Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a implementace neuronové sítě YOLO3 pro rozpoznávání zvolených objektů v reálném čase

Bc. Vít Hlaváček

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Vít Hlaváček

Systémové inženýrství a informatika
Informatika

Název práce

Návrh a implementace neuronové sítě YOLO3 pro rozpoznávání zvolených objektů v reálném čase

Název anglicky

Design and implementation of the YOLO3 neural network for real-time recognition of selected objects

---

Cíle práce

Cílem práce je prakticky ověřit schopnost a rychlost rozpoznat naučené objekty v reálném čase konvoluční neuronovou sítí YOLO3.

Metodika

1) Prostudujte technologii Darknet.

2) Naprogramujte nástroj učení sítě YOLO3 (předučený model získáte na stránkách Darknet)

3) Vytvořte vhodnou učební sadu nad zvolenými objekty

4) Objekty naučte

5) Vytvořte vhodný video záznam, který bude naučené objekty obsahovat

6) Otestujte síť s využitím videozáznamu

7) Výsledky shrňte a definujte závěry.

Doporučený rozsah práce

53

Klíčová slova

YOLO, Konvoluční neuronová síť, Umělá inteligence, Darknet, C, programování, rozpoznávání objektů

Doporučené zdroje informací

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

Předběžný termín obhajoby

2019/20 ZS – PEF (únor 2020)

Vedoucí práce

Ing. Josef Pavlíček, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 21. 3. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 21. 3. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 31. 03. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a implementace neuronové sítě YOLO3 pro rozpoznávání zvolených objektů v reálném čase" jsem vypracoval 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 autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31. 03. 2020 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Josefu Pavlíčkovi PhD., za ochotu a trpělivost při tvorbě práce. Své rodině, Pavle a Karlovi Hlaváčkům za podporu a Alešovi Hlaváčkovi za to, že mi jde příkladem. V neposlední řadě chci poděkovat Dejvovi Francovi za morální podporu a všechnu tu vynaloženou energii, kterou klidně mohl napájet menší město.

Návrh a implementace neuronové sítě YOLO3 pro rozpoznávání zvolených objektů v reálném čase

Abstrakt

Cílem diplomové práce je prakticky ověřit schopnost a rychlost rozpoznat naučené objekty v reálném čase neuronovou sítí YOLOv3.

První část se zabývá teorií neuronových sítí, strojového učení a hlubokého učení. Dále rozbor nejdůležitějších modelů neuronových sítí a jejich využití v praxi.

Druhá část obsahuje kroky vedoucí k naplnění cílů. Patří tam výběr dat, technologie a prostředí, implementace konvoluční sítě na cloud, trénování modelu i jeho ověření.

V závěru je práce shrnuta a zhodnocena.

Klíčová slova: YOLO, Darknet, TensorFlow, neuronová síť, konvoluční neuronová síť, rozpoznání obrazu, Umělá inteligence, Google Colaboratory, Google Colab, Python, Keras

Design and implementation of the YOLO3 neural network for real-time recognition of selected objects

Abstract

The aim of this master's thesis is to implement and test the ability and performance of YOLOv3 neural network to detect objects in real-time

First part is scientific research of neural networks, machine learning and deep learning. Following part is a breakdown of the most important neural network models and their use-cases in practice.

Second part contains specific steps which lead to reaching our goals. In particular: dataset selection, technologies and the environment, CNN cloud implementation, model training and it's verification.

At the closing part there is a summary and the thesis is appraised.

Keywords: YOLO, Darknet, TensorFlow, neural network, convolutional neural network, CNN, computer vision, Artificial Inteligence, Google Colaboratory, Google Colab, Python, Keras

Obsah

Seznam obrázků

Úvod

K výběru téma diplomové práce výrazně přispěl autorův silný zájem o umělou inteligenci. Moderní technologie, konkrétně umělá inteligence a neuronové sítě, jsou velmi rozsáhlé obory, jejichž vývoj postupuje neskutečně rychle. Informace mohou mnohdy být zastaralé dříve, než je samouk stíhá vstřebat ve volném čase. Závěrečná práce je ideální záminkou, proč se studiu technologie věnovat plnohodnotně.

Tyto technologie autor považuje za stavební kameny informačních technologií, které přetvoří spoustu oblastí a odvětví lidských životů. Ačkoliv se v již objevují v našem každodenním životě, jejich přínosnost má jistě ještě velký potenciál.

Tato diplomová práce se zabývá návrhem a implementací neuronové sítě YOLO3. V této práci lze naleznout spoustu kvalitních poznatků mého výzkumu a následné práce.

První kapitoly teoretické části se zabývají základními definicemi pojmů, definicí Neuronová síť, strojové učení, umělá inteligence a hluboké učení.

V další části pokračuje popis konceptů umělých neuronových sítí.

Poslední část teoretické části obsahuje detailní popis modelů neuronových sítí.

Praktická část začíná datovým souborem, pokračuje posloupností přípravy dat, anotací a augmentací.

Následující část zahrnuje kompletní programovou základnu, konfiguraci a implementaci Frameworku Darknet na domácí stanici. Ze zde vysvětlených důvodů se od této implementace upustí.

Kapitola Cloud: Google Colaboratory klíčovou kapitolou pro praktickou část. Jedná se o řešení, umožňující implementaci YOLOv3 za použití frameworku Darknet v prostředí Jupyter Notebook na cloudové službě Google Colaboratory. Kapitola podrobněji popisuje prostředí Google Colab, porovnává jeho výkon a grafickou akceleraci.

Navazující kapitoly obsahují samotnou implementaci detektoru objektů YOLOv3 s frameworkem Darknet53. Následuje ověření detekcí objektů ze snímků a videí a ověření rychlosti detektoru na kvalitní a redukované verzi příslušného média.

Trénování modelu se zabývá primárně konfigurací konvolučních vrstev, konfiguračních souborů, přípravě dat, natrénováním modelu a odzkoušením jeho schopnosti a rychlosti rozpoznat naučené objekty v reálném čase.

V části výsledků a diskuse je vyhodnocení praktické části ve formě 3 zajímavých poznatků vycházejících ze splnění cílů.

Cíl práce a metodika

1.1 Cíl práce

Diplomová práce je zaměřená na prostudování technologie Darknet a YOLOv3.

Cílem diplomové práce je prakticky ověřit schopnost a rychlost rozpoznat naučené objekty v reálném čase neuronovou sítí YOLOv3.

Mezi další cíle této práce se řadí získání potřebných teoretických znalostí v oblasti neuronových sítí, strojového učení, hlubokých konvolučních sítí nebo přípravy učební sady, nezbytných pro realizaci vlastní práce.

1.2 Metodika

Rešeršní část bude obsahovat studium relevantních technologií, literatury a Darknetu, naprogramování či implementaci nástroje učení YOLOv3 (předučený model je k dispozici na stránkách Darknet). Následuje získávání a transformace učební sady nad zvolenými daty a jejich popis. Naučení modelu obnáší konfiguraci a přetrénování konvoluční sítě. K vytvoření fotografických snímků a testovacích videí bude použit telefon IPhone 6. Otestování sítě pomocí připravených materiálů. V závěru bude souhrn výsledků a uzavření.

Teoretická východiska

1.3 Matematický kontext Neuronových sítí

Mezi nezbytný teoretický základ k pochopení neuronových sítí patří pár základních konceptů lineární algebry.

1.3.1 Lineární algebra

Lineární algebra je široký matematický obor. Zabývá se studiem operací na množinách různých prvků a vlastnostmi struktur, které takto vznikají. Známými představiteli těchto struktur jsou grupa a těleso. Lineární algebra se zabývá vektory, maticemi, soustavami lineárních rovnic a vektorovými prostory. Pro účel této práce se budeme zabývat jen její částí, kterou je teorie matic. (HAŠEK, 2016)

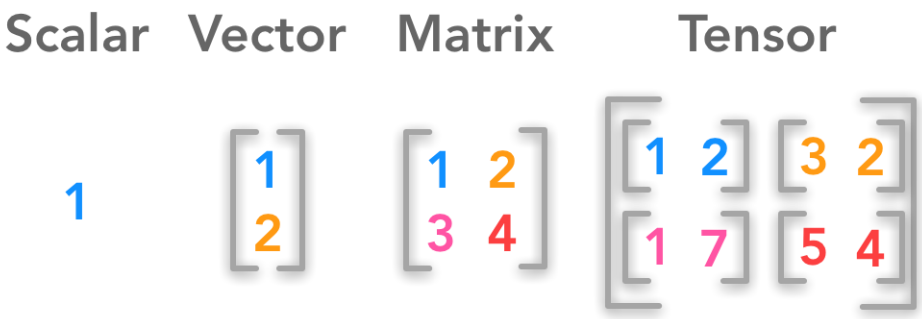
1.3.1.1 Matice – definice

Matice typu (m, n) je uspořádaná m-tice prvků z R^n. Jednotlivé složky této m-tice nazýváme řádky matice. Nechť $a_r = (a_{r,1}, a_{r,2}, ..., a_{r,n})$ je r-tý řádek matice typu (m, n). s-tá složka tohoto řádku $a_{r,s} \in R$ se nazývá (r, s)-tý prvek matice. (Olšák, 2007)

1.3.1.2 Tenzor, Vektor, Skalár

V matematice je tenzor v podstatě obecná forma vektoru či matice. Tenzor může být troj i více rozměrný. Pro dimenze či rozměry se u tenzoru používá slovo řád. Slovo tenzor pochází z latinského tenze neboli napětí. Poprvé byly totiž tenzory zavedeny v souvislosti s popisem napětí.

Tenzor nultého řádu má 3^0 složek, jedná se tedy o skalár.
Tenzor prvního řádu má $3^1 = 3$ složky, jedná se o trojrozměrný vektor.
Tenzor druhého řádu má $3^2 = 9$ složek.
Tenzor třetího řádu $3^3 = 27$ složek.
Tenzor čtvrtého řádu $3^4 = 81$ složek.
(Šlégr, 2012)



Obrázek 1 Tenzor (Steinke, 2017)

1.3.1.3 Tenzor vs Matice v Neuronových sítích

Jakýkoliv tenzor druhého řádu může být reprezentován jako matice, ale ne každá matice může být tenzorem druhého řádu.

Tenzor je matematická entita, která interaguje s okolní strukturou a jejími dalšími entitami. Pokud transformujeme jednu entitu, tenzor se také odpovídajícně pozmění. Tato „dynamická“ vlastnost je hlavní, kterou se liší tenzor druhého řádu od matice.

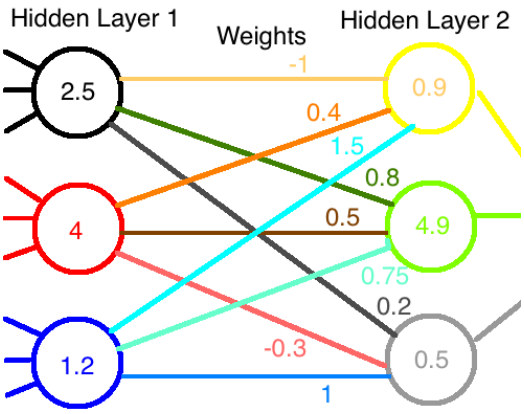
Názorná ukázka rozdílu

Pro ukázku máme neuronovou síť o 3 vstupech, 1. skrytá vrstvě se třemi neurony, 2. skrytá vrstva se třemi neurony a jedním výstupem. V této síti máme polo-náhodně zvolené hodnoty neuronů a jejich váhy. Pro první skrytou vrstvu máme výstupy neuronů (vektor): $L_1 = \begin{bmatrix} 2.5 \\ 4 \\ 1.2 \end{bmatrix}$ a váhy(matice): $W_{12} = \begin{bmatrix} -1 & 0.4 & 1.5 \\ 0.8 & 0.5 & 0.75 \\ 0.2 & -0.3 & 1 \end{bmatrix}$.

Vynásobením získáme hodnoty neuronů druhé skryté vrstvy:

$$W_{12}L_1 = L_2 \rightarrow \begin{bmatrix} -1 & 0.4 & 1.5 \\ 0.8 & 0.5 & 0.75 \\ 0.2 & -0.3 & 1 \end{bmatrix} \begin{bmatrix} 2.5 \\ 4 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.9 \\ 4.9 \\ 0.5 \end{bmatrix}$$

Diagram znázorňující aktuální stav:



Obrázek 2 tenzor diagram (Steinke, 2017)

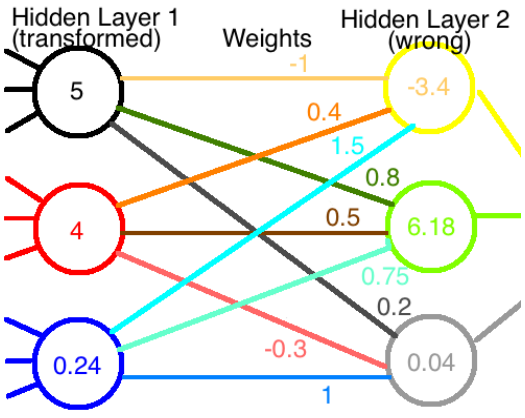
Předpokládejme situaci, kdy bychom chtěli ručně změnit jednotlivé aktivační funkce neuronů (např.: vynásobíme jednotlivě vstupní funkce čísly 2, 1 a 1/5).

V našem příkladě se touto operací změnil výstup neuronové sítě na nežádoucí.

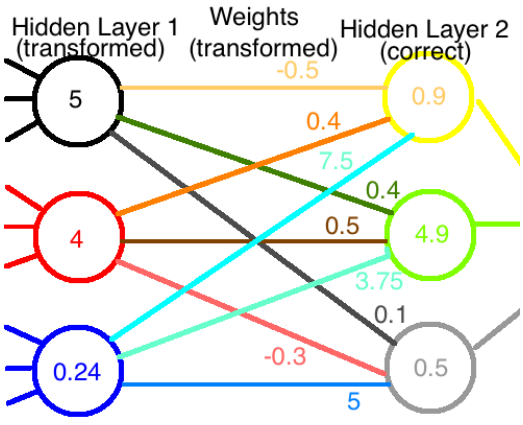
$$A\ L_1 = L_1' \rightarrow \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.2 \end{bmatrix} \begin{bmatrix} 2.5 \\ 4 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \\ 0.24 \end{bmatrix}$$

Abychom získali na výstupu neuronové sítě původní data, vynásobíme výstupní váhy změněných neuronů převrácenou hodnotou (A^{-1}), než byla hodnota A změny vstupu.

$$W_{12}A^{-1} = W_{12}' \rightarrow \begin{bmatrix} -1 & 0.4 & 1.5 \\ 0.8 & 0.5 & 0.75 \\ 0.2 & -0.3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix} = \begin{bmatrix} -0.5 & 0.4 & 7.5 \\ 0.4 & 0.5 & 3.75 \\ 0.1 & -0.3 & 5 \end{bmatrix}$$



Obrázek 3 tenzor diagram nefunkční (Steinke, 2017)



Obrázek 4 tenzor diagram opravený (Steinke, 2017)

1.3.1.3.1 Kovariance, Kontravariance

Kvůli transformaci jednoho z vektorů bylo pro zachování konzistence nutno kompenzovat transformací jeho vah obráceným způsobem. Tato propojená struktura povýší pouhou matici na tenzorový objekt. Pokud původní transformaci neuronů první vrstvy nazveme kovariantní (měnící se spolu s neuronem), pak změnu v jejich výstupních vahách (sloužící jako rovnováha) nazveme kontravariační transformací (měnící se proti neuronu). (Steinke, 2017)

1.4 Neuronová síť

Neuronová síť je matematický model používající principy lineární algebry, biologie i statistiky za účelem řešení daného problému. První formální neuron nazvaný Threshold Logic Unit (TLU) byl poprvé navrhnut pány McCulloh a Pitts v roce 1943.

Je důležité si uvědomit, že umělá neuronová síť je silně založena na biologických procesech v mozku. Neurony v mozku získávají data od ostatních propojených neuronů skrze vlákna zvaná axony. Určité neurony mají spojení silnější či četnější než jiné, a mozek jako celek je složen velkého spletitých sestav neuronů a axonů. Odhadované množství neuronů v lidském mozku je 86miliard. Umělá neuronová síť byla vytvořena jako snaha o simulaci fungování mozku. (Azevedo, F.A., Carvalho, L.R., Grinberg, L.T., Farfel, J.M., Ferretti, R.E., Leite, R.E., Filho, W.J., Lent, R. and Herculano-Houzel, S., 2009)

Neuronové sítě řeší problémy aproximací odpovědí, oproti „klasickému“ způsobu odpovědi deterministicky algoritmicky spočítat. Na neuronovou síť by se dalo nahlížet, jako na adaptivní systém, který postupně sám sebe upravuje za účelem přiblížení se k řešení. Do sítě vstupuje určité množství vstupů, které je postupně transformováno do přesně určeného množství výstupů s cílem se co nejpřesněji přiblížit zamýšlenému výsledku.

Neuronové sítě jsou zejména vhodné pro řešení problémů, v kterých nemáme plné porozumění, takže sestavit věrný matematický model je téměř nemožné, ale je k dispozici velké množství dat. K takovým problémům patří zejména rozpoznávání vzorů (pattern recognition), lineární klasifikace (linear classification), aproximace křivek/funkcí (data fitting, non-linear function approximation), modelování nelineárních systémů (system modeling, system prediction), a další. (Andrea Tettamanzi, Marco Tomassini, 2001) (Shafer, 2016) (Itzhak Fried, Ueli Rutishauser, Moran Cerf and Gabriel Kreiman, 2014)

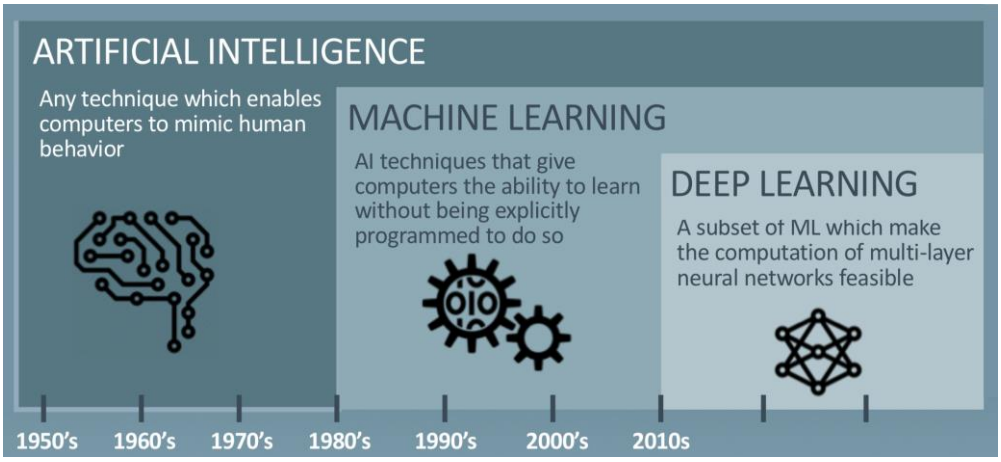
1.4.1 Vztah Umělé inteligence, Strojového učení a Hlubokého učení

1.4.1.1 Umělá inteligence (česky UI, anglicky AI)

John McCarthy, uznáván jako jeden ze zakladatelů oboru umělé inteligence, ji v roce 1955 definoval jako „věda a technika vytváření inteligentních strojů se schopnostmi dosáhnout cílů, jako lidé“.

Cílem bylo (a stále je), přimět počítače vykonávat činnosti označované jako výhradně lidské. Zpočátku to znamenalo například řešení logických problémů či hraní deskové hry „Dáma“. Rané úspěchy těchto metod zvýšily zájem o obor mezi veřejností i vědeckou obcí. Obzvlášť když začal mít počítač v daných problémech převahu nad člověkem. (Jeffcock, 2018)

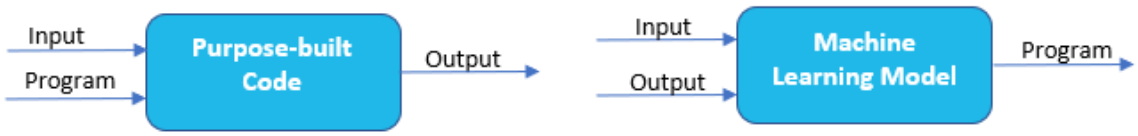
Termín Umělá inteligence nemá nic společného s metodou řešení problému. Umělou inteligencí můžeme nazvat i počítač hrající piškvorky[3x3] pomocí naivního algoritmu s maticí všech pohybů a předepsaných reakcí. K nazvání počítače Umělou inteligencí stačí, že jeho chování imituje/vypadá jako lidské.



Obrázek 5 Umělá inteligence a její podmnožiny (Jeffcock, 2018)

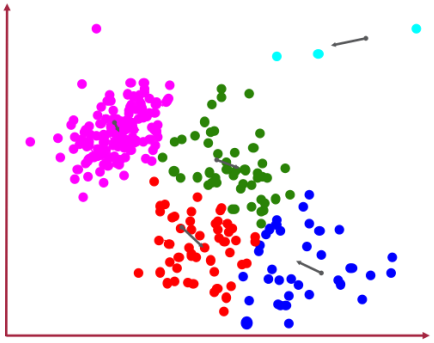
1.4.1.2 Strojové učení

Strojové učení je vědní obor, který dává počítači schopnost se učit bez nutnosti být explicitně naprogramován. To znamená, že se program od doby jeho vytvoření bude schopen naučit vykonávat určitou inteligentní aktivitu, která nebyla v rámci původní programové výbavy.



Obrázek 6 Ručně napsaný algoritmus vs. Strojové učení z příkladů a vzorů (Agrawal, 2018)

Strojové učení je tedy metoda, jakou lze dosáhnout umělé inteligence. Dalo by se říct, že strojové učení je snaha o napodobení principu, jakým se lidé učí věci. Nejznámějšími technikami strojového učení je regrese (prognózy a předpovědi ze závislosti dat), klasifikace (zařazování vzorků do kategorií), shluková analýza (třídění jednotek do shluků) a detekce odchylek.



Obrázek 7 Shluková analýza (Jeffcock, 2017)

1.4.2 Využití strojového učení

Mnoho druhů úkolů může být vyřešeno strojovým učením. Mezi nejběžnější využití pro strojové učení patří:

1.4.2.1 Třídění (Classification)

V tomto druhu úloh je počítač požádán stanovit, do které kategorie vstup patří. Příkladem klasifikační úlohy je rozpoznávání objektů, kde vstupem je obrázek či fotografie (většinou popsána jako množina pixelů) a výstupem je číselný kód, sloužící jako identifikátor objektu v obrazu.

1.4.2.2 Třídění s neúplnými vstupy (Classification with missing inputs)

Klasifikace se stane mnohem více náročným úkolem, pokud počítač nemá zaručeno, že ve svém vstupním vektoru vždy dostane všechny naměřené hodnoty. K vyřešení klasifikační úlohy stačí učícímu algoritmu definovat jednu funkci, jež tranformuje vstupní vektor na konkrétní kategorický výstup. V případě, kdy by mohly některé ze vstupů chybět, místo jedné klasifikační funkce se musí algoritmus naučit řadu funkcí, kde každá odpovídá klasifikaci s konkrétní kombinací daných a chybějících vstupů.

Tento druh situace nastává často v lékařské diagnostice, protože spousta zdravotních testů jsou nákladné či invazivní.

1.4.2.3 Regresní úloha

V regresní úloze se po počítači chce, aby předpověděl číselnou hodnotu na základě daného vstupu. Úloha je podobná klasifikační, liší se však formátem výstupu.

Tento druh předpovědí se používá pro výpočet pojistného, předpovídání ceny cenných papírů a zejména pro algoritmické obchodování.

1.4.2.4 Přepis (transkripce)

V této úloze je systém strojového učení požádán, aby sledováním relativně nestrukturovaně zobrazených dat nějakého druhu a přepsal informace do zřetelné (většinou) textové podoby.

Typický příklad je přepis textu z fotografie, v optickém rozpoznávání znaků OCR (Optical Character Recoginition) Google StreetView používá metody hlubokého učení pro zpracování adres tímto způsobem. Dalším příkladem je rozpoznávání řeči, kde počítačovému programu poskytneme průběh hlasové vlny a on vydá sekvenci znaků nebo (popisné) identifikační kódy slov.

1.4.2.5 Strojový překlad (machine translation)

V úloze na strojový překlad, vstup je již ve formátu sekvence symbolů v určitém jazyce, počítačový program pak musí převést tuto sekvenci symbolů v jednom jazyce v odpovídající sekvenci symbolů v jiném jazyce. Metoda je běžně používaná pro přirozené jazyky.

1.4.2.6 Strukturovaný výstup

Tato široká kategorie zahrnuje i již zmíněný Přepis (1.4.2.4) a Strojový překlad (1.4.2.5) ale i mnoho dalších. Jedná se o zastřešující kategorii spíše pro předpovídání struktury objektů než konkrétních oddělených hodnot.

Jedním příkladem je parsování (rozbor) a mapování vět přirozeného jazyka do stromů, které popisují jejich gramatickou strukturu označením uzlů stromů jako slovesa, podstatná jména, příslovce atd. Další využití kromě zpracování přirozeného jazyka je i v rozpoznávání řeči, bioinformatice a počítačovém vidění (získávání informací ze zachyceného obrazu).

1.4.2.6.1 Detekce anomálií

Odhalování odchylek spočívá v důkladném prozkoumávání událostí či objektů a označení některých z nich, jako mimořádné či neobvyklé, vybočující z řady.

Příklad úlohy na odhalování anomálií je detekce podvodů s kreditními kartami. Vymodelováním zákazníkových nákupních zvyků může společnost odhalit zneužití jeho karty. Pokud je karta nebo její údaje odcizeny, jsou zlodějovi nákupy často z jiné části pravděpodobnostní distribuce než původního zákazníka.

1.4.2.6.2 Syntéza a vzorkování

V tomto druhu úkolu je algoritmus strojového učení požádán, aby vygeneroval nové příklady, které budou podobné tréninkovým datům.

V některých případech chceme, aby proces syntézy nebo vzorkování vygeneroval konkrétní druh výstupu, podle námi zadaného vstupu.

Například, v úloze syntézy programu do vstupu zadáme psanou formou text, který požadujeme transformovat na zvukovou vlnu mluvené řeči. V komplexnějších modelech je možno dodat jako vstup zvukovou stopu obsahující styl řeči a intonaci reálného člověka, kterého má program co nejvěrněji napodobit. (Louppe, 2019)

1.4.2.6.3 Nahrazení chybějících hodnot (Imputation of missing values)

Algoritmus je učen na příkladech z dat, z kterých jsou záměrně vyjmuty náhodné záznamy. Předpovědi se pak v testovací fázi ověřují oproti kompletním sadám. Prospěšnost tkví v problémech, které nekompletní sady dat představují pro další analýzu, jakými jsou: výrazná zaujatost (pozitivní či negativní diskriminace) jiných hodnot, značné znesnadnění práce s daty a snížení účinnosti modelu.

1.4.2.7 Odstraňování šumu (Denoising)

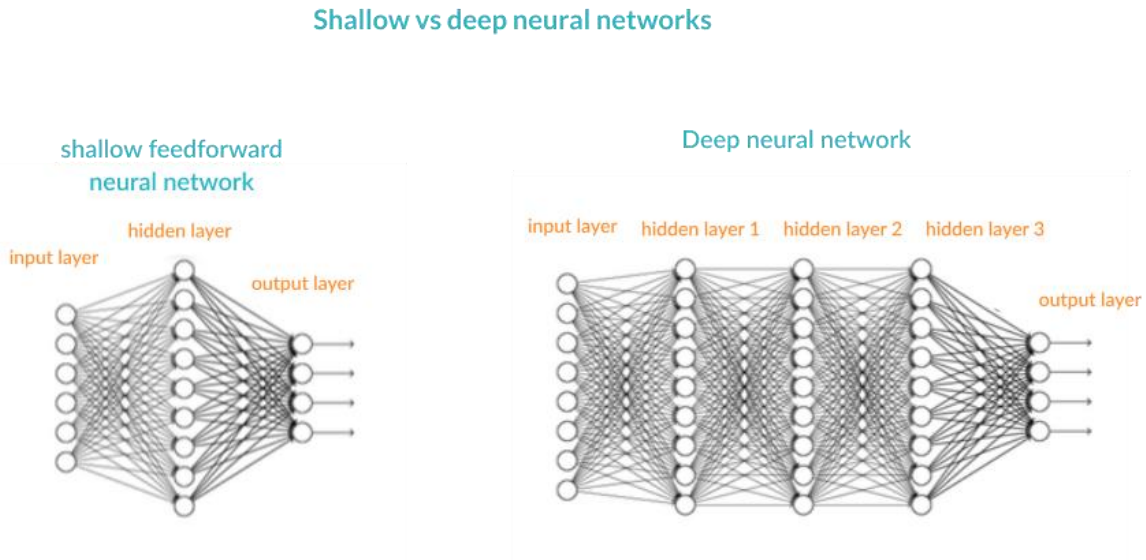
V úkolech na odstraňování šumu má algoritmus strojového učení jako vstup poničený příklad, poškozený nespecifikovaným procesem, zatímco původní nepoškozená verze slouží jako testovací data. Algoritmus musí předpovědět vyčištěný výsledek z vložených poškozených dat. (Andrea Tettamanzi, Marco Tomassini, 2001) (Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016)

1.4.2.8 Mozkem inspirované Neuronové sítě

Lidský mozek je jedním z nejlepších nám známých „strojů“ pro řešení problémů. Mozkem inspirované neuronové sítě jsou tedy opravdu odvozené z principů fungování lidského mozku. Hlavním výpočetním elementem mozku je neuron, komplexně propojená síť z neuronů tvoří základ všech rozhodnutí, založené na různorodých nasbíraných datech. Přesně o toto se umělá neuronová síť také snaží.

1.4.2.9 Hluboké učení

Implementací technik strojového učení došlo v rámci oboru neuronových sítí k rozšíření strojového učení na techniku Hluboké učení (Deep Learning). Jinak řečeno, hluboké učení je technika pro implementaci strojového učení, v které neuronové sítě obsahují více než 3 vrstvy (neboli více než jednu Skrytou vrstvu). Tyto neuronové sítě s hlubokým učení se nazývají Hluboké neuronové sítě (Deep neural network, DNN).



Obrázek 8 rozdíl mezi mělkou neuronovou sítí (ANN) a hlubokou (DNN) (MissingLink dev team, 2018)

Hluboké učení je v podstatě snaha o napodobení fungování lidského mozku na velmi základní úrovni. To nám sice dovoluje řešit komplexnější problémy, ale ani robustní hluboká neuronová síť s 1000 neurony a poměrně naivním (neindividuálním) propojení mezi těmito neurony nemůže dosahovat schopností lidského mozku, obsahujícího okolo 86 miliard velmi komplexně propojených neuronů a dalších faktorů.

Díky hlubokému učení existuje mnoho úloh, které stroj vyřeší efektivněji a lépe, než dokáže člověk. Již v roce 2015 překonaly modely pro klasifikaci obrázků pěti procentní chybovost lidí a od té doby se modely ještě výrazně zpřesnily.
(Agrawal, 2018) (Jeffcock, 2018) (Jeffcock, 2017) (Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016)

1.4.2.10 AI, ML, DL ve zkratce

- Umělá inteligence je přimět počítač nějakým způsobem napodobit lidské chování.
- Strojové učení je podmnožina umělé inteligence a skládá se z technik, které dovolují počítačům porozumět podstatě čtených dat a toto porozumění prakticky aplikovat. Dalo by se říct, že strojové učení je snaha o napodobení principu, jakým se lidé učí.
- Hluboké učení je podmnožinou strojového učení, které dovoluje počítačům vyřešit mnohem komplexnější problémy. Zde by se dalo říct, že hluboké učení je snaha o napodobení fungování lidského mozku na velmi základní úrovni.

1.4.3 Koncepty umělé neuronové sítě

1.4.3.1 Vstupy

Zdrojová data, kterými je krmena neuronová síť za účelem tvoření rozhodování či predikcí z takových dat. Vstupem do neuronové sítě je typicky množina reálných hodnot, každá hodnota vstupuje do jednoho z neuronů ve vstupní vrstvě.

1.4.3.2 Trénovací sada

Množina vstupů, pro které známe správné výstupy, která je použita k trénování neuronové sítě.

1.4.3.3 Výstupy

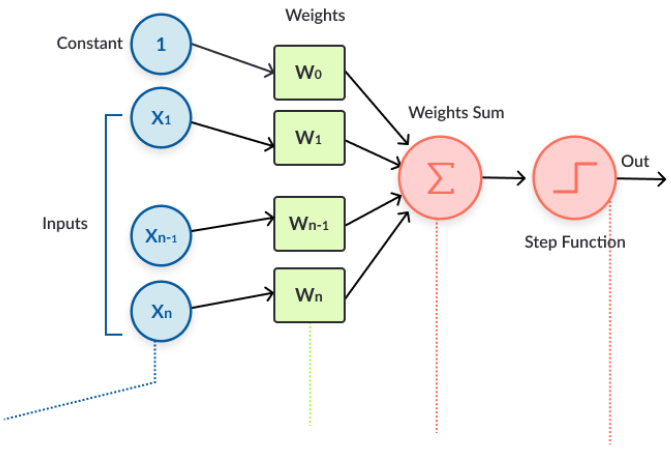
Neuronové sítě generují své predikce ve formě reálných nebo boolean hodnot. Každá výstupní hodnota je vypočítána jedním neuronem ve výstupní vrstvě sítě.

1.4.3.4 Skrytá vrstva

Stejně jako vstupy a výstupy, jde o vrstvu neuronů transformující průchozí informace. Nachází se mezi vstupy a výstupy, a proto je pro svět mimo model „skrytá“. Skládá se z neuronů, které transformují procházející data. Síť může obsahovat libovolné množství skrytých vrstev.

1.4.3.5 Neuron/perceptron

Model biologického neuronu (viz Obrázek 10), základní jednotka sítě, která má vstupy a generuje predikci.
Každý z neuronů přijme část vstupu a nechá je projít **aktivační funkcí**. Běžnými aktivačními funkcemi jsou sigmoid, TanH a funkce ReLu (viz 1.4.4). Význam aktivační funkce spočívá v generování výstupů neuronů ve správném rozsahu, tato normalizaci výstupů neuronů a nelineární podoba těchto funkcí je klíčová pro trénování sítí.



Obrázek 9 Detail Perceptronu / nejjednodušího neuronu (MissingLink dev team, 2018)

1.4.3.6 Hyperparametry

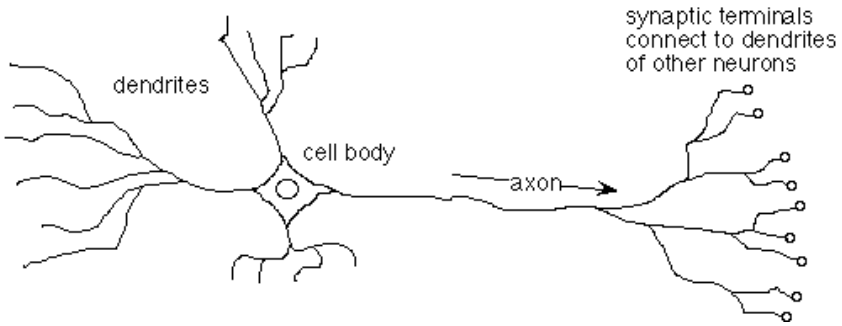
Hyperparametr je nastavení, které ovlivňuje strukturu nebo fungování neuronové sítě. V hlubokém učení (deep learning) jsou hyperparametry základním způsobem, jak sestavit síť poskytující přesné predikce. Mezi hyperparametry patří množství skrytých vrstev, aktivační

funkce a množství, kolikrát by měl být tréninkový proces opakován neboli množství epoch. (MissingLink dev team, 2018)

1.4.3.7 Váha (Weight)

Váha v umělých neuronových sítích je číselný parametr, spojený s propojením každého jednoho neuronu k jinému dalšímu neuronu. Toto koresponduje se synapsemi v biologickém neuronu a udává, jak velkou pozornost by měl věnovat k obdrženým signálům od takto propojeného neuronu.

Váha může být kladná i záporná, v případě kladné se spojení nazývá **excitatorní**, u záporné váhy nazýváme napojení **inhibitorní**. V přirovnání k biologickému neuronu je váha u formálního neuronu spojením dendritu a axonu. (Wilson, 2012)



Obrázek 10 Biologický neuron (Wilson, 2012)

1.4.3.8 Zpětná propagace (Backpropagation)

K tomu, aby síť získala optimální váhy neuronů, provádí se zpětný průchod vrstvami, směrem od predikce směrem k neuronům, které ji vygenerovaly. Zpětná propagace pracuje se vstupními složkami aktivačních funkcí v každém navazujícím neuronu za účelem snížit pomocí vah chybovou funkci na minimum. Zde použitý matematický proces se nazývá *Gradient descent* a hlavní výhodou jeho varianty *Stochastic gradient descent* je, že při vyhledávání globálního minima funkce má malý risk zaseknout se v minimech lokálních. (Orr, 1999)

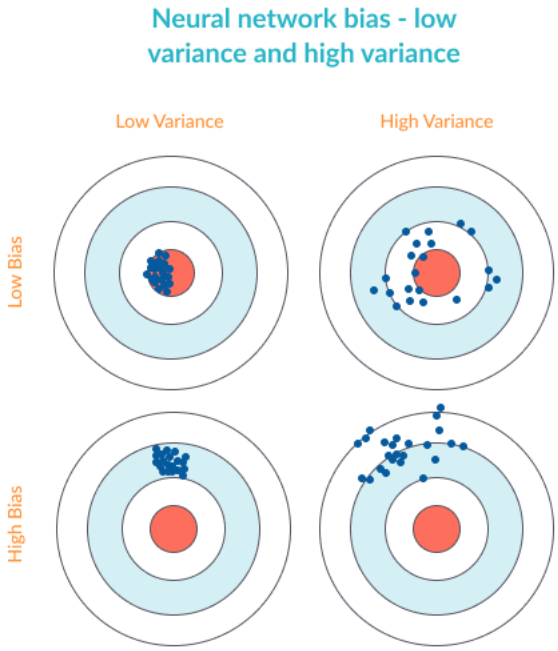
1.4.3.9 Bias a Odchylka

Bias (zaujatost, diskriminace) udává, jak přesně je model schopen předpovídat známé výstupy tréninkového setu, nízký bias znamená, že chybovost modelu bude malá.

Bias však také může být doplňkový parametr, který by se dal přirovnat ke konstantě v lineární funkci: output = sum (weights * inputs) + bias

Obrázek 11 Bias

Odchylka měří, jak (odmocnina z odchylky se je směrodatná chyba), jak je model schopen poradit si novými daty mimo tréninkovou sadu.

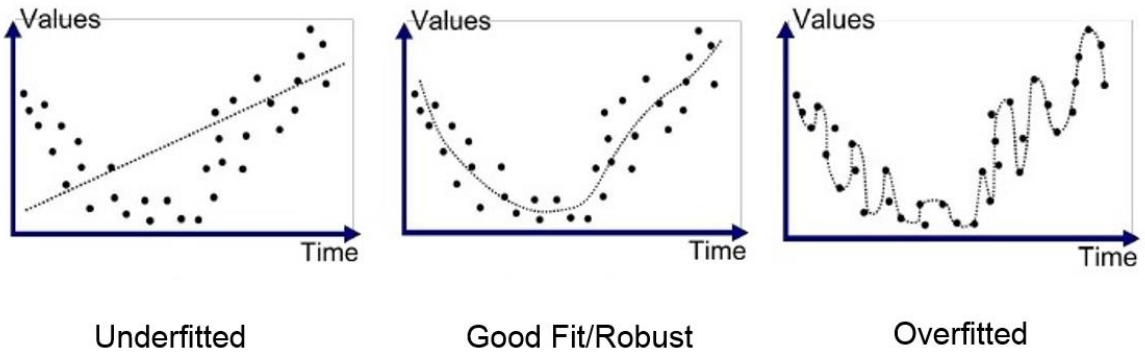


Obrázek 12 Znázornění biasu a odchylky (MissingLink dev team, 2019)

1.4.3.10 Overfitting, Underfitting

> *"Simpler solutions are more likely to be correct than complex ones."*
>
> *– Ockhamova břitva, William Ockham, 14.století*

- underfitting (vysoký bias, nízká odchylka)
  - neúspěch v obsáhnutí vztahů v tréninkových datech
- overfitting (neboli přetrénování) (nízký bias, vysoká odchylka)
  - přílišná důvěra v tréninková data
- vysoká odchylka: model na základě tréninkových dat reaguje přehnaně
- vysoký bias: doměnky o modelu vedou k ignorování tréninkových dat

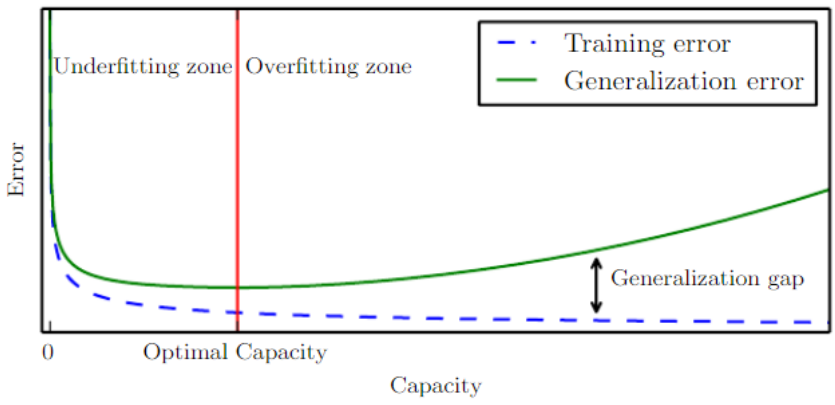


Obrázek 13 Graf: underfitting (vysoký bias, nízká odchylka) vs overfitting (nízký bias, vysoká odchylka) (Koehrsen, 2018)

Metody, jak se vyhnout overfittingu (přetrénování):

- netrénovat příliš dlouho (velmi subjektivní)
- přetrénování neuronové sítě znovu, ale s jinými výchozími vahami
    - takové trénování sítí více sítí stejné architektury může běžet i paralelně již od prvního trénování
- včasné zastavení učení modelu
    - optimální bod pro zastavení učení modelu je takové místo, v kterém se dalším trénováním na tréninkových datech sice snižuje chyba a zároveň se chyba na všeobecných, sítí zatím neznámých datech, se začíná zvyšovat (viz Obrázek 14)
    - za tato data můžeme považovat testovací sadu za předpokladu, že je kvalitní a dostatečně obecná

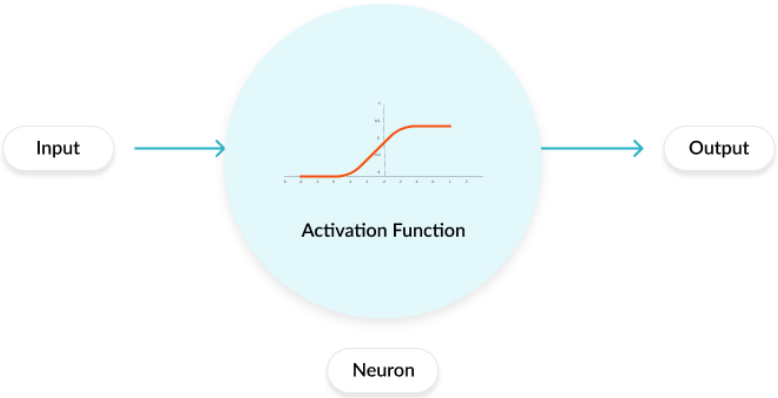
(MissingLink dev team, 2019) (Koehrsen, 2018) (Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016)



Obrázek 14 Bod optima učení

1.4.4 Aktivační funkce

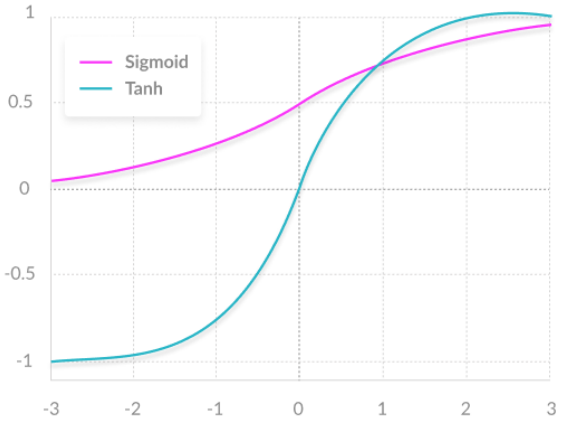
Aktivační funkce (Activation function nebo také Transfer function) je matematická rovnice, která určí výstup každého uzlu (perceptronu nebo neuronu) v neuronové síti. Přijímá vstupy od každého neuronu a transformuje je do jeho výstupu, který je většinou v intervalu 0 a 1 nebo -1 a 1.



Obrázek 15 aktivační funkce (MissingLink dev team, 2018)

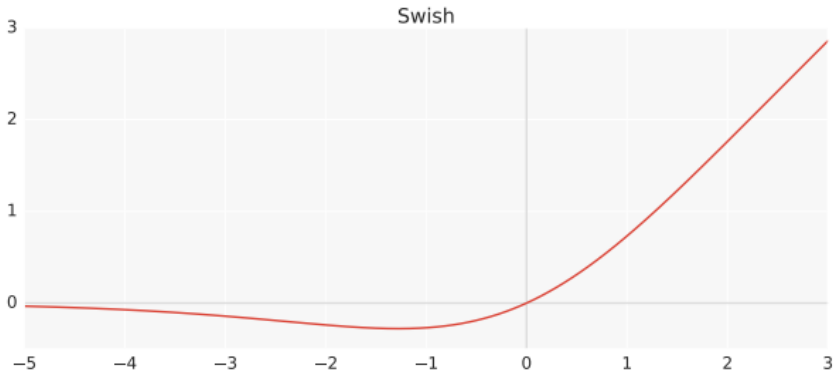
Nejpoužívanější aktivační funkce jsou:

- jednotkový skok („heavisideova funkce“)
    - nespojitá; 0 pro zápornou hodnotu argumentu; rovná 1 pro kladnou
- sigmoida („jemný krok“)
    - spojitá, hladký plynulý gradient od 0 do +1
    - velmi výpočetně pomalá
- tanh (hyperbolický tangens)
    - spojitá, hladký plynulý přechod od -1 do +1
    - vhodná pro modely se silně kladnými a zápornými vstupy



Obrázek 16 funkce sigmoid a tanh (MissingLink dev team, 2018)

- ReLU (rectified linear unit)
  - spojitá, 0 pro záporné hod., lineární 0-1 pro kladné
  - výpočetně velmi rychlá a osvědčená, ale není vhodná pro záporné vstupy
- Leaky ReLU
  - ReLU funkce, která má pro záporné hodnoty malý lineární spád (0,1x)
- Parametric ReLU (PReLU)
  - Leaky ReLU funkce, s lineárním sklokem 1x v pro kladné argumenty a parametrický koeficient αx pro záporné argumenty, tento parametr je předmětem učení neuronové sítě
- Swish
  - nová (2017) Aktivační funkce od Google Brain týmu
  - $f(x)=x\cdot sigmoid(\beta x)$
  - lze jí snadno vyměnit s ReLU, podle týmu je funkce jednoduchá a podobná funkci ReLU a zároveň vykazuje lepší výsledky v hlubokých modelech (Prajit Ramachandran, Barret Zoph, Quoc V. Le, 2017)



Obrázek 17 Aktivační funkce Swish (Prajit Ramachandran, Barret Zoph, Quoc V. Le, 2017)

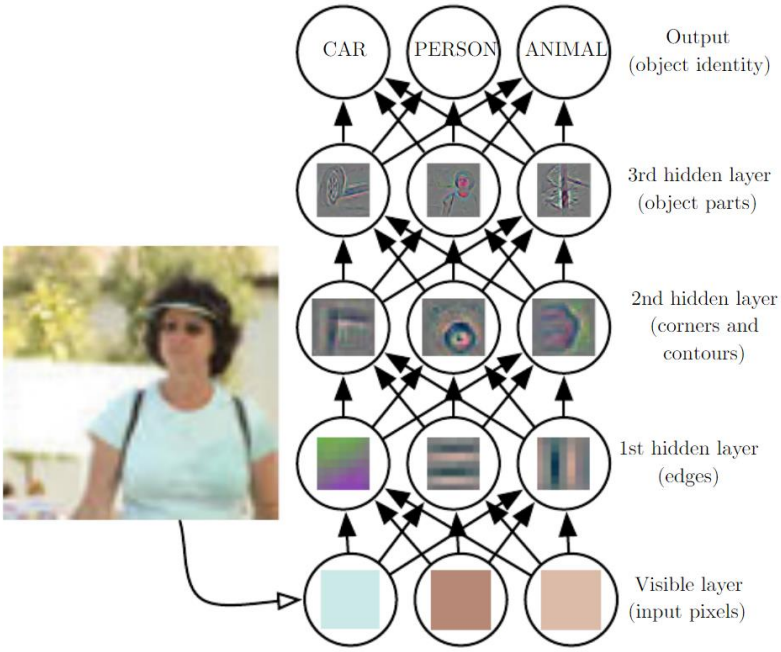
- Softmax
  - funkce <0;1> používaná v případě, že nějaký z výstupů není normalizovaný

1.4.5 Účel skrytých vrstev v neuronových sítích

Tato kapitola plynule navazuje na kapitolu Hluboké učení (viz 1.4.2.9).

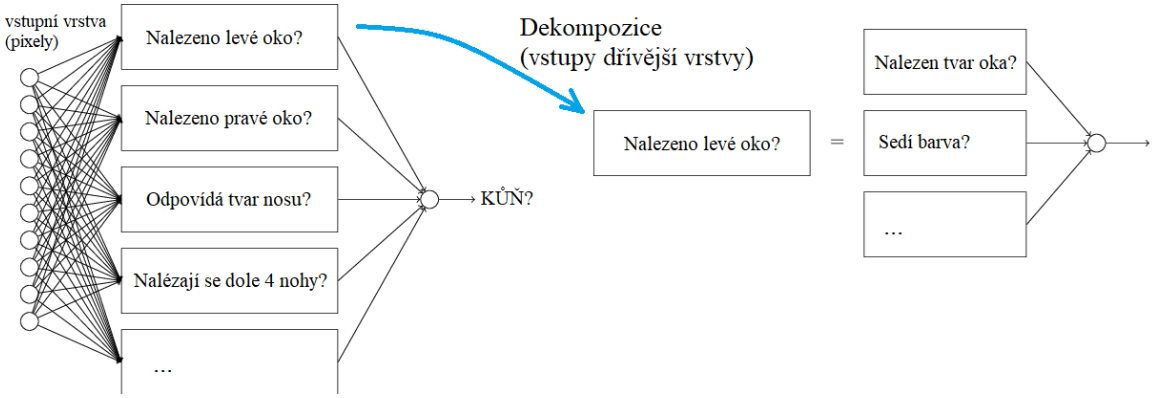
Hluboká neuronová síť je za pomoci skrytých vrstev schopna dekompozice problému na menší logické části, jako hrany, rohy, obrysy a tvary, jejichž skládáním se realita co nejvěrněji popisuje (viz Obrázek 18).

První skrytá vrstva, s ohledem na pixely vstupních dat, dokáže snadno identifikovat hrany tak, že porovná jas sousedících pixelů. Do druhé skryté vrstvy vstupuje popis obrázku z první vrstvy a detekuje z něj rohy a obrysy. Třetí vrstva z nich pak dokáže složit určité tvary objektů tak, že najde soubor konkrétních znaků typických pro tyto objekty. Tento popis obrazu pomocí částí objektů, které obsahuje, se použije pro rozpoznání celých objektů vyskytujících se na vstupu.



Obrázek 18 Rozpoznání obrazu identifikací jeho částí a objektů

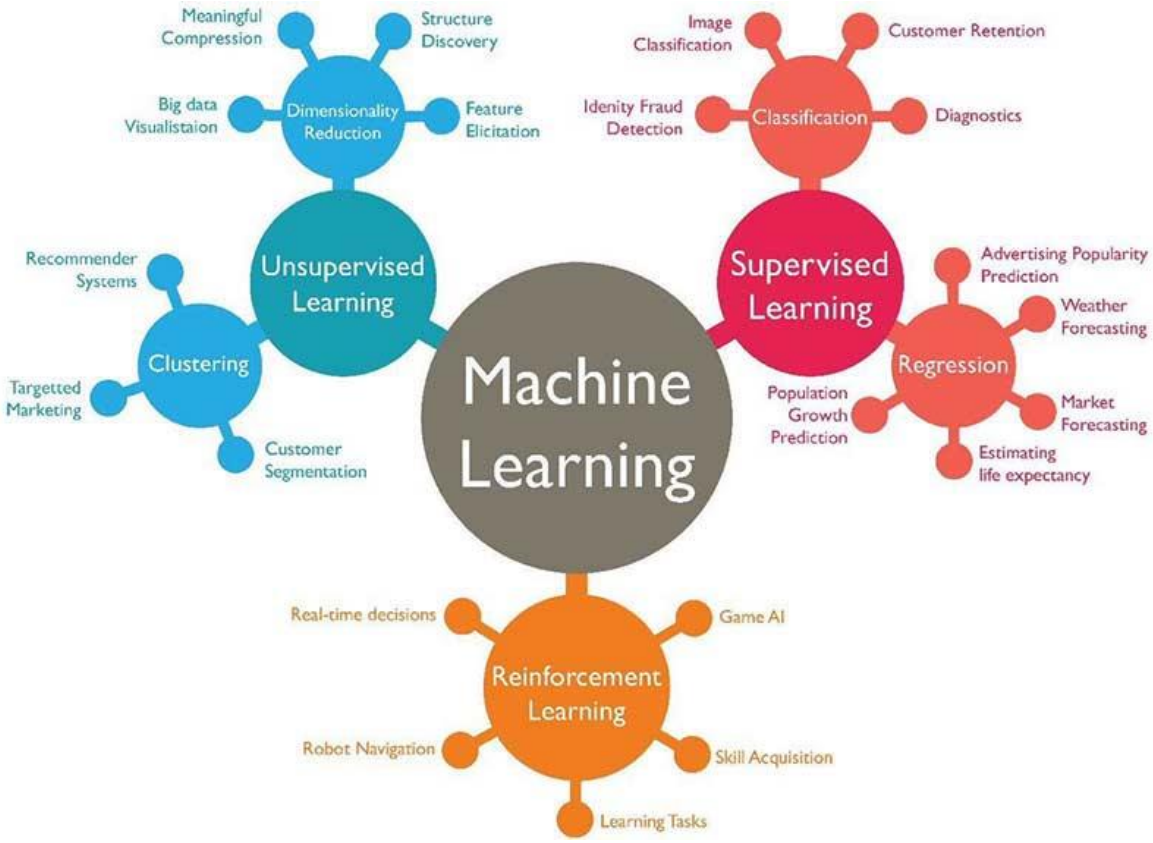
Například, pokud člověk dostane obrázky koní, rozpozná, že se jedná o koně, ať už jsou nakreslení, oblečení v kostýmu hrocha nebo sedí na křesle. Je to proto, že člověk rozpozná rozličné prvky, které koně definují: jako je tvar jeho hlavy a nosu, množství a rozmístění nohou atd. Hluboké učení je toto schopno simulovat tak, že v určitých neuronech a vrstvách dokáže rozpoznat objekty na obrázku a podle nich složit odhad, co se v něm nachází. (Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016)



Obrázek 19 Význam neuronů (Vlastní práce)

1.5 Modely neuronových sítí

Modely v této kapitole nejsou kompletním výčtem všech existujících modelů, jedná se o takové modely, které autor považuje za relevantní a zajímavé. Obrázek 20 nádherně zobrazuje nejznámější strukturu, podle které se modely typicky řadí.



Obrázek 20 Modely dle stylu učení (Jha, 2017)

1.5.1 Učení (training)

Většinu algoritmů strojového učení zle rozdělit do kategorií učení s učitelem (supervised learning) a učení bez učitele (unsupervised learning) podle toho, jaké mají k dispozici zkušenosti pro svůj učební proces.

1.5.1.1 Učení bez učitele (unsupervised)

Algoritmy učení bez učitele dostanou kolekci dat obsahující spoustu rysů, z nichž se naučí užitečné vlastnosti o struktuře těchto vstupních dat.

Příklady algoritmů využívající tréninkovou metodu učení bez učitele jsou:

- shluková analýza (viz Obrázek 7),

- klasifikace (viz 1.4.2.1, 1.4.2.2)
- a regrese (viz 1.4.2.3).

Obecně, učení bez učitele tedy zahrnuje snahu naučit se distribuci pravděpodobnosti vstupních dat nebo nějakých zajímavých vlastností v té distribuci.

Jedná-li se o hluboké učení, snažíme se model naučit celou distribuci pravděpodobnosti, která vytvořila data, ať už přímo (přirozená data) nebo nepřímo (námi transformovaná data pro účely syntézy či odstraňování šumu).

1.5.1.2 Učení s učitelem (supervised)

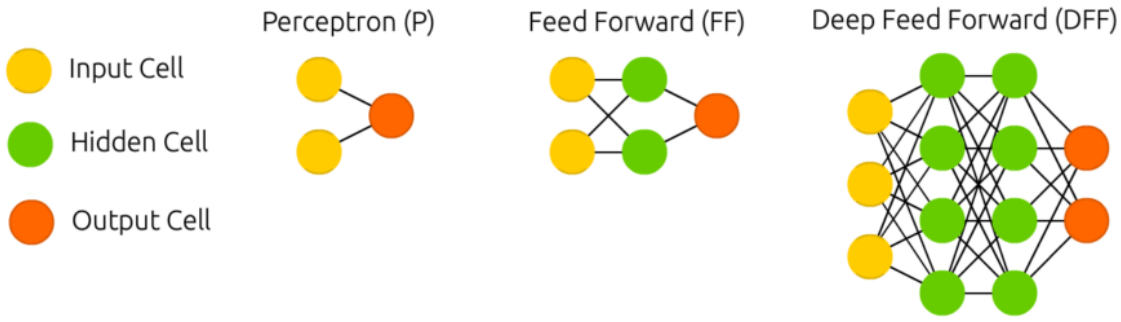
Algoritmus učení s učitelem má zajištěn datový soubor obsahující rysy, ale každý z příkladů má přidružený buďto popis, nebo požadovanou cílovou podobu. Například neuronová síť, které bychom vkládaly datovou kolekci obsahující různé rostliny stejného rodu s označením, o který druh se jedná, dokáže studiem těchto dat rozčlenit budoucí rostliny naučeného druhu na základě tvarů a rozměrů.

Učení s učitelem zahrnuje sledování vstupních dat, z jejichž základu se učí předpovídat data výstupní. (Ian Goodfellow and Yoshua Bengio and Aaron Courville, 2016)

1.5.2 Dopředná architektura neuronových sítí

Těmito velmi přímočarými sítěmi prostupuje informace vždy od vstupu k výstupu. Dopředné neuronové sítě mají plné propojení přiléhajících vrstev. To znamená, že každý neuron je propojen s každým neuronem v následující vrstvě.

Trénink dopředných sítí většinou probíhá pomocí zpětné propagace (backpropagation) (viz 1.4.3.8) tak, že síti poskytneme páry vstupních dat a podobu požadovaného výstupu. Tomuto učebnímu principu se říká výuka s učitelem (supervised learning, viz 1.5.1.2).



Obrázek 21 NN s dopřednou architekturou (Van Veen, F. & Leijnen, S., 2019), (vlastní práce)

1.5.2.1 Perceptron (P)

Nejelementárnější a nejjednodušší jednotka („základní stavební kámen“) neuronových sítí (popis viz 1.4.3.5).

Nejjednodušší v praxi použitelná síť má dva vstupní neurony a jeden výstupní, použitelná pro modelování logických hradel.

1.5.2.2 Vícevrstvý perceptron (MLP / FF nebo FFNN)

Nazývaný také dopředná neuronová síť (feed forward neural network). Vlastnost vícevrstvého perceptronu je šíření signálu právě pouze jedním směrem, proto jsou názvy MLP a FFNN zaměnitelné.

1.5.2.3 Hluboké (dopředné) neuronové sítě

Hluboké učení je podrobně popsané v dřívější kapitole (viz 1.4.2.9).

Známý problém Hlubokých dopředných sítí je, že s přílišnou hloubkou (velkým množstvím skrytých vrstev) se informace ztrácí.

Většina algoritmů hlubokého učení používají v rámci zpětné propagace optimalizační algoritmus zvaný *stochastic gradient descent* (detailní popis viz 1.4.3.8).

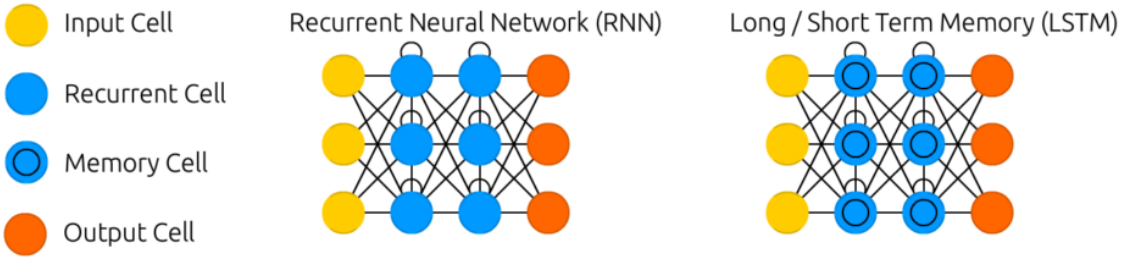
1.5.3 Rekurentní architektura neuronových sítí

Rekurentní neuronové sítě berou v potaz koncept času, protože kromě propojení uzlu se všemi uzly přiléhající vrstvy jako v FFNN, ale dostanou údaje svého vlastního vstupu z předchozího vstupu. (Tento feedback nemusí být neuron sám do sebe, ale i třeba neuron následující vrstvy, předávající svou hodnotu minulého vstupu.) Toto znamená, že je tato neuronová síť citlivá na pořadí tréninkových dat a jejich proházení by mohlo přinést jiné výsledky.

1.5.3.1 Rekurentní neuronové sítě (RNN)

Obdobně jako příliš hlubokých DFFNN, v RNN s příliš velkým množstvím vstupních dat (a tedy moc velkým časovým rozsahem) rapidně ztrácí informaci, protože váha zpětné vazby neuronu získá převahu nad ostatními vstupními vahami do neuronu.

Rekurentní sítě se dají využít i případě, že data nejsou časového charakteru, ale záleží na jejich pořadí. Typickým příkladem, v čem RNN vynikají je pokračování řady či dokončování informací, jako je to v textovém našeptávači zpráv (autocompletition / T9).



Obrázek 22 NN s rekurentní architekturou (Van Veen, F. & Leijnen, S., 2019), (vlastní práce)

1.5.3.2 Neuronové sítě Dlouhodobé a krátkodobé paměti (LSTM)

Dlouhodobé/krátkodobé sítě jsou inspirovány elektrickými obvody, kde přidáním hradla a paměti pro předchozí hodnoty bojují s problémem převahy váhy zpětné vazby (který vzniká při velkém množství dat) tak, že si neuron stanovuje míru, jakou část předchozí informace si zapamatuje. Tento princip má využití pro velmi dlouhé sekvence.

LSTM se dokáže naučit komplexní sekvence, takže dokáže replikovat styly jako psaní jako Shakespear nebo skládání hudby.

1.5.4 Modely učení bez učitele (unsupervised)

Zatímco předchozí zmíněné typy neuronových sítí bývají trénovány metodou učení s učitelem (supervised learning), tyto typy sítí trénované učením bez učitele nepoužívají zároveň trénovací a testovací sadu dat, ale vstupuje do nich pouze soubor dat bez jakýchkoliv anotací či dalších informací (detailně popsáno viz 1.5.1.1).

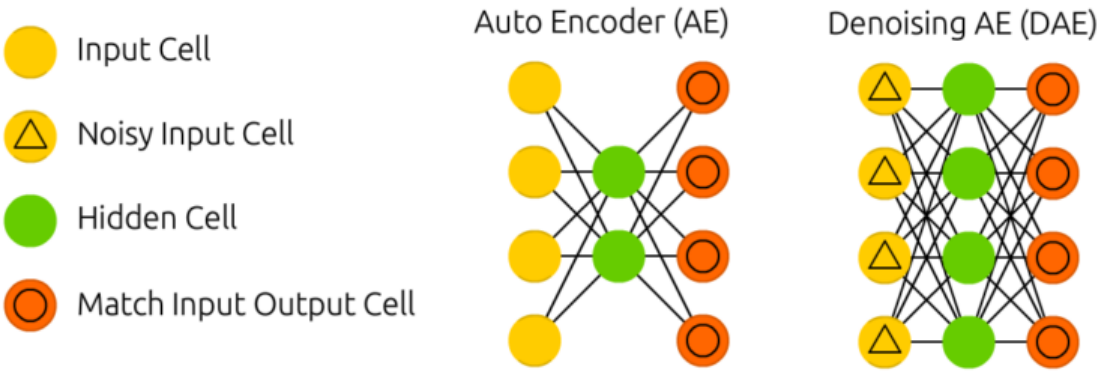
1.5.4.1 Autoenkóder (AE)

Autoenkodér je architekturou trochu podobný dopředné neuronové síti (FFNN), ale má jiné využití. Hlavní pointa architektury autoenkoderů je automaticky zakódovat (myšleno komprimovat, zestručnit) informace. Cílem je naučit se znaky a rysy dat pro určitou sadu vstupů.

Neuronová síť by tedy měla být schopna zestručnit vstupní data a následnou symetrickou expanzí získat na výstupu původní data v co nejvěrnější podobě.

Tvar celé sítě se podobá přesýpacím hodinám, kdy skryté vrstvy jsou vždy menší než vstupy a výstupy. Autoenkodery jsou také vždy symetrické okolo prostřední vrstvy. Nejmenší vrstva (vrstvy) je vždy uprostřed, to znamená, že je v tomto místě informace nejvíce zkomprimovaná (nejužší místo sítě).

Vše před středem neuronové sítě je kompresní část sítě, všechny vrstvy za středem se nazývají rekonstrukční (dekódovací) část sítě.



Obrázek 23 NN Autoenoderu, metoda bez učitele (Van Veen, F. & Leijnen, S., 2019), (vlastní práce)

Trénink proběhne technikou zpětné propagace (backpropagation, viz 1.4.3.8) tak, že chyba je rozdíl mezi vloženými vstupními daty a výstupními daty sítě (jelikož se snažíme, aby zpětná expanze odpovídala vstupu). Symetrie AE neuronové sítě se dá použít i na vahách a to tak, že dekódovací váhy mohou být stejné, jako kompresní.

(Adam Coates and Andrew Y. Ng, 2012) (Van Veen, F. & Leijnen, S., 2019)

1.5.4.2 Denoising autoencoder (DAE)

Variací AE je neuronová síť Autoencodéru pro odstraňování šumu. Tato síť je zcela identická s klasickým autoencodérem (viz Obrázek 23), ale neuronovou síť DAE učíme na datech, do kterých pro vstup přidáme uměle vygenerovaný šum (například zrnitost obrazu). Od výstupu však očekáváme co nejbližší podobu původnímu, nepoškozenému vstupu. Proto výpočet chyby pro zpětnou propagaci probíhá z nepoškozených dat bez šumu.

Tento způsob učení podnítí neuronovou síť neučit se detaily ale širší vlastnosti, protože naučení se detailů nepomůže s rekonstrukcí, při stále měnícím se šumu.

1.5.5 Reinforcement Learning (RL) (semi-supervised)

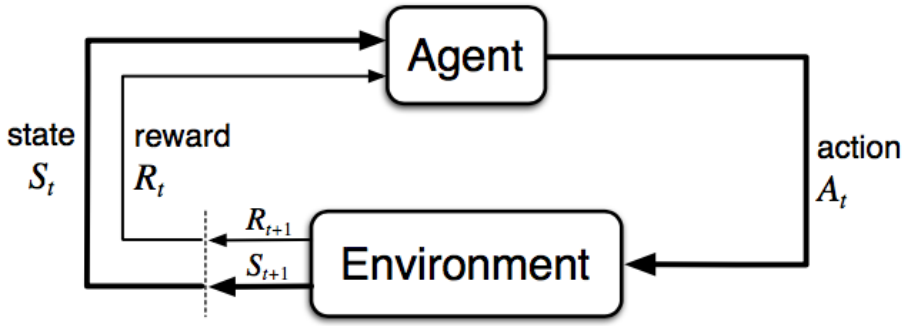
Protože strojového učení nemusí vždy dostat neměnný soubor dat, reinforcement learning algoritmy interagují se svým okolím, čímž získají pro svůj naučený systém zpětnou vazbu ze svých zkušeností.

Reinforcement learning je algoritmus zaměřený na cíl, který se učí metodou pokus-omyl, a používá se k vytváření procesu rozhodování v sekvenčních problémech.

Nespadá pod metody učení s učitelem, ani učení bez učitele, a je spíše známý jako „částečně vedený“ model (semi-supervised).

Reinforcement learning znamená, že je síť motivovaná dosahováním výsledků. Algoritmus je odměňován a penalizován podle toho, jak jedná (podle implikací jeho výstupů). Učí se z přímé interakce se svým prostředím, aniž by spoléhal na předpřipravená popsaná vstupní data. Algoritmus se učí sledy akcí, které dosahují nejlepších výsledků – snaží se maximalizovat funkci, která vyhodnocuje (okamžité i budoucí) odměny za agentem právě provedenou akci v prostředí.

Cíl RL je tedy naučit se dobrou **strategii** agenta pro interakci s jeho prostředím. S optimální strategií je agent schopen se přizpůsobovat prostředí k (budoucímu) získávání odměn. (Weng, 2018)



Obrázek 24 Diagram Reinforcement learningu (Weng, 2018)

Agent:

Agent vykonává akce v daném prostředí. Agentem může být buďto samotný algoritmus sítě, nebo i jiný zprostředkovatel, jako například ovládací rozhraní serva / krokového motoru.

Prostředí:

Prostředí reaguje na akce prováděné agentem a vrací agentovi odměnu (ohodnocení, zda jeho akce prospěla či uškodila) a nový pozměněný stav prostředí. Příklad prostředí mohou být fyzikální zákony reálného světa, jeho virtuální simulace nebo třeba prostředí konkrétního herního světa (herní engine). Ve většině situací se prostředí mění samo od sebe, ať už agent přispívá či nepřispívá akcemi. Ne úplně intuitivní, zároveň ale extrémně zajímavý koncept je, že součást prostředí mohou být i **další agenti**.

Interpret – podle složitosti prostředí může být nutné vytvořit interpret, který převede prostředí zpět na pro agenta čitelné stavy a vypočítá odměnu.

Akce:

Množina všech možných tahů, které může agent provést. Agenti si pak vybírají ze seznamu všech možných akcí. V prostředí jednoduché počítačové hry by seznam takových akcí mohl obsahovat běh vlevo, běh vpravo, skočit, dlouze skočit, skrčit se a žádná akce (nepohnout se).

Stav:

Stav je vypozorovaná situace, získaná z aktuálního stavu prostředí, ve které se agent nachází.

Například ve hře Šachy či hře Go je stav reprezentován pozicí všech figur na herní ploše a vyhozených figurkách (perfect information game).

Ve hře BlackJack či Poker obsahuje stav pouze ty informace, které jsou možné znát, ale už ne třeba následující taženou kartu (neúplné informace, imperfekt information game).

Odměna:

Odměna je zpětná vazba prostředí na agentovu akci, sloužící k efektivnímu zhodnocení této akce (a všech předchozích).

Dále pojmy: **Strategie** (taktika založená na aktuálním stavu a dřívějších odměnách), **Hodnota V** (dlouhodobá odměna) a **Q-Hodnota** (okamžitá očekávaná odměna).

(MissingLink dev team, 2018-2020)

Reinforcement learning je ze všech metod pro učení umělých neuronových sítí nejvěrnější tomu, jak se v reálu rozhoduje živý člověk. Je to velmi mocná metoda tréninku, která dokáže vykazovat skvělé výsledky a ve správných podmínkách ohromně převýšit schopnosti člověka. Bohužel, propastný rozdíl mezi reálným a simulovaným prostředím dělá trénování modelů pro reálný svět velmi složité.

1.5.5.1 Deep Reinforcement Learning (DRL)

Hluboké učení a reinforcement učení se vzájemně doplňují. DRL je technologie kombinující tyto dvě metody.

Algoritmus Reinforcement learningu se stará o průběh posloupností důsledků zvolené akce, ohodnocování výsledků a výběr nejvhodnější následující akce.

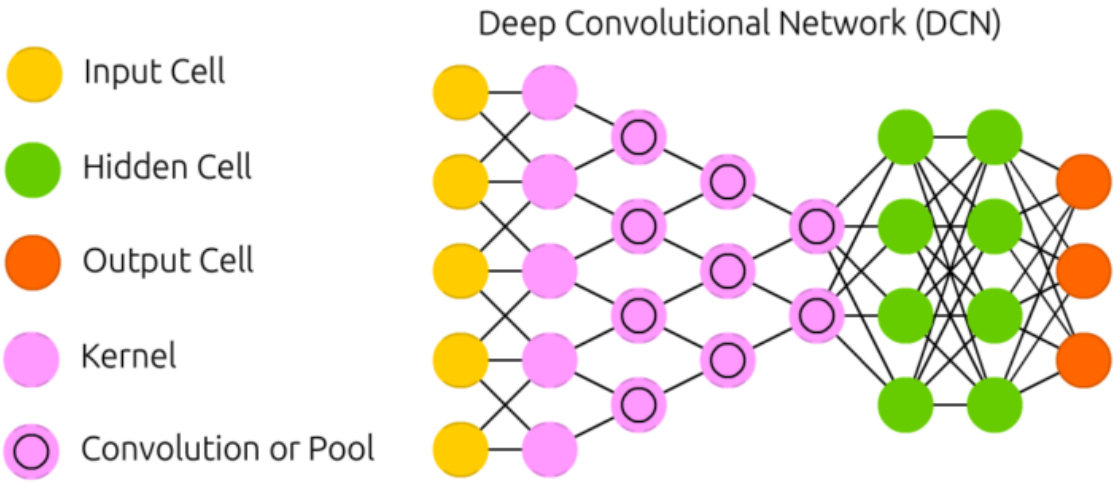
Hluboké učení je právě ten mechanismus, kterým RL používá k výběru následující akce. Protože je to nejmocnější dostupný nástroj k naučení se nejlepšího výsledku na základě předchozích dat.

1.5.6 Konvoluční neuronové sítě

Konvoluční sítě se výrazně liší od většiny ostatních sítí. Nejčastěji se využívají pro zpracování obrazu a jeho klasifikaci, ale vstupní data mohou být i jiná, například audio data.

Rozdíl v klasifikaci obrazu pomocí konvoluční neuronové sítě a čistě hlubokou neuronovou sítí je ten, že hluboká neuronová síť vyžaduje, aby byl klasifikovaný objekt jediným objektem v obraze a aby byl ideálně vycentrovaný (podle tréninku).

Rozdíl konvolučních vrstev oproti vrstvám hlubokého učení je v tom, že konvoluční neurony jsou typicky propojené pouze s malým množstvím „nejbližších“ neuronů v sousedních vrstvách.



Obrázek 25 Model konvoluční neuronové sítě (Van Veen, F. & Leijnen, S., 2019), (vlastní práce)

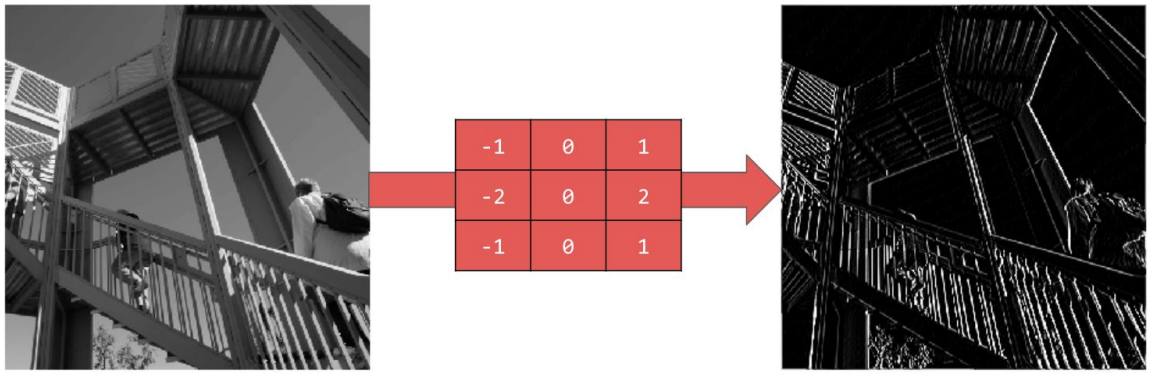
1.5.6.1 Konvoluce

Konvoluce je matematická metoda pro smíchání dvou signálů do jednoho (například spojením rezonance a zvuku kytary vznikne zvuk jak z koncertního sálu). Konvoluce se dokonce dají použít na rozmazání nebo zostření snímku.

Pointa konvoluční sítě je, že před trénováním pomocí hluboké neuronové sítě se použije na data filtr, který projde obrázkem pixel po pixelu. Díky tomuto filtru se rysy obrazu zvýrazní. Natrénovaný filtr bude velmi efektivně schopný získat z obrazu hrany (ze kterých se snadněji identifikují vlastnosti a objekty) a ignorovat téměř vše ostatní.

Princip konvoluce je aplikace filtru na každý pixel obrazu.

Filtr je v podstatě pole parametrů, kterými transformuji (vynásobím) každý pixel tak, aby v sobě obsahoval i informaci z okolních pixelů. Ačkoliv to zní divně, výsledky některých filtrů jsou velmi zajímavé. Jednoduchý filtr ((-1;-2;-1);(0;0;0);(-1;-2;-1)) odstraní z obrázku skoro všechno kromě horizontálních čar. Filtr pro vertikální čáry viz Obrázek 26.



Obrázek 26 Konvoluce - filtr (Google LLC, 2019)

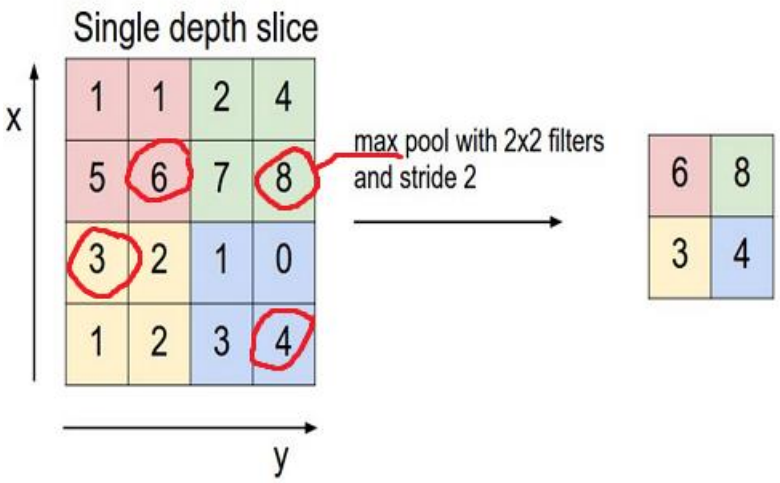
Jak je vidět, konvoluce pomocí filtru je mimořádně efektivní v detekci hran. Takto jednoduchý filtr je však jen příklad, v praxi necháme neuronovou síť parametry filtru najít samu tréninkem pro ideální detekci hran.

1.5.6.1.1 Pooling (subsampling)

Za konvoluční vrstvou typicky následuje pooling vrstva, která velmi přispívá k detekci rysů. Jejím cílem je snížit celkové množství informací obrazu a zároveň neodstranit detekované rysy.

Principem je vzít vždy vlastnosti ze skupiny pixelů a spojit je do jednoho pixelu následující vrstvy. Skupinou pixelů se myslí 2x2, 3x3, 5x5, 7x7 transformovaných v jednu

hodnotu. Nejznámější používanou metodou je technika *Max Pooling*, kde se nejvyšší hodnota použije jako hodnota pixelu reprezentující všechny, ale existují i další metody.

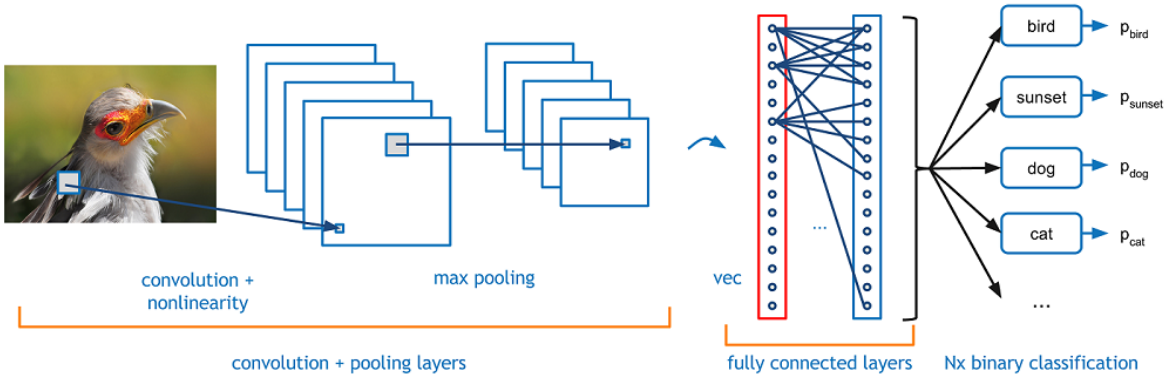


Obrázek 27 Pooling (Deshpande, 2016)

Díky pooling vrstvám mají konvoluční sítě sklon scvrkávat dimenzi (rozlišení), typicky snadno dělitelným číslem vstupu (20x20 pixelů vstup by se mohl v průběhu jedné subsampling vrstvy na 10px a v následující pooling vrstvě na 5px). Často používanými čísly jsou mocniny dvou (256, 128, 64, …). Tato redukce také snižuje riziko overfittingu.
(Van Veen, F. & Leijnen, S., 2019) (Google LLC, 2019)

1.5.6.1.2 Vrstvy konvoluční neuronové sítě

Konvoluční neuronová síť obsahuje konvoluční vrstvy (zvýrazňující hrany), které mohou být následovány pooling vrstvami (zjednodušující učební proces). Tento výsledek je pak klasifikován navazující hlubokou neuronovou sítí s plně propojenými neurony.



Obrázek 28 Vrstvy konvoluční neurové sítě (Deshpande, 2016)

Praktická část

1.6 Datový soubor

Konvoluční neuronová síť vyžaduje učební data, na nichž může trénovat za účelem zvyšování přesnosti predikcí. Sada tréninkových dat se dělí na tři podmnožiny:

- Množina tréninkových dat
    - Dataset, ze které se systém trénuje své váhy a bias.
- Testovací soubor dat
    - Menší soubor dat, který algoritmus ještě neviděl, ze kterého může ohodnotit, jak dobře je naučen.
- Validační data
    - Validační data slouží k nezávislému posouzení, jak dobře sedí model a vyladit podle toho hyperparametry.

Mezi hyperparametry patří množství skrytých vrstev, počet neuronů, funkce ztráty, aktivační funkce a množství, kolikrát by měl být tréninkový proces opakován neboli množství epoch. Dále učící krok (learning rage), batch size (množství paralelně vyhodnocovaných výsledků v jednom učebním kroku) i optimalizér.

V případě výběru tak velmi specifické sítě, jakou je YOLOv3 ve frameworku Darknet, jsou hyperparametry účelně vyladěné a není vhodné je jakkoliv měnit. Proto budou zmíněné jen okrajově.

Pro vyladění vnitřních nastavení sítě lze použít intuici, experimentaci, validační set, ale i podívání se na jiné architektury a vypůjčením si jejich nápadů, popřípadě čerpání z literatury.

Princip ručního návrhu, jak rozhodnout počtet skrytých vrstev a množství jejich neuronů je podrobně sepsán viz tento článek (Gad, 2018).

1.6.1 Příprava dat

Množství potřebných dat pro síť je závislé na počtu druhů klasifikovaných objektů a složitosti rysů těchto objektů. Díky faktu, že banán je jediným důležitým klasifikačním

objektem této práce, a díky jednoduchosti jeho tvaru, jsou požadavky na velikost datového setu malé.

Optimální počet byl zvolen na 150 fotografií. Tohoto počtu bylo dosaženo pořízením 30 fotografií. Dataset byl následně rozšířen za pomoci augmentačních metod.

Pro potřeby práce byly fotoaparátem snímány regály s banány. Snímky obsahující banány byly pořízeny z více úhlů. Na fotografiích jsou banány jak zvlášť, tak i pohromadě. Součástí sbíraných dat jsou dále snímky částečně a zcela zakrytých banánů. A obecně ve všech možných variacích v jakých se nacházely.

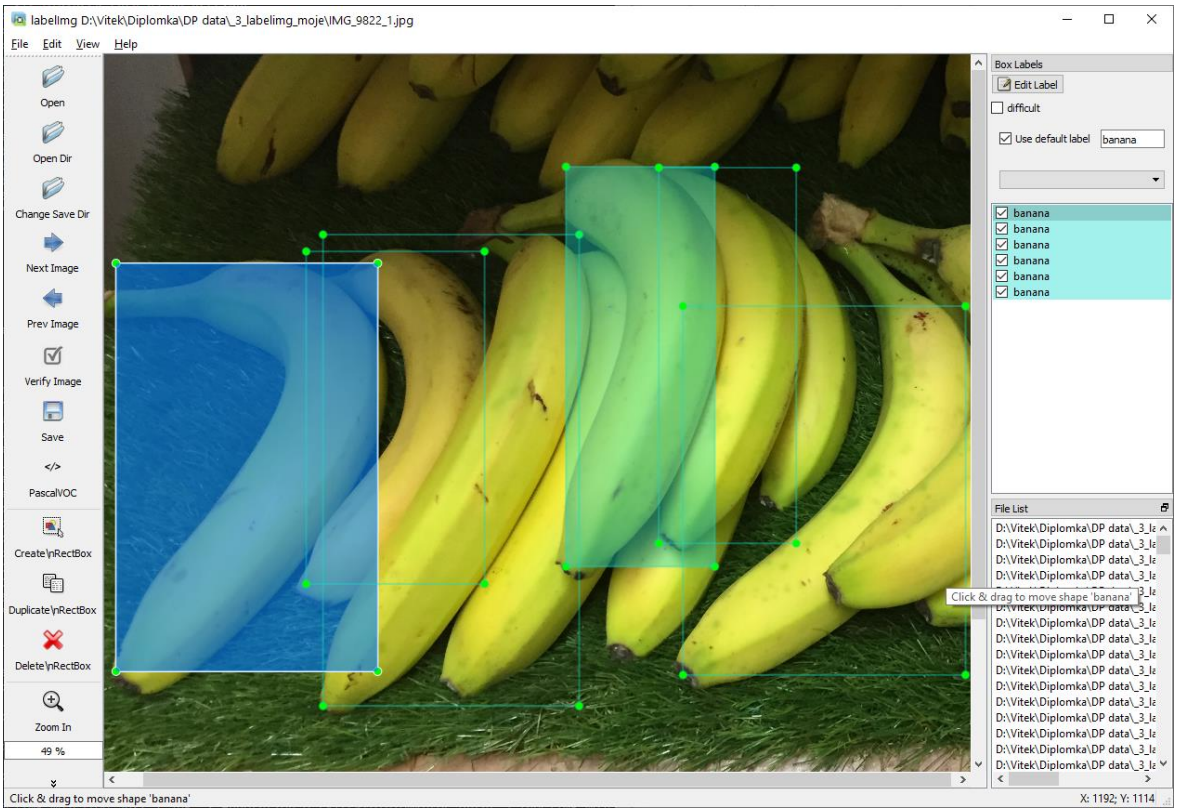
Z každého z původních 44 snímků bylo za pomoci programu Zoner Photo Studio X ručně oříznuto jedna až 6 nových snímků tak, aby na nich byly přítomny sledované objekty. Zároveň ořez sloužil k zajištění poměru stran 1:1. Tímto se kolekce dat rozrostla na 140 fotografií.

V následujícím kroku byl soubor dat sjednocen na rozlišení 416x416 pomocí online služby https://bulkresizephotos.com/.

1.6.2 LabelIMG

Data jsou nyní ve správném formátu a je zapotřebí k nim vytvořit popisky.
Pro vytvoření anotací byl zvolen program LabelImg (viz 1.7.1).
Tím, že označíme pouze zřetelně viditelné banány, se model naučí nacházet jen ty v popředí, což je požadovaná situace.

Aby se model naučil detekovat hlavně banány v popředí, při označování ignorujeme zakryté banány. Takové chování je pro naše využití vhodnější než přesné počty.



Obrázek 29 Program LabelImg (vlastní tvorba)

1.6.3 Augmentace dat

Augmentace dat je důležitá technika pro trénování modelu počítačového vidění. Jedná se o doplnění vstupních dat jejich (náhodné) transformované variace.

Mezi takové patří: přiblížení, vertikální / horizontální převrácení, rotace, změna jasu, kontrastu aj. (Shaw, 2018)

Obrázek 30 Augmentace vstupnich dat - změna jasu (vlastní tvorba)

Pro augmentaci dat mohou sloužit nástroje FastAI, Keras či RoboflowAI. Pro naše data jsme zvolili augmentaci náhodné rotace obrazu, vertikálního převrácení a jasu. Datová kolekce se takto rozšířila na transformací 840 snímků.

1.7 Výběr prostředí, technologie

1.7.1 Základní prostředí a programové vybavení

Jazyk Python

https://www.python.org/, svobodný software (BSD)

- je velmi univerzální moderní programovací jazyk, relativně snadný a pro většinu běžných potřeb i dostatečně výkonný. (https://www.python.org/)

Anaconda

https://www.anaconda.com/, svobodný software (BSD)

- robustní enterprise platforma obsahující mj. distribuci pythonu
- změna verze pythonu: 3.7

Pandas

https://pandas.pydata.org/, open-source software

- vysoce výkonná knihovna pro efektivní práci s datovými strukturami

Jupyter Notebook

https://jupyter.org/, open-source software

- živý dokument umožňující psát, programovat, vizualizovat data a další.

LabelImg

https://github.com/tzutalin/labelImg, svobodný software (MIT)

- nástroj s grafickým rozhraním pro popisování objektů na obrázcích pomocí čtvercových ohraničení

Qt5 (multiplatformní framework, https://wiki.qt.io/About_Qt, svobodný GPL)

1.7.2 Cesta 1 – Localhost: Framework Darknet na domácím prostředí

Obrázek 31 Darknet (Redmon, 2016)

Rozšířené prostředí pro hluboké učení

Open-source framework neuronových sítí napsaný v C a CUDA (Redmon, 2016)

- CMake – opensource crossplatform kompilátor
- NVidia CUDA – rozhraní pro paralelní zpracovávání grafickými procesory
- NVidia cuDNN – CUDA Deep Neural Network library
- NVidia CUDA Toolkit – instrukce pro CUDA
- GitHub – repozitáře programů
- OpenCV – Open Source Computer Visuon Library – široká podpora mediálních formátů.

  - složitý instalační, konfigurační proces na dosažení na MS Windows (ani po 2 týdnech se nepodařilo najít funkční FOURCC filtr)

image.c | demo.c | Makefile | rTargetPointer.lua | 2017-12-08-06-36-59_1.xml | detector.c

```
206     CvVideoWriter* output_video_writer = NULL;     // cv::VideoWrite
207     if (out_filename && !flag_exit)
208     {
209         CvSize size;
210         size.width = det_img->width, size.height = det_img->height;
211         int src_fps = 25;
212         src_fps = get_stream_fps(cap, cpp_video_capture);
213
214         //const char* output_name = "test_dnn_out.avi";
215         output_video_writer = cvCreateVideoWriter(out_filename,
216             CV_FOURCC('H', '2', '6', '4'), src_fps, size, 1);
217         output_video_writer = cvCreateVideoWriter(out_filename,
218             CV_FOURCC('D', 'I', 'V', 'X'), src_fps, size, 1);
219         output_video_writer = cvCreateVideoWriter(out_filename,
220             CV_FOURCC('M', 'J', 'P', 'G'), src_fps, size, 1);
221         output_video_writer = cvCreateVideoWriter(out_filename,
222             CV_FOURCC('M', 'P', '4', 'V'), src_fps, size, 1);
223         output_video_writer = cvCreateVideoWriter(out_filename,
224             CV_FOURCC('M', 'P', '4', '2'), src_fps, size, 1);
225         output_video_writer = cvCreateVideoWriter(out_filename,
226             CV_FOURCC('X', 'V', 'I', 'D'), src_fps, size, 1);
227         output_video_writer = cvCreateVideoWriter(out_filename,
228             CV_FOURCC('W', 'M', 'V', '2'), src_fps, size, 1);
```

Báze programu

Ruční stažení frameworku Darknet, dle návodů pokus o správnou úpravu konfiguračních souborů a souborů makefile, kompilace zdrojových kódů, stažení všech různých variant předučených vah.

V této cestě se autor po několika týdnech dostal do slepé uličky, protože požadavky na běh hluboké sítě s předučenými váhami překračovaly hardwarové prostředky relativně nové herní stanice.

Z důvodu nedostatku systémových prostředků a složité konfiguraci bylo zapotřebí najít jiné řešení.

1.7.3 Cesta 2 – Cloud: Google colaboratory

Obrázek 32 Logo Google Colab a Jupyter

Google colaboratory (krátce Colab) je kulminací několika koncepcí. Tento velmi moderní projekt začal vnikat poté, co Microsoft v na konci roku 2015 vydali TensorFlow jako open-source. Jeho největší výhodou jsou nulové nároky na lokální stroj, přenositelnost a veliké množství vývojových nástrojů.

Google Colab

- Python 2 a 3 (verze 2 ztrácí od začátku roku 2020 podporu)
- framework neuronových sítí TensorFlow
- vlastní implementace Jupyter Notebooku, fungující jako všezahrnující textový editor, vývojové prostředí, kompilátor, poznámkový blok, statistický program, nástroj sdílení i kolaborace – to **vše 100 % v cloudu**
- další knihovny Keras, PyTorch, OpenCV
- velmi rychlé připojení a administrátorská práva
- každý má v prostředí svůj vlastní Virtuální Stroj

Přechod z původního způsobu práce na lokálním prostředí je krokem k modernizaci.

1.7.3.1 Parametry Google Colab

Služba je zdarma, limitovaná pouze maximálními zdroji (viz Obrázek 33), ale tento rozsah je zatím tak veliký, že při osobních projektech nijak omezuje. Instance virtuálního stroje se každých 12 hodin přemaže, ale lze použít přes API propojený Google drive pro ukládání rozpracovaných dat.

Colab vs. Colab Pro

	Price	GPU	Runtime	Memory
Colab	Free	K80	Up to 12 hours	12GB
Colab Pro	$9.99/m (before tax)	T4 & P100	Up to 24 hours	25GB with high memory VMs

Obrázek 33 Colab Pro (Kim, 2020)

Colab Pro za 10$/měsíčně je zajímavá nabídka. Pro porovnání, před ~5 lety obdobný cloud 1000$/měsíčně. (Kim, 2020)

1.7.3.2 Srovnání výkonu Colab Free a soukromé domácí herní stanice

- K80 24GB RAM ~3000GFLOPS ($\times 10^9$ početních operací/s)
- GTX 1060 3GB RAM ~123GFLOPS

1.7.3.3 CPU vs GPU vs TPU: výkon a co zvolit

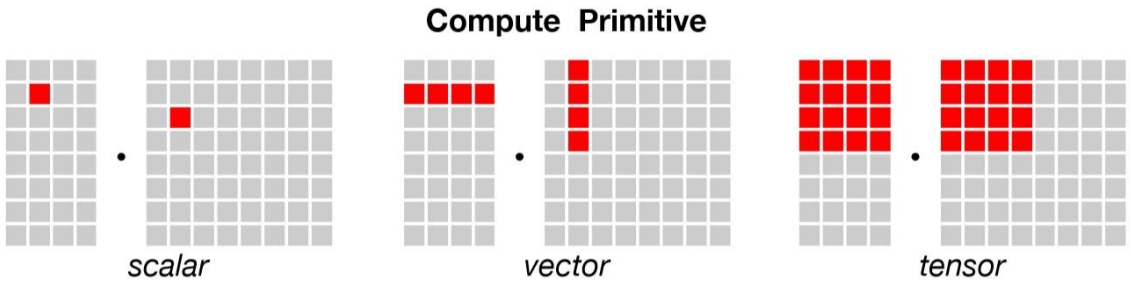
Trénování modelů hlubokého učení je v podstatě násobení matic. To by zdálo jako jednoduchá úloha, ale přidejme do kontextu rozsáhlost hlubokého učení: VGG16 (běžné používaná CNN s 16 skrytými vrstvami) obsahuje ~140 milionů parametrů, které je potřeba trénováním nastavit.

Konvenčními metodami by se taková síť trénovala roky. (Shaikh, 2017)

Vysvětlení na příběhu:

Scénář: Chceme převést zboží z místa A do B. Možnosti dopravy sportovní Porshe nebo kamion. Porshe by bylo extrémně rychlé a převezlo by zboží okamžitě, ale množství za jednu cestu je malé a spotřebovaná energie (palivo) velké. Kamion by byl pomalejší, ale energeticky úspornější a množství zboží výrazně lepší.

Ale samozřejmě existují situace, kde je výhodnější využít Porshe.



Obrázek 34 CPU vs GPU vs TPU (OpenGenus Foundation, 2018)

	CPU	GPU	TPU
(základní = nejmenší) komputační jednotka	1 × 1	1 × N	N × N
Výkon (instrukce /takt)	desítky	tisíce	128 000
účel	obecnost	grafika, ML, paralelizace	ML: TensorFlow

Obrázek 35 Tabulka CPU, GPU, TPU (Vlastní zpracování)

1.8 Implementace detektoru YOLOv3 s frameworkem Darknet

1.8.1 Nastavení prostředí

a. jazyk = Python 3
b. hardwarova akcelerace = GPU.

1.8.2 Framework Darknet

Jedná se o hlubokou konvoluční neuronovou síť, která je známá svou (oproti konkurenci) vysokou rychlostí, díky které dokáže detekovat objekty v reálném čase 30FPS.

Přestože se síť soustředí na rychlost (4 až 5krát rychlejší než ostatní detektory), v přesnosti predikcí před špičkou své konkurence zaostává jen o 0–10 %.

V srpnu 2018 vyšla nová architektura neuronové sítě s označením Darknet53, podle množství vrstev. I přes značné zvýšení vrstev (z Darknet19) si drží špičkovou rychlost.

Method name	Speed (FPS)	Input size
YOLOv3 (Darknet53)	**45.5**	320x320
YOLOv3 (Darknet53)	**34.5**	416x416
YOLOv3 (Darknet53)	19.6	608x608

Obrázek 36 Výkon nové iterace Darknetu53 (Abbasi, 2019)

Další srovnání výkonnosti existujících sítí viz (Abbasi, 2019).

```
# stahnu si repozitar Darknetu
!git clone https://github.com/AlexeyAB/darknet
```

(…)

```
# soubor makefile dle instrukci na githubu pro zapnuti GPU a OPENCV
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

(…)

```
# overeni CUDA
!/usr/local/cuda/bin/nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2018 NVIDIA Corporation
Built on Sat_Aug_25_21:08:01_CDT_2018
Cuda compilation tools, release 10.0, V10.0.130
```

```
# zkompiluj darknet
!make
```

1.8.3 YOLOv3 – detektor objektů v reálném čase

Zde používáme předučený model, detektor YOLOv3.

Neuronová síť byla předučena na rozsáhlé datové kolekci COCO:

- obsahuje 330 tisíc obrázků,
- s více než 200 tisíci anotacemi,
- dohromady označeno 1,5 milionu objektů,
- celkem v 80 kategoriích. (http://cocodataset.org/#home)

```
# stahni vahy yolov3 predtrenovane na datove sade COCO
!wget https://pjreddie.com/media/files/yolov3.weights

# definice pomocnych funkci
def imShow(path):
  import cv2
  import matplotlib.pyplot as plt
  %matplotlib inline

  image = cv2.imread(path)
  height, width = image.shape[:2]
  resized_image = cv2.resize(image,(3*width, 3*height), interpolation =
 cv2.INTER_CUBIC)

  fig = plt.gcf()
  fig.set_size_inches(18, 10)
  plt.axis("off")
  plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
  plt.show()

# upload funkce
def upload():
  from google.colab import files
  uploaded = files.upload()
  for name, data in uploaded.items():
    with open(name, 'wb') as f:
      f.write(data)
      print ('saved file', name)

# download funkce
def download(path):
  from google.colab import files
  files.download(path)
```

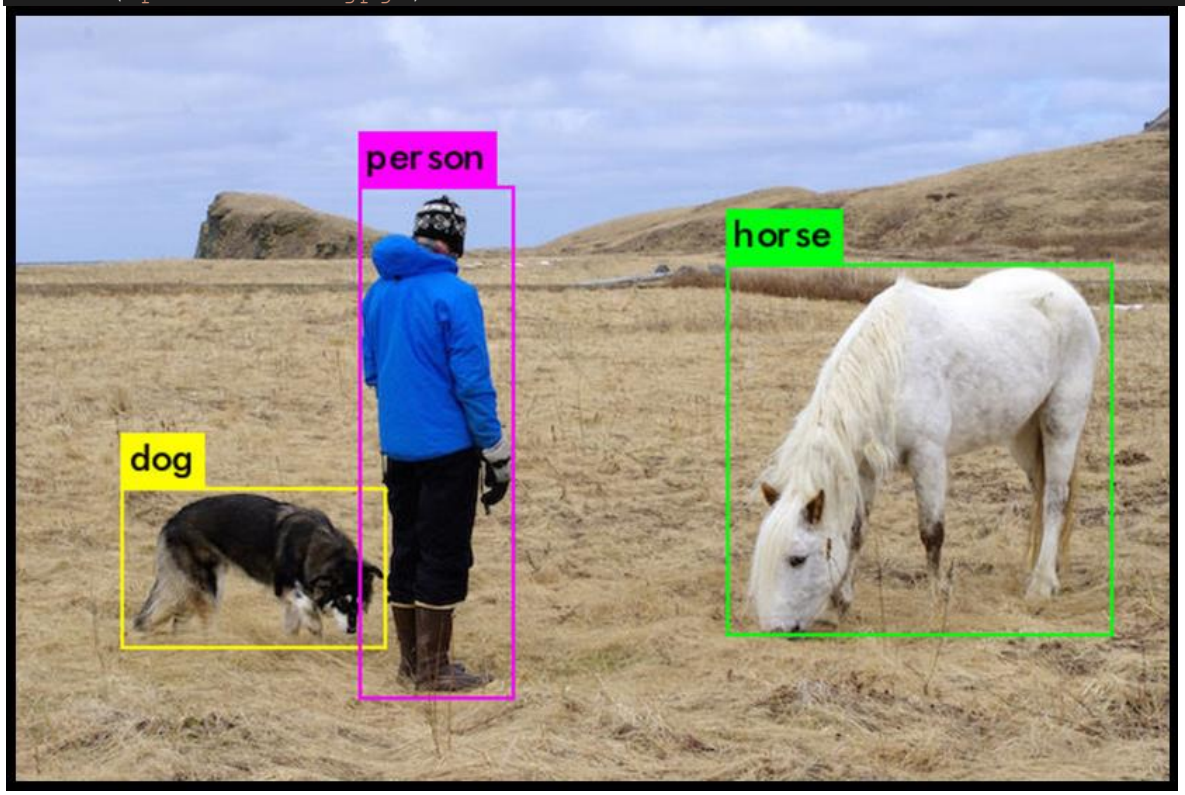
1.8.4 Detekce objektů sady

V cloudu běžící darknet dokáže z vah funkcí *detect* detekovat 80 tříd z COCO.

```
# parametry <path to config> <path to weights> <path to image>
```

```
!./darknet detect cfg/yolov3.cfg yolov3.weights data/person.jpg
Done! Loaded 107 layers from weights-file
data/person.jpg: Predicted in 21.690000 milli-seconds.
dog: 99%
person: 100%
horse: 100%
Unable to init server: Could not connect: Connection refused
(predictions:1288):Gtk-WARNING **: 16:08:28.264: cannot open display:
```

```
# ve virtualnim stroji se nam vystupy neukazuji automaticky, proto:
imShow('predictions.jpg')
```



Obrázek 37 Vnitřní ukázkový příklad YOLOv3

Jak je popsáno v kapitole viz 1.7.3.1, instance VM se nejpozději za 12hodin vyčistí. Proto je vhodné napojit k notebooku Google Drive a data ukládat i číst odtamtud.

Druhá poznámka, pole kódu Jupyter notebooku se chovají jako terminál. Jdou v něm používat windows i linux příkazy (`%příkaz`), ale hlavně jsou pole citlivá na to, v jaké jsme složce. Pokud vystoupíme ze složky `%cd ..`, pak se musíme i vrátit abychom mohli spustit `!./darknet detect`.

(Redmon, 2016), (Ibáñez, 2019), (The AI Guy, 2020), (AlexeyAB, Joseph Redmon, 2016-2020)

1.8.5 Nalezena chyba ve zdrojovém kódu YOLO (VIZ Error! Reference source not found.)

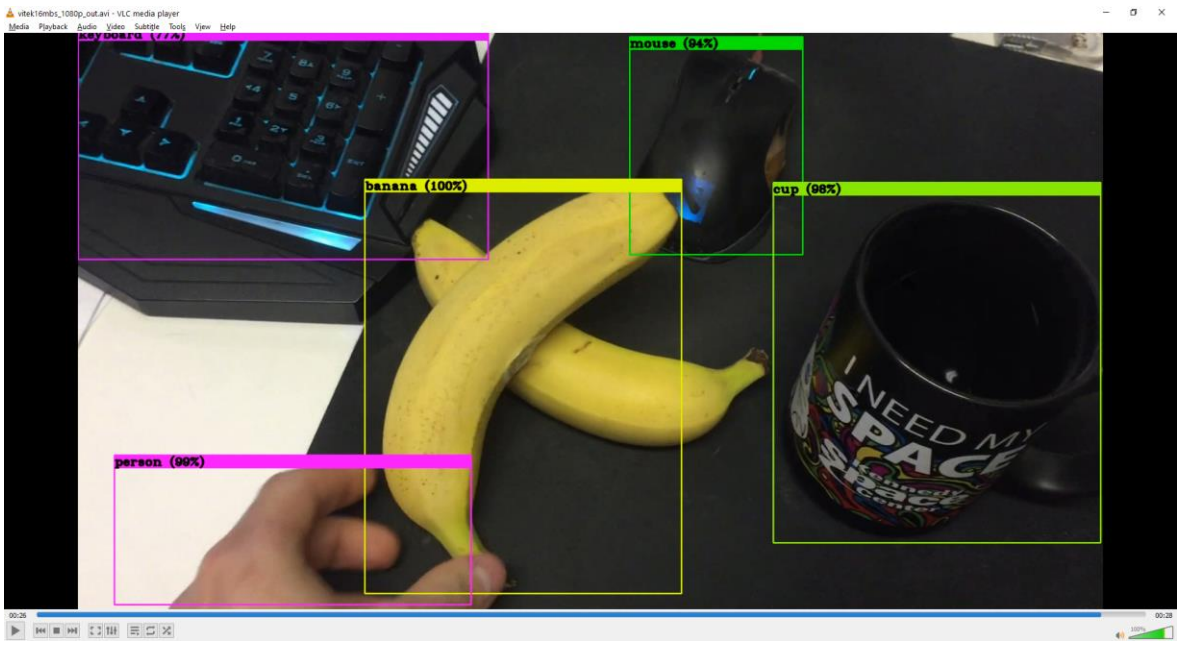
V tomto místě se autor rozhodl podívat se do kódu, aby našel odpověď na otázku: jaké všechny parametr přijímá funkce *./darknet detect*; a lze parametricky pojmenovat výstupní soubor?

Z důvodu nespojitosti formátování je kapitola o nalezené chybě vedena jako příloha; viz **Error! Reference source not found.**)

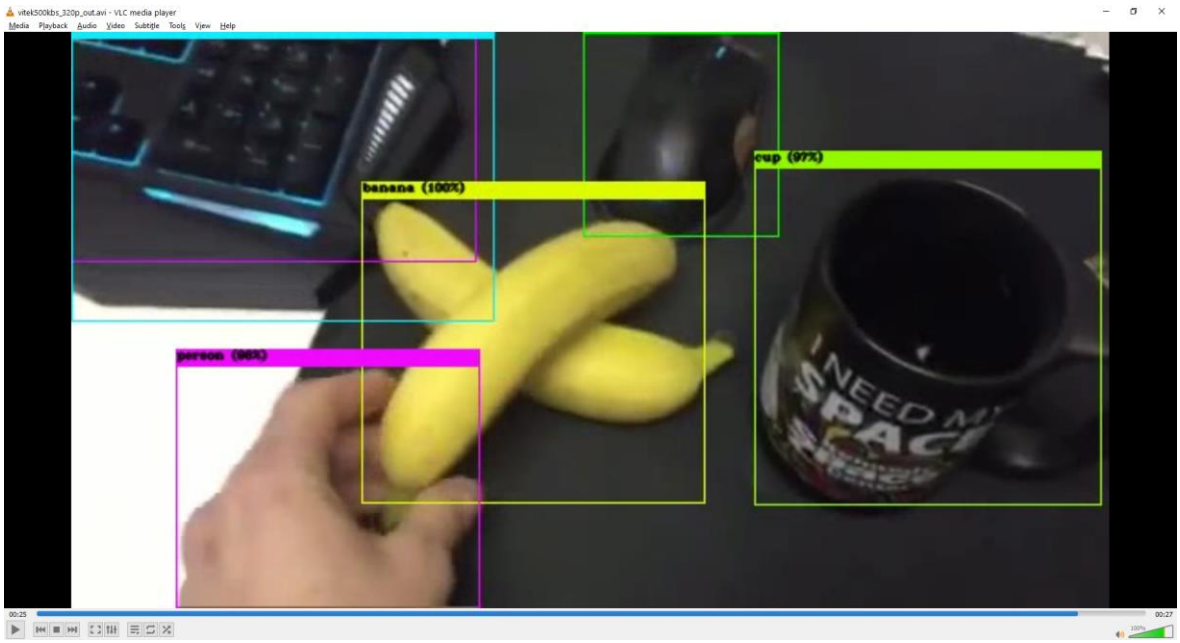
1.8.6 Detekce objektů z videa

```
# vlastni video
# /darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights <v
ideo file>
!./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -
dont_show data/video/vitek16mb-s,1080p.MOV -i 0 -
out_filename data/video/vitek16mbs_1080p_out.avi
!./darknet detector demo cfg/coco.data cfg/yolov3.cfg yolov3.weights -
dont_show data/video/vitek500kb-s,320p.mp4 -i 0 -
out_filename data/video/vitek500kbs_320p_out.avi
```

Na dvou následujících snímcích z videa vysoké a nízké kvality, také natočeného na mobilní telefon, se přesnost téměř neliší. Původní model, trénovaný na datasetu COCO také dokázal rozpoznávat třídu „banana“.



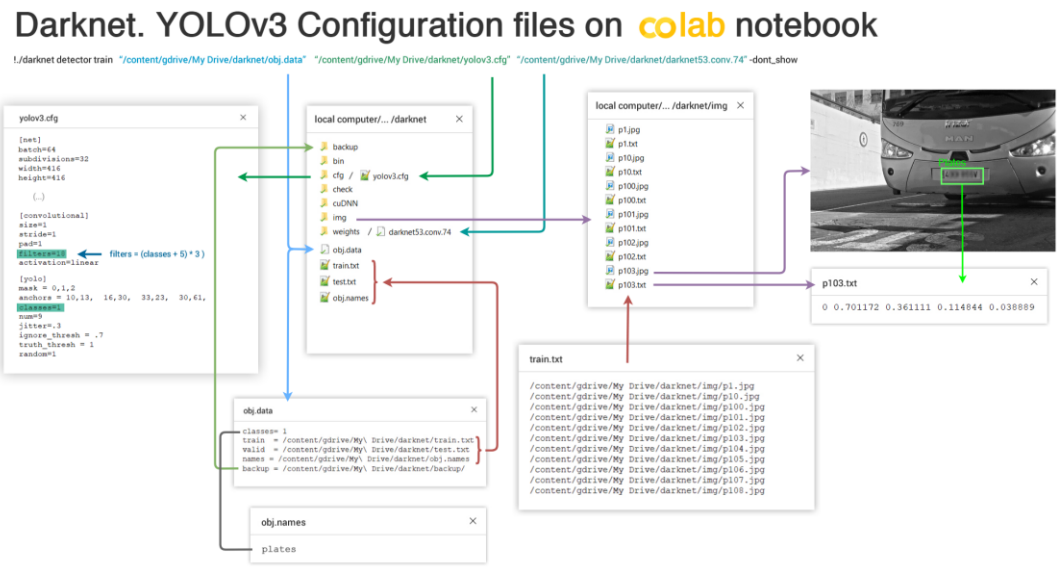
Obrázek 38 Video uvnitř, vysoká kvalita (vlastní tvorba)



Obrázek 39 Video uvnitř, nízká kvalita, téměř stejný moment (vlastní tvorba)

1.9 Trénink modelu

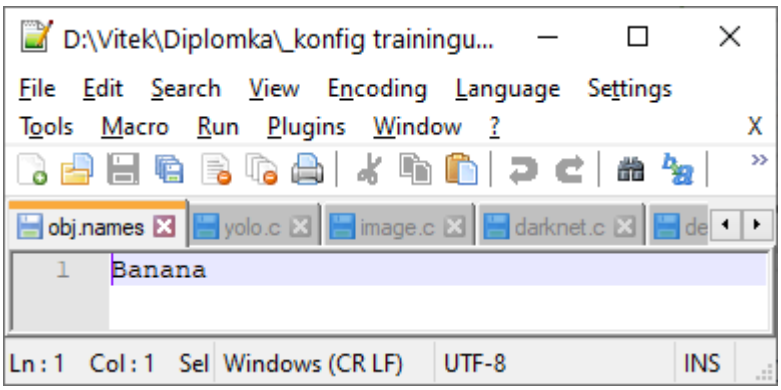
1.9.1 Konfigurace



Obrázek 40 Konfigurační postup trénování YOLO (Ibáñez, 2019)

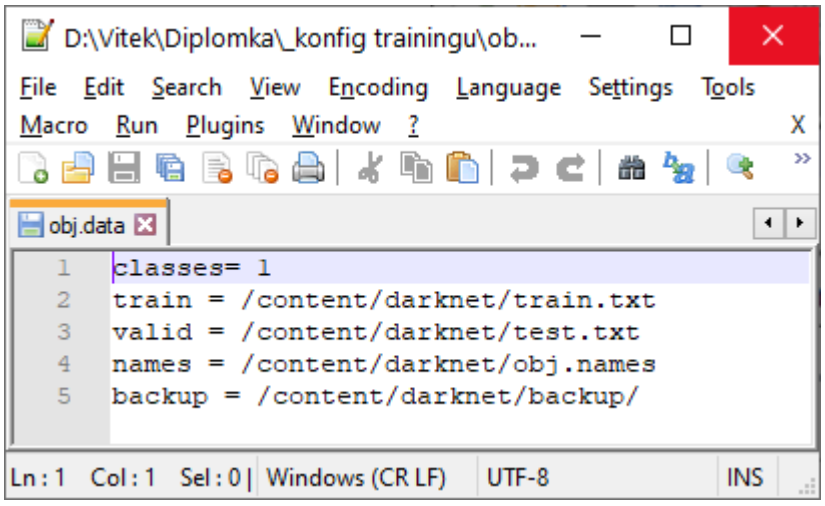
Nejprve je zapotřebí adekvátně upravit konfigurační soubory, které vstupují do trénovací funkce v bodě 1.9.3.

vytvoříme textový soubor s příponou obj.names



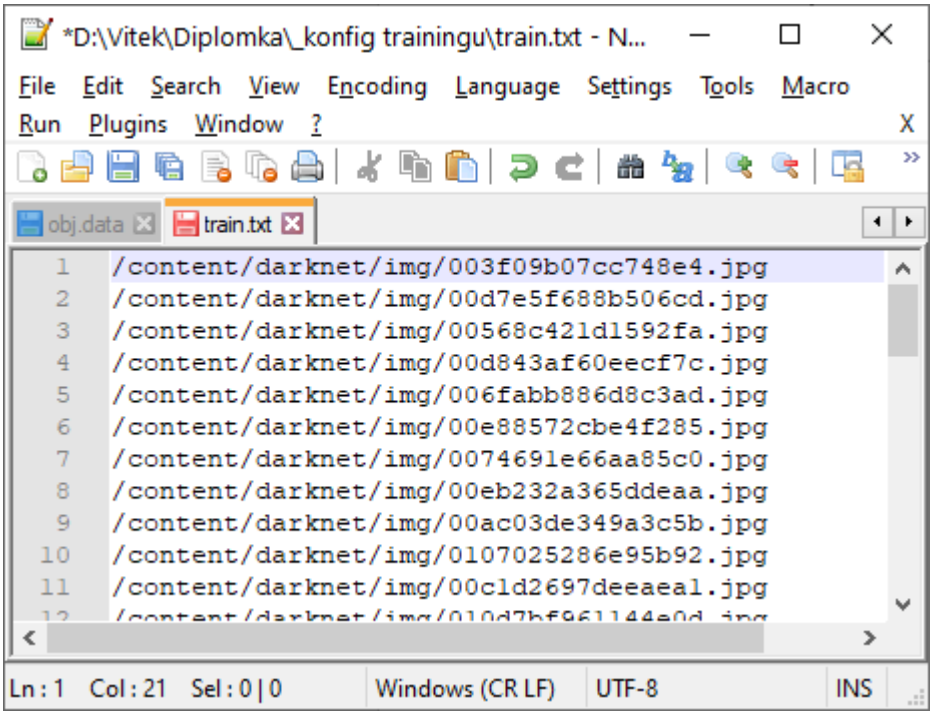
Obrázek 41 Soubor obj.names

vytvoříme textový soubor s příponou obj.data



Obrázek 42 Soubor obj.data

soubor train.txt



Obrázek 43 Soubor train.txt

yolov3.cfg

před úpravou zálohuji konfigurační soubor yolov3

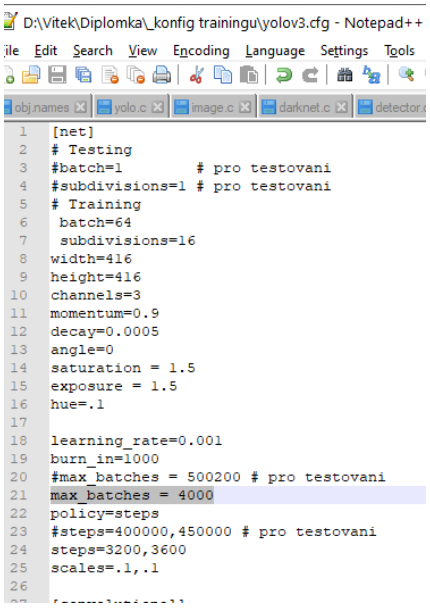
```
!cp cfg/yolov3.cfg cfg/yolov3_backup.cfg
```

nastavíme batch=64

nastavíme subdivision=16

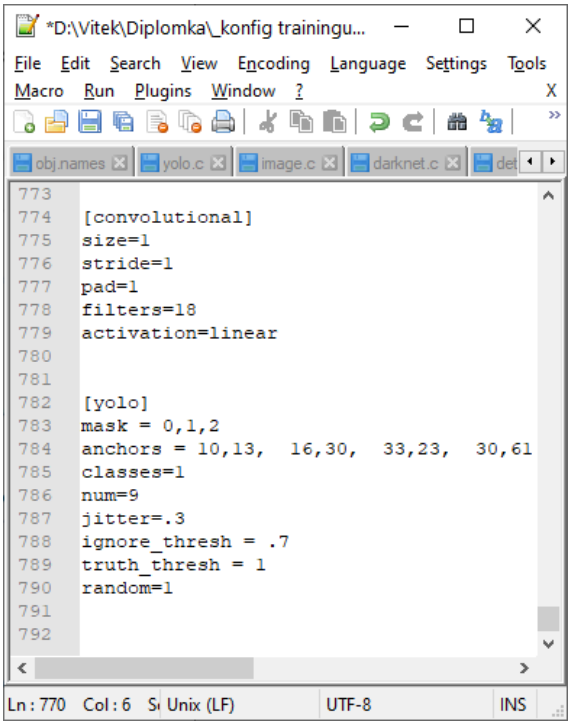
max_batches=4000 (doporučený údaj pro 1 třídu)

steps=3200,3600 (doporučené dle 80 a 90 % max_batches)



Obrázek 44 Konfiguracni soubor yolov3.cfg

Vyhledáme všechny 3 vrstvy [yolo] a upravím počet tříd na 1, změním parametr filters na 18.



Obrázek 45 Konfigurace vrstev v yolov3.cfg

Stáhnout předtrénované konvoluční váhy: darknet53.conv.74 do složky \darknet\weights\

Obrázek 46 Předtrénované váhy imagenetu

1.9.2 Nahrání tréninkových dat (po augmentaci, viz 1.6.3)

Virtuální stroj Colabu je vhodné propojit se službou Google Drive pro snazší správu souborů.

```
from google.colab import drive
drive.mount('/content/drive')
```

Tréninkové data vložíme do Google Drive do složky: darknet/img/

Nahraná tréninková data zkopírujeme do prostředí Google Colab.

```
# Copy fils from Google Drive to the VM local filesystem
!cp -r "/content/gdrive/My Drive/darknet/img" ./img
```

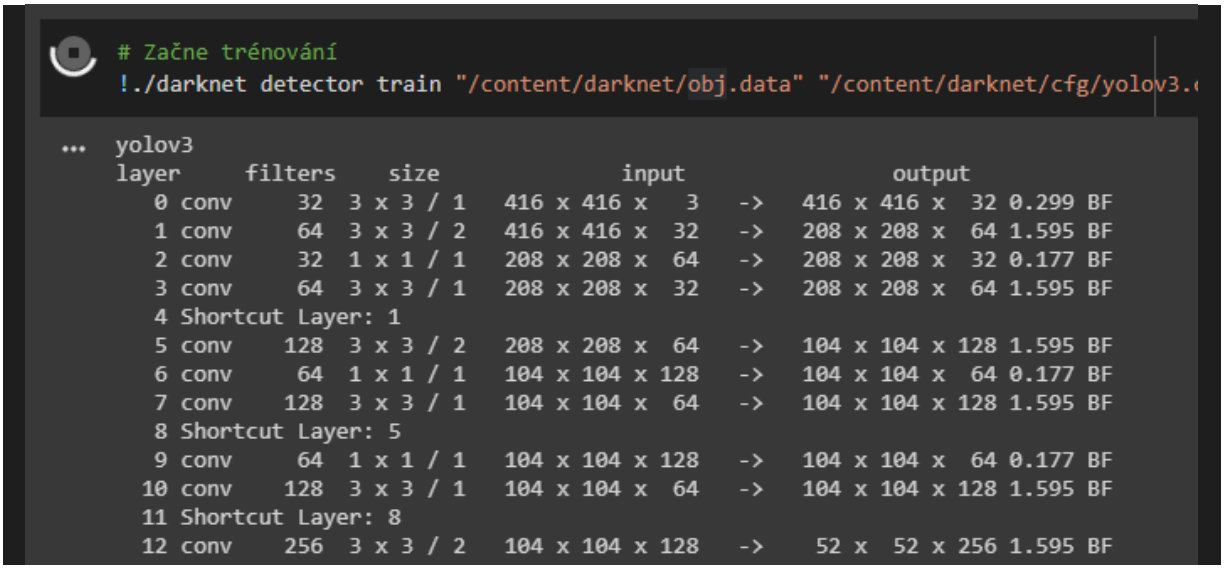
1.9.3 Natrénování modelu

Spuštěním příkazu Darknet detector train začne trénink.

Každá dokončená epocha vypíše na výstupní konzoli aktuální stav učícího procesu.

(Ibáñez, 2019)

```
# Začne trénování
!./darknet detector train "/content/darknet/obj.data" "/content/darknet
/cfg/yolov3.cfg" "/content/darknet/weights/darknet53.conv.74" -
dont_show
```



```
# Začne trénování
!./darknet detector train "/content/darknet/obj.data" "/content/darknet/cfg/yolov3.
```

```
yolov3
layer     filters    size              input                output
   0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32 0.299 BF
   1 conv     64  3 x 3 / 2   416 x 416 x  32   ->   208 x 208 x  64 1.595 BF
   2 conv     32  1 x 1 / 1   208 x 208 x  64   ->   208 x 208 x  32 0.177 BF
   3 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1
   5 conv    128  3 x 3 / 2   208 x 208 x  64   ->   104 x 104 x 128 1.595 BF
   6 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64 0.177 BF
   7 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5
   9 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64 0.177 BF
  10 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8
  12 conv    256  3 x 3 / 2   104 x 104 x 128   ->    52 x  52 x 256 1.595 BF
```

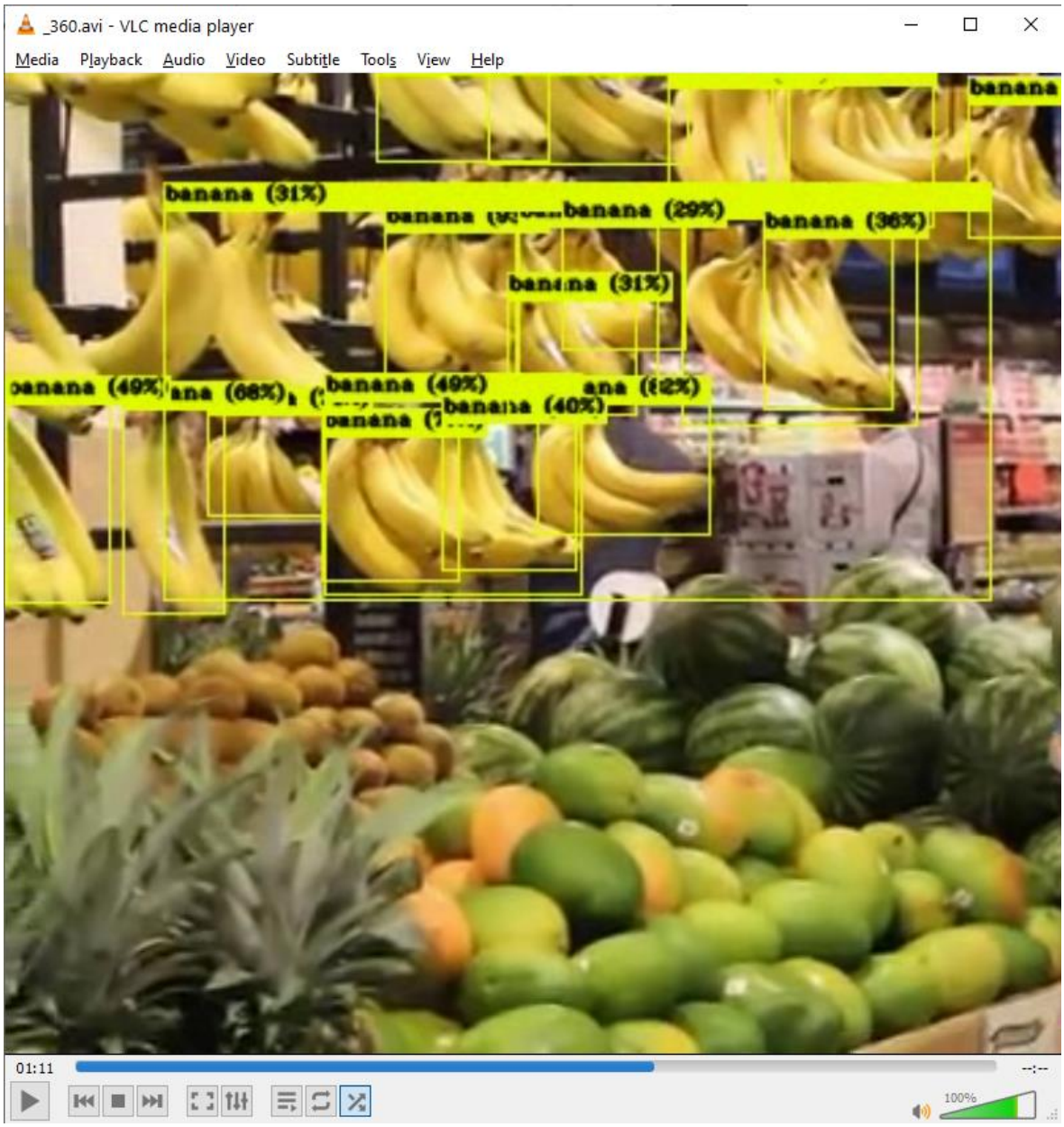
Obrázek 47 Implicitní sestava modelu při trénování (Vlastní tvorba)

1.9.4 Detekce objektů z videa

K natočení použit telefon. Testována byla přesnost i rychlost detekce u 1080p velkého videa a následně vždy i u identického videa, který byl ztrátově zkomprimován na 360p.

Zpracování videa trvá přesně tak dlouho, jak je dlouhé, protože si je YOLO při detekci objektů přehrává v reálném čase.

Obrázek 48 Detekce objektů na přeučené síti, vysoké rozlišení (Vlastni tvorba)



Obrázek 49 Detekce objektů na přeučené síti, nízké rozlišení (Vlastni tvorba)

Z obrázků Obrázek 49 a **Error! Reference source not found.**, že model detekoval banány v obou rozlišeních velmi dobře. Čas výpočtu byl podle očekávání v obou případech shodný s délkou videa.

Výsledky a diskuse

Výstupem kroků rozebraných v předchozích kapitolách je pořízení vlastní datové sady, optimalizace této sady pro trénink, kvalifikovaná volba vhodné softwarové základny, implementace předtrénované konvoluční neuronové sítě a demonstrace klasifikace objektů.

1.10 Transformace oštítkovaných dat včetně anotací

V praktické části bylo nejprve úkolem připravit data. To znamená použít augmentaci obrazu a označit data štítky.

Původním předpokladem bylo, že pokud by se provedla augmentace nad oštítkovanými daty, tak by se o tyto štítky přišlo, protože by nepodléhaly transformacím.

Proto byla provedena augmentace následovaná ručním označit násobě většího množství snímků.

Později bylo zjištěno, že nástroje jako RoboFlowAI umí provádět Augmentace se zachováním anotací. Tato zkušenost výrazně zjednoduší a zrychlí budoucí přípravu dat.

1.11 Výběr programové základny

Na základě rešerše a dostupných informací byla učiněna rozhodnutí vedoucí k implementaci Darknet YOLOv3 v lokálním prostředí.

Z důvodu složitého konfiguračního procesu a systémových požadavků převyšujících možnosti lokálního prostředí bylo zapotřebí najít jiné řešení.

Nejvhodnějším řešením se ukázala implementace YOLOv3 za použití frameworku Darknet v prostředí Jupyter Notebook na cloudové službě Google Colaboratory.

1.12 YOLOv3 detektor

Ve virtuálním prostředí Google Colab bylo dosaženo vlastní implementace předučeného detektoru YOLOv3. Na něm ověřena funkčnost detekce objektů z videa v reálném čase.

Následně jsme porovnali detekci objektů pouze třídy banánů na námi dotrénované síti (viz 1.9.4) oproti předtrénované. Úroveň detekcí banánů na těchto modelech je srovnatelná. Důvodů pro toto chování může být několik.

Tvar banánu je velmi jednoduchý, a protože má tak snadno rozlišitelné rysy tvaru a barvy je tudíž jeho klasifikace triviální a s vysokou pravděpodobností predikce.

1.12.1 Nalezena CHYBA ve zdrojových souborech modulu YOLO, viz:

Error! Reference source not found. **(viz** Error! Bookmark not defined.**)**

1.12.2 Free software a Open-source

Ve většině kapitol, které obsahují instalaci či inicializaci:

➔ většina použitého softwaru je buďto free sortware licence nebo přímo open-source.

➔ využití pro veřejnost.

Závěr

Cílem této diplomové práce bylo prakticky ověřit schopnost a rychlost rozpoznat naučené objekty v reálném čase konvoluční neuronovou sítí YOLOv3.

Na začátku této práce je popsáno několik konceptů lineární algebry, které autor považuje za podstatnější. Z lineární algebry vyzdvihuje zejména teorii matic. Klade obzvlášť důraz na vysvětlení role matic a tenzorů.

V hned následující části je podrobně popsána neuronová síť, v krátkosti její historie a její podobnost k biologické neuronové síti. Následuje definice umělé inteligence, její cíl a rozdíl od deterministického programování.

Kapitola s popisem strojového učení obsahuje i vysvětlené nejběžnější druhy úkolů, které se přesně hodí pro strojové učení. Konkrétně jde o třídění, třídění s neúplnými vstupy, regresní úloha, přepis, strojový překlad, strukturovaný výstup, detekce anomálií, syntéza a vzorkování, algoritmus nahrazení chybějících hodnot a odstraňování šumu.

Na toto téma plynule navazuje hluboké učení, obsahující informace o hlubokém učení a jeho vztahu k lidskému mozku a k vrstvám.

Následující kapitola pojednává o konceptech neuronové sítě, kde kromě zcela základních jsou popsány váhy, perceptron, zpětná propagace, hyperparametry, underfitting a overfitting včetně způsobů, jak se mu vyhnout.

Kapitola „Účel skrytých vrstev“ názorně popisuje význam jednotlivých skrytých vrstev a jaké informace se udržují v neuronech.

Následuje skupina kapitol zaměřených na popis a charakteristiku modelů neuronových sítí. Prvním tématem jsou algoritmy učení bez učitele a učení s učitelem. Druhým tématem je Dopředná a Rekurentní architektura neuronových sítí.

Další kapitola Reinforcement learning popisuje svůj princip a pojmy.

Poslední kapitolou rešerše pojednává o konvoluční neuronových sítích.

Praktická část začíná datovým souborem, pokračuje posloupností přípravy dat, anotací a argumentací.

Následující část zahrnuje kompletní programovou základnu, konfiguraci a implementaci Frameworku Darknet na domácí stanici. Ze zde vysvětlených důvodů se od této implementace upustí.

Kapitola Cloud: Google Colaboratory je klíčovou kapitolou pro praktickou část. Jedná se o řešení umožňující implementaci YOLOv3 za použití frameworku Darknet v prostředí Jupyter Notebook na cloudové službě Google Colaboratory. Kapitola podrobněji popisuje prostředí Google Colab, porovnává jeho výkon a grafickou akceleraci.

Navazující kapitoly obsahují samotnou implementaci detektoru objektů YOLOv3 s frameworkem Darknet53. Následuje ověření detekcí objektů ze snímků i videí a ověření rychlosti detektoru na kvalitní a redukované verzi příslušného média.

Trénování modelu se zabývá primárně konfigurací konvolučních vrstev, konfiguračních souborů, přípravě dat, natrénováním modelu a odzkoušením jeho schopnosti a rychlosti rozpoznat naučené objekty v reálném čase.

V části výsledků a diskuse je vyhodnocení praktické části ve formě 3 zajímavých poznatků vycházejících ze splnění cílů.

Autorovou myšlenkou k účelu práce bylo získat schopnost prakticky využít neuronové sítě a dosáhnout v tomto oboru hlubšímu porozumění, protože ačkoliv téma umělé inteligence byl pro autora velký koníček, tak jej pro jeho rozsáhlost a komplexnost neovládal. V ideálním případě by autor rád pokračoval ve výzkumu nejrůznějších využití v běžné praxi a v dalším profesním životě.

Z tohoto hlediska autor hodnotí práci velmi optimisticky. Podařilo se nastudovat množství odborné literatury zabývající se tímto problémem, zpracovat rozsáhlou literární rešerši a zkusit si znalosti implementovat na praktickém příkladě.

Osobní radostí je, že se autorovi podařilo splnit předsevzetí ze závěru bakalářské práce:

> *(...) vypracovávání této práce autora skutečně bavilo, ať už prozkoumávání krás matematických vzorců a obrazců(...) či řešení algoritmických oříšků. Pravděpodobně to autora podnítí k dalšímu samostudiu těchto problematik.*
>
> *– (Hlaváček, 2017)*

Seznam použitých zdrojů

1. **Itzhak Fried, Ueli Rutishauser, Moran Cerf and Gabriel Kreiman. 2014.** *Single Neuron Studies of the Human Brain: Probing Cognition (The MIT Press).* místo neznámé : The MIT Press, 2014. ISBN-13: 978-0262027205.
2. **Abbasi, Ashkan. 2019.** Comparison of Fast Deep Learning based Object Detectors. *Medium.com.* [Online] University of Isfahan, 2019. https://medium.com/@ashkan.abbasi/comparison-of-fast-deep-learning-based-object-detectors-eeac9a0a0ebb.
3. **Adam Coates and Andrew Y. Ng. 2012.** *Learning Feature Representations with K-means.* Standford CA : Stanford University, 2012. [online] https://www-cs.stanford.edu/~acoates/papers/coatesng_nntot2012.pdf.
4. **Agrawal, Rachit Kumar. 2018.** Difference between Machine Learning, Deep Learning and Artificial Intelligence. *Medium.com.* [Online] Udacity India, 2018. https://medium.com/@UdacityINDIA/difference-between-machine-learning-deep-learning-and-artificial-intelligence-e9073d43a4c3.
5. **AlexeyAB, Joseph Redmon. 2016-2020.** Yolo-v3 and Yolo-v2 for Windows and Linux. *GitHub.com.* [Online] GitHub, Inc., 2016-2020. https://github.com/AlexeyAB/darknet.
6. **Anderson, James A. 1995.** *An Introduction to Neural Networks.* Cambridge, Massachusetts : The MIT Press, 1995. 612.8-dc20 94-30749CIP.
7. **Andrea Tettamanzi, Marco Tomassini. 2001.** *Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems.* místo neznámé : Springer, 2001. ISBN-13: 978-3540422044.
8. **Azevedo, F.A., Carvalho, L.R., Grinberg, L.T., Farfel, J.M., Ferretti, R.E., Leite, R.E., Filho, W.J., Lent, R. and Herculano-Houzel, S. 2009.** *Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain.* místo neznámé : Instituto de Ciências Biomédicas, Universidade Federal do Rio de Janeiro, 2009. 513: 532-541. doi:10.1002/cne.21974.
9. **Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, Dan Mané. 2016.** *Concrete Problems in AI Safety.* Standford, Berkley : Cornel University, Standford, Berkley, 2016. arXiv:1606.06565v2.

10. **Deshpande, Adit. 2016.** A Beginner's Guide To Understanding Convolutional Neural Networks Part 2. *ADeshpande3.github.io.* [Online] 2016. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.
11. **François, Chollet. 2019.** *Deep learning v jazyku Python.* 2019. ISBN: 978-80-247-3100-1.
12. **Gad, Ahmed. 2018.** Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?". *towardsthedatascience.com.* [Online] Towards The Data Science, 2018. https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e.
13. **Google LLC. 2019.** Learn Tensorflow 4: Convolutional Neural Networks (CNNs). *codelabs.developers.google.com.* [Online] Google LLC, 2019. https://codelabs.developers.google.com/codelabs/tensorflow-lab3-convolutions/#0.
14. **HAŠEK, Roman. 2016.** *Lineární Algebra.* 2016. [ONLINE] https://docplayer.cz/119735235-Linearni-algebra-kma-la-roman-hasek.html.
15. **Hlaváček, Vít. 2017.** *Programování s vlákny v jazyce C#.* Praha : autor neznámý, 2017.
16. **Chollet, Francois. 2019.** *Deep learning v jazyku Python - Knihovny Keras, TensorFlow.* Praha : Grada Publishing, 2019. ISBN 978-80-247-3100-1.
17. **Ian Goodfellow and Yoshua Bengio and Aaron Courville. 2016.** *Deep Learning.* Boston, MA : MIT Press, 2016. ISBN-13: 978-0262035613.
18. **Ibáñez, David. 2019.** How to train YOLOv3 using Darknet on Colab 12GB-RAM GPU notebook and speed up load times. *DEV-ibanyez.info.* [Online] 2019. http://blog.ibanyez.info/blogs/coding/20190410-run-a-google-colab-notebook-to-train-yolov3-using-darknet-in/.
19. **Jeffcock, Peter. 2017.** 4 Machine Learning Techniques You Should Recognize. *oracle.com.* [Online] Oracle, 2017. https://blogs.oracle.com/bigdata/machine-learning-techniques.
20. **—. 2018.** What's the Difference Between AI, Machine Learning, and Deep Learning? *oracle.com.* [Online] Oracle, 2018. https://blogs.oracle.com/bigdata/difference-ai-machine-learning-deep-learning.

21. **Jha, Vishakha. 2017.** Machine Learning Algorithm - Backbone of emerging technologies. *TechLeer Media.* [Online] 2017. https://www.techleer.com/articles/203-machine-learning-algorithm-backbone-of-emerging-technologies/.
22. **Kim, Buomsoo. 2020.** Google newly launches Colab Pro! - comparison of Colab and Colab pro. *https://buomsoo-kim.github.io/.* [Online] 2020. https://buomsoo-kim.github.io/colab/2020/03/15/Google-newly-launches-colab-pro.md/.
23. **Koehrsen, Will. 2018.** Overfitting vs. Underfitting: A Conceptual Explanation. *TowardsDataScience.com.* [Online] 2018. https://towardsdatascience.com/overfitting-vs-underfitting-a-conceptual-explanation-d94ee20ca7f9.
24. **Louppe, Gilles. 2019.** *Master thesis : Automatic Multispeaker Voice Cloning.* Liège, Belgium : Université de Liège, Liège, Belgique, 2019.
25. **MissingLink dev team. 2018.** Complete Guide to Deep Reinforcement Learning: Concepts, Process, and Real World Applications. *MissingLink.ai.* [Online] 2018. https://missinglink.ai/guides/neural-network-concepts/complete-guide-deep-reinforcement-learning-concepts-process-real-world-applications/.
26. **—. 2019.** Neural Network Bias: Bias Neuron, Overfitting and Underfitting. *MissingLink.ai.* [Online] 2019. https://missinglink.ai/guides/neural-network-concepts/neural-network-bias-bias-neuron-overfitting-underfitting/.
27. **—. 2018-2020.** Tensorflow Reinforcement Learning: Introduction and Hands-On Tutorial. *MissingLink.ai.* [Online] 2018-2020. https://missinglink.ai/guides/deep-learning-healthcare/tensorflow-reinforcement-learning-introduction-and-hands-on-tutorial/.
28. **—. 2018.** The Complete Guide to Artificial Neural Networks: Concepts and Models. *MissingLink.ai.* [Online] 2018. https://missinglink.ai/guides/neural-network-concepts/complete-guide-artificial-neural-networks/.
29. **Nakamoto, Pat. 2017.** *Neural Networks and Deep Learning: Deep Learning Explained to Your Granny; a Visual Introduction for Beginners Who Want to Make Their Own Deep Learning Neural Network.* místo neznámé : CreateSpace Independent Publishing Platform, 2017, 2017. ISBN 1981614060, 9781981614066.
30. **Olšák, Petr. 2007.** *Úvod do algebry, zejména lineární.* Praha : FEL ČVUT, 2007. ISBN 978-80-01-03775-1.

31. **OpenGenus Foundation. 2018.** https://iq.opengenus.org/cpu-vs-gpu-vs-tpu/. *iq.OpenGenus.org.* [Online] OpenGenus IQ, 2018. https://iq.opengenus.org/cpu-vs-gpu-vs-tpu/.
32. **Orr, Genevieve. 1999.** Neural networks - Momentum and Learning Rate Adaptation. *willamette.edu.* [Online] Willamette University, 1999. https://willamette.edu/~gorr/classes/cs449/momrate.html.
33. **Prajit Ramachandran, Barret Zoph, Quoc V. Le. 2017.** *Searching for Activation Functions.* Ithaca, New York : Cornell University, 2017. [online] https://arxiv.org/abs/1710.05941.
34. **Redmon, Joseph Chet. 2016.** https://pjreddie.com/darknet/. *pjreddie.com.* [Online] University of Washington & Allen Institute for Artificial Intelligence, 2016. https://pjreddie.com/darknet/install/.
35. **Shafer, Casey. 2016.** *The Neural Network, its Techniques and Applications.* Walla Walla, Washington : Whitman college, 2016. [Online] https://www.whitman.edu/Documents/Academics/Mathematics/2016/Schafer.pdf.
36. **Shaikh, Faizan. 2017.** Why are GPUs necessary for training Deep Learning models? *analyticsvidhya.com.* [Online] Analitics Vidhya, 2017. https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/ .
37. **Shaw, Andrew. 2018.** Vision transform. *Fast.AI.* [Online] Fast AI, 2018. https://docs.fast.ai/vision.transform.html#Data-augmentation .
38. **Steinke, Steven. 2017.** What's the difference between a matrix and a tensor? *Medium.com.* [Online] 2017. https://medium.com/@quantumsteinke/whats-the-difference-between-a-matrix-and-a-tensor-4505fbdc576c.
39. **Šlégr, Jan. 2012.** *Tenzory a tenzorový počet.* Hradec Králové : Katedra fyziky PřF UHK, 2012. [Online] https://lide.uhk.cz/prf/ucitel/slegrja1/tenzory.pdf.
40. **The AI Guy. 2020.** YOLOv3-Cloud-Tutorial. *https://github.com/theAIGuysCode.* [Online] The AI Guys Code, 2020. https://github.com/theAIGuysCode/YOLOv3-Cloud-Tutorial.
41. **Van Veen, F. & Leijnen, S. 2019.** The Neural Network Zoo. *AsimovInstitute.org.* [Online] The Asimov Institue, 2019. https://www.asimovinstitute.org/neural-network-zoo/.

42. **Weng, Lilian. 2018.** A (Long) Peek into Reinforcement Learning. *LilianWeng.github.io.* [Online] 2018. https://lilianweng.github.io/lil-log/2018/02/19/a-long-peek-into-reinforcement-learning.html#what-is-reinforcement-learning.
43. **Wilson, Bill. 2012.** The Machine Learning Dictionary. *cse.unsw.edu.au.* [Online] University of New South Wales, Sydney, 2012. http://www.cse.unsw.edu.au/~billw/mldict.html.