. [4]

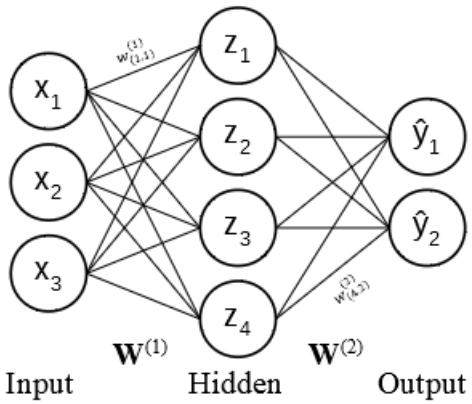
Input layer je první, vstupní vrstva sloužící pro příjem dat a jejich vyslání do sítě, ovšem i ona data musí být samozřejmě přizpůsobena pro model. Například pokud je zdrojem informací množina bodů, musíme předat důležité informace o nich, třeba jejich souřadnice. Pokud je zdrojem informací obraz, může být potřeba ho převést „do řeči modelu“, třeba vytvořit matici dimenze 3, odpovídající RGB hodnotám pro každý pixel.

Output layer je poslední, výstupní vrstva, jejíž podoba závisí na účelu celého modelu. Tuto podobu může být nutné znát pro případ potřeby interpretace výstupu. Například pro síť, jejímž účelem je klasifikace obrazu, může mít jako výstup vektor hodnot. Právě až znalost podoby výstupní vrstvy nám umožní tyto podoby interpretovat jako pravděpodobnosti náležitosti k určité klasifikační třídě. [4]

Hidden layer se nachází mezi Input a Output vrstvami. V této vrstvě probíhá drtivá většina výpočetní činnosti modelu. Na rozdíl od ostatních vrstev, jejichž funkce a chování jsou jasné, jí nemůžeme chování přímo přisoudit, jelikož by se v průběhu učení mělo měnit. Proto je označována jako hidden, přeloženo do češtiny – schovaná, jak kdybychom k neuronové síti přistupovali jako k černé skříňce. [4]

3.3.2.1 Příklad – Fully connected (dense) single layer NN

Vezměme jako příklad neuronovou síť s jednou vstupní, výstupní a schovanou vrstvou, kde mezi po sobě jdoucími vrstvami jsou všechny neurony navzájem propojeny. Takto koncipovaná síť je označována jako fully connected single layer neural network, doslovně přeloženo jako plně propojená (někdy také používáno dense – hustá) neuronová síť o jedné (schované) vrstvě.



Obr. 4: Schéma sítě z příkladu 3.3.2.1

Výpočetní proces zde můžeme matematicky obecně vyjádřit jako:

$$z_i = g_z(w_{(0,i)}^{(1)} + \sum_{j=1}^{m} w_{(j,i)}^{(1)} * x_j) \quad \text{a} \quad \hat{y}_i = g_{\hat{y}}(w_{(0,i)}^{(2)} + \sum_{j=1}^{m} w_{(j,i)}^{(2)} * x_j)$$

kde:

- z_i je výstup i-tého neuronu ve schované vrstvě,
- g_z je aktivační funkce neuronu schované vrstvy,
- $w_{(0,i)}^{(k)}$ je bias i-tého neuronu k-té vrstvy (k počítáno od první schované vrstvy),
- $w_{(j,i)}^{(k)}$ je váha i-tého vstupu j-tého neuronu k-té vrstvy,
- x_j je j-tý vstup neuronu,
- $\hat{y}_i$ je výstup i-tého neuronu ve výstupní vrstvě,
- a $g_{\hat{y}}$ je aktivační funkce neuronu výstupní vrstvy.

Jak napovídá značení g_z a $g_{\hat{y}}$, všechny vrstvy nemusí používat stejnou aktivační funkci. Jednotlivé vrstvy pak mohou být kromě jejich umístění (input – hidden – output) nazývány také podle jejich funkce vyplývající z jejich aktivační funkce. Takto právě dále v této práci narazíme například na konvoluční, pooling (doslovně přeloženo – shromažďující), nebo softmax vrstvy.

3.3.3 **Deep neural network**

Síť v předchozím příkladu může také být označena jako shallow network (plytká, mělká, nejdoucí do hloubky), protože má pouze jednu schovanou vrstvu. Taková síť může zvládat nějaké problémy, ale s rostoucí komplexitou problémů již neudrží krok. Pro takové problémy je potřeba jít více do hloubky, uvědomit si například nějaké souvislosti atp. Toho lze dosáhnout právě prohloubením vlastní neuronové sítě – přidáním dalších schovaných vrstev. Pro síť je pak možné, jednoduše řečeno, dělat si „mezivýpočty“, složitěji řečeno, postupně zjišťovat komplexnější vlastnosti. Například síť rozeznávající objekty na fotografii v ní pak může prvně najít jednoduché vzorce (třeba hrany a rohy), podle nich pak rozeznat složitější útvary (dejme tomu tvary) a nakonec si z nich poskládat komplexní objekty, například části obličeje (a vztahy mezi nimi!), z nichž pak dokáže rozeznat obličej. [2]

3.3.4 **Trénování**

Máme tedy komplexní, hlubokou neuronovou síť, která dokáže zpracovávat data. To ale nestačí. Ještě se musíme postarat o to, aby věděla, jak data zpracovávat, přesněji řečeno, co si z nich vzít, co v nich hledat, která informace je jak důležitá. Hned mezi základními stavebními kameny neuronové sítě se nám vyskytlo něco, co ovlivňovalo důležitost informací – váhy synapsí a biasy. Právě ty umožňují síti, respektive jejím jednotlivým neuronům, rozeznat důležité informace. Potřebné hodnoty vah a biasů ale neznáme, musíme se je nějak naučit.

Jak již bylo zmíněno, učení v machine learningu stojí na datech. Je tedy potřeba mít pro model připraven dostatek dat, z nichž bude zaučen. Neuronové sítě jsou trénovány na datasetech vytvořených právě pro ten účel. I když by se dalo říci, že neuronové sítě jsou vlastně napodobením lidského mozku, je zde podstatný rozdíl – jsou to stále pouze matematické modely, které se (alespoň zatím) nedokáží vyrovnat letům evoluce kognice. Oproti mozku ale v současné době (a o to více v budoucnosti) mají jednu podstatnou výhodu, kterou je čistá výpočetní kapacita. Neuronová síť je model naprogramovaný v počítači, který má spousty výpočetních jednotek. Jak bylo ukázáno na funkci neuronů, základní operace neuronových sítí navíc jdou většinou rozložit na sčítání a násobení, v čemž v dnešní době vynikají grafické jednotky. Operace v rámci jedné vrstvy na sebe nemusí čekat, což umožňuje využití potenciálu paralelizace, který grafické jednotky nabízejí. Neuronová síť pak může prostě udělat spoustu chyb a poučit se z nich.

Je tedy potřeba určit si, co to chyba je. To samozřejmě záleží na povaze řešeného problému. Definujeme si tedy nákladovou funkci (kde velikost chyby je náklad, který chceme minimalizovat). Obecně takovou funkci můžeme popsat jako:

$$L(f(x^{(i)}; \boldsymbol{W}), y^{(i)})$$

kde:

- $x^{(i)}$ je i-tá položka trénovacího datasetu (vstup pro síť),
- $\boldsymbol{W}$ je matice vah synapsí,
- $f(x^{(i)}; \boldsymbol{W})$ je hodnota odhadnutá sítí (neboli výstup sítě s daným vstupem $x^{(i)}$ a vahami $\boldsymbol{W}$),
- a $y^{(i)}$ je očekáváná (správná) hodnota pro i-tou položku trénovacího datasetu.

Tak obecně vypadá naše nákladová funkce pro jeden konkrétní tréninkový případ. Aby ale naše síť dokázala být přesná pro jakýkoli příklad, poučit se z jednoho samozřejmě nestačí, je jich potřeba mnohem více. Proto si definujeme také účelovou funkci, obecně:

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)}; \boldsymbol{W}), y^{(i)})$$

kde jednoduše nasčítáme chyby pro každou položku z tréninkového datasetu a výsledek vydělíme jejich počtem, čímž získáme průměrnou empirickou chybu, a vzhledem k tomu, že hodnoty $x^{(i)}$ a $y^{(i)}$ jsou pevně dané body, jediným parametrem této funkce je právě matice vah $\boldsymbol{W}$, což nám umožňuje sledovat dopad hodnot jednotlivých vah na její hodnotu, kterou se snažíme minimalizovat za účelem maximalizace přesnosti odhadů naší sítě. [2] [4]

Což se konkrétní podoby nákladové a účelové funkce týče, volba její podoby záleží čistě na nás a určení správné funkce je umění, možností je mnoho a problémy bývají moc komplexní na to, aby šlo na první pohled vidět, která funkce bude nejlepší. Pro představu – pokud síť odhaduje nějakou spojitou veličinu, můžeme použít střední kvadratickou chybu:

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} - f(x^{(i)}; \boldsymbol{W})\right)^2$$

Nebo pokud máme klasifikační model, jehož výstupem jsou dvě pravděpodobnostní hodnoty, můžeme využít funkci křížové entropie[1]:

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} \left(y^{(i)} \ln\left(f(x^{(i)}; \boldsymbol{W})\right) + (1 - y^{(i)}) \ln(1 - f(x^{(i)}; \boldsymbol{W}))\right)$$

3.3.4.1 Gradient descent

Síť tedy učíme tak, že jí říkáme, jaké dělá chyby, a podle toho měníme váhy jejích synapsí. Snažíme se tedy najít optimální podobu $\boldsymbol{W}$, ve které $J(\boldsymbol{W})$ dosahuje svého minima. K tomu můžeme využít tzv. gradient descent (sestup podle gradientu), kde začneme s náhodnými vahami, jejichž hodnoty pak iterativně zlepšujeme.

Algoritmus vypadá zhruba takto[1]:

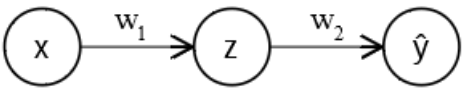
1. Inicializuj neuronovou síť s náhodnými hodnotami vah
2. Dělej do konvergence (hodnoty $J(\boldsymbol{W})$):
    1) Spočti gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
    2) Aktualizuj váhy $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Vrať $\boldsymbol{W}$

Gradient je zobecnění derivace pro více proměnných[2]. Spočtením gradientu tedy zjistíme, jaký dopad mají jednotlivé váhy na růst hodnoty účelové funkce $J(\boldsymbol{W})$ a podle toho hodnoty vah změníme opačným směrem, protože chceme, aby hodnota $J(\boldsymbol{W})$ klesala, proto je gradient od $\boldsymbol{W}$ odečten. Konvergencí rozumíme bod, kde je sklon $J(\boldsymbol{W})$ nulový. Pro úplnost – η je míra učení – hyperparametr, kterým můžeme ovlivnit velikost změny vah. Podrobněji je popsán v kapitole o optimalizaci modelu.

3.3.4.2 Backpropagation

Pro spočtení gradientu můžeme využít tzv. backpropagation algoritmus. Přeloženo přímo z názvu, jde o zpětnou propagaci (chyby). Algoritmus s využitím řetízkového pravidla postupně od posledních vrstev napočítává hodnoty skrze síť, až ke konkrétní váze. Ukažme jeho efektivitu na jednoduchém příkladu:

Mějme zjednodušenou neuronovou síť s jedním vstupním neuronem x, jedním schovaným neuronem z, jedním výstupním neuronem $\hat{y}$ a obecnou účelovou funkcí $J(\boldsymbol{W})$. Označme si váhu synapse vedoucí od x do z jako w_1, váhu synapse mezi z a $\hat{y}$ jako w_2.



Obr. 5: Schéma sítě pro příklad z kapitoly 3.3.4.2

Dopad w_2 na $J(\boldsymbol{W})$ můžeme podle tohoto algoritmu pomocí řetízkového pravidla rozložit na dopad $\hat{y}$ na $J(\boldsymbol{W})$ a dopad w_2 na $\hat{y}$, tedy:

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Podobně pak můžeme odvodit vliv w_1 na $J(\boldsymbol{W})$ jako:

$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_1}$$

Všimněme si, že hodnotu $\frac{\partial J(\boldsymbol{W})}{\partial \hat{y}}$ ve výpočtu pro w_1 máme již zjištěnou z výpočtu pro w_2. Pro napočítání jednotlivých hodnot nám tedy stačí pouze projít jednou od konce celou neuronovou síť. [2] [3] [4]

3.4 Optimalizace modelu

V této části jsou uvedeny příklady možností, jak je dále stavěno na výše popsaných konceptech neuronových sítí, a jejich algoritmů, a jak jsou dále upravovány a optimalizovány.

3.4.1 Stochastic a mini-batch gradient descent

I když je gradient descent relativně výpočetně jednoduchý, vyžaduje určení $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ pro celý trénovací dataset (Batch gradient descent), ve kterém chceme mít co nejvíce dat, což může být obzvláště s rostoucí velikostí modelu velmi výpočetně náročné. Na druhou stranu, měnit váhy podle jednotlivých položek z datasetu (Stochastic gradient descent) může způsobit velmi klikatou cestu za minimem. Možným řešením tohoto problému je kompromis mezi Batch gradient descentem a Stochastic gradient descentem, algoritmem nazývaným Mini-batch gradient descent. Dalo by se říci, že jde o střední cestu mezi počítáním gradientu pro celý trénovací dataset a pro jednotlivé případy. Místo výpočtu gradientu na základě celého datasetu, v jednotlivých iteracích spočítá jeho odhad na základě náhodné podmnožiny datasetu o určité velikosti. Každá iterace je pak méně výpočetně náročná, ale na úkor vypovídající hodnoty spočteného gradientu, což může mít vliv na rychlost konvergence. Proto je potřeba, v závislosti na dostupné výpočetní kapacitě, najít účinnou velikost dávek.

Modifikovaný algoritmus:

1. Inicializuj neuronovou síť s náhodnými hodnotami vah

2. Dělej do konvergence:
    1) Vyber dávku B položek z datasetu
    2) Spočti gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} \cong \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
    3) Aktualizuj váhy $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Vrať $\boldsymbol{W}$

Kde $J_k(\boldsymbol{W})$ je k-tá položka z dávky B položek z trénovacího datasetu.

Všimněme si, že jednotlivé iterace na sebe nemusí čekat, můžeme je tedy provádět paralelně a tím celý proces značně zrychlit. [2]

3.4.2 **Míra učení**

Máme tedy výpočetně jednoduchý postup, který síti umožňuje z dat zjistit, jak si upravit váhy synapsí pro lepší odhady. Pomocí gradient descentu je tak zmenšována hodnota $J(\boldsymbol{W})$ až do její konvergence. Zde se ale naskytuje následující problém – gradient descent je tzv. hladový algoritmus, sice nás zavede do nějakého minima, ale nemusíme mít záruku, že se jedná o minimum globální. Stejně tak může nastat problém, pokud narazíme na oblast, kde je hodnota $J(\boldsymbol{W})$ konstantní, v takovém případě bude její sklon nulový a může se zdát, že máme hotovo. Teoreticky může pomoct jednoduše začít na různých místech a doufat, že některá z cest pak dopadne dobře, toto řešení ovšem vyžaduje mnohem více výpočtů a není spolehlivé.

Pro tento případ nám může pomoct již dříve zmíněný hyperparametr η – míra učení, který upravuje velikost kroku během učení, tedy o kolik jsou jednotlivé váhy posunuty v každé iteraci[2]. Určení konkrétní hodnoty η je velmi choulostivé, správná hodnota může napomoct vyhnout se lokálnímu minimu, případně také pozitivně ovlivnit rychlost konvergence. Špatná hodnota naopak může vést až k divergenci, která může být způsobena například tzv. přestřelením optima v případě příliš velké hodnoty. [4]

Jednou možností, jak určit míru učení, je prostě vyzkoušet různé hodnoty a sledovat, která bude mít nejlepší efekt. Takový přístup je sice velmi jednoduchý, ale takto tahat hodnoty z klobouku nejspíše není úplně spolehlivé. Konstantní míra učení navíc může způsobit oscilaci kolem minima. Proto lze použít postupy, které míru učení mění postupně, nebo až adaptivně, například předem naplánovat její hodnoty v závislosti na epoše trénování, nebo ji snižovat nějakým faktorem třeba v závislosti na hodnotě účelové funkce. Některé optimalizační algoritmy si ji mohou měnit samy.

3.4.3 Adaptivní optimalizační algoritmy

- **Momentum** – hybnost, vylepšení založené na stejnojmenné fyzikální veličině. Určuje velikost kroku nejen na základě samotného gradientu v dané iteraci, ale také bere v potaz gradienty z předchozích iterací. Posiluje tedy změny ve stejném směru. Snižuje tak oscilaci a podporuje hladší cestu za optimem zrychlováním posunu v jeho směru. [4]
- **Root Mean Square Propagation** (RMSprop) – adaptivní metoda měnící míru učení pro každý jednotlivý parametr. Spočte exponenciální průměr druhých mocnin minulých gradientů pro daný parametr, na jehož základě určí velikost kroku a následně aktualizuje váhu. Stejně jako Momentum snižuje oscilaci, čímž podporuje hladší cestu za optimem, dosahuje toho ale oslabováním posunů mimo směr postupu. [11] [2]
- **Adagrad** – spravuje míru učení pro každý parametr zvlášť podle toho, jak často je parametr aktualizován, čím více je parametr měněn, tím menší míra učení.[8] [10]
- **Adadelta** – rozšíření algoritmu Adagrad, které nebere celou historii parametru, ale pohyblivou část, učení je pak účinné i po velkém počtu aktualizací. [9] [10]
- **Adam** – Adaptive Moment Estimation je kombinací metod AdaGrad a RMSProp. [6] [2]

3.4.4 Overfitting

Neuronovou síť cvičíme za účelem obecného využití naučených informací. Náš model tedy musí být co nejpřesnější obecně, ne pouze pro případy, se kterými se setkal v rámci učení.[1] Jednou možností je prostě mu poskytnout více dat, což nemusí být vždy možné. Alternativou je využití různých doplňujících postupů během trénování, například:

- **Regulace vah** – zajištěna přídatným výrazem v účelové funkci, který trestá příliš velké hodnoty vah, které mohou způsobit velkou citlivost vůči konkrétním případům, snižujíc tak jejich dopad na overfitting. Příkladem tohoto postupu je regulace označovaná jako L2, která způsobuje minimalizaci sumy čtverců vah. Alternativní a primitivnější postup mající stejný efekt jako regulace L2 je tzv. weight decay (chátrání vah), kdy po každé změně vah jsou nové váhy trochu zmenšeny. [1] [2] [4]
- **Dropout** – dodatečný hyperparametr, který určuje, jak velká část sítě bude v každé iteraci trénování náhodně „vypnuta“. Například pokud je dropout 0.5, znamená to, že v každé iteraci bude aktivace poloviny náhodně vybraných schovaných neuronů

(ze specifikovaných vrstev) nastavena na nulu. Tím je síť donucena nespoléhat se na jednotlivé cesty nebo neurony, jinak řečeno, řešit určité případy „zpaměti“. [2]

- **Early stop** – tato metoda se snaží zastavit učení dříve, než může k overfittingu dojít. Dosahuje toho tak, že kromě trénovacího datasetu má taky dataset validační (může jít prostě o nezávislou a v trénování nevyužitou podmnožinu trénovacího datasetu). Během trénování pak sleduje hodnotu účelové funkce pro oba datasety a zastaví učení, když se hodnoty začnou rozcházet, tedy v momentě, kdy se zlepšuje účelová funkce již pouze pro trénovací dataset. [4]

3.4.5 Modifikace dat

Pro vylepšení vlastností modelu je také možné modifikovat vstupní data:

- **Normalizace** – vstupní data mohou obsahovat rozličné veličiny o všelijakých hodnotách, např. vysoké hodnoty, které mohou způsobit, že během trénování se model naučí veliké váhy, což může vést k jeho nestabilitě.[2] Možným řešením tohoto problému je normalizace dat. Data mohou být transformována např. pomocí min-max normalizace, nebo z-skóre.[2]
- **Augmentace** – trénovací data je možno rozšířit na základě dostupných dat pomocí jejich triviálních transformací, např. promíchání před každou epochou, nebo u obrazů pomocí posunů, rotací, převrácení atd. Mohou tak vzniknout přijatelná další data, z hlediska počítače úplně nové obrázky, umožňující modelu naučit se rozpoznávat objekty v různých polohách. Je ovšem potřeba zvýšená opatrnost, nesmí být narušena vypovídací hodnota trénovacích dat o realitě. Například pokud reálná data nemohou obsahovat obrazy, které jsou „vzhůru nohama“, převracet data vertikálně pak spíše uškodí, protože tak získáme něco, co model v realitě nemůže potkat. [1] [2]

3.4.6 Transfer learning

Vzhledem k tomu, že cílem trénování je získat obecné znalosti, dobře vycvičené modely mohou být použity pro více případů, jedinou podmínkou je dostatečná shoda dat, na kterých byl model vycvičen, a dat na která je aplikován. Obecně řečeno, předpokládá se, že pokud se například model během trénování naučil, jak poznat kočku, pozná ji i v našich datech.

Pro běžné klasifikační problémy je možné využít hlubokých modelů, předem vytrénovaných na velkých a náročných klasifikačních datasetech. Proto jsou využívány modely vycvičené například na datasetu ImageNet, kolekci fotografií obsahující 1000 klasifikačních tříd. Tyto modely jsou trénovány až týdny s využitím výkonného hardwaru a poskytnuty pro využití. Patří mezi ně například:

- VGG [16]
- Inception [17]
- ResNet[18]

Podrobné porovnání populárních předem vytrénovaných modelů provedli Canziani et al.[19]

3.5 Architektura NN pro klasifikaci obrazu

Tato část je zaměřena na adaptaci neuronové sítě pro klasifikaci obrazu.

3.5.1 Klasifikace obrazu

Klasifikace obrazu je problém z oblasti počítačového vidění. Přesněji jde o klasifikaci objektu (případně objektů) nacházejících se na obraze. Výstupem pak může být třída (např. stůl, letadlo, kočka), případně pravděpodobnost náležitosti k dané třídě. Každé třídě odpovídá množina charakteristik, nebo vlastností, a vztahů mezi nimi. Pro rozpoznání třídy pak je tedy potřeba znát, čím je charakterizována. Pokud předem známe vlastnosti dané třídy, můžeme zahrnout prostředky pro jejich rozpoznání přímo do kódu programu například ve formě filtrů, kde každý reaguje na nějakou vlastnost (na nejnižší úrovni například hranu), a podle toho které filtry reagují, třídu odhadnout, rozpoznat.[1]

V reálném světě se ale většina věcí vyskytuje v různých podobách, tím rozumíme v různých situacích, poloze a velikosti (nebo jen bodu, odkud byl obraz pořízen), jasech, barvách, navíc se ani vzájemně nemusí moc podobat, vezměme si jako příklad člověka – liší se nejen vzrůstem, barvou pleti nebo pohlavím, ale stále může jít o stejnou klasifikační třídu (pokud se třeba nesnažíme rozeznat různé lidi). Vytvořit tedy filtr, který si s touto variabilitou dokáže poradit je pak velmi těžké. Nezapomínejme také, že počítač „vidí“ obraz vlastním způsobem. Je to pro něj jen pole hodnot odpovídajících jednotlivým barevným kanálům.

3.5.2 **Přizpůsobení NN pro práci s obrazem**

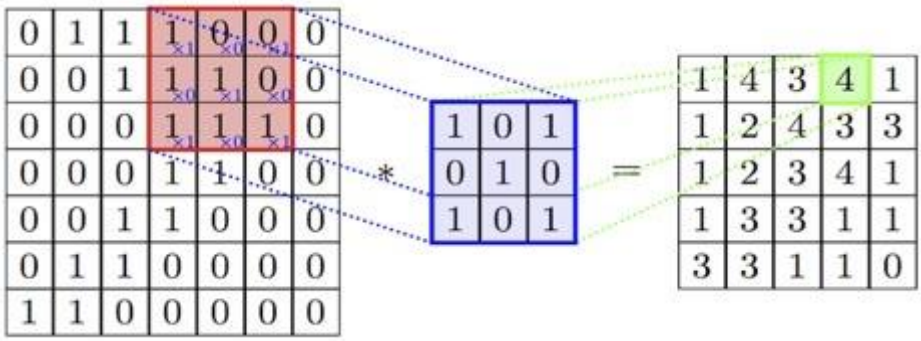
Pokud bychom se pro klasifikaci obrazu pokusili využít neuronovou síť tak, jak byla v této práci popsána do teď, setkáme se s následujícím problémem – máme hlubokou plně propojenou síť, které dáme jako vstup vektor hodnot. Naším vstupem je obraz, který předáváme jako vektor pixelů. V naší síti jsou všechny vstupní neurony propojeny se všemi neurony z následující vrstvy. Řekněme, že vstupní obrázek je černobílý, o rozměru třeba 28x28 pixelů, a první schovaná vrstva obsahuje 10 neuronů. To by pak znamenalo, že pro velmi malý obrázek máme již mezi prvními dvěma vrstvami 7840 parametrů (vah synapsí). Pokud by navíc byl barevný, s kanály RGB (red, green, blue), zvětšil by se tento počet parametrů trojnásobně. Navíc, pokud rozpojíme pixely do vektoru a propojíme každý jednotlivý pixel s každým neuronem následující vrstvy, ztratíme tak většinu prostorových informací. Je tedy potřeba alespoň první část sítě práci s obrazem nějak přizpůsobit.

3.5.3 **Konvoluční filtry**

Stejně jako je základní princip neuronových sítí vzdáleně inspirován mozkem, tak je jím inspirováno i řešení problému, jak síti informace z obrazu předat, které využívá neuronů s modifikovanou funkcí.[1] Za účelem zachování prostorových informací napojíme jednotlivé tyto neurony na části obrazu. Představme si, že máme malé okénko, které postupně po obrazu posouváme, a jeho obsah vždy propojíme s jedním neuronem, dokud nepokryjeme celý obraz. Toto okénko je pak tzv. recepčním polem daného neuronu. Prostorové informace jsou dále zachovány tím, že sousedící recepční pole se navzájem částečně překrývají. Velikost posunu recepčního pole mezi dvěma po sobě jdoucími neurony je pak pro síť důležitým hyperparametrem.

Účelem těchto neuronů je analyzovat své recepční pole z hlediska nějaké vlastnosti. Každý neuron pak hledá nějakou vlastnost pouze ve svém recepčním poli, které lze chápat také jako tabulku nebo matici hodnot, a to s využitím již dříve zmíněných filtrů. Filtr je v tomto případě tabulkou vah aplikovanou na tabulku recepčního pole. Vzato matematicky, dochází k diskrétní konvoluci, jež má pro schovaný neuron (m,n) s recepčním polem 3x3 pixelů a tedy filtrem stejných rozměrů, s přidáním biasu b, podobu:

$$\sum_{i=1}^{3}\sum_{j=1}^{3} w_{i,j}x_{i+m,j+n} + b$$



Obr. 6: Vizualizace konvoluce s konvolučním filtrem bez biasu[46]

Jsou tedy navzájem roznásobeny prvky (hodnota z kanálu obrazu a váha z filtru) na stejném místě, výsledné součiny jsou sečteny a je přidán bias. Filtr i neuron, případně celá vrstva neuronů, jsou pak nazývány podle této funkce – konvoluční. Užívanou aktivační funkcí konvolučních neuronů je dříve zmíněná ReLU funkce. Konvoluční vrstva má 2 rozměry závislé na velikosti obrazu a recepčního pole svých neuronů, a třetí závislý na počtu konvolučních filtrů. Vstup může být upraven např. doplněním nul tak, aby byla konvoluce provedena i s hodnotami na okrajích v jejím středu a její výstup měl stejné rozměry. Této technice se říká padding, možno přeložit jako vycpávání.

Velkou výhodou sítě využívající konvolučních vrstev, a vlastně i důvodem našeho snažení, je, že ony váhy v konvolučních filtrech není potřeba předem určovat, můžeme je inicializovat s náhodnými hodnotami, které následně budou optimalizovány během učení!

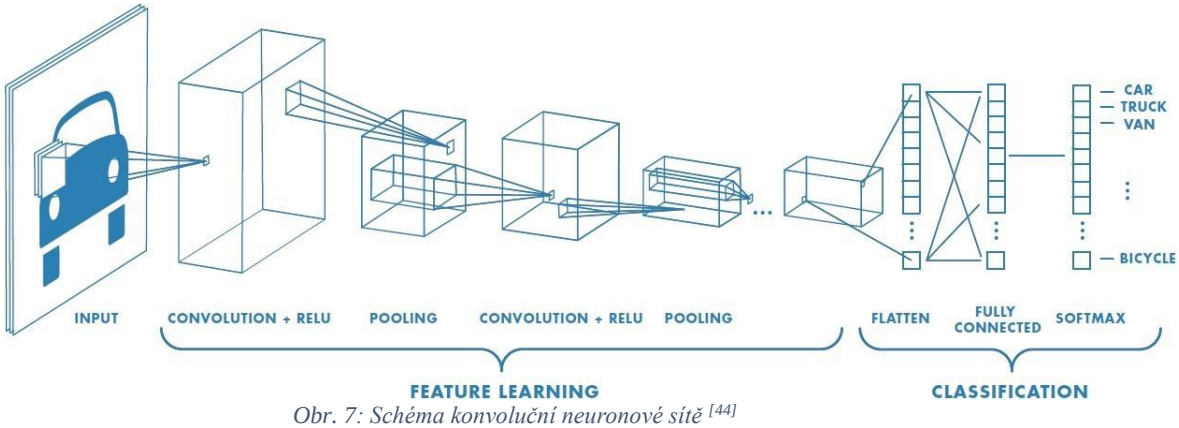
3.5.4 **Konvoluční neuronová síť**

Neuronová síť využívající konvoluci může podle toho být nazvána konvoluční. Pro klasifikaci obrazu má dvě hlavní části.

První část analyzuje obraz, pomocí konvolučních filtrů v něm hledá základní vlastnosti nebo rysy, v nichž pak s každou další konvoluční vrstvou nachází vlastnosti komplexnější, proto s hloubkou roste počet konvolučních filtrů. Může také využívat tzv. pooling, doslovně přeloženo – shromažďujících, vrstev, které snižují výpočetní náročnost snižováním rozměrů vrstev tak, že „mačkají“ hodnoty do sebe například jejich průměrováním. Pro tento účel bývají také využívané max-pooling vrstvy, které oblast reprezentují vždy nejvyšší aktivační hodnotou, která se v nich vyskytuje. [5]

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Druhá část vyhodnocuje nalezené vlastnosti. Jde o plně propojené vrstvy. Výstupem je vektor pravděpodobností odpovídajících jednotlivým třídám. Pro poslední vrstvu proto bývá jako aktivační funkce používána funkce softmax, která zajistí, že výstup odpovídá normalizovanému pravděpodobnostnímu rozdělení, tedy že hodnoty výstupu jsou čísla od nuly do jedné a jejich součet je roven jedné.



Obr. 7: Schéma konvoluční neuronové sítě [44]

Síť je trénována na obrazech, optimalizovány jsou jak váhy v konvolučních filtrech, tak i váhy synapsí plně propojených vrstev. Jako účelovou funkci lze použít variantu dříve zmíněné funkce křížové entropie pro více klasifikačních tříd:

$$J(\boldsymbol{\theta}) = \sum_{i} \left(y^{(i)} \ln\left(\hat{\boldsymbol{y}}^{(i)}\right) \right)$$

kde:

- $\boldsymbol{\theta}$ obsahuje jak váhy z plně propojených vrstev, tak i z konvolučních filtrů
- $y^{(i)}$ je předpokládané rozdělení pravděpodobností
- $\hat{\boldsymbol{y}}^{(i)}$ je výstup sítě

3.6 Užité nástroje

3.6.1 Python

Python je open-source vysokoúrovňový programovací jazyk podporující objektově orientované programovaní. Velmi přístupným ho mimo jiné činí jednoduchá čitelnost kódu, dynamická kontrola typů, nebo automatická správa paměti (tzv. garbage collection). Verze 2.7 přestala být podporována s rokem 2020 a v současnosti se používá verze 3.x.

Vzhledem k tomu, že je implementován v programovacím jazyce C, může být jeho interpret zabudován do jiné aplikace psané v jazycích C nebo C++, kde je ho pak možno využít pro pružné rozšiřování funkčnosti aplikace bez zasahování do zdrojového kódu.

3.6.2 Tensorflow a Keras

Tensorflow je open-source software knihovna pro numerickou matematiku využívající grafy toku dat. Mimo jiné umožňuje práci i na grafických procesorech a větší časovou efektivitu s tím spojenou. Samotné transformační operace jsou velmi účinně implementovány v C++, nad nimiž jsou vytvořeny front endy v C++ nebo Pythonu. Nad nimi dále existují API vyšších úrovní, např. Keras.

Keras je open-source vysokoúrovňová API navržená pro jednoduché a rychlé prototypování modelů neuronových sítí. Podle jeho hlavního autora F. Cholleta je míněn spíše jako rozhraní, front end, pro deep learning než jako samostatný framework. Jako back end může mimo jiné využívat dříve zmíněný Tensorflow.

Základní datová struktura Kerasu je označena klíčovým slovem model. Tato třída má ve své nejjednodušší podobě, označené slovem sequential, podobu zásobníku vrstev neuronů. Model pak má metody, které umožňují jednoduše:

- přidávat jednotlivé vrstvy neuronů (add),
- konfigurovat proces učení (compile),
- trénovat model na trénovacím datasetu (fit),
- zhodnotit výkon vytrénovaného modelu na testovacím datasetu (evaluate),
- aplikovat vytrénovaný model (predict).

3.6.3 Jupyter Notebook

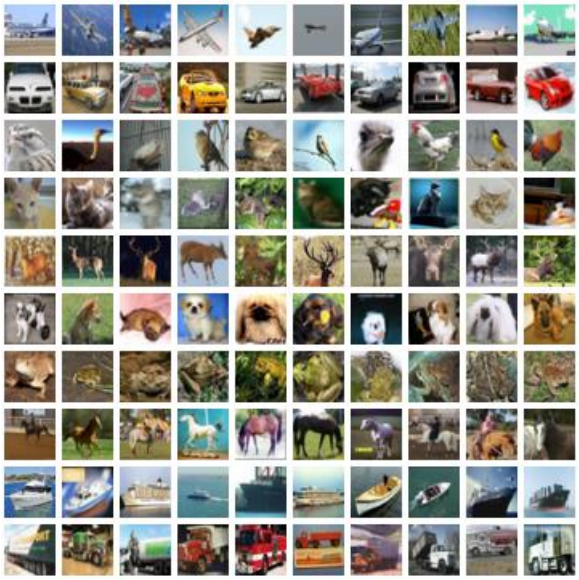
Jupyter Notebook je open-source webová aplikace umožňující vytváření JSON dokumentů obsahujících uspořádaný seznam jednotlivých buněk, které mohou nést text, výpočty, vizualizace, nebo i zdrojový kód. Podporuje různé programovací jazyky, mimo jiné i Python, kde napodobuje jeho konzoli s tím rozdílem, že příkazy a jejich výstupy se v rámci buňky ukládají.

Google Colaboratory umožňuje zdarma spouštět Jupyter Notebook v cloudu a ve spojení používat Python, Tensorflow a Keras, a to i s využitím GPU a napojením na Google Drive.

3.7 CIFAR10

CIFAR-10 and CIFAR-100 jsou podmnožinami datasetu 80 million tiny images. CIFAR-10 dataset obsahuje 60000 barevných obrázku o velikosti 32x32 pixelů, označených podle toho, co se na nich vyskytuje. Na obrázku je vždy jedna věc odpovídající jedné z deseti klasifikačních tříd, (6000 obrázků pro každou třídu). Data jsou rozdělena na 50000 trénovacích a 10000 testovacích (nebo také validačních) obrázků. Klasifikační třídy jsou brány v pořadí: letadlo, auto, pták, kočka, vysoká zvěř, pes, žába, kůň, loď, nákladní automobil.

Žádná z klasifikačních tříd se nepřekrývá, například nákladním automobilem se rozumí pouze velké tzv. TIRáky, třeba vozidla typu pick-up v datasetu proto nejsou.[13] [14]



Obr. 8: Příklad obrázků z datasetu CIFAR10 [13]

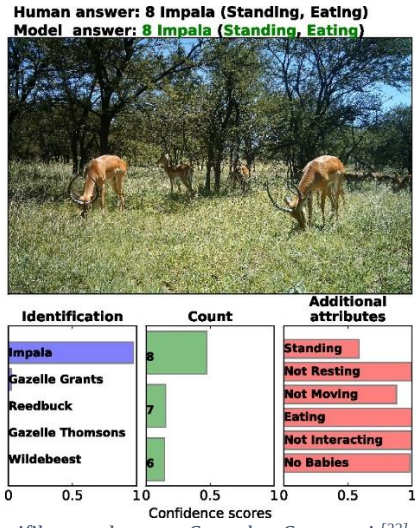
3.8 Aplikace CNN ve Smart Agriculture

Pro využití konvolučních sítí a jejich schopnosti provádět automatizované digitální zpracování obrazu se v zemědělství nabízí několik příležitostí. Uvedeme několik příkladů z posledních let, které ukazují potenciál využití konvolučních neuronových sítí v této oblasti.

3.8.1 Rozpoznávání a identifikace zvířat

Konvoluční neuronové sítě mohou provádět automatické zpracování obrazových dat pořízených například drony nebo fotografickými pastmi. Verma et al.[21] dosáhl přesnosti klasifikace fotografií zvířat z fotografických pastí přes 90 %. Trnovszky et al.[24] ukázal, že konvoluční sítě mohou podat lepší výkon než jiné

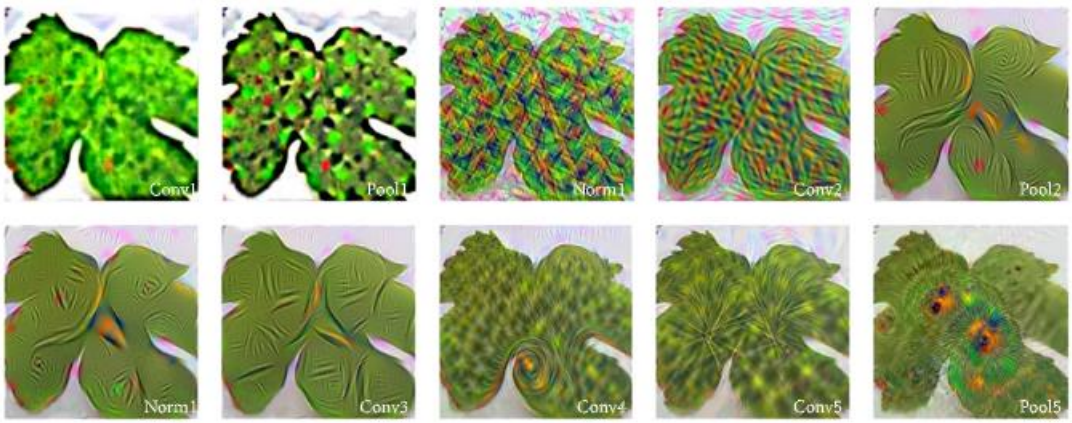
systémy pro rozpoznávání zvířat. Norouzzadeh et al.[22] dokázal klasifikovat fotografie z datasetu Snapshot Serengeti[23] se stejnou přesností jako dobrovolníci vytvářející označení pro jednotlivé položky datasetu. U obrazu byly určovány také další vlastnosti, například počet nebo činnost přítomné zvěře.



Obr. 9: Ukázka klasifikace z datasetu Snapshot Serengeti [22]

3.8.2 Kontrola zdraví rostlin

Manuálně kontrolovat, zdraví pěstovaných rostlin je s rostoucím objemem velmi časově náročné. Vzhledem k tomu, že mnoho informací o zdraví rostliny jde získat z jejího vzhledu, lze kontrolu provádět zpracováním obrazu, pomocí klasifikace listů rostlin a rozpoznávání vzorců[26]. Model od Sladojevic et al.[27], dokáže na obrazu list identifikovat a rozeznat 13 různých typů viditelných rostlinných chorob s konečnou průměrnou přesností 96,3 %. Model pracoval s databází čítající 3000 obrazů listů vzatých z internetu, rozšířenou na více než 30000 obrazů pomocí augmentace dat.



Obr. 10: Výstupy konvolučních a poolingových vrstev (Sladojevic et al.) [27]

Fuentes et al.[20] vytvořil model schopný v reálném čase rozpoznávat napadení nemocí, či škůdcem u listů rajčat.

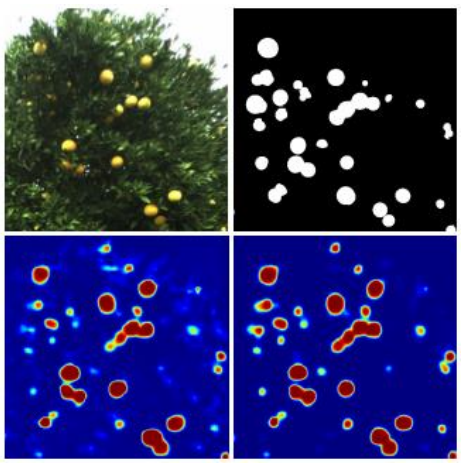
Sun et al.[29] navrhl architekturu pro hodnocení vzhledu rostliny na základě tří fotografií pořízených z různých úhlů. Model bere jako vstup 3 fotografie najednou a dosáhl přesnosti 89,6 %.

3.8.3 **Rozpoznávání rostlin**

Další možnou aplikací je samotné rozpoznávání rostlin, například identifikace plevelu. Tang et al.[28] dosáhl přesnosti 92,89 % při rozpoznávání plevelů spjatých s pěstováním sóji využitím algoritmu shlukové analýzy nazývaného k-means pro nenáhodnou inicializaci parametrů před trénováním. Yalcin et al.[30] ukázal, že pro problém rozpoznávání fenologie rostlin dokáže předem trénovaná konvoluční neuronová síť založená na síti AlexNet dosáhnout lepších výsledků než algoritmy strojového učení založené na manuálně vytvořených kritériích pro rozpoznávání.

3.8.4 **Počítání plodů**

Chen et al.[31] dosáhl slibných výsledků v práci, kde řešil problém počítání jablek a pomerančů na fotografiích pomocí dvou neuronových sítí. Jedna prováděla detekci identifikací tvarů odpovídajících „kaňkám“ na fotografii (anglicky označeno – blob) a druhá pak počítání, protože ovoce se na fotografiích samozřejmě může překrývat, jedna „kaňka“ tedy může odpovídat většímu počtu kusů. Museli se potýkat s chybami označení trénovacích dat, přesné určení počtu na fotografii může být velmi těžké i pro člověka, jak je možno vidět na následujícím obrázku (nahoře vlevo původní obrázek, vpravo očekávaný výsledek; dole vlevo výsledek segmentace po 2500 iteracích, vpravo po 50000 iteracích).



Obr. 11: Chen et al.[31] - ukázka segmentace

3.8.5 **Analýza dat o půdě pořízených satelity či bezpilotními letouny**

Kussul et al.[32] ukazuje možnost aplikace na satelitní snímky pro oblast monitorování území, využití půdy, či klimatických změn, využívajíc víceúrovňového hierarchického modelu. Postup je rozdělen do čtyř kroků: filtrování šumu a shlukování dat, klasifikace území, post-processing a další filtrování, analýza prostorových dat.

Lu et al.[33] využil bezpilotních letadel (UAV), pro pořízení obrazů s vysokým rozlišením z nízké letové hladiny, ze kterých pak získal data o orné půdě pomocí konvoluční neuronové sítě a transfer learnignu.

3.8.6 **Další aplikace**

Mezi další aplikace se řadí rozpoznávání objektů, například detekce překážek v cestě autonomních strojů pro předcházení haváriím, či předpověď počasí. [15] [34] [35] [36]

Kamilaris et al.[25] provedl průzkum týkající se využití nejen konvolučních neuronových sítí v zemědělství.

4 Vlastní práce

Všechny modely (Model 0-3) jsou trénovány s využitím GPU. Počáteční architektura neuronové sítě je založena na příkladu sítě pro CIFAR10 dostupné v dokumentaci Kerasu.

4.1 Tvorba počátečního Modelu 0

Model je inicializován a následně jsou přidávány jednotlivé vrstvy. Podoba vstupní vrstvy je určena podle rozměrů první položky trénovacích dat pomocí parametru input_shape. Jde o vrstvu provádějící dvojrozměrnou konvoluci o rozměru 3x3, obsahující 32 filtrů. Padding je nastaven 'same', vstup je tedy doplněn nulami na takové rozměry, aby výstup konvolučního filtru měl stejné rozměry jako vstup. Aktivační funkce je nastavena na ReLU a vždy po dvou konvolucích je aplikován 2x2 MaxPooling pro snížení počtu parametrů, a tedy výpočetní náročnosti. Následuje vrstva zprostředkovávající dropout. Tato sekvence je pak zopakována s jedním rozdílem, kterým je dvojnásobný počet konvolučních filtrů.

```
# Inicializace modelu
model = Sequential()

# Konvoluční část
model.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

Obr. 12: Model 0 - inicializace a přidání konvoluční části

Po takto vytvořené konvoluční části neuronové sítě následuje klasifikační část. Výstup konvoluční části je převeden do jednoho rozměru, aby na něj mohla být napojena plně propojená vrstva čítající 512 neuronů s aktivační funkcí ReLU, pak dropout, a nakonec výstupní vrstva s deseti neurony, každým odpovídajícím pravděpodobnosti náležitosti k jedné z klasifikačních tříd, s aktivační funkcí softmax.

```
# Klasifikační část
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

Obr. 13: Model 0 - přidání klasifikační části

Dataset je rozdělen na trénovací a validační data. Správné odpovědi v něm mají podobu pořadového čísla odpovídající klasifikační třídy, pro možnost porovnání s výstupem sítě při výpočtu hodnoty účelové funkce jsou převedeny do podoby nulového vektoru s hodnotou 1 na pozici odpovídající pořadovému číslu správné klasifikační třídy. Data obrázků jsou normalizována pomocí min-max normalizace.

```
# Rozdělení dat do proměnných
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Úprava podoby očekávaných odpovědí
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Normalizace dat
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

Obr. 14: Model 0 - Příprava dat

Pro optimalizaci je použit algoritmus RMSprop inicializován s mírou učení 0.0001 a přidaným jejím malým „chátráním“. Během kompilace modelu je mu určena účelová funkce, optimalizační algoritmus pro trénování, a funkce sledující výkon modelu. Následně je spuštěno trénování s dávkami o velikosti 32 položek z datasetu po 100 epoch, trénovací data jsou každou epochu náhodně promíchána.

```
batch_size = 32
epochs = 100

# Inicializace RMSprop
opt = keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

# Určení nákladové funkce, optimalizačního algoritmu a sledované metriky
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

# Trénování
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test),
          shuffle=True)
```

Obr. 15: Model 0 - RMSprop, kompilace a spuštění trénování

Po skončení trénování je model zhodnocen s využitím validačních dat z datasetu, sledována je přesnost klasifikace a hodnota účelové funkce.

```
# Vyhodnocení modelu
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

Obr. 16: Model 0 - vyhodnocení

4.1.1 Rozměry a počet parametrů

Vstupem pro síť je obrázek 32x32 se třemi kanály podle barev (RGB). Jde vyjádřit trojrozměrným polem o rozměrech (32, 32, 3). S ohledem na počet filtrů a padding první konvoluční vrstvy, který každý prvek doplní nulami na rozměry 34x34, je jejím výstupem pole o rozměrech (32, 32, 32), rozměry se mění i s druhou konvolucí, která nemá padding, proto má její výstup rozměr (30, 30, 32). Ke změně rozměrů dále dochází s poolingovou vrstvou, která zmenší rozměry aktivací filtrů na polovinu, jejím výstupem je tedy pole o rozměrech (15x15x32). Následující konvoluční vrstva má padding a 64 filtrů, změní tedy rozměry na (15x15x64), po ní jdoucí konvoluce na (13x13x64) a poolingová vrstva na (6x6x64).

Vícerozměrný výstup konvoluční části neuronové sítě je pak převeden do vstupní vrstvy klasifikační části tak, že je „zploštěn“ do jednoho rozměru. Vstupní vrstva klasifikační části neuronové sítě tedy obsahuje 2304 neuronů, další vrstvy mají počet neuronů pevně dán (512 a 10).

Což se počtu parametrů týče, první konvoluční vrstva obsahuje 32 filtrů velikosti 3x3, které jsou aplikovány na 3 barevné kanály a mají bias. První vrstva má tedy 896 trénovatelných parametrů, druhá 9248, tato vrstva je aplikována na aktivace filtrů předchozí vrstvy. Poolingová vrstva, stejně jako dropout, trénována není. Následující konvoluce přidávají postupně 18496 a 36928 parametrů. První plně propojená vrstva má 1180160 parametrů a výstupní vrstva 5130. Celkový počet trénovatelných parametrů je tedy 1250858. Jde o váhy v jednotlivých filtrech a jejich biasy.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
activation_4 (Activation)    (None, 13, 13, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
activation_5 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
_________________________________________________________________
```

Obr. 17: Model 0 - schéma

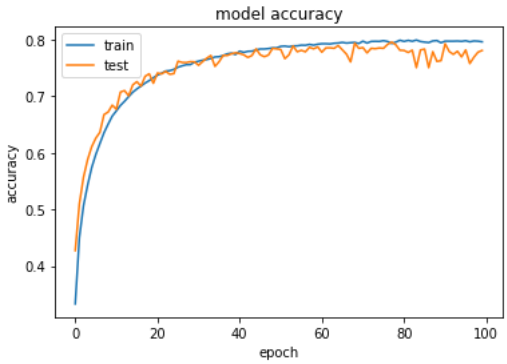
4.1.2 **Zhodnocení Modelu 0**

Tento model obsahuje 2 dvojice konvolučních vrstev, po kterých vždy následuje MaxPooling a dropout. Klasifikační část modelu obsahuje dvě plně propojené vrstvy – jednu schovanou a jednu výstupní. Aktivační funkcí je ReLU až na poslední vrstvu, kde je pro vytvoření výstupu v podobě pravděpodobnostního rozdělení využita aktivační funkce softmax. Jako účelová funkce využita křížová entropie, optimalizační algoritmus RMSprop.

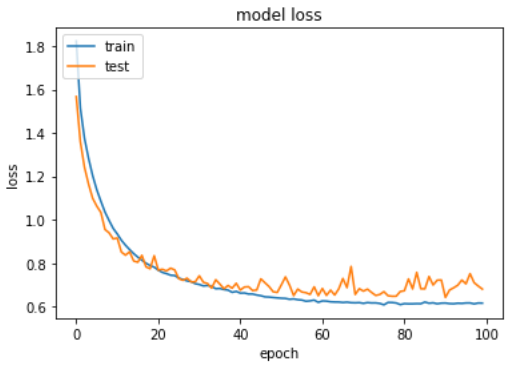
Celkový počet parametrů modelu je 1250858, všechny trénovatelné. Síť byla trénována po 100 epoch, jedna epocha zabrala v průměru 9 sekund, celkem učení zabralo zhruba 15 a půl minuty. Míra učení konstantně 0,0001.

Výsledky na testovacích datech:

- hodnota účelové funkce – 0,680431597328186
- přesnost klasifikace – 78,04 %



Obr. 18: Model 0 – přesnost



Obr. 19: Model 0 – nákladová funkce

4.2 Model 1

Počáteční Model 0 tedy dosáhl přesnosti 78,04 %, což nelze pokládat za spolehlivé. Pokusíme se tedy využít postupů popsaných v teoretické části a přesnost zvýšit.

Pro začátek se zaměřme na trénovací data. Síť je každou epochu trénována na stejné množině obrázků, funkce Kerasu model.fit(), která provádí trénování, sice má implicitně nastaveno, aby data pro každou epochu promíchala, můžeme ale jejich vypovídající hodnotu dále zvýšit, a to s využitím lepší normalizace a augmentace dat.

Pro normalizaci použijeme z-skóre. S pomocí knihovny NumPy spočteme aritmetický průměr a směrodatnou odchylku trénovacích dat. Data pak transformujeme tak, že odečteme průměr a výsledek vydělíme směrodatnou odchylkou. Jednotlivé hodnoty pak zachycují vzdálenost od průměru ve směrodatných odchylkách. Nesmíme zapomenout transformovat i testovací data, aby se jejich podoba shodovala s podobou dat, na kterých bude model vycvičen.

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

#z-score
mean = np.mean(x_train,axis=(0,1,2,3))
std = np.std(x_train,axis=(0,1,2,3))
x_train = (x_train-mean)/(std)
x_test = (x_test-mean)/(std)

#Augmentace dat
datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True)

datagen.fit(x_train)
```

Obr. 20: Model 1 – z-skóre a augmentace dat

Což se augmentace dat týče, Keras má třídu ImageDataGenerator, která obsahuje vše potřebné. S augmentací je opatrnost na místě, nesmíme změnit povahu trénovacích dat až příliš, tedy do bodu, kdy už nemusí odpovídat testovacím, potažmo reálným, datům. Zvolme tedy, že obrázek může být posunut v jakémkoli směru o maximálně desetinu své velikosti, otočen do patnácti stupňů a převrácen vodorovně. Při posunu je potřeba doplnit vzniklý prázdný prostor, způsob doplnění nechme na základním nastavení – opakování poslední hodnoty.

Pro trénování bude potřeba zavolat metodu počítající s generátorem, které bude instance jeho třídy na základě trénovacích dat generovat dávky. Vzhledem k tomu, že nevyužíváme výpočetní kapacitu ani zdaleka naplno, můžeme také zvětšit dávky na 64 položek.

Přístup k délce trénování lze také vylepšit. Pevně daný počet epoch lze těžko jen tak trefit na optimální, můžeme ho tedy nastavit na nějaké vysoké číslo, abychom měli jistotu, že trénování nebude přerušeno příliš brzy, a zároveň můžeme využít early stopping, aby trénování neprobíhalo zbytečně a byl omezen overfitting. Budeme tedy sledovat hodnotu účelové funkce pro testovací data každou epochu a pokud se nezlepší po 5 epoch, trénování bude přerušeno a načtou se hodnoty parametrů z nejlepší epochy. Zkusme také použít jiný optimalizační algoritmus, který se nám postará o míru učení – Adadelta.

```
batch_size = 64
num_epochs = 200

# Inicializace Adadelta
opt_adadelta = Adadelta(lr=1.0, rho=0.95)

# Inicializace EarlyStopping
earlystop = EarlyStopping(
              monitor='val_loss',
              min_delta=0,
              patience=5,
              verbose=1,
              mode='min',
              baseline=None,
              restore_best_weights=True)

# Kompilace modelu
model.compile(loss='categorical_crossentropy',
              optimizer=opt_adadelta,
              metrics=['accuracy'])

# Trénování
history = model.fit_generator(
                  datagen.flow(x_train, y_train, batch_size=batch_size),
                  steps_per_epoch=x_train.shape[0] // batch_size,
                  epochs=num_epochs,
                  verbose=1,
                  validation_data=(x_test,y_test),
                  callbacks=[earlystop])
```

Obr. 21: Model 1 - Adadelta, EarlyStopping, kompilace, trénování

Dále přejděme k samotné architektuře neuronové sítě. Zde můžeme zkusit model rozšířit do hloubky, což by mu mělo umožnit zachytit komplexnější vztahy. Přidejme tedy do konvoluční části další dva bloky obsahující dvojici konvolucí, maxpooling a dropout.

S teoreticky rostoucí kapacitou modelu může vzrůst i jeho schopnost přizpůsobit se trénovacím datům až moc do detailu (overfitting), musíme tedy proti tomu zavést opatření. Zaveďme regulaci vah L2, abychom předešli velkým hodnotám vah, které by mohly přinést velké gradienty v nežádoucích směrech.

Po každé aktivaci můžeme data znovu normalizovat, abychom předešli deformaci rozdělení dat v důsledku změn ve váhách během trénování. Na aktivace také bude mít vliv změna normalizace dat. Nově, s využitím z-skóre, se mohou v datech objevit záporná čísla, pro která je aktivace ReLU vždy nulová. Přejděme tedy na podobnou funkci, která je ale citlivá i pro záporné hodnoty – ELU. Dejme také každé konvoluci stejný padding, aby byla zachována vypovídací hodnota dat na okrajích a rozměry v rámci jednoho bloku.

```
model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))
```

Obr. 22: Model 1 - blok konvoluční části

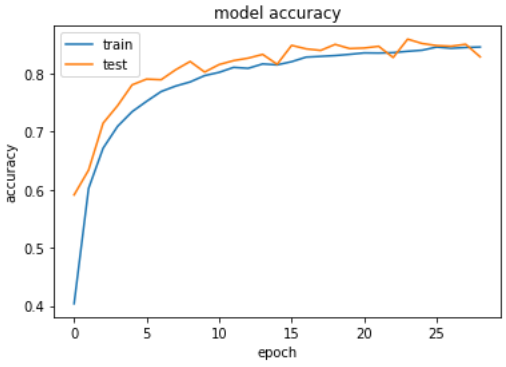
4.2.1 Zhodnocení Modelu 1

Tento model obsahuje 4 dvojice konvolučních vrstev, po kterých vždy následuje MaxPooling a dropout. Klasifikační část modelu obsahuje dvě plně propojené vrstvy – jednu schovanou a jednu výstupní. Aktivační funkcí je ELU až na poslední vrstvu, kde je pro vytvoření výstupu v podobě pravděpodobnostního rozdělení využita aktivační funkce softmax. Jako účelová funkce využita křížová entropie, optimalizační algoritmus Adadelta.

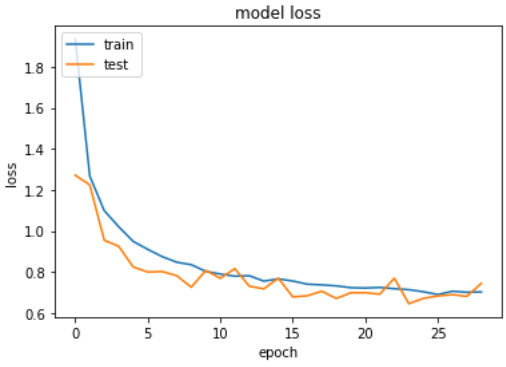
Celkový počet trénovatelných parametrů modelu je 1706026. Trénování bylo přerušeno v epoše 24, konečné parametry tedy odpovídají epoše 19. Jedna epocha zabrala v průměru 29 sekund, celkem učení zabralo zhruba pod 12 minut.

Výsledky na testovacích datech:

- hodnota účelové funkce – 0,657147852897644
- přesnost klasifikace – 85,68 %



Obr. 23: Model 1 - přesnost



Obr. 24: Model 1 - nákladová funkce

4.3 Model 2

I přes výrazně větší přesnost nového modelu je ihned zřejmý alespoň jeden problém. Hodnota účelové funkce testovacího datasetu relativně hodně osciluje, což pravděpodobně vedlo k příliš brzkému zastavení trénování. Je tedy potřeba zvýšit trpělivost metody EarlyStopping. Zároveň vzhledem k tomu, že trénování zatím neprobíhá moc dlouho, je asi

zbytečné užívat algoritmus Adadelta, nejspíše postačí jednodušší RMSprop s adaptivními změnami míry učení. Nastavíme ji na 0,001 a budeme sledovat hodnotu účelové funkce testovacích dat. Pokud se během 8 epoch nezlepší, snížíme míru učení na polovinu až do hodnoty 0,0001. Samozřejmě nechceme zastavit trénování před dosažením nejnižší hodnoty míry učení, je tedy potřeba také zvětšit trpělivost metody EarlyStopping, třeba na 16 epoch.

```
# Adaptivní změna míry učení
reduce_lr = ReduceLROnPlateau(
                monitor='val_loss',
                factor=0.5,
                patience=8,
                verbose=1,
                mode='min',
                min_delta=1e-4,
                cooldown=1,
                min_lr=1e-4)

# EarlyStopping
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=16, ...
```

Obr. 25: Model 2 - kontrola míry učení a větší trpělivost

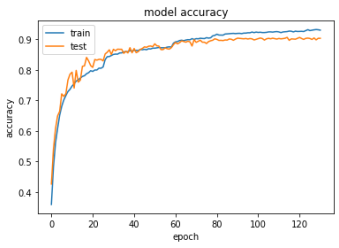
4.3.1 Zhodnocení Modelu 2

Tento model má stejnou architekturu, jako model předchozí, změněn byl pouze optimalizační algoritmus na RMSprop, bylo přidáno adaptivní snižování míry učení a zvýšena trpělivost zastavení trénování.

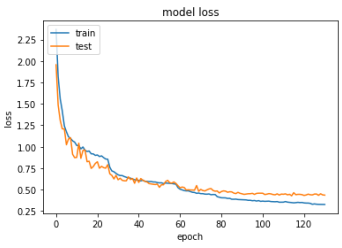
Trénování bylo přerušeno v epoše 131, konečné parametry tedy odpovídají epoše 115. Jedna epocha zabrala v průměru 27 sekund, celkem učení zabralo zhruba necelou hodinu.

Výsledky na testovacích datech:

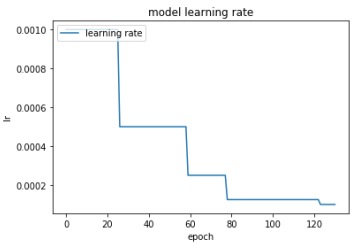
- hodnota účelové funkce – 0,42731796441078185
- přesnost klasifikace – 90,59 %



Obr. 26: Model 2 - přesnost



Obr. 27: Model 2 - nákl. funkce



Obr. 28: Model 2 - míra učení

4.4 Model 3

Trénování předchozího modelu bylo ukončeno velmi krátce po dosažení minimální hodnoty míry učení, neuškodí asi pro jistotu ještě trochu zvýšit trpělivost, třeba na 20 epoch. Z hodnot nákladové funkce pro trénovací a testovací data je zároveň vidět, že trénování bylo zastaveno právě, když se začínal projevovat overfitting – hodnota se zlepšovala již pouze pro trénovací data. Zkusíme využít další metodu pro zvětšení robustnosti modelu, přidáme za každý blok šum.

```
model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3,3), padding='same', kernel_regularizer=regularizers.l2(weight_decay)))
model.add(Activation('elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(GaussianNoise(0.1))
model.add(Dropout(0.4))
```

Obr. 29: Model 3 - blok konvolucí a přidaný šum

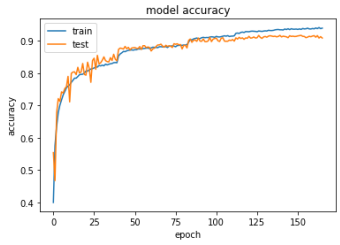
4.4.1 Zhodnocení Modelu 3

Architektura byla lehce pozměněna přidáním šumu za každou dvojici konvolucí. Také byla zvýšena trpělivost zastavení trénování.

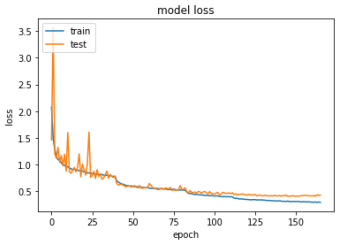
Trénování bylo přerušeno v epoše 166, konečné parametry tedy odpovídají epoše 146. Jedna epocha zabrala v průměru 27 sekund, celkem učení zabralo zhruba hodinu a čtvrt.

Výsledky na testovacích datech:

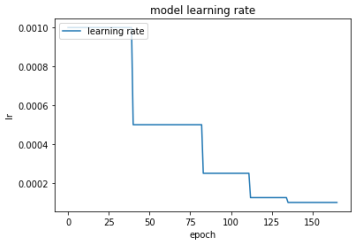
- hodnota účelové funkce – 0,4040874563694
- přesnost klasifikace – 91,53 %



Obr. 30: Model 3 - přesnost



Obr. 31: Model 3 - nákl. funkce



Obr. 32: Model 3 - míra učení

4.5 Porovnávané sítě

Model VGG-16 obsahuje 15,001,418 parametrů a pro dataset CIFAR10 byl trénován necelých 5 hodin na jedné GPU. Jeho architektura je podobná Modelu 3, ale mnohem hlubší, obsahující bloky dvou dvojic a tří trojic konvolucí. [16]

Struktura modelu ResNet je složitější než u předešlých modelů – kromě lineárního sledu konvolucí, využívá „zkratek“ v podobě přidaných spojů mezi konvolučními vrstvami, které přeskakují více následujících vrstev a posílají výstup určitých vrstev vzdálenějším vrstvám. Právě podle využívání předávání těchto „zbytkových informací“ (z angličtiny – residuals) a počtu vrstev je pojmenována. ResNet1001 v2 pak obsahuje zhruba 10,2 milionů parametrů a jeho autoři uvádějí, že pro dataset CIFAR10 byl trénován 27 hodin na dvou GPU. [37]

Na koncept ResNet navazuje a dále ho rozvádí DenseNet. Každý blok je zde napojen na aktivace všech bloků po něm jdoucích. Tato architektura dosahuje minimálně stejných výsledků jako ResNet, ale se zhruba poloviční náročností. Model DenseNet-BC (L=190, k=40) obsahuje zhruba 25,6 milionů parametrů. Autoři této architektury dali k dispozici před-trénované modely s optimalizovaným využitím paměti. Také pro využití jejich architektury doporučují využít efektivnější modely Wide-Densenet-BC, kde model o 4,3 milionech parametrů dosahuje přesnosti klasifikace non-wide modelu. [45]

Opravdu velmi hluboký model AmoebaNet-B obsahuje zhruba 557 milionů parametrů, byl vytrénován na datasetu ImageNet ILSVRC-2012 a pak s využitím transfer learningu přizpůsoben pro dataset CIFAR10. Jeho účelem bylo demonstrovat účinnost knihovny GPipe navržené pro účinné trénování velmi hlubokých modelů jejich rozdělením mezi více akcelerátorů, v tomto případě 4, a využitím paralelismu. [38]

5 Výsledky a diskuse

5.1 Výsledky

5.1.1 Přehled modelů

Tabulka 1: Přehled vytvořených modelů

Model	Počet epoch	Přesnost	Nákl. funkce	Počet parametrů
0	100	0,7804	0,680431597	1250858
1	24	0,8568	0,657147852	1706026
2	115	0,9059	0,427317964	1706026
3	146	0,9153	0,404087456	1706026

5.1.2 Přehled porovnávaných sítí

Tabulka 2: Přehled porovnávaných sítí

Model	Přesnost	Počet parametrů (miliony)
Model 3	0,9153	1,7
VGG-16[16][40]	0,9356	15
A. Karpathy[39]	0,9400	člověk
Resnet1001 v2[37][10]	0,9508	10,2
DenseNet-BC[45]	0,9654	25,6
Wide-DenseNet-BC[45]	0,9599	4,3
AmoebaNet-B[38]	0,9900	557

5.2 Diskuse

5.2.1 Model 3

Další prohloubení Modelu 3 do hloubky i šířky nepřineslo zlepšení, ani výrazné zhoršení, změna architektury potřebná pro lepší výkon by tedy musela být výraznější.

Využití metody EarlyStopping silně snižuje pravděpodobnost zlepšení s prodloužením trénování. Nutno ale zmínit, že náchylnost k silnějším výkyvům hodnoty nákladové funkce pro validační data může teoreticky způsobit příliš brzké zastavení trénování, čemuž lze předejít nastavením větší trpělivosti metody.

Posilování augmentace a šumu sice přiblížilo přesnost klasifikace testovacích a validačních dat, nejspíše ale nejde o zlepšení, což se učení z dat či snížení overfittingu týče, spíše jde o to, že testovací data obsahují stejné množství důležitých informací, ale je pak těžší je klasifikovat. Tento závěr podporuje fakt, že hodnota nákladové funkce zůstala přibližně nezměněna.

Vzhledem k tomu, že trénování bylo prováděno v cloudu Google Colab, nebyla přímá kontrola nad hardwarem. Někdy časy trénování bez jakýchkoli změn modelu vzrostly až o 20 sekund, na vině by teoreticky mohlo být rozdělování výpočetní kapacity uživatelům.

Ve shrnutí výsledků lze vidět zvláštní skok hodnoty nákladové funkce mezi modely 1 a 2 nepřiměřený stoupající přesnosti. Zde je nutno se zamyslet nad povahou jednotlivých veličin. Přesnost odráží pouze, jestli se model trefil, její určení tedy bere v potaz pouze stav ano nebo ne. Odhady modelu jsou ale vyhodnocovány na základě pravděpodobností, které lze interpretovat jako nakolik si je model jist, že klasifikovaný objekt náleží nějaké klasifikační třídě. Pokud se model trefí, přesnost už nezajímá, jak moc si byl svým odhadem jist. To ale nákladové funkci neunikne, protože se snaží přesně určit, jak velkou chybu model udělal. Proto nelze od těchto dvou veličin čekat navzájem úplně přiměřené změny. Onen skok hodnoty lze s přihlédnutím na přesnost vysvětlit tak, že model nejen klasifikoval přesněji, ale odhady vytvářel s větší jistotou.

5.2.2 **Porovnání modelů**

Pro srovnání, je v Tabulce 2 zahrnuta i běžně uváděná lidská přesnost klasifikace pro dataset CIFAR10 – 94 %, na základě blogu A. Karpathyho.[39]

Přesnost 91,53 %, které dosáhl Model 3 značně zaostává za lidskou přesností. Lze čekat, že s rostoucí komplexitou řešených problémů budou růst nároky nejen na robustnost ale také i na hloubku užívaného modelu. Na druhou stranu dosažení skvělé přesnosti vyžaduje nejen velmi hlubokou znalost problematiky a velmi komplexní model, ale také prostředky pro jeho realizaci, jako je hardware a čas.

Jako nejlepší volba se jeví architektura DenseNet. Pro tvorbu vlastních modelů se nabízí rychlá a na paměť relativně nenáročná verze Wide-DenseNet-BC. K dispozici jsou také modely, s optimalizovaným využitím paměti, předem vytrénované na datasetu ImageNet.

Jak ukazuje Norouzzadeh et al.[22], dobré výsledky může přinést také průměrování odhadů různých modelů.

6 Závěr

V této práci byly charakterizovány konvoluční neuronové sítě a analyzovány možnosti jejich aplikace v rámci Smart Agriculture v oblastech rozpoznávání a identifikace divokých zvířat, identifikace a kontrola zdraví rostlin, předpověď objemu sklizně, analýza dat o půdě pořízených satelity či bezpilotními letouny.

S využitím veřejně přístupných prostředků byla vytvořena, vytrénována a postupně zlepšena jednoduchá konvoluční neuronová síť, která na datasetu CIFAR10 dosáhla přesnosti klasifikace 91,53 %.

Na základě porovnání architektur vytvořené sítě, VGG-16, ResNet, DenseNet a AmoebaNet je formulován závěr, že dosažení nejlepší přesnosti je velmi náročné na prostředky, a proto je jako nejlepší určena architektura DenseNet, jejíž modifikace Wide-DenseNet-BC dosáhla přesnosti klasifikace na datasetu CIFAR10 95,99 % s využitím mnohem menšího počtu parametrů než ostatní porovnávané sítě dosahující podobných nebo lepších výsledků. Pro architekturu DenseNet-BC jsou také dostupné modely s optimalizovaným využitím paměti předem vytrénované na datasetu ImageNet, což umožňuje její užití pro transfer learning.

7 Seznam použitých zdrojů

[1] GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. Deep learning. MIT press, 2016. ISBN 978-0262035613.

[2] CHOLLET, François. Deep learning with Python. Shelter Island, New York: Manning Publications Co., 2018. ISBN 978-1617294433.

[3] RASHID, Tariq. Make your own neural network. CreateSpace Independent Publishing Platform, 2016. ISBN 978-1530826605.

[4] DEMUTH, Howard B., et al. Neural network design. Martin Hagan, 2014. ISBN-10: 0-9717321-1-6

[5] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012. p. 1097-1105.

[6] KINGMA, Diederik P.; BA, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] RAMACHANDRAN, Prajit; ZOPH, Barret; LE, Quoc V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.9

[8] DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 2011, 12.Jul: 2121-2159.

[9] ZEILER, Matthew D. ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.

[10] CHOLLET, François. Keras documentation. keras. io, 2015.

[11] TIELEMAN, Tijmen; HINTON, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 2012, 4.2: 26-31.

[12] CLEVERT, Djork-Arné; UNTERTHINER, Thomas; HOCHREITER, Sepp. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015.

[13] KRIZHEVSKY, Alex; NAIR, Vinod; HINTON, Geoffrey. The CIFAR-10 dataset. online: http://www. cs. toronto. edu/kriz/cifar. html, 2014, 55.

[14] TORRALBA, Antonio; FERGUS, Rob; FREEMAN, William T. 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE transactions on pattern analysis and machine intelligence, 2008, 30.11: 1958-1970.
[15] ZHU, Nanyang, et al. Deep learning for smart agriculture: Concepts, tools, applications, and opportunities. International Journal of Agricultural and Biological Engineering, 2018, 11.4: 32-44.
[16] SIMONYAN, Karen; ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
[17] SZEGEDY, Christian, et al. Going deeper with convolutions. arXiv. arXiv preprint arXiv:1409.4842, 2014.
[18] HE, Kaiming, et al. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p. 770-778.
[19] CANZIANI, Alfredo; PASZKE, Adam; CULURCIELLO, Eugenio. An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678, 2016.
[20] FUENTES, Alvaro, et al. A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition. Sensors, 2017, 17.9: 2022.
[21] VERMA, Gyanendra K.; GUPTA, Pragya. Wild animal detection using deep convolutional neural network. In: Proceedings of 2nd international conference on computer vision & image processing. Springer, Singapore, 2018. p. 327-338.
[22] NOROUZZADEH, Mohammad Sadegh, et al. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. Proceedings of the National Academy of Sciences, 2018, 115.25: E5716-E5725.
[23] SWANSON, Alexandra, et al. Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. Scientific data, 2015, 2: 150026.
[24] TRNOVSZKÝ, Tibor, et al. Animal recognition system based on convolutional neural network. 2017.
[25] KAMILARIS, Andreas; PRENAFETA-BOLDÚ, Francesc X. Deep learning in agriculture: A survey. Computers and electronics in agriculture, 2018, 147: 70-90.
[26] YALCIN, Hulya; RAZAVI, Salar. Plant classification using convolutional neural networks. In: 2016 Fifth International Conference on Agro-Geoinformatics (Agro-Geoinformatics). IEEE, 2016. p. 1-5.

[27] SLADOJEVIC, Srdjan, et al. Deep neural networks based recognition of plant diseases by leaf image classification. Computational intelligence and neuroscience, 2016.
[28] TANG, JingLei, et al. Weed identification based on K-means feature learning combined with convolutional neural network. Computers and Electronics in Agriculture, 2017, 135: 63-70.
[29] SUN, Yu, et al. Multi-input convolutional neural network for flower grading. Journal of Electrical and Computer Engineering, 2017, 2017.
[30] YALCIN, Hulya. Plant phenology recognition using deep learning: Deep-Pheno. In: 2017 6th International Conference on Agro-Geoinformatics. IEEE, 2017. p. 1-5.
[31] CHEN, Steven W., et al. Counting apples and oranges with deep learning: A data-driven approach. IEEE Robotics and Automation Letters, 2017, 2.2: 781-788.
[32] KUSSUL, Nataliia, et al. Deep learning approach for large scale land cover mapping based on remote sensing data fusion. In: 2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS). IEEE, 2016. p. 198-201.
[33] LU, Heng, et al. Cultivated land information extraction in UAV imagery based on deep convolutional neural network and transfer learning. Journal of Mountain Science, 2017, 14.4: 731-741.
[34] SALMAN, Afan Galih; KANIGORO, Bayu; HERYADI, Yaya. Weather forecasting using deep learning techniques. In: 2015 international conference on advanced computer science and information systems (ICACSIS). IEEE, 2015. p. 281-285.
[35] SCHER, Sebastian; MESSORI, Gabriele. Predicting weather forecast uncertainty with machine learning. Quarterly Journal of the Royal Meteorological Society, 2018, 144.717: 2830-2841.
[36] RODRIGUES, Eduardo Rocha, et al. DeepDownscale: a deep learning strategy for high-resolution weather forecast. In: 2018 IEEE 14th International Conference on e-Science (e-Science). IEEE, 2018. p. 415-422.
[37] HE, Kaiming, et al. Identity mappings in deep residual networks. In: European conference on computer vision. Springer, Cham, 2016. p. 630-645.
[38] HUANG, Yanping. Introducing GPipe, an Open Source Library for Efficiently Training Large-scale Neural Network Models [online]. [cit. 2020-01-22]. Dostupné z: https://ai.googleblog.com/2019/03/introducing-gpipe-open-source-library.html

[39] KARPATHY, Andrej. Lessons learned from manually classifying CIFAR-10 [online]. [cit. 2020-01-22]. Dostupné z: http://karpathy.github.io/2011/04/27/manually-classifying-cifar10/
[40] LIU, Shuying; DENG, Weihong. Very deep convolutional neural network based image classification using small training sample size. In: 2015 3rd IAPR Asian conference on pattern recognition (ACPR). IEEE, 2015. p. 730-734.
[41] SILVER, David, et al. Mastering the game of Go with deep neural networks and tree search. nature, 2016, 529.7587: 484.
[42] DEAN, Jeff; HÖLZLE, Urs. Build and train machine learning models on our new google cloud tpus, 2017 [online]. [cit. 2020-01-22]. Dostupné z: https://blog.google/products/google-cloud/google-cloud-offer-tpus-machine-learning/
[43] TAIGMAN, Yaniv, et al. Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2014. p. 1701-1708.
[44] ASIRI, Sidath. Building a Convolutional Neural Network for Image Classification with Tensorflow, 2019 [online]. [cit. 2020-01-22]. Dostupné z: https://towardsdatascience.com/building-a-convolutional-neural-network-for-image-classification-with-tensorflow-f1f2f56bd83b
[45] HUANG, Gao, et al. Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017. p. 4700-4708.
[46] NATAKARNKITKUL, Satsawat. Beginner Guides to Convolutional Neural Network from scratch-Kuzushiji-MNIST.” 2019 [online]. [cit. 2020-01-22]. Dostupné z: https://medium.com/@net_satsawat/beginner-guides-to-convolutional-neural-network-from-scratch-kuzushiji-mnist-75f42c175b21