Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra Informačních Technologií

Aplikace konvolučních neuronových sítí pro autonomní vozidla

Diplomová práce

Autor práce: Bc. Dennis Tschamler
Studijní obor: Aplikovaná Informatika

Vedoucí práce: prof. RNDr. PhDr. Antonín Slabý, CSc.

Hradec Králové 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury

.....

Dennis Tschamler

24. dubna 2022

Poděkování

Děkuji vedoucímu diplomové práce prof. RNDr. PhDr. Antonín Slabý, CSc. za metodické vedení práce, trpělivost, cenné rady a vstřícnost při konzultacích.

Anotace

Diplomová práce se zabývá problematikou konvolučních neuronových sítí a jejich použití v autonomních vozidlech. V práci je na začátku popsáno fungování umělé neuronové sítě, nejpoužívanější aktivační funkce, ztrátové funkce a optimalizátory. Dále je popsána architektura konvoluční neuronové sítě, představeno několik existujících modelů a zmíněny některé klasifikační algoritmy. Poté následuje popis implementace klasifikace dopravních značek pomocí knihovny PyTorch a frameworku PyTorch Lightning. V závěru jsou provedeny experimenty s modely ResNet50, VGG16 a AlexNet v klasifikaci dopravních značek. Každý model je natrénován v rámci čtyř experimentů. Následně jsou výsledky porovnány a zvolen nejlepší model pro tuto úlohu. Součástí experimentů jsou i vizualizace konvolučních filtrů a výsledných 2D aktivačních map.

Anotation

Title: Application of convolutional neural networks for autonomous vehicles

The diploma thesis deals with the issue of convolutional neural networks and their use in autonomous vehicles. The work initially describes the operation of an artificial neural network, the most commonly used activation functions, loss functions and optimizers. Then the architecture of the convolutional neural network is described, several existing models and classification algorithms are introduced. This is followed by a description of the implementation of traffic sign classification using the PyTorch library and the PyTorch Lightning framework. Finally, experiments are performed with models ResNet50, VGG16 and AlexNet in the classification of traffic signs. Each model is trained in four experiments. Subsequently, the results are compared and the best model for this task is chosen. The experiments also include visualizations of convolutional filters and the resulting 2D activation maps.

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

1 Úvod

Konvoluční neuronové sítě mění současný svět v mnoha oblastech. Velký dopad má na odvětví autonomních vozidel, kde se vývoj velice posunul kupředu. Přesto zatím nejsou k dispozici vozidla, která by byla schopná řídit v běžném provozu zcela bez zásahu řidiče. The Society of Automotive Engineers definuje 6 úrovní (0 až 5) autonomního řízení. Každá země si určuje svá pravidla pro akceptaci vozidla pro danou úroveň. V současné době se podařilo dosáhnout v USA úrovně 2 (Audi A8L — dodáván bez hardwaru a softwaru potřebného pro úroveň 3) a v Německu úrovně 3 (Audi A8L).

Samotné téma bylo vybráno z důvodu jeho široké použitelnosti a jeho dopadem na budoucí vývoj v mnoha oblastech. Konvoluční neuronové sítě jsou velice zajímavé z hlediska jejich široké využitelnosti napříč mnoha obory. Pomocí předpřipravených dat je lze relativně rychle naučit rozpoznávat text, mluvenou řeč nebo objekty na obrázcích. V případě autonomních vozidel lze pomocí nich řešit úkony rozpoznávání dopravních čar, detekce vozidel a chodců, rozpoznávání semaforů a dopravních značek. Na základě takto získaných dat je pak důležité, aby autonomní vozidlo bylo schopné provádět správná rozhodnutí. Omezením lidského faktoru se zásadně sníží nehodovost na silnicích.

V první kapitole je podrobně popsána umělá neuronová síť. Další část se věnuje konvolučním neuronovým sítím, zejména její architektuře. Také jsou představeny některé známé modely, klasifikační algoritmy a závěrem popsán proces vytvoření modelu pro klasifikaci obrazu. Poté je nastíněno využití konvolučních neuronových sítí pro autonomní vozidla. V posledních dvou kapitolách je představena knihovna PyTorch a implementace modelu pro klasifikaci dopravních značek.

Cílem práce bude představit problematiku konvolučních neuronových sítí a jejich využití v autonomních vozidlech. Na závěr bude implementováno trénování a testování tří modelů pomocí programovacího jazyka Python a knihovny PyTorch. Pro každý model budou provedeny čtyři experimenty a následně bude vybrán nejlepší z nich.

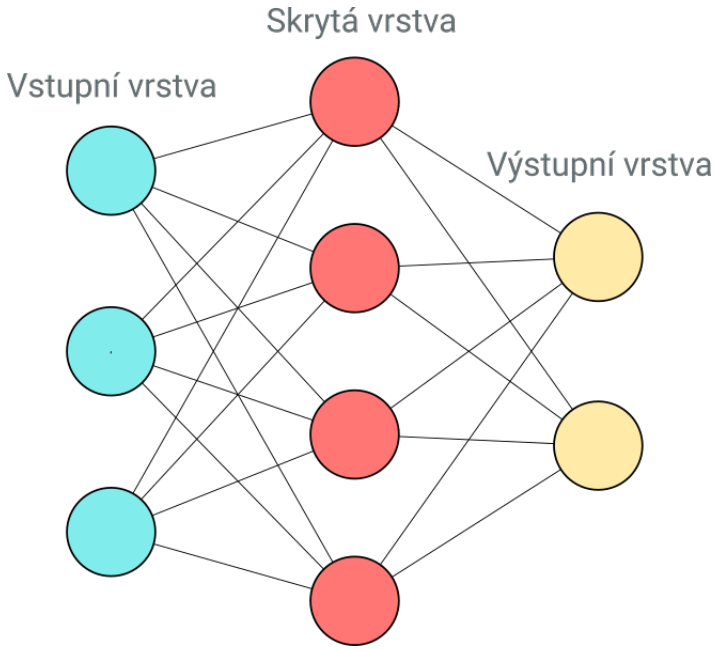
2 Cíl práce

Cílem práce je uvést čtenáře do problematiky konvolučních neuronových sítí a jejího využití v oblasti autonomních vozidel. V první části bude popsána umělá neuronová síť a konvoluční neuronová síť. Druhá část bude zaměřená na aplikaci konvolučních neuronových sítí pro autonomní vozidla. Následně bude představena knihovna PyTorch a pomocí ní bude implementováno trénování sítě pomocí existujících modelů. Budou zvoleny tří modely a každý model bude trénován v rámci čtyr experimentů. V závěru budou tyto experimenty mezi sebou porovnány a vyhodnoceny.

3 Umělé neuronové sítě

Umělá neuronová síť se svým fungováním snaží přiblížit fungování lidského mozku. Skládá se z navzájem propojených neuronů, které tvoří neuronovou síť. Hlavním cílem sítě je naučit se správně predikovat výsledek na základě předešlých zkušeností. Tedy postupným učením zlepšovat přesnost své predikce. Každý neuron na základě svého vstupu produkuje výstup.

Neuron může mít několik vstupů a jeden výstup, který lze dále odeslat do ostatních neuronů. K odeslání svého výstupu dochází v závislosti na výsledku aktivační funkce, která rozhodne, jestli je výstup neuronu přínosný (viz kapitola 3.2). Neurony lze sdružovat do vrstev — mezi základní patří vstupní (input), skrytá (hidden) a výstupní (output) viz obrázek 1. [1]



Obrázek 1: Umělá neuronová síť navzájem propojených neuronů. Každý kruh reprezentuje neuron a čára symbolizuje propojení výstupu jednoho neuronu na vstup druhého neuronu (zleva doprava). Neurony jsou podle barev rozděleny do jednotlivých vrstev

Zdroj: podle [1]

3.1 Neuron

Úkolem neuronu je převzít data na vstupu, vypočítat výstupní hodnotu a odeslat ji na výstup. Na vstupu může neuron dostat data v různé podobě — obrázek, dokument, zvuk nebo výstup z jiného neuronu. Spojení mezi neurony má tzv. váhu (značí se w), která se během procesu učení (viz kapitola 3.3) mění a optimalizuje pro optimální přesnost predikce. Pro výpočet výstupní hodnoty neuronu je potřeba rovnice 1, kde symboly mají následující význam:

- x — vstupní hodnota
- w — váha vstupní hodnoty

- Θ — bias — posun funkce o konstantu
- S — aktivační funkce (viz kapitola 3.2)

$$Y = S\left(\sum_{i=1}^{N}(w_i x_i) + \Theta\right) \tag{1}$$

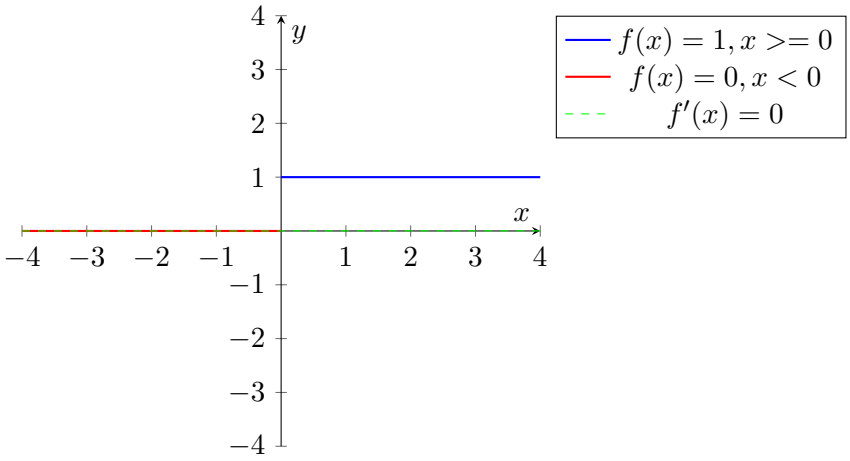
Tedy nejprve se vypočte vážený součet všech vstupních hodnot (pomocí váh w). Poté se k této hodnotě přičte *bias*, který hodnotu posune doleva/doprava. Výsledná hodnota je jako parametr použita v aktivační funkci. Na základě výsledku aktivační funkce se rozhodne, jestli bude neuron aktivován tzn. je důležitý pro proces predikce správného výstupu. Tento výpočet probíhá v dopředné propagaci procesu učení neuronové sítě (viz kapitola 3.3.2). V následující kapitole budou představeny nejčastěji používané aktivační funkce.

3.2 Aktivační funkce

Aktivační funkce je finálním krokem při výpočtu výstupu neuronu. Na základě její hodnoty se rozhodne, jestli se odešle dále na vstup do dalších propojených neuronů. Použití aktivační funkce v neuronové síti je velice důležité, protože se do modelu dostane nelinearita. Díky tomu se může síť učit a rozpoznávat komplexní vazby v datech. Jedním z hlavních požadavků na aktivační funkci je její diferencovatelnost, která je důležitou podmínkou pro zpětnou propagaci chyby a optimalizování hyperparametrů pomocí optimalizátorů (viz kapitola 3.3.3 a 3.5). Na další straně budou představeny základní aktivační funkce, které se běžně používají. U každé funkce je modrou (nebo i červenou) barvou vyznačen graf funkce a zelenou barvou derivace funkce. [2]

3.2.1 Binary Step Function

Je to jedna z nejjednodušších aktivačních funkcí — lze ji implementovat pomocí konstrukce if-else. Většinou se používá pro binární klasifikaci — nelze ji použít pro klasifikaci více tříd. První derivace se rovná nule, tedy může dojít k problému při učení v kroku zpětné propagace (pro jakýkoliv vstup se výsledek derivované funkce nemění). [2]

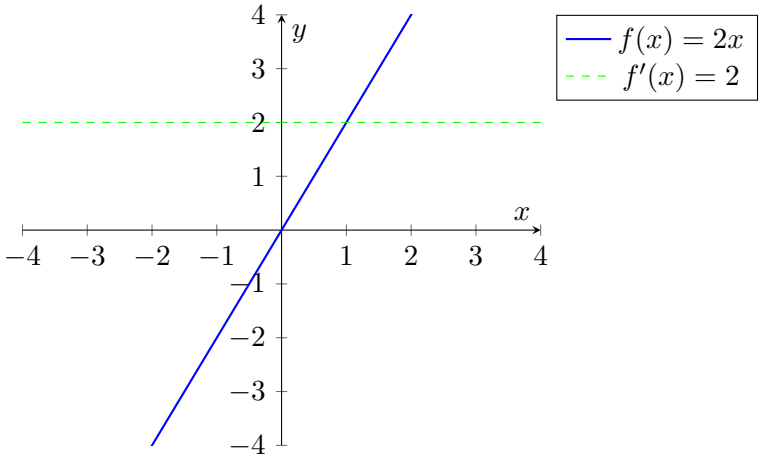


Graf 1: Aktivační funkce binary step function

Zdroj: vlastní zpracování

3.2.2 Linear

Lineární aktivační funkce je přímo úměrná jejímu vstupu. Gradient v tomto případě lze ovlivnit použitím parametru a: $F(x) = ax$. Díky tomu se budou váhy a biasy aktualizovat v kroku zpětné propagace, ale kvůli konstantní derivaci pořád stejně. Proto je lineární funkce vhodná pro jednodušší úlohy. [2]

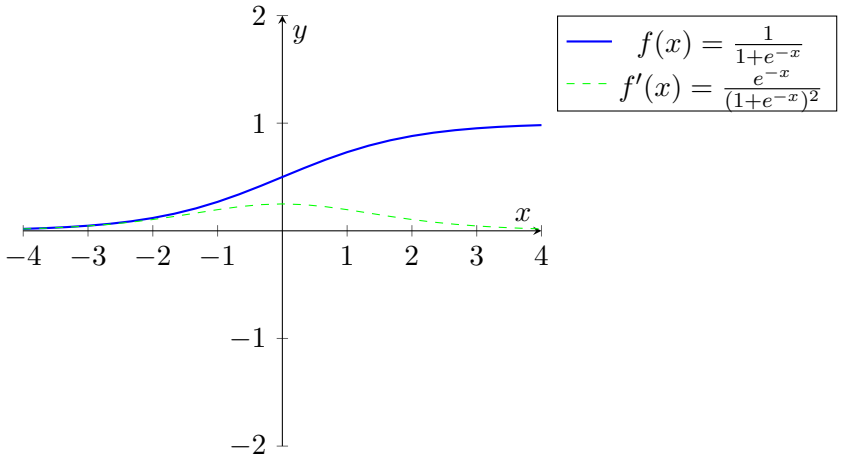


Graf 2: Aktivační funkce linear

Zdroj: vlastní zpracování

3.2.3 Sigmoid

Sigmoid je jedna z nejpoužívanějších aktivačních funkcí. Všechny vstupní hodnoty transformuje do intervalu od 0 do 1, tedy je velice vhodná pro predikci pravděpodobnosti. Používá se zejména pro klasifikaci obrazu. [2]

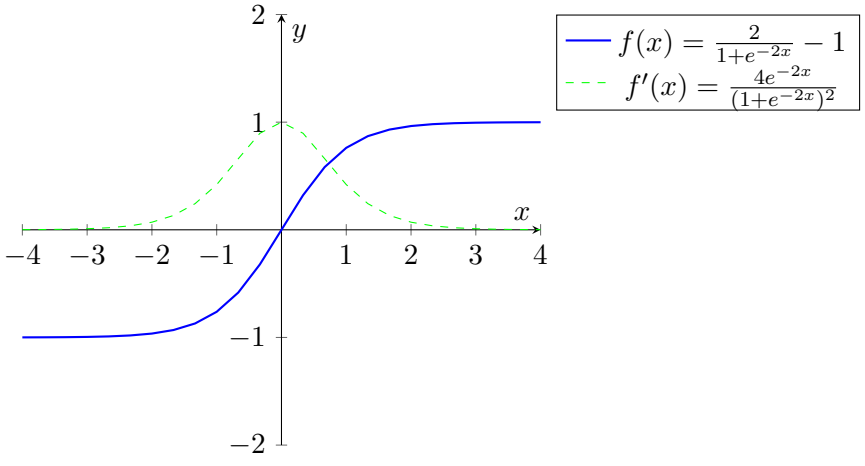


Graf 3: Aktivační funkce sigmoid

Zdroj: vlastní zpracování

3.2.4 Tanh

Je to hyperbolická tangent funkce, která je podobná sigmoid funkci. Hlavním rozdílem je její symetrie kolem 0 (může docházet ke změně znamínek mezi vrstvami). Rozsah hodnot se pohybuje od −1 do 1. [2]

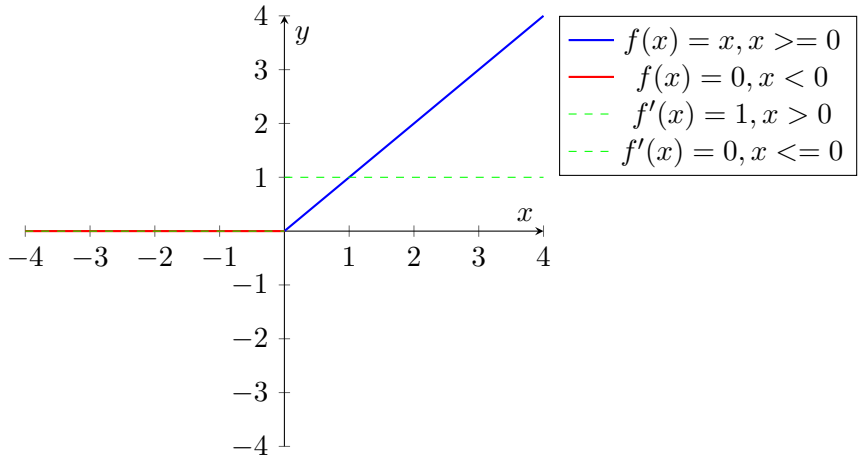


Graf 4: Aktivační funkce tanh

Zdroj: vlastní zpracování

3.2.5 ReLU (Relative Linear Unit)

Opět velice často používaná aktivační funkce. Výhodou je, že ne všechny neurony jsou aktivovány ve stejný čas. Tedy pokud je výsledek aktivační funkce roven 0, tak je neuron deaktivován. Nevýhodou je, že v některých případech je gradient roven 0 (váhy a biasy nejsou aktualizovány během zpětné propagace). Může dojít k tzv. umření neuronu, kdy se z tohoto stavu již nelze dostat. [2]

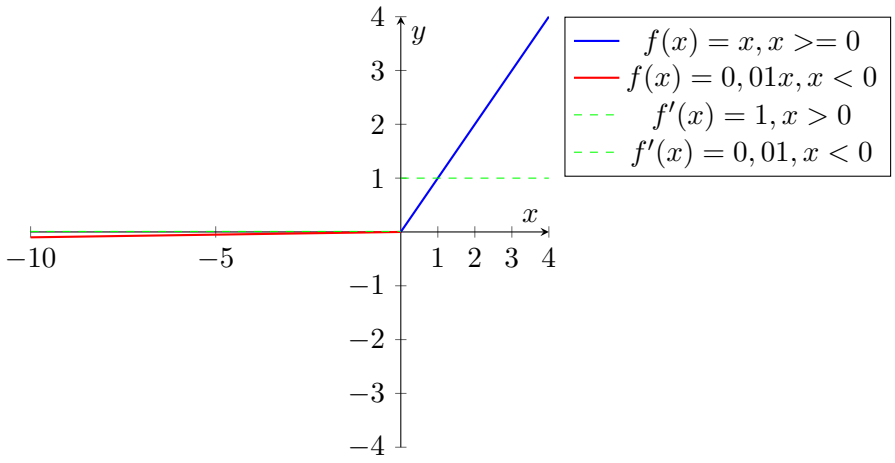


Graf 5: Aktivační funkce ReLU

Zdroj: vlastní zpracování

3.2.6 Leaky ReLU

Oproti klasické ReLU funkci se zde negativní vstupní hodnoty nerovnají 0, ale nabývají nízkých negativních hodnot. Díky tomu částečně řeší problém umření neuronu. [2]

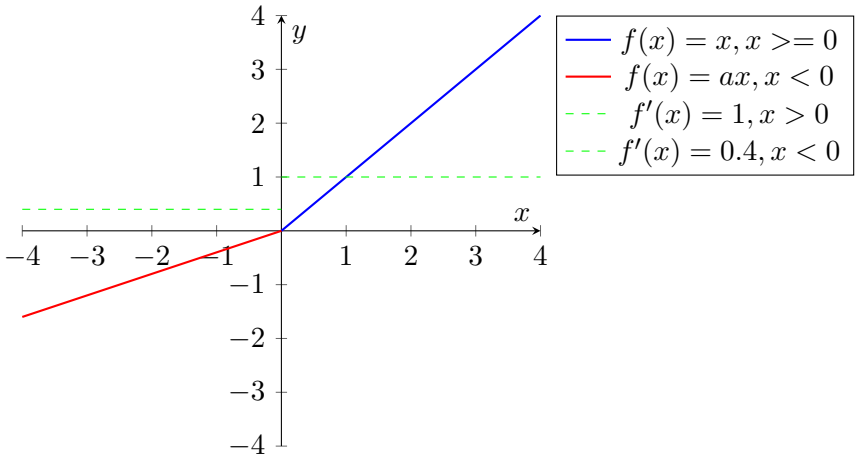


Graf 6: Aktivační funkce leaky ReLU

Zdroj: vlastní zpracování

3.2.7 Parametrized ReLU

Další varianta ReLU funkce, kde negativní vstupní hodnoty jsou transformovány dle lineární funkce s parametrem a. Tento parametr může být zadán nebo jej lze nechat neuronovou síť naučit (viz kapitola 3.3.1). [2]

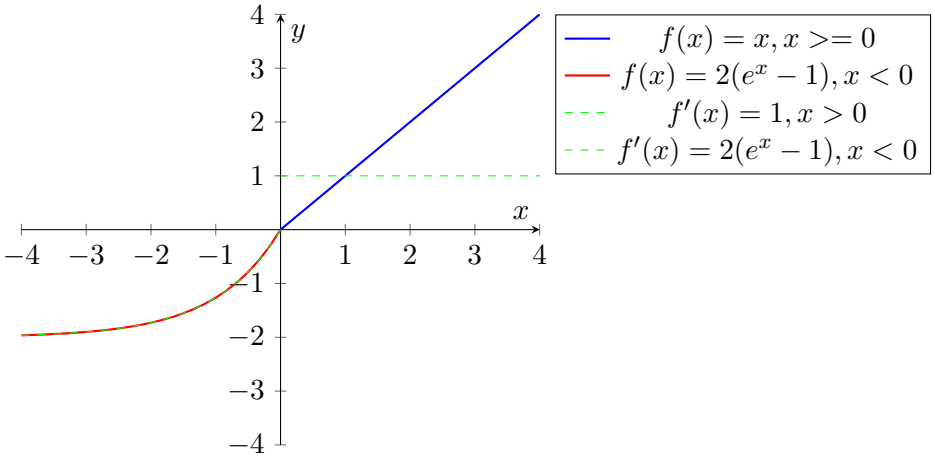


Graf 7: Aktivační funkce parametrized ReLU

Zdroj: vlastní zpracování

3.2.8 ELU (Exponential Linear Unit)

Exponenciální varianta ReLU funkce, která pro negativní vstupní hodnoty používá logaritmickou křivku. Podobně jako parametrizovaná ReLU nemá problém s umíráním neuronů. [2]

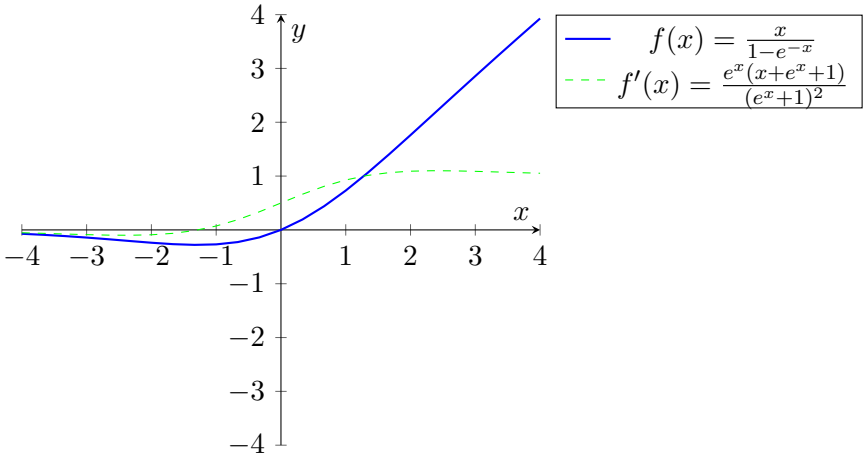


Graf 8: Aktivační funkce ELU

Zdroj: vlastní zpracování

3.2.9 Swish

Poměrně nová aktivační funkce, kterou objevili výzkumníci ze společnosti *Google*. Není to monotónní funkce, tedy výstupní hodnota se může snížit i přesto, že vstupní hodnota roste. [2]



Graf 9: Aktivační funkce swish

Zdroj: vlastní zpracování

3.2.10 Softmax

Softmax funkce je kombinací několika sigmoid funkcí. Na rozdíl od sigmoid funkce ji lze použít pro klasifikaci více tříd. Při sestavení neuronové sítě bude mít výstupní vrstva stejný počet neuronů jako je počet tříd, které se budou klasifikovat. [2]

3.3 Proces učení (trénování)

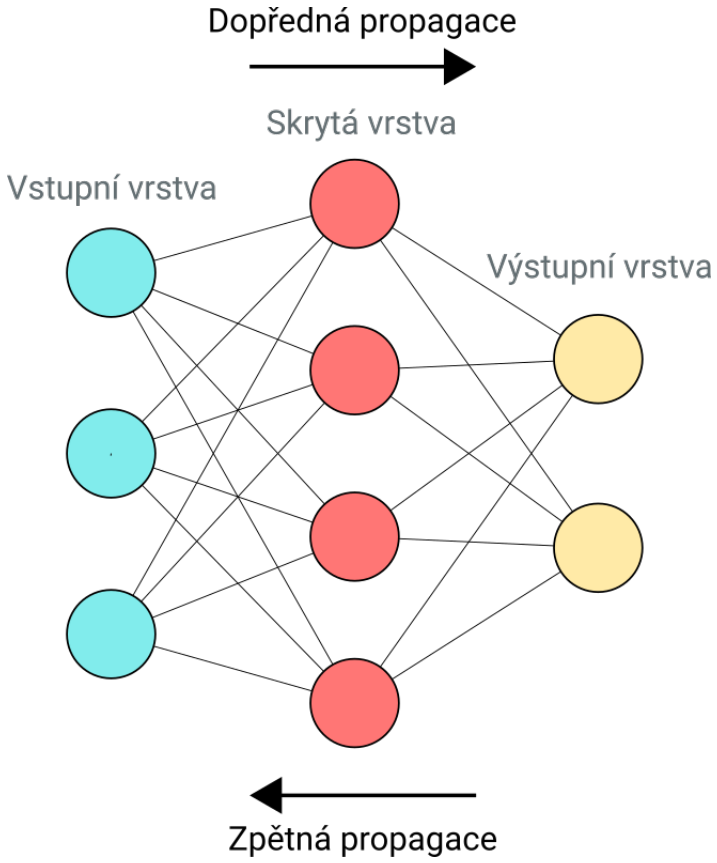
V předchozích kapitolách byl vysvětlen neuron a jak funguje jeho aktivace. Bylo představeno několik základních aktivačních funkcí. Nyní je možné popsat, jak se neuronová síť naučí predikovat správné výsledky.

Proces učení lze rozdělit na dva kroky — dopřednou propagaci (forward propagation) a zpětnou propagaci (backward propagation) viz obrázek 2. Než se začne neuronová síť učit, je nejprve potřeba inicializovat parametry modelu (váhy, bias) na náhodné hodnoty.

V prvním kroku se během dopředné propagace posílají data do vstupní vrstvy. U každého neuronu se provede kalkulace z rovnice 1 a výsledek se použije při výpočtu v následujícím propojeném neuronu. Takhle se postupně výsledky propagují až do výstupní vrstvy, kde se porovnají s očekávaným výsledkem. Výpočet rozdílu mezi predikovaným a očekávaným výsledkem zajišťuje ztrátová funkce.

Poté lze přejít k druhému kroku učení — zpětné propagaci. V tomto kroku je potřeba optimalizátor, který na základě hodnoty ztrátové funkce upraví během zpětné propagace parametry modelu (váhy, bias, atd.). Tyto kroky se opakují pro všechna data a několik epoch, dokud se nám už dále nezlepšuje přesnost predikce. V tento moment je možné ukončit učení.

V této části budou popsány podrobně jednotlivé kroky procesu učení. Před zahájením samotného učení je nutné nastavit tzv. hyperparametry, které jsou během učení neměnné.



Obrázek 2: Dopředná a zpětná propagace v neuronové síti

Zdroj: podle [1]

Ty mohou zásadně ovlivnit průběh učení, proto je jejich volba zásadní. Mezi hyperparametry patří například rychlost učení, velikost dávky, počet epoch a počet skrytých vrstev (v závislosti na použitém modelu sítě, ztrátové funkci, optimalizátoru a dalších proměnných).

3.3.1 Hyperparametry

Rychlost učení je jeden z nejdůležitějších hyperparametrů, který zajišťuje změnu vah modelu o danou hodnotu. Pokud se nastaví na nízkou hodnotu, může učení trvat dlouhou dobu nebo se zcela zaseknout. Naopak pokud se nastaví na vysokou hodnotu, můžou se váhy postupně nastavit na ne zcela optimální hodnoty a celkově způsobit špatné trénování. Hodnota se nastavuje v rozmezí 0.0 a 1.0 — zpravidla se nastavuje na hodnotu *0.001*.

Učení lze rozdělit do jednotlivých fázích tzv. epoch. Během jedné epochy celý náš dataset projde jednou krokem dopředné a zpětné propagace. Počet potřebných epoch na natrénování neuronové sítě nelze dopředu určit. Vždy záleží na řešeném problému. Dataset lze rozdělit na více částí — dávek. Počet dávek, které se zpracují během jedné epochy, se nazývá iterace.

3.3.2 Dopředná propagace

Během dopředné propagace se provádějí kalkulace ve směru od vstupní vrstvy k výstupní vrstvě. U každého neuronu je potřeba určit jeho hodnotu pomocí rovnice 1. Pokud se neuron

aktivuje, je tato hodnota dále posílána ve směru k výstupní vrstvě. Na konci dopředné propagace se musí vyhodnotit správnost predikcí pomocí ztrátové funkce.

Ztrátová funkce vyjadřuje rozdíl mezi predikovaným a očekávaným výstupem. Se ztrátovou funkcí úzce souvisí optimalizátory (viz kapitola 3.5), které se snaží upravit parametry modelu tak, aby se ztrátová funkce co nejvíce minimalizovala. Výběr ztrátové funkce závisí na řešeném problému:

- Regrese
  - Střední kvadratická chyba (mean squared error)
  - Huberova ztráta (Huber loss)
    - kombinace střední kvadratické chyby a střední absolutní chyby
    - méně citlivé na odlehlé hodnoty oproti střední kvadratické chybě
    - parametr δ určuje, jaká bude vypočtená chyba
      - menší než δ — kvadratická chyba
      - větší než δ — absolutní chyba
- Binární klasifikace
  - Binární křížová entropie (Binary cross-entropy)
  - Hinge loss — primárně se používá společně s metodou podpůrných vektorů
    - predikuje hodnoty v rozmezí -1 a +1
      - pokud je hodnota pozitivní, tak je zařazena do třídy +1
      - pokud je hodnota negativní, tak je zařazena do třídy -1
    - při použití s metodou podpůrných vektorů se při kalkulaci počítá s marginem — Hinge loss poté penalizuje špatné i správné predikce
      - skutečná hodnota je +1, ale predikovaná je -0.3 — dochází k velké penalizaci
      - skutečná hodnota je +1, margin je 0.2 a predikovaná hodnota je 0.15 — sice je predikce správná, ale je pod hodnotou marginu, tedy dochází k menší penalizaci
- Vícetřídní klasifikace
  - Vícetřídní křížová entropie (Multi-Class cross-entropy)
  - Kullbackova–Leiblerova divergence
    - určuje, jak moc se distribuční funkce P odlišuje od distribuční funkce Q
    - distribuce P a Q se rovnají skoro všude, když je divergence rovna 0
    - jinak je divergence kladná

Hodnota ztrátové funkce bude velice důležitá při zpětné propagaci.

3.3.3 Zpětná propagace

Ve zpětné propagaci budou probíhat kalkulace v opačném směru — tedy od výstupní vrstvy ke vstupní vrstvě. Během zpětné propagace dochází k ladění vah a biasu na základě vypočtené ztrátové funkce z předchozí epochy. Cílem je taková optimalizace parametrů, aby

se co nejvíce minimalizovala hodnota ztrátové funkce. K tomu nám pomůže optimalizátor (viz kapitola 3.5), který se pomocí první derivace rozhoduje, jak upraví váhy a bias. Tyto kalkulace se postupně propagují až ke vstupní vrstvě. Po tomto kroku lze opět začít nový cyklus učení (novou epochu) a zjistit, jak se zlepšila/zhoršila predikce.

3.4 Metody učení

Výběr metody učení pro neuronovou síť závisí na řešeném problému a vstupních datech. Metody učení lze rozdělit na tři základní metody — učení s učitelem, učení bez učitele a zpětnovazební učení . [3]

3.4.1 Učení s učitelem (Supervised learning)

Pro učení s učitelem je potřeba mít definovaný označený dataset, na kterém se může naučit neuronová síť predikovat správné výsledky a poté je aplikovat na neznámých datech. Přesnost predikce se určuje pomocí ztrátové funkce, kterou se síť snaží co nejvíce minimalizovat. Učení s učitelem se dále dělí na klasifikaci a regresi. Klasifikace rozděluje vstupní data do dvou nebo více tříd. Regrese odhaduje číselnou hodnotu výstupu v závislosti na velikosti vstupu. [3]

Během procesu klasifikace dokáže neuronová síť zařadit testovací data do konkrétních kategorií (viz kapitola 4.3). Typickým příkladem klasifikace je rozpoznávání číslic, kde na vstupu je dataset s obrázky číslic od 0 do 9 (ideálně rovnoměrně zastoupené). Po dokončeném učení stačí takové síti předložit obrázek číslice a ta ho dokáže s určitou přesností správně zařadit. Mezi klasifikační algoritmy (rozhodují, jak se mají data rozřadit do kategorií) patří lineární klasifikátory, metoda podpůrných vektorů, rozhodovací stromy a metoda nejbližších sousedů (viz kapitola 4.3).

Regrese dokáže porozumět vztahům mezi závislou proměnnou a jednou nebo několika nezávislými proměnnými. Nejčastěji se používá pro predikci dat (predikce závislé proměnné na základě jedné nebo několika nezávislých proměnných). Mezi regresní algoritmy patří lineární regrese, logistická regrese a polynomiální regrese.

3.4.2 Učení bez učitele (Unsupervised learning)

Tyto sítě převezmou neoznačené vstupy a samy se snaží najít v datech souvislosti a vzory, podle kterých data roztřídí do jednotlivých kategorií. Učení bez učitele lze dále rozdělit na shlukovou analýzu, asociační pravidla a zmenšení rozměrů. [3]

Shluková analýza (clustering) dokáže seskupit neoznačená data do skupin (shluků) na základě jejich podobných/rozdílných vlastností. Mezi shlukovací algoritmy patří hierarchické shlukování, shlukování metodou nejbližších středů a pravděpodobností shlukování.

Pomocí asociačních pravidel se dají hledat vztahy mezi proměnnými. Zejména se používá při analýze spotřebního koše (market basket analysis), kdy lze lépe pochopit vztahy mezi jednotlivými produkty (např.: jaké produkty se většinou kupují společně). Mezi nejpoužívanější algoritmus patří apriori.

3.4.3 Zpětnovazební učení (Reinforcement learning)

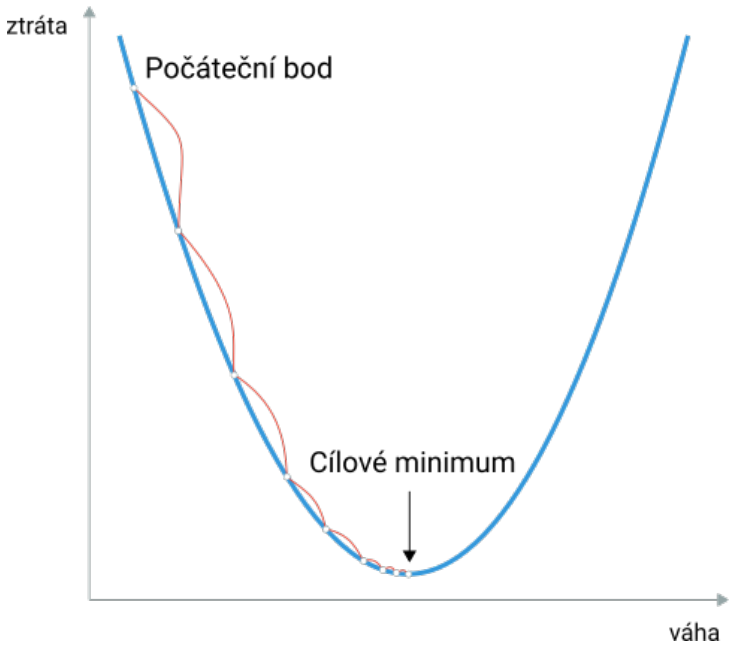
Zpětnovazební učení funguje na principu zpětné vazby. Během učícího procesu dostává neuronová síť pozitivní a negativní zpětnou vazbu, na základě které upravuje své učení. Základním cílem je tedy maximalizace pozitivní zpětné vazby. [3]

3.5 Optimalizátory

Optimalizátory jsou důležitou součástí modelu v kroku zpětné propagace (viz kapitola 3.3.3). Během zpětné propagace má za úkol pomocí hodnoty ztrátové funkce upravit hyperparametry a váhy tak, aby se ztrátová funkce co nejvíce minimalizovala. Tedy snaží se tyto parametry optimalizovat. Její výběr je jeden z rozhodujících faktorů rychlosti učení neuronové sítě a přesnosti finální predikce modelu. Níže budou představeny některé optimalizační algoritmy. [4]

3.5.1 Gradientní sestup

Iterativní optimalizační algoritmus, který se používá pro nalezení lokálního minima/maxima dané funkce (viz obrázek 3). V případě strojového učení a hlubokého učení se používá pro hledání minima. Většina ostatních optimalizátorů vychází právě z gradientního sestupu. Abychom tento algoritmus mohli použít, musí být daná funkce diferencovatelná a konvexní. [5]



Obrázek 3: Ukázka algoritmu gradientního sestupu

Zdroj: podle [6]

Celý algoritmus lze popsat v několika krocích:

1. inicializace startovacího bodu
2. výpočet gradientu v tomto bodě (1. derivace funkce — funkce musí být diferencovatelná)
3. podle vypočteného gradientu se posune v opačném směru (tento posun se škáluje pomocí nastavené rychlosti učení viz kapitola 3.3.1)
4. opakování bodu 2 a 3, dokud se nesplní alespoň jedna z následujících podmínek:

- byl dosažen nám daný maximální počet iterací
- vypočtený posun je menší než námi daná tolerance

3.5.2 Stochastický gradientní sestup (Stochastic gradient descent)

Stochastický gradientní sestup je vylepšením gradientního sestupu. Funguje na obdobném principu, ale nepočítá se pro všechna data. Místo toho se při každé iteraci vybere náhodná dávka dat (podmnožina), která se použije pro výpočet. Díky tomu je výpočet mnohem rychlejší (je potřeba méně iterací). Rychlost učení se definuje na začátku a už se v průběhu algoritmu nemění. [7]

$$w = w - \eta \nabla Q(w) \tag{2}$$

Pro výpočet se používá rovnice 2, kde:

- w — váhy
- η — rychlost učeni
- ∇ — gradient
- Q — ztrátová funkce

3.5.3 Optimalizátor Momentum

Další nadstavbou nad gradientním sestupem je optimalizátor momentum (gradientní sestup s momentem), který funguje na principu momenta (hybnosti). Algoritmus Momentum je rychlejší oproti gradientnímu sestupu díky tomu, že pracuje s historií předešlých kroků a využívá momentum. Pokud se pohybuje stále ve stejném směru, postupně zvětšuje kroky (získává momentum). [8] Jako hyperparametr lze v rozmezí 0,0 a 1,0 nastavit váhu předešlého kroku.

$$w = w - \eta \nabla Q(w) + \gamma v_t \tag{3}$$

Pro výpočet se používá rovnice 3, kde:

- w — váhy
- η — rychlost učení
- ∇ — gradient
- Q — ztrátová funkce
- γ — momentum
- v_t — poslední změna, která byla provedena u vah

3.5.4 Optimalizátor Adam (Adaptive moment estimation)

Optimalizátor Adam je rozšířením stochastického gradientního sestupu. Rozdílem je, že každá váha v sítí má přidělenou svou rychlost učení (learning rate) a ta je postupně zvlášť adaptována v průběhu učení. Jednou z hlavních výhod tohoto optimalizátoru oproti ostatním je jeho rychlost. [9]

Chování optimalizátoru Adam lze nastavit těmito vstupními proměnnými:

- α — rychlost učení
- β_1 — rychlost exponenciálního rozpadu pro
- β_2 — rychlost exponenciálního rozpadu pro
- ϵ — velmi nízké číslo, které zabrání dělení nulou v implementaci

Autoři doporučují následující nastavení: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ a $\epsilon = 10^{-8}$.

3.5.5 Optimalizátor AdaGrad (Adaptive gradient)

Algoritmus AdaGrad je dalším vylepšením gradientního sestupu. AdaGrad adaptuje rychlost učení pro každý parametr. Čím častěji se parametr aktualizuje, tím se jeho hodnota upravuje menší hodnotou. Díky tomu dokáže rychleji nalézt optimální parametry. Hlavní nevýhodou je neustále se zmenšující rychlost učení díky akumulaci hodnot v sumě. [10]

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{\epsilon + \sum_{T=1}^{t}(\nabla Q(w_{T,i}))^2}} \nabla Q(w_{t,i}) \tag{4}$$

Pro výpočet se používá rovnice 4, kde:

- w — váhy
- η — rychlost učení (doporučuje se hodnota 0,01)
- ∇ — gradient
- ϵ — velmi nízké číslo (zamezení dělení nulou)
- Q — ztrátová funkce
- v_t — poslední změna, která byla provedena u vah

3.5.6 Optimalizátor RMSprop

Funguje podobně jako optimalizátor AdaGrad, ale místo sumy všech gradientů do času t, používá exponenciálně klesající průměr. Byl vytvořen jako vylepšení optimalizátoru AdaGrad a řeší problém postupně zmenšující se rychlosti učení. [5]

$$\begin{gathered} w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{\epsilon + E[g^2]_t)}} \nabla Q(w_{t,i}) \\ E[g^2]_t = (1-\gamma)g^2 + \gamma E[g^2]_{t-1} \\ g = \nabla Q(w_{t,i}) \end{gathered} \tag{5}$$

Pro výpočet se používá rovnice 5, kde:

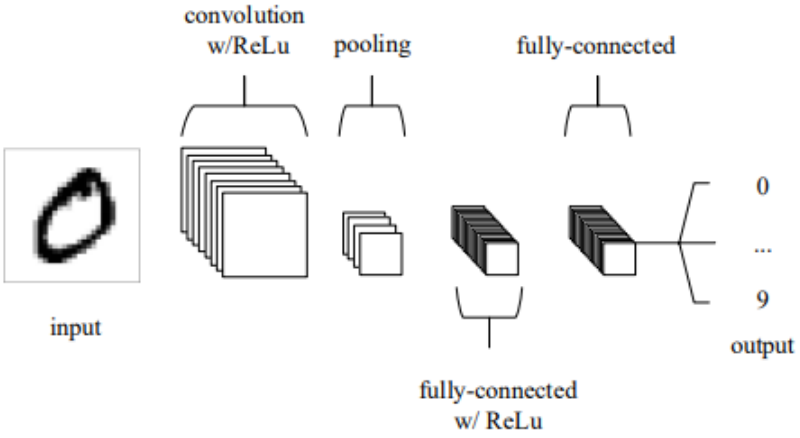
- w — váhy
- η — rychlost učení (doporučuje se hodnota 0,001)
- ∇ — gradient
- ϵ — velmi nízké číslo (zamezení dělení nulou)
- Q — ztrátová funkce
- γ — momentum (doporučuje se hodnota 0,9)

4 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou speciálním případem neuronových sítí. Hlavním rozdílem jsou konvoluční vrstvy, které jsou vhodné pro klasifikaci obrazu (viz kapitola 4.3). [11]

4.1 Architektura

Základním stavebním kamenem každé konvoluční neuronové sítě jsou jednotlivé neurony, kterou jsou seskupeny do vrstev. Ukázka takové sítě je vidět na obrázku 4. Tato síť se skládá ze vstupní, konvoluční, pooling a dvou plně propojených vrstev.

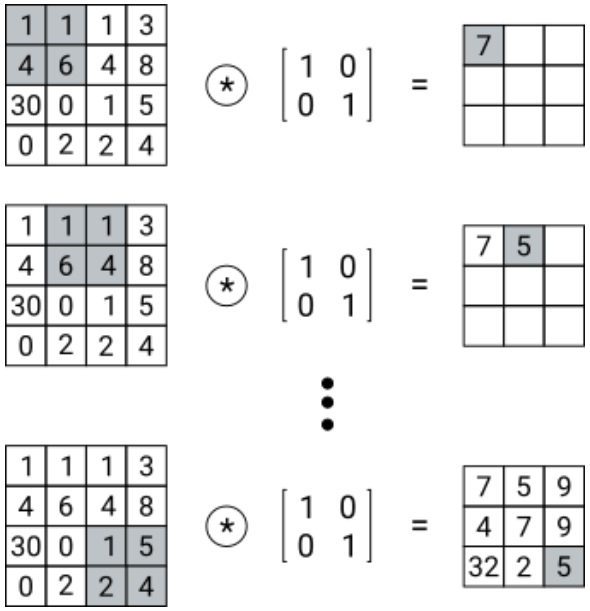


Obrázek 4: Jednoduchá konvoluční neuronová síť, která se skládá z pěti vrstev

Zdroj: převzato z [11]

4.1.1 Konvoluční vrstva

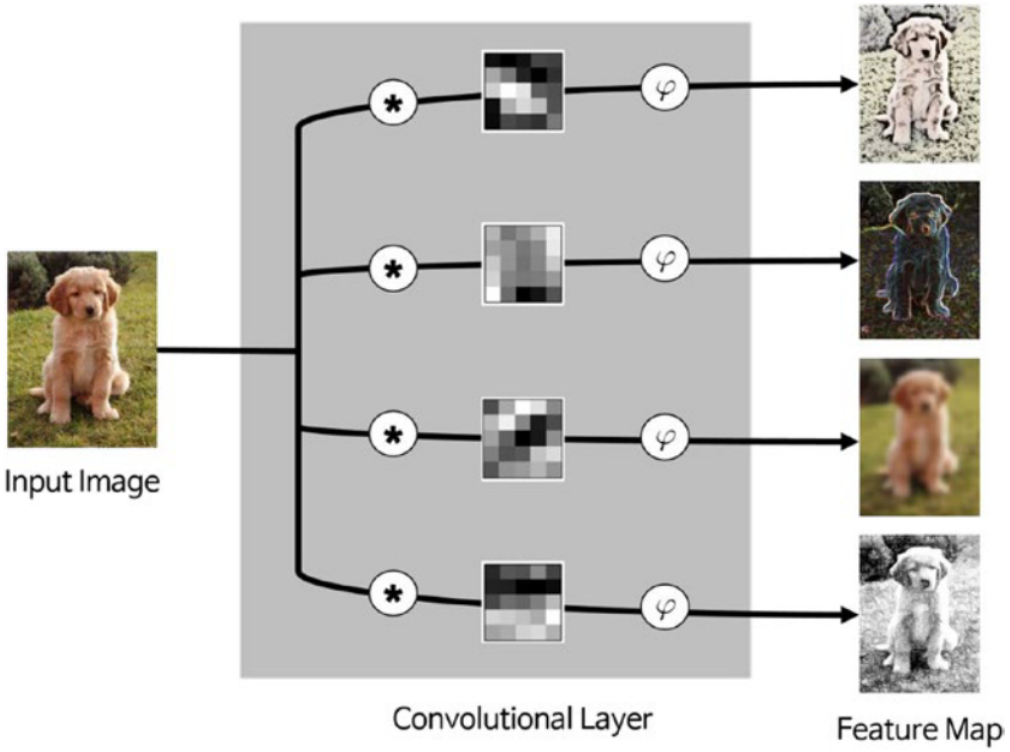
Konvoluce kombinuje matice – při dopředném průchodu konvoluje každý filtr přes šířku a hloubku, počítá skalární součin mezi konvolučním filtrem a vstupem a produkuje 2D aktivační mapu filtrů (viz obrázek 5, kde se výsledek konvoluce počítá jako $(1 \times 1) + (1 \times 0) + (4 \times 0) + (6 \times 1) = 7$). První vrstva extrahuje nejprimitivnější vlastnosti, jako jsou rohy, barva atd. Vrstvy postupně zjišťují detailnější vlastnosti. Počet výsledných 2D aktivačních map odpovídá počtu konvolučních filtrů. Před příchodem konvolučních neuronových sítí bylo potřeba definovat tyto extraktory příznaků manuálně. [12]



Obrázek 5: Ukázka konvoluce. Levá matice symbolizuje vstupní obrázek (šedá oblast je aktuální podmatice, s kterou se počítá aktuální konvoluce). Prostřední matice symbolizuje konvoluční filtr. V pravé matici jsou výsledky konvolucí (šedá oblast značí výsledek aktuální konvoluce).

Zdroj: podle [12]

Na obrázku 6 lze vidět výsledek procesu konvoluční vrstvy, kde $*$ značí konvoluci a φ značí aktivační funkci. Mezi $*$ a φ je obrázek, který reprezentuje konvoluční filtr. Výsledkem konvolucí na vstupním obrázku jsou 2D aktivační mapy zcela vpravo.



Obrázek 6: Vizualizace fungování konvoluční vrstvy

Zdroj: převzato z [12]

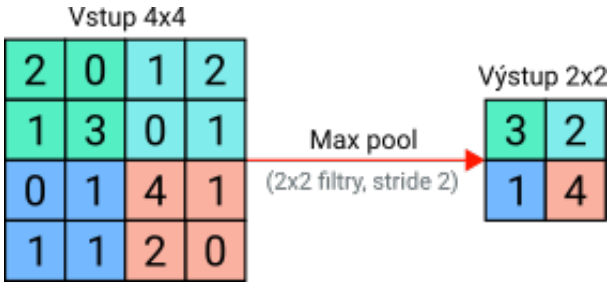
4.1.2 Pooling vrstva

Pooling vrstva slouží k redukci velikosti obrázku. To dokáže díky kombinaci několika sousedících pixelů do jednoho pixelu. Pro fungování pooling vrstvy je potřeba zajistit, jak se budou sousedící pixely vybírat a jak bude vypočtena výsledná hodnota pixelu. Sousedící pixely jsou zpravidla vybrány ze čtvercové matice a tyto čtvercové matice se nepřekrývají. Pro výpočet výsledné hodnoty pixelu se většinou používá průměr nebo maximum (viz obrázek 7) ze zvolených pixelů. Díky pooling vrstvě lze ušetřit výpočetní výkon a navíc dokáže zabránit přeučení. [12]

$$\frac{h - f + 1}{s} * (w - f + 1) * c \tag{6}$$

Význam jednotlivých parametrů:

- w — šířka mapy příznaků
- h — výška mapy příznaků
- c — počet kanálů



Obrázek 7: Ukázka max pooling vrstvy, kde je dimenze obrázku snížena ze 4x4 pixelů na 2x2 pixelů. Z každé barevné čtvercové matice je použita maximální hodnota.

Zdroj: podle [13]

- f — počet filtrů
- s — délka kroku

4.1.3 Plně propojená vrstva (Fully connected layer)

V této vrstvě jsou všechny výstupy neuronů propojeny se vstupy všech neuronů v další vrstvě. Zpravidla jsou na konci a produkují finální výsledek klasifikace na základě získaných map příznaků z předešlých vrstev. Pro vícetřídní klasifikaci se používá softmax aktivační funkce (viz kapitola 3.2.10). [13]

4.2 Modely

Nejvíce modelů začalo vznikat v poslední dekádě v rámci různých výzev, kde bylo úkolem vytvořit model s vysokou přesností na datasetu ImageNet (dataset s více jak 14 miliony obrázků a více jak 20.000 třídami).

4.2.1 AlexNet

Alex Krizhevksky a kolektiv v roce 2012 vymysleli nový způsob, jak paralelizovat učení konvolučních neuronových sítí přes více grafických karet [14]. Tato neuronová síť se učila na 1,2 milionech obrázků a ty byly klasifikovány do 1000 různých tříd. Skládala se z 60 milionů parametrů a 650.000 neuronů. Celá architektura je k vidění na obrázku 12, kde je vidět rozdělení na horní a dolní část. Každou část měla na starost jedna grafická karta a v některých vrstvách docházelo k vzájemné komunikaci. V rámci pěti nejlepších predikcí tříd dosáhl chybovosti 15,3%.

S počtem 60 milionů parametrů je potřeba vyřešit problém přetrénování. To bylo vyřešeno díky dvěma metodám. Pomocí augmentace dat — vytvoření dalších dat pomocí extrakce náhodných výřezů 224x224 z 256x256 obrázku a změnou intenzity RGB kanálů v trénovacích obrázcích. Druhou metodou bylo použití techniky dropout, která s pravděpodobností 0,5 deaktivovala neurony v prvních dvou plně propojených vrstvách.

4.2.2 ResNet

Kaiming He a kolektiv vyhráli se svou konvoluční neuronovou sítí první místo na soutěži ILSVRC 2015 [15]. V architektuře této sítě se vyskytuje unikátní prvek — reziduální blok

(viz obrázek 13) , který zajišťuje během v určitých případech přeskočení vah. Díky tomu řeší problém velmi nízkých gradientů (tedy už se přestanou měnit váhy) a také díky přeskočení náročných vrstev ušetří výkon. V rámci pěti nejlepších predikcí tříd dosáhl chybovosti 3,57%.

4.2.3 DenseNet

Gao Huang a kolektiv vytvořili konvoluční neuronovou síť, která vytváří více přímých spojení mezi vrstvami a tím optimalizuje samotné učení [16]. Normálně konvoluční neuronová síť s L vrstvami potřebuje L spojení. DenseNet používá $\frac{L(L+1)}{2}$ přímých spojení (viz obrázek 14), kde je k vidění blok 5 vrstev. Každá vrstva přijímá na vstupu aktivační mapy ze všech předešlých vrstev.

4.2.4 GoogLeNet

Christian Szegedy a kolektiv navrhli architekturu neuronové sítě, s kterou vyhráli první místo na soutěži ILSVRC 2014 [17]. Díky *inception* bloku (viz obrázek 15) se jim podařilo razantně snížit počet parametrů (pouze 4 miliony). V bloku jsou k vidění tři různé velikosti konvolučních filtrů, které zajistí lepší extrakci vlastností pro různé velikosti objektů v obrázku (objekty mohou být více v popředí/pozadí a pro každý takový objekt je vhodnější jiná velikost konvolučního filtru). Celá neuronová síť se skládá z 22 vrstev.

4.2.5 VGGNet

Konvoluční neuronová síť, kterou vytvořili A. Zisserman a K. Simonyan [18]. Existují dvě varianty — VGG16 a VGG19, kde čísla značí počet vrstev. VGG16, která se skládá z 13 konvolučních vrstev a 3 plně propojených vrstev (viz obrázek 16), dokáže klasifikovat obrázky až do 1000 tříd. V rámci pěti nejlepších predikcí tříd dosáhl chybovosti 6,8%.

4.3 Klasifikace

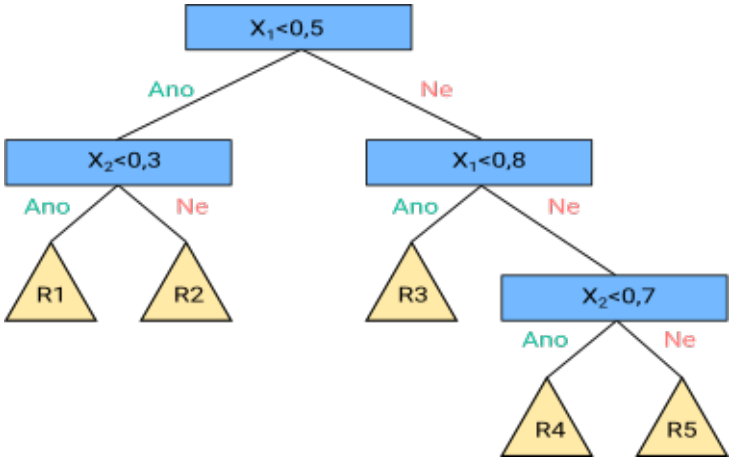
Konvoluční neuronové sítě se hlavně používají pro vizuální úkony, kde je potřeba klasifikovat jednotlivé objekty na obrázku. Úkolem je tedy najít společné znaky, podle kterých lze data rozdělit do jednotlivých tříd. Níže budou představeny různé klasifikátory (algoritmy, které mapují vstupní data do jednolivých tříd).

4.3.1 Rozhodovací strom

Cílem je vytvoření modelu, který dokáže predikovat proměnnou na základě jednoduchých rozhodovacích pravidel. Rozhodovací pravidla jsou naučena na základě testovacích dat. Při rozhodování o zařazení položky se postupuje od kořenového uzlu dál směrem k jednomu koncovému uzlu. Během cesty se v každém uzlu testuje rozhodovací pravidlo a na základě výsledku se zvolí další odpovídající potomek (uzel). [19]

Na obrázku 8 si lze tento proces předvést pro položku ($x1 = 0,9$,$x2 = 0.5$):

1. v kořenovém uzlu rozhodne pro pravého potomka $(0,9 > 0,5)$
2. v dalším uzlu se rozhodne pro pravého potomka $(0,9 > 0,8)$
3. v dalším uzlu se rozhodne pro levého potomka $(0,5 < 0,7)$
4. dostane se do koncového uzlu R4 — do této třídy je položka zařazena



Obrázek 8: Ukázka rozhodovacího stromu

Zdroj: podle [19]

4.3.2 Logistická regrese

Predikuje pravděpodobnost příslušnosti ke třídě nebo pravděpodobnost jevu (závislé proměnné) na základě nezávislých proměnných. Používá logistickou funkci $g(z) = \frac{1}{1+e^{-}z}$. Zásadní je příprava dat (například odstranění extrémů). Zpravidla se používá pro binární klasifikaci, ale lze ji použít i pro vícetřídní klasifikaci.

4.3.3 Metoda podpůrných vektorů (Support vector machines)

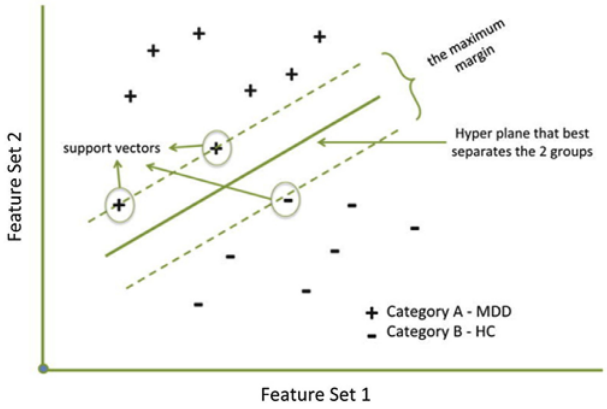
Jeden z nejpopulárnějších algoritmů, který se používá pro klasifikaci. Snaží se balancovat mezi maximalizací správně zařazených tříd (optimalizace přesnosti) a zajištění toho, aby byl dostatečně obecný pro nová data (reprodukovatelnost). Cílem je nalezení takové nadroviny, která rozdělí trénovací data na dva poloprostory (odlišné třídy), které jsou od sebe vzdálené na maximální možnou vzdálenost. Tato nadrovina je k vidění na obrázku 9, kde plná čára označuje nadrovinu a vzdálenost mezi čárkovanými čárami je maximální možná vzdálenost. Symboly plus a mínus označují dvě odlišné třídy a zakroužkované symboly značí podpůrné vektory. [20]

4.3.4 Bayesovský klasifikátor

Bayesovský klasifikátor (někdy také naivní bayesovský klasifikátor) přiřadí nejpravděpodobnější třídu k dané položce na základě vektorů příznaků. Tedy na základě nezávislých proměnných (každá přispívá stejným dílem pro správnou predikci) predikuje třídu. Vychází z Bayesovy věty o podmíněných pravděpodobnostech. [21]

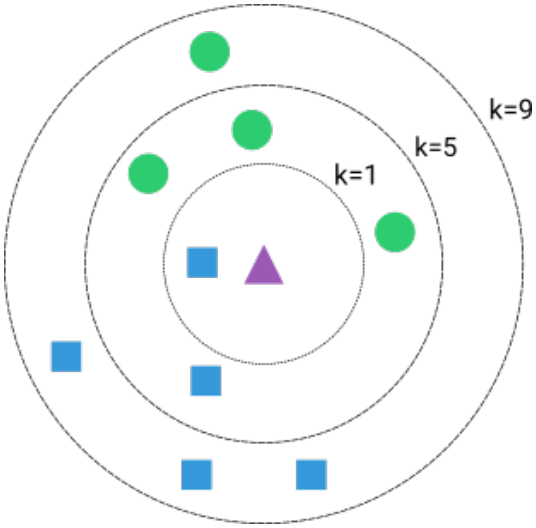
4.3.5 Algoritmus k-nejbližších sousedů

Tento algoritmus lze použít jak pro klasifikaci, tak i pro regresi. Při klasifikaci je na základě přilehlých sousedních dat přiřazena objektu třída (nejčastěji zastoupená mezi k sousedy). Proměnná k nabývá kladných hodnot (např. pro hodnotu 1 se zvolí třída podle jednoho nejbližšího souseda). Při volbě třídy lze brát v potaz i vzdálenost od sousedních dat (bližší sousedi mají větší váhu na finální volbu třídy). [22]



Obrázek 9: Metoda podpůrných vektorů — ukázka rozdělení prostoru pomocí nadroviny na dva poloprostory

Zdroj: převzato z [20]



Obrázek 10: Ukázka algoritmu k-nejbližších sousedů

Zdroj: podle [23]

Na obrázku 10 lze vidět ukázku algoritmu k-nejbližších sousedů. Je potřeba klasifikovat fialový trojúhelník (nezařazený objekt) buď do třídy zelených kruhů nebo modrých čtverců. Přerušované kružnice reprezentují zvolený počet nejbližších sousedů (proměnná k). Pro dané k bude fialový trojúhelník zařazen do třídy:

- $k = 1$ — modrý čtverec
- $k = 5$ — zelený kruh
- $k = 9$ — modrý čtverec

	Predikovaný pozitivní	Predikovaný negativní
Skutečně pozitivní	a	b
Skutečně negativní	c	d

Tabulka 1: Matice záměn pro binární klasifikaci

4.3.6 Matice záměn (Confusion matrix)

Pomocí matice záměn lze vizualizovat hodnocení klasifikátoru. Na diagonále se nacházejí všechny správné predikce. Matice není limitována svou velikostí — lze použít jak na binární klasifikaci (viz tabulka 1) tak i na vícetřídní klasifikaci. [24]

Z matice záměn (viz tabulka 1) lze vypočítat několik vlastností:

- **prevalence** — podíl správných predikcí $\frac{a+d}{a+b+c+d}$
- **preciznost** — podíl skutečně pozitivních mezi všemi pozitivně predikovanými $\frac{a}{a+c}$
- **senzitivita** — podíl predikovaných pozitivních mezi všemi skutečně pozitivními $\frac{a}{a+b}$
- **specificita** — podíl predikovaných negativních mezi všemi skutečně negativními $\frac{d}{c+d}$

4.4 Vytvoření modelu pro klasifikaci obrazu (učení s učitelem)

Proces vytvoření modelu pro klasifikaci obrazu se skládá z několika kroků. V průběhu se lze vracet a měnit/optimalizovat různé části. Mezi první krok se řadí příprava dat, která je esenciální pro správné a přesné natrénování modelu.

4.4.1 Příprava dat

Příprava dat je zásadním krokem při trénování modelu. Je potřeba vytvořit soubor obrázků a zařadit je do příslušných tříd. Takový soubor obrázků se nazývá dataset. V ideálním případě by měl mít dataset řádově tisíce až desetitisíce obrázků. Několik vytvořených datasetů lze pod určitou licencí získat na platformě *Kaggle*[1], která mimo jiné zprostředkovává i pracovní prostředí pro vývoj a trénování neuronových sítí. Také Google má svůj vyhledávací engine přímo dedikovaný pro vyhledávání datasetů[2]. S takto připraveným datasetem lze dále obrázky předzpracovat.

4.4.2 Předzpracování dat

Nyní je potřeba připravit dataset pro správné natrénování. V ideálním případě by měl být dataset vybalancovaný — tedy každá třída bude rovnoměrně zastoupena. Pokud je dataset nevybalancovaný, může docházet k přetrénování. Řešením je uměle dodat další obrázky pomocí augmentace (změna kontrastu, rotace, ořez) již existujících obrázků.

Celkově budou potřeba tři datasety, které budou sloužit pro trénování sítě a následné vyhodnocení správnosti predikcí:

[1] Kaggle `https://www.kaggle.com/datasets`
[2] Google Dataset Search `https://datasetsearch.research.google.com/`

- trénovací dataset — neuronová síť se na těchto datech učí predikovat (prochází dopřednou a zpětnou propagací)
- validační dataset — na těchto datech se ověřuje správnost predikce z procesu učení, na základě výsledků se následně ladí hyperparametry (nejvíce nápomocný ve fázi vývoje modelu)
- testovací dataset — po dokončeném učení slouží k finálnímu ověření správnosti predikce na neznámých datech

V této fázi jsou kompletně připravené datasety, které budou potřeba při trénování. V dalším kroku se vybere existující model nebo se vytvoří vlastní architektura konvoluční neuronové sítě.

4.4.3 Model

V tomto kroku se zvolí již existující model (viz kapitola 4.2) nebo se může nadefinovat vlastní architektura sítě (viz kapitola 4.1). Pro každý problém klasifikace je vhodné zvolit správnou architekturu, která bude co nejvíce efektivní.

Poté je potřeba vybrat ztrátovou funkci s ohledem na druh klasifikace (binární nebo vícetřídní klasifikace). Společně se ztrátovou funkcí je potřeba zvolit i optimalizátor, který má zásadní vliv na průběh učení (viz kapitola 3.5). Změnami těchto a dalších parametrů lze docílit zásadního zlepšení přesnosti modelu.

Nyní lze tento model trénovat a pozorovat dosažené přesnosti.

4.4.4 Trénování modelu

Po přípravě datasetu a vytvoření modelu je možné začít s trénováním modelu. Celý proces trénování lze rozdělit do jednotlivých epoch (jedna epocha se rovná jednomu průchodu celého trénovacího datasetu krokem dopředné a zpětné propagace viz kapitola 3.3). Dále je vhodné nastavit ukončující podmínku, kdy se síť přestane trénovat:

- dokud nebyl dosažen maximální počet epoch
- dokud se zvětšuje validační přesnost
- dokud se zmenšuje validační ztráta
- apod.

Po dokončeném trénování je na řadě vyhodnocení modelu, tedy jestli jeho predikce obstojí na zcela neznámých datech.

4.4.5 Vyhodnocení modelu a optimalizace přesnosti

Na závěr se musí trénování modelu vyhodnotit a ověřit přesnost predikce. K tomu nám poslouží testovací dataset, který obsahuje zcela neznámé obrázky, které neuronová síť nezpracovávala. Do natrénované sítě se odešlou testovací obrázky a na základě celkové přesnosti se vyhodnotí kvalita modelu.

Při vyhodnocení modelu je určitě vhodné použít vizualizační prostředky. Pomocí grafů lze sledovat průběh trénování/validace po jednotlivých epochách (trénovací přesnost a ztráta,

validační přesnost a ztráta). Dalším nástrojem může být matice záměn (viz kapitola 4.3.6), která přehledně zobrazí správné/špatné predikce na testovacím datasetu. Lze z ní vyčíst, jaké třídy byly často špatně zařazeny a naopak, jaké třídy byly zařazeny správně.

Pokud nejsou výsledky natrénovaného modelu uspokojující (nízká přesnost na testovacích obrázcích, špatná generalizace, přetrénování), lze se vrátit k předešlým krokům a model patřičně upravit a znovu natrénovat. Tím se může razantně zlepšit přesnost modelu. V tento moment je model připravený pro reálné nasazení v aplikaci, kde bude predikovat třídy objektů na zcela neznámých obrázcích.

5 Využití konvolučních neuronových sítí pro autonomní vozidla

S rostoucím počtem značek a vizuálního smogu v okolí silnic musí být řidič stále více na pozoru. I přes maximální pozornost může nastat situace, kdy řidič nestihne zareagovat na okolní dění a může dojít k nehodě. Naštěstí dnešní automobily disponují několika asistenčními systémy, které značně zvyšují bezpečnost vozidla a jeho řízení. Za většinu nehod může lidský faktor a pomocí autonomních vozidel by se mohla razantně snížit úmrtnost na silnicích. V posledních letech nastává mezi automobilkami boj o vývoj takového vozidla, které by řidiče vůbec nepotřebovalo (vozidlo zcela bez řidiče určené pouze pro přepravu lidí — bez volantu a pedálů).

5.1 Autonomní vozidla

Automobilky Audi, Volkswagen, Tesla a Google se snaží o vývoj plně autonomních vozidel, která budou provozu schopná zcela bez zásahu řidiče [25]. Pro autonomní vozidlo je velice důležité, aby mělo přehled o svém okolí — dopravní značky, semafory, chodci a především ostatní vozidla. Tedy potřebuje vnímat své okolí, rozpoznávat na něm objekty a na základě všech těchto informací provádět správná rozhodnutí. V další kapitole budou popsány senzory, které vozidlu pomůžou ve vnímání svého okolí.

5.2 Vnímání

Pro autonomní vozidlo je důležité, aby dokázalo vnímat své okolí. Na základě získaných dat o svém okolí se pak může správně rozhodovat. Proto musí vozidlo disponovat senzory, které tyto informace dokáží zpracovat. Mezi základní senzory vnímání patří kamera, LiDAR a RADAR.

5.2.1 Kamera

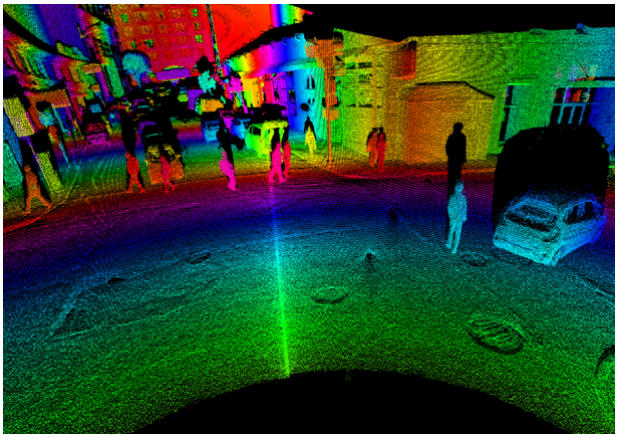
Pro rozpoznání objektů před vozidlem je zapotřebí kamera, která snímá prostor před vozidlem. Jednotlivé snímky lze pak použit pro klasifikaci, segmentaci a lokalizaci různých objektů.[26] Je důležité, aby kamery zabíraly kompletní oblast okolo vozidla (360 stupňů).

Hlavní nevýhodou je omezené fungování během extrémních podmínek jako je mlha, déšť nebo sníh (na obrázku je mnoho šumu a nelze se na snímky z kamery stoprocentně spolehnout). Tento problém lze řešit pomocí LiDARu nebo RADARu.

5.2.2 LiDAR (Light Detection And Ranging)

Pomocí laserového paprsku dokáže změřit vzdálenost, kterou počítá podle času, za který se paprsek odrazí od povrchu v závislosti na rychlosti šíření světla ve vzduchu. Díky tomu dokáže vnímat své okolí jako prostorovou informaci. Na obrázku 11) jsou k vidění jednotlivé odrazy paprsků jako body v 3D prostoru (vznikají shluky bodů). Podle vzdálenosti bodu od LiDARu se určí jeho barva. Pro tento druh informace existuje datový formát, který má na starosti ukládání 3D shluků bodů (přípona souboru .LAS). V kombinaci s kamerou dostane auto relativně dobrou představu o svém okolí. [27]

Při učení konvoluční neuronové sítě rozpoznávat např. chodce lze použitím snímků z kamery a dat z LiDARu docílit větší přesnosti klasifikace a předejít falešně pozitivním výsledkům (detekce chodce tam, kde žádný chodec není).



Obrázek 11: Ukázka 3D shluků bodů z LiDARu, kde barva bodu koresponduje se vzdáleností bodu od LiDARu

Zdroj: převzato z [28]

5.2.3 RADAR (Radio Detection And Ranging)

RADAR vysílá radiové vlny a ty se odrážejí od okolních objektů zpět do RADARu (díky tomu je RADAR zcela nezávislý na okolních podmínkách a funguje vždy). Na základě této doby lze vypočítat vzdálenost k objektu, od kterého se vlna odrazila. Data jsou strukturována podobně jako u LiDARu do 3D shluků bodů. Před použitím těchto dat je potřeba jej očistit (RADAR zachytává mnoho šumu) — tedy použijí se pouze silnější signály (zadefinuje se pomocí práhu). V kombinaci s daty z LiDARU lze získat poměrně přesnou představu o okolním prostředí vozidla. [29]

5.3 Lokalizace

Lokalizační algoritmy kalkulují pozici a orientaci vozidla na základě snímků z kamery — této problematice se věnuje věda vizuální odometrie. Na základě jednotlivých snímků mapuje klíčové body na obrázku a pomocí mapovacího algoritmu dokáže určit pozici a orientaci objektů. [26]

Díky konvoluční neuronové síti lze významně vylepšit výkon vizuální odometrie a navíc lze poté i tyto objekty klasifikovat do různých tříd.

5.4 Predikce

Zásadní úlohou autonomního vozidla je schopnost predikovat události na základě získaných dat:

- možné vběhnutí chodce do vozovky
- jiné vozidlo nedá přednost

Algoritmus hlubokého učení dokáže na základě obrázku a bodových dat z LiDARU a RADARu modelovat možné scénáře, které mohou v budoucnu nastat (např. auto náhle zabrzdí). Tedy na základě všech předešlých dat z různých senzorů se správně rozhodnout, pokud nastane určitá situace . [30]

5.5 Rozhodování

Vozidlo se musí rozhodnout na základě získaných dat a zároveň predikovat chování ostatních lidských řidičů. Pomocí připravených scénářů pak lze provést správné rozhodnutí, aby se například zabránilo nehodě. Samotné rozhodování je o tom, aby vozidlo učinilo to nejlepší rozhodnutí ze všech možných. Pro rozhodování se používá Markovův rozhodovací proces, který dokáže modelovat stavy a akce. Na základě vybrané akce proběhne přesun z jednoho stavu do druhého a uživatel získá odpovídající užitek. [31]

5.6 Klasifikace dopravních značek

V následující kapitole bude řešen problém klasifikace dopravních značek. Mezi sebou budou porovnány tři modely a vybere se ten nejlepší podle zvolených kritérií.

6 Metodika – testování konvolučních neuronových sítí pro autonomní vozidla (klasifikace dopravních značek)

V této části bude představen postup pro testování konvolučních neuronových sítí pro klasifikaci dopravních značek. Celkově budou porovnány 3 modely (ResNet50, VGG16 a AlexNet a každý model bude trénován během 4 experimentů. Následně se u každého modelu vyberou nejlepší experimenty a ty budou navzájem porovnány. Jako model s nejlepší testovací přesností je očekáván model ResNet50, který dosáhl nejmenší chybovosti 3,57% (VGG16 6,8% a AlexNet 15,3%) v rámci pěti nejlepších predikcí tříd (viz kapitoly 4.2.2, 4.2.5 a 4.2.1). Testování bude probíhat v jazyce Python.

Programovací jazyk Python je díky své jednoduchosti a rychlosti nejpoužívanějším programovacím jazykem pro strojové učení. Další nespornou výhodou je absolutní nezávislost na platformě. Existuje několik knihoven, které tuto práci výrazně zjednodušují. Pro testování konvolučních neuronových sítí byla vybrána knihovna PyTorch a framework PyTorch Lightning.

6.1 PyTorch

PyTorch je knihovna pro hluboké učení pomocí grafických karet nebo procesorů. Disponuje všemi důležitými funkcemi, které jsou potřeba pro vytvoření, trénování a testování neuronových sítí. Tím je ušetřeno mnoho času a lze se soustředit na samotné trénování a optimalizaci přesnosti.

Základním stavebním kamenem knihovny PyTorch jsou tzv. Tensory[3]. Jedná se o datovou strukturu, která je velmi podobná polím nebo maticím. Používá se pro zakódování vstupních a výstupních dat modelu (např.: obrázku). Hlavní výhodou je možnost zpracování tensorů na grafické kartě.

6.2 PyTorch Lightning

PyTorch Lightning je framework postavený na základech knihovny PyTorch. Vyniká zejména svým objektovým přístupem a odstíněním od některých kroků, které by se museli řešit pouze při použití knihovny PyTorch. Díky tomu jsou jednotlivé části dobře znovupoužitelné a případně nahraditelné např. jiným datasetem nebo zcela jiným modelem. Níže budou představeny jednotlivé části, které jsou důležité pro sestavení modelu:

- třída Dateset
  - zajišťuje získání položky z datasetu
  - aplikace transformací na položky (v případě klasifikace obrázků)
- třída LightningDataModule
  - má na starost načtení trénovacích, validačních a testovacích datasetů
  - nastavení velikosti jedné dávky
- třída LightningModule
  - nastavení modelu

[3] `https://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html`

  - konfigurace optimalizátorů
  - výběr ztrátové funkce

- třída Trainer
  - zajišťuje trénování sítě
  - nastavení parametrů trénování
  - umožňuje následně otestovat sít na testovacím datasetu

6.3 Klasifikace dopravních značek

Problém klasifikace dopravních značek je jedním z úkolů, s kterým si musí poradit autonomní vozidlo. Níže budou představeny jednotlivé části kódu (pro použití s frameworkem PyTorch Lightning), které budou následně použity při trénování tří různých modelů.

6.3.1 Dataset

Pro klasifikaci dopravních značek bude nejdříve potřeba dataset. Na platformě Kaggle[4] je k dispozici dataset německých dopravních značek[5]. Obsahuje 43 tříd a celkově více jak 50.000 obrázků (jednotlivé třídy nesou označení od 0 do 42 viz tabulka7). Dataset je rozdělen na dvě části — trénovací (39.209 obrázků) a testovací (12.631 obrázků). Tento dataset bude potřeba načíst a pak s ním dále pracovat.

K tomuto účelu je vytvořena třída TrafficSignDataset, která dědí z třídy torch.utils.data.Dataset. Níže je zobrazen celý kód této třídy. Za zmínku stojí zejména metoda __getitem__, která zajistí získání obrázku z datasetu dle indexu a následně aplikuje transformace na obrázek. Tyto transformace jsou předány v konstruktoru.

Pro modely z knihovny *PyTorch* jsou zapotřebí následují transformace[6]:

- obrázky musí mít minimální výšku 224 a minimální šířku 224
- musí být v rozsahu 0 až 1 — to zajistí metoda torchvision.transforms.ToTensor[7], která převede *PIL Image* z rozsahu 0 až 255 na torch.FloatTensor v rozahu 0 až 1
- normalizace pomocí metody torchvision.transforms.Normalize (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]

```
1  import os
2  from torch.utils.data import Dataset
3  from PIL import Image
4
5  class TrafficSignDataset(Dataset):
6      def __init__(self, X: list, y: list, root: str, transforms=None):
7          self.X = list(X)
8          self.y = list(y)
9
```

[4] https://www.kaggle.com/
[5] https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign
[6] https://pytorch.org/vision/stable/models.html
[7] https://pytorch.org/vision/main/generated/torchvision.transforms.ToTensor.html

```
        self.root = root
        self.transforms = transforms

    def __len__(self):
        return len(self.X)

    # Získání obrázku z datasetu
    def __getitem__(self, idx):
        filename = os.path.join(self.root, self.X[idx])
        image = Image.open(filename)

        label = self.y[idx]

        # Aplikace transformací na obrázek
        if self.transforms is not None:
            image = self.transforms(image)

        return image, label

    def get_labels(self):
         return self.y
```

Dále bude potřeba trénovací, validační a testovací dataset načíst. Tedy vzniknou dvě instance třídy TrafficSignDataset — jedna pro účely testování (načtou se testovací obrázky) a druhá se rozdělí na dataset trénovací a validační (načtou se obrázky pro trénování a ty se v určitém poměru rozdělí).

6.3.2 LightningDataModule

Nyní se bude dataset načítat pomocí třídy pytorch_lightning.LightningDataModule. Ta disponuje několika metodami, které lze implementovat — setup, train_dataloader, val_dataloader a test_dataloader. V metodě setup se načtou datasety, vytvoří instance třídy TrafficSignDataset a následně se rozdělí trénovací dataset na trénovací a validační v určitém poměru pomocí metody torch.utils.data.random_split.

```
import os
import pytorch_lightning as pl
import pandas as pd
from torch.utils.data import DataLoader, random_split
from torchvision import transforms as T

# Cesta k root složce datasetu
ROOT = os.path.join('..', 'input', 'gtsrb-german-traffic-sign')

class TrafficSignDataModule(pl.LightningDataModule):
    def __init__(self, data_dir: str = "path/to/dir", batch_size: int =
        32):
        super().__init__()
        self.data_dir = data_dir
        self.batch_size = batch_size
```

```
        self.transform = T.Compose([
            T.Resize((224, 224)),
            T.ToTensor(),
            T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
                0.225])
        ])

    # Příprava dat
    def setup(self):
        train_data = pd.read_csv(os.path.join(ROOT, 'Train.csv'))
        self.num_classes = max(train_data.ClassId.unique())+1

        X_train, y_train = train_data.Path, train_data.ClassId
        full_dataset = TrafficSignDataset(X_train, y_train, root=ROOT,
            transforms=self.transform)

        # Rozdělení datasetu na trénovací a validační dataset
        self.train_dataset, self.val_dataset = random_split(full_dataset,
            [35000, 4209])

        test_data = pd.read_csv(os.path.join(ROOT, 'Test.csv'))
        X_test, y_test = test_data.Path, test_data.ClassId

        self.test_dataset = TrafficSignDataset(X_test, y_test, root=ROOT,
            transforms=self.transform)

    # Inicializace loaderu pro trénovací dataset + použití sampleru pro
        vybalancování dat
    def train_dataloader(self):
        return DataLoader(self.train_dataset, batch_size=self.batch_size,
            sampler=ImbalancedDatasetSampler(self.train_dataset))

    # Inicializace loaderu pro validační dataset
    def val_dataloader(self):
        return DataLoader(self.val_dataset, batch_size=self.batch_size)

    # Inicializace loaderu pro testovací dataset
    def test_dataloader(self):
        return DataLoader(self.test_dataset, batch_size=self.batch_size)
```

6.3.3 LightningModule

Dále bylo potřeba vytvořit třídu TrafficSignModel, která má na starosti definování modelu. Zde jsou definovány metody pro inicializaci jednotlivých modelů (ResNet50, VGG16 a AlexNet) a přiřazení správného počtu tříd do jejich klasifikační vrstvy. Jsou zde použity předtrénované modely (parametr pretrained=True), které zajistí inicializaci vah na optimální hodnoty (z trénování sítě na určitém datasetu).

```
import torch
```

```
2  from torchvision import transforms as T, models, utils
3  import torchmetrics as tm
4  from torch import nn
5  import torch.nn.functional as F
6
7  LEARNING_RATE = 1e-4
8
9  class TrafficSignModel(pl.LightningModule):
10     def __init__(self, num_of_classes: int):
11         super().__init__()
12
13         self.num_of_classes = num_of_classes
14
15         self.train_acc = tm.Accuracy()
16         self.val_acc = tm.Accuracy()
17
18     def setup(self, stage = None):
19         self._create_model()
20
21     # Výběr modelu
22     def _create_model(self):
23         self._use_resnet50()
24         # self._use_vgg16()
25         # self._use_alexnet()
26
27     # Inicializace modelu alexnet + nastavení správného počtu tříd pro
           klasifikátor
28     def _use_alexnet(self):
29         self.model = models.alexnet(pretrained=True)
30         inputs = self.model.classifier[6].in_features
31         self.model.classifier[6] = nn.Linear(inputs, self.num_of_classes)
32
33     # Inicializace modelu resnet50 + nastavení správného počtu tříd pro
           klasifikátor
34     def _use_resnet50(self):
35         self.model = models.resnet50(pretrained=True)
36         inputs = self.model.fc.in_features
37         self.model.fc = nn.Linear(inputs, self.num_of_classes)
38
39     # Inicializace modelu vgg16 + nastavení správného počtu tříd pro
           klasifikátor
40     def _use_vgg16(self):
41         self.model = models.vgg16(pretrained=True)
42         inputs = self.model.classifier[6].in_features
43         self.model.classifier[6] = nn.Linear(inputs, self.num_of_classes)
44
45     # Dopředný krok
46     def forward(self, x):
```

```
47          return self.model(x)
48
49      # Trénovací krok
50      def training_step(self, batch, batch_idx):
51          images, labels = batch
52          outputs = self(images)
53
54          # Kalkulace ztráty pomocí křížové entropie
55          loss = F.cross_entropy(outputs, labels)
56          self.train_acc(outputs, labels)
57
58          self.log('train_loss', loss, on_step=False, on_epoch=True,
                prog_bar=True, sync_dist=True)
59          self.log('train_acc', self.train_acc, on_step=False, on_epoch=
                True, prog_bar=True, sync_dist=True)
60
61          return {'loss': loss}
62
63      # Validační krok
64      def validation_step(self, batch, batch_idx):
65          images, labels = batch
66          outputs = self(images)
67
68          # Kalkulace ztráty pomocí křížové entropie
69          loss = F.cross_entropy(outputs, labels)
70          self.val_acc(outputs, labels)
71
72          self.log("val_loss", loss, on_step=False, on_epoch=True, prog_bar
                =True, sync_dist=True)
73          self.log('val_acc', self.val_acc, on_step=False, on_epoch=True,
                prog_bar=True, sync_dist=True)
74
75          return {'val_loss': loss}
76
77      # Testovací krok, volá po zavolání metody Trainer.fit
78      def test_step(self, batch, batch_idx):
79          images, labels = batch
80          outputs = self(images)
81
82          # Kalkulace ztráty pomocí křížové entropie
83          loss = F.cross_entropy(outputs, labels)
84          _, preds = torch.max(outputs, 1)
85          correct = torch.sum(preds == labels.data)
86
87          return {
88              'test_loss': loss,
89              'test_correct': correct
90          }
```

```
91
92      def test_epoch_end(self, outputs):
93          avg_loss = torch.stack([x['test_loss'] for x in outputs]).mean()
94          avg_acc = torch.stack([x['test_correct'].float() for x in outputs
                ]).sum() / 12630
95
96          self.log("test_loss",avg_loss)
97          self.log('test_acc', avg_acc)
98          return {
99              'avg_test_loss': avg_loss,
100             'avg_test_acc': avg_acc
101         }
102
103     # Konfigurace optimalizátorů
104     def configure_optimizers(self):
105         return torch.optim.Adam(self.parameters(), lr=LEARNING_RATE)
```

6.3.4 Logger

Pro vizualizaci veškerých dat během trénování byla zvolena platforma Comet[8]. Nejdříve je potřeba vytvořit instanci třídy pytorch_lightning.loggers.CommetLogger (je potřeba vygenerovat API klíč), kterou následně předáme v konstruktoru třídě pytorch_lightning.Trainer v následující kapitole.

```
1  import comet_ml
2  from pytorch_lightning.loggers import CometLogger
3  from kaggle_secrets import UserSecretsClient
4
5  us = UserSecretsClient()
6
7  comet_logger = CometLogger(
8      api_key=us.get_secret("COMET_API_KEY"),
9      project_name=us.get_secret("COMET_PROJECT"),
10     workspace=us.get_secret("COMET_WORKSPACE"),
11 )
```

6.3.5 Trainer

Pro samotné trénování bude potřeba vytvořit instanci třídy pytorch_lightning.Trainer, která kromě trénování zajistí i validaci a testování. V konstruktoru lze předat několik parametrů, kterými můžeme ovlivnit chování[9]. Za zmínku stojí parametr limit_train_batches, který při hodnotě 0,1 použije při testování pouze 0,1 náhodně vybraných obrázků z trénovacího datasetu (tím se urychlí trénování sítě a zároveň každý experiment bude natrénován pomocí jiného souboru trénovacích dat). Pomocí parametru callbacks lze nastavit zpětná volání v reakci na různé události (začátek trénování, konec trénování, konec trénovací epochy atd.). Zde je nastaven pytorch_lightning.callbacks.early_stopping.EarlyStopping, který

[8] https://comet.ml/
[9] https://pytorch-lightning.readthedocs.io/en/stable/common/trainer.html

zajistí předčasné ukončení trénování. Volá se na konci každého validačního kroku a dokud se validační ztráta zlepšuje, tak trénování pokračuje.

```
import pytorch_lightning as pl
import torch
from pytorch_lightning.callbacks.early_stopping import EarlyStopping

trainer = pl.Trainer(
    limit_train_batches=0.1,
    logger=comet_logger,
    fast_dev_run=False,
    callbacks=[EarlyStopping(monitor="val_loss")],
    max_epochs=50,
    gpus=min(1, torch.cuda.device_count()),
    accelerator="gpu",
    precision=16,
    profiler="advanced"
)


dm = TrafficSignDataModule()
dm.prepare_data()
dm.setup()
model = TrafficSignModel(dm.num_classes)

# Trénování a validace modelu
trainer.fit(model, dm)

# Testování modelu po ukončeném trénování
trainer.test(datamodule=dm)
```

7 Provedené experimenty a dosažené výsledky

V následujících částech budou vyhodnoceny výsledky několika experimentů v klasifikaci dopravních značek. Pro tyto experimenty byly vybrány modely ResNet50 (viz kapitola 4.2.2), VGG16 (viz kapitola 4.2.5) a AlexNet (viz kapitola 4.2.1).

Dále bude popsán samotný průběh porovnání modelů.

7.1 Porovnání modelů

Celkově budou porovnány 3 modely z hlediska počtu epoch, trénovací přesnosti, validační přesnosti, testovací přesnosti a délky experimentu:

- počet epoch — celkový počet epoch, po kterých bylo ukončeno trénování
- trénovací přesnost — konečná přesnost po poslední epoše na trénovacích datech
- validační přesnost — konečná přesnost po poslední epoše na validačních datech
- testovací přesnost — přesnost po natrénování sítě na testovacích (neznámých) datech
- délka experimentu — zahrnuje délku trénování sítě, validace na konci každé epochy, finální testování sítě a generování matice záměn

Každý model byl natrénován v rámci 4 experimentů. Veškeré experimenty probíhaly v prostředí Kaggle[10], kde byly modely trénovány na jedné grafické kartě.

Dále budou vizualizovány konvoluční filtry jednotlivých modelů a jejich výsledné 2D aktivační mapy (vlastní aplikace). Součástí vizualizací bude i matice záměn, která bude vytvořena na základě výsledků finálního testování modelu. Všechny tyto vizuální výstupy experimentů jsou k dispozici v příloze C. V závěru bude zvolen nejlepší model pro klasifikaci dopravních značek podle dosažené testovací přesnosti a podle rychlosti.

Pro všechny modely byly zvoleny stejné parametry na základě empirického výzkumu:

- ztrátová funkce — vícetřídní křížová entropie (viz kapitola 3.3.2)
- optimalizátor — ADAM s výchozími parametry ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ a $\epsilon = 10^{-8}$ viz kapitola 3.5.4)
- velikost dávky — 32
- předčasné zastavení trénování, pokud se nezlepšuje validační ztráta

Nyní lze přejít k jednotlivým modelům a podívat se na jednotlivé experimenty.

[10] `https://kaggle.com/`

7.1.1 ResNet50

Model ResNet50 vychází z konvoluční neuronové sítě ResNet (viz kapitola 4.2.2). Číslovka v názvu značí celkový počet vrstev v síti.

V tabulce 2 jsou výsledky 4 experimentů. Natrénovaný model z experimentu 4 s testovací přesností 97,9% se jeví jako nejúspěšnější model. Této přesnosti dokázal docílit během 8 epoch. Vývoj trénovací a validační přesnosti po jednotlivých epochách je vidět v grafu na obrázku 10. Vývoj trénovací a validační ztráty po jednotlivých epochách je vidět v grafu na obrázku 11.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Počet epoch	8	10	6	8
Trénovací přesnost	99%	99,29%	98,7%	99,3%
Validační přesnost	97,5%	99,4%	97,8%	99,5%
Testovací přesnost	95,2%	97,1%	95,6%	97,9%
Délka experimentu	11m a 11s	12m a 15s	9m a 23s	10m a 52s

Tabulka 2: Čtyři experimenty trénování modelu ResNet50 na klasifikaci dopravních značek. Zeleně vyznačeny nejlepší výsledky pro daný parametr.

Na obrázku 17 je k vidění ukázka matice záměn prvních 16 tříd. Z této ukázky je patrná vysoká přesnost natrénovaného modelu, ale také pár špatně zařazených testovacích obrázků. Například z počtu 450 testovacích obrázků, které skutečně patří do třídy 3 (značka 60km/h), bylo 20 obrázků zařazeno do třídy 5 (značka 80km/h). Tato nepřesnost mohla vzniknout z vysoké podobnosti číslovek 6 a 8.

Na obrázku 18 je vidět konvoluční filtr z první konvoluční vrstvy (64 kernelů s rozměry 7x7). Jednotlivé kernely se snaží zachytit rozmanité tvary a obrysy. Na obrázku 19 lze pozorovat část 2D aktivační mapy po průchodu první konvoluční vrstvou (detekce nákladního vozidla a kruhu).

7.1.2 VGG16

Model VGG16 vychází z konvoluční neuronové sítě VGGNet (viz kapitola 4.2.5). Číslovka v názvu (podobně jako u modelu ResNet) značí celkový počet vrstev v síti.

V tabulce 3 jsou výsledky 4 experimentů. Natrénovaný model z experimentu 3 s testovací přesností 97,1% se jeví jako nejúspěšnější model. Této přesnosti dokázal docílit během 10 epoch. U experimentu 1 a 2 dochází k přetrénování (sice má větší trénovací přesnost než experiment 3, ale nedokáže dobře generalizovat na neznámých datech). Vývoj trénovací a validační přesnosti po jednotlivých epochách je vidět v grafu na obrázku 12. Vývoj trénovací a validační ztráty po jednotlivých epochách je vidět v grafu na obrázku 13.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Počet epoch	10	9	10	9
Trénovací přesnost	99%	98,9%	98,6%	98,5%
Validační přesnost	98,5%	98,5%	99,3%	98,2%
Testovací přesnost	96,2%	96%	97,1%	96,2%
Délka experimentu	15m a 33s	13m a 51s	15m a 53s	13m a 58s

Tabulka 3: Čtyři experimenty trénování, validace a testování modelu VGG16 na klasifikaci dopravních značek. Zeleně vyznačeny nejlepší výsledky pro daný parametr.

Na obrázku 20 je k vidění ukázka matice záměn prvních 16 tříd. Z této ukázky je patrná vysoká přesnost natrénovaného modelu, ale také pár špatně zařazených testovacích obrázků. Například z počtu 450 testovacích obrázků, které skutečně patří do třídy 3 (značka 60km/h), bylo 41 obrázků zařazeno do třídy 5 (značka 80km/h), 4 obrázky zařazeny do třídy 2 (50 km/h) a 1 obrázek zařazen do třídy 10 (zákaz předjíždění osobní aut nákladními vozidly).

Na obrázku 21 je vidět konvoluční filtr z první konvoluční vrstvy (64 kernelů s rozměry 3x3). Jednotlivé kernely se snaží zachytit rozmanité tvary a obrysy. Na obrázku 22 lze pozorovat část 2D aktivační mapy po průchodu první konvoluční vrstvou (detekce nákladního vozidla a kruhu).

7.1.3 AlexNet

Model AlexNet vychází z konvoluční neuronové sítě AlexNet (viz kapitola 4.2.1).

V tabulce 4 jsou výsledky 4 experimentů. I když se vždy mění výběr trénovacích obrázků z datasetu, tak všechny experimenty dopadly totožným výsledkem.

Všechny natrénované modely dosáhly testovací přesnosti 94,2%. Této přesnosti dokázaly docílit během 12 epoch, ale za poměrně krátkou dobu. Vývoj trénovací a validační přesnosti po jednotlivých epochách je vidět v grafu na obrázku 14. Vývoj trénovací a validační ztráty po jednotlivých epochách je vidět v grafu na obrázku 15.

	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Počet epoch	12	12	12	12
Trénovací přesnost	98,4%	98,4%	98,4%	98,4%
Validační přesnost	98,6%	98,6%	98,6%	98,6%
Testovací přesnost	94,2%	94,2%	94,2%	94,2%
Délka experimentu	7m a 33s	7m a 28s	7m a 18s	7m a 17s

Tabulka 4: Čtyři experimenty trénování, validace a testování modelu AlexNet na klasifikaci dopravních značek. Zeleně vyznačeny nejlepší výsledky pro daný parametr.

Na obrázku 23 je k vidění ukázka matice záměn prvních 16 tříd. Z této ukázky je poměrně vysoká přesnost natrénovaného modelu, ale také pár špatně zařazených testovacích obrázků. Například z počtu 450 testovacích obrázků, které skutečně patří do třídy 3 (značka 60km/h), bylo 40 obrázků zařazeno do třídy 5 (značka 80km/h) a 7 obrázků zařazeno do třídy 2 (50 km/h).

Na obrázku 24 je vidět konvoluční filtr z první konvoluční vrstvy (64 kernelů s rozměry 11x11). Jednotlivé kernely se snaží zachytit rozmanité tvary a obrysy. Na obrázku 25 lze pozorovat část 2D aktivační mapy po průchodu první konvoluční vrstvou (detekce nákladního vozidla a kruhu).

7.2 Porovnání nejlepších experimentů

Jako nejlepší experiment se jeví model ResNet, který se dokázal během 8 epoch natrénovat a dosáhl testovací přesnosti 97, 9%. Dle tabulky 6 dokázal porazit i ostatní modely v průměrné testovací přesnosti (96, 45%). Rychlejší byl akorát experiment s modelem AlexNet, který trval pouhých 7 minut a 18 sekund. Kompletní porovnání je k dispozici v tabulce 5. Na obrázcích 16, 17, 18 a 19 jsou k dispozici grafy pro porovnání trénovací a validační přesnosti a grafy pro porovnání trénovací a validační ztráty.

	ResNet50	VGG16	AlexNet
Počet epoch	8	12	12
Trénovací přesnost	99,3%	98,6%	98,4%
Validační přesnost	99,5%	99,3%	98,6%
Testovací přesnost	97,9%	97,1%	94,2%
Délka experimentu	10m a 52s	15m a 53s	7m a 18s

Tabulka 5: Porovnání nejlepších experimentů modelu ResNet50, VGG16 a AlexNet na klasifikaci dopravních značek. Zeleně vyznačeny nejlepší výsledky pro daný parametr.

	ResNet50	VGG16	AlexNet
Průměrný počet epoch	8	9,5	12
Průměrná trénovací přesnost	99,07%	98,75%	98,4%
Průměrná validační přesnost	98,55%	98,63%	98,6%
Průměrná testovací přesnost	96,45%	96,38%	94,2%
Průměrná délka experimentu	10m a 55s	14m a 49s	7m a 7s

Tabulka 6: Porovnání průměrných hodnot napříč experimenty modelu ResNet50, VGG16 a AlexNet na klasifikaci dopravních značek. Zeleně vyznačeny nejlepší výsledky pro daný parametr.

8 Shrnutí výsledků, závěry a doporučení

Celkem byly provedeny s každým modelem (ResNet50, VGG16 a AlexNet) čtyři experimenty. Nejlepších výsledků dosáhl model ResNet50 s nejlepší testovací přesností $97,9\%$ a průměrnou testovací přesností $96,45\%$. Jako druhý nejlepší model skončil model VGG16 s nejlepší testovací přesností $97,1\%$ a průměrnou testovací přesností 96.38%. Posledním modelem se stal model AlexNet s nejlepší testovací přesností $94,2\%$ a průměrnou testovací přesností 94.2%.

Kód aplikace pro testování modelů je připraven tak, aby bylo možné učinit porovnání i s dalšími modely z knihovny PyTorch. Zajímavým rozšířením by bylo vytvoření vlastní architektury konvoluční neuronové sítě a pokusit se dosáhnout lepší testovací přesnosti než které dosáhl nejlepší experiment modelu ResNet50. Další možností by bylo zaměření se i na data z ostatních senzorů (LiDAR a RADAR), které by poskytly novou výzvu v podobě trojrozměrných dat. V kombinaci se snímky z kamery by se mohlo jednat o poutavou úlohu.

To samé platí i pro vizualizační aplikaci, kde by nebyl problém provést vizualizace konvolučních filtrů a aktivačních map i pro jiné konvoluční neuronové sítě. Dále by se dala tato aplikace rozšířit o další vizualizace, které by poskytly lepší náhled do fungování konvoluční neuronové sítě.

Celkově se víceméně podařilo dosáhnout veškerých cílů práce. Obecně byla představena umělá neuronová síť a její fungování. Poté byla popsána architektura konvoluční neuronové sítě a její využití pro autonomní vozidla (tato část by se dala v budoucnu dále rozpracovat). Na závěr se podařilo provést několik experimentů s třemi modely a navzájem je porovnat (v budoucnu by bylo možné vytvoření vlastní architektury konvoluční sítě a její porovnání s existujícími modely).

Téma konvolučních neuronových sítí je velice zajímavé a určitě bude mít v budoucnu velký dopad v mnoha oblastech vědy a výzkumu.

Literatura

[1] Wu, Y.-c.; Feng, J.-w. Development and Application of Artificial Neural Network. *Wireless Personal Communications*, volume 102, no. 2, Sept. 2018: pp. 1645–1656, ISSN 1572-834X, doi:10.1007/s11277-017-5224-x. Available from: https://doi.org/10.1007/s11277-017-5224-x

[2] Sharma, S.; Sharma, S.; Scholar, U.; et al. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. volume 4, no. 12, 2020: p. 7.

[3] Zhang, Y. *New Advances in Machine Learning*. BoD – Books on Demand, Feb. 2010, ISBN 978-953-307-034-6, google-Books-ID: XAqhDwAAQBAJ.

[4] Choi, D.; Shallue, C. J.; Nado, Z.; et al. On Empirical Comparisons of Optimizers for Deep Learning. *arXiv:1910.05446 [cs, stat]*, June 2020, arXiv: 1910.05446. Available from: http://arxiv.org/abs/1910.05446

[5] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv:1609.04747 [cs]*, June 2017, arXiv: 1609.04747. Available from: http://arxiv.org/abs/1609.04747

[6] What is Gradient Descent? Oct. 2020. Available from: https://www.ibm.com/cloud/learn/gradient-descent

[7] Robbins, H.; Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, volume 22, no. 3, 1951: pp. 400–407, ISSN 0003-4851, publisher: Institute of Mathematical Statistics. Available from: https://www.jstor.org/stable/2236626

[8] Polyak, B. T. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, volume 4, no. 5, Jan. 1964: pp. 1–17, ISSN 0041-5553, doi:10.1016/0041-5553(64)90137-5. Available from: https://www.sciencedirect.com/science/article/pii/0041555364901375

[9] Kingma, D. P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Jan. 2017, arXiv: 1412.6980. Available from: http://arxiv.org/abs/1412.6980

[10] Zou, F.; Shen, L. *On the Convergence of Weighted AdaGrad with Momentum for Training Deep Neural Networks*. Aug. 2018.

[11] O'Shea, K.; Nash, R. An Introduction to Convolutional Neural Networks. *arXiv:1511.08458 [cs]*, Dec. 2015, arXiv: 1511.08458. Available from: http://arxiv.org/abs/1511.08458

[12] Kim, P. Convolutional Neural Network. In *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*, edited by P. Kim, Berkeley, CA: Apress, 2017, ISBN 978-1-4842-2845-6, pp. 121–147, doi:10.1007/978-1-4842-2845-6_6. Available from: https://doi.org/10.1007/978-1-4842-2845-6_6

[13] Sakib, S.; Ahmed, N.; Kabir, A. J.; et al. An Overview of Convolutional Neural Network: Its Architecture and Applications. Feb. 2019,

doi:10.20944/preprints201811.0546.v4, publisher: Preprints. Available from: https://www.preprints.org/manuscript/201811.0546/v4

[14] Krizhevsky, A.; Sutskever, I.; Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*, volume 25, Jan. 2012, doi:10.1145/3065386.

[15] He, K.; Zhang, X.; Ren, S.; et al. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. Available from: http://arxiv.org/abs/1512.03385

[16] Huang, G.; Liu, Z.; van der Maaten, L.; et al. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, Jan. 2018, arXiv: 1608.06993. Available from: http://arxiv.org/abs/1608.06993

[17] Szegedy, C.; Liu, W.; Jia, Y.; et al. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, Sept. 2014, arXiv: 1409.4842. Available from: http://arxiv.org/abs/1409.4842

[18] Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, Apr. 2015, arXiv: 1409.1556. Available from: http://arxiv.org/abs/1409.1556

[19] SONG, Y.-y.; LU, Y. Decision tree methods: applications for classification and prediction. *Shanghai Archives of Psychiatry*, volume 27, no. 2, Apr. 2015: pp. 130–135, ISSN 1002-0829, doi:10.11919/j.issn.1002-0829.215044. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/

[20] Pisner, D. A.; Schnyer, D. M. Chapter 6 - Support vector machine. In *Machine Learning*, edited by A. Mechelli; S. Vieira, Academic Press, Jan. 2020, ISBN 978-0-12-815739-8, pp. 101–121, doi:10.1016/B978-0-12-815739-8.00006-7. Available from: https://www.sciencedirect.com/science/article/pii/B9780128157398000067

[21] Rish, I.; others. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, 2001, pp. 41–46, issue: 22.

[22] Kramer, O. K-Nearest Neighbors. In *Dimensionality Reduction with Unsupervised Nearest Neighbors*, edited by O. Kramer, Berlin, Heidelberg: Springer, 2013, ISBN 978-3-642-38652-7, pp. 13–23, doi:10.1007/978-3-642-38652-7_2. Available from: https://doi.org/10.1007/978-3-642-38652-7_2

[23] AnAj, A. A. Example of k-nearest neighbour classification. May 2007. Available from: https://commons.wikimedia.org/wiki/File:KnnClassification.svg

[24] Deng, X.; Liu, Q.; Deng, Y.; et al. An improved method to construct basic probability assignment based on the confusion matrix for classification problem. *Information Sciences*, volume 340-341, May 2016: pp. 250–261, ISSN 0020-0255, doi:10.1016/j.ins.2016.01.033. Available from: https://www.sciencedirect.com/science/article/pii/S002002551600044X

[25] Al Zaabi, A. O.; Yeun, C. Y.; Damiani, E. Autonomous Vehicle Security: Conceptual Model. In *2019 Ieee Transportation Electrification Conference and Expo, Asia-Pacific (itec Asia-Pacific 2019): New Paradigm Shift, Sustainable E-Mobility*, New York: Ieee, 2019, ISBN 978-1-72812-124-6, pp. 188–192, iSSN: 2693-3586 WOS:000650969400035. Available from: https://www.webofscience.com/wos/woscc/full-record/WOS:000650969400035

[26] Schwarting, W.; Alonso-Mora, J.; Rus, D. Planning and Decision-Making for Autonomous Vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, volume 1, no. 1, 2018: pp. 187–210, doi:10.1146/annurev-control-060117-105157, _eprint: https://doi.org/10.1146/annurev-control-060117-105157. Available from: https://doi.org/10.1146/annurev-control-060117-105157

[27] Royo, S.; Ballesta-Garcia, M. An Overview of Lidar Imaging Systems for Autonomous Vehicles. *Applied Sciences*, volume 9, no. 19, Jan. 2019: p. 4093, ISSN 2076-3417, doi:10.3390/app9194093, number: 19 Publisher: Multidisciplinary Digital Publishing Institute. Available from: https://www.mdpi.com/2076-3417/9/19/4093

[28] Hecht, J. Lidar for self-driving cars. *Optics and Photonics News*, volume 29, no. 1, 2018: pp. 26–33, publisher: Optical Society of America.

[29] Stroescu, A.; Cherniakov, M.; Gashinova, M. Classification of High Resolution Automotive Radar Imagery for Autonomous Driving Based on Deep Neural Networks. In *2019 20th International Radar Symposium (IRS)*, June 2019, pp. 1–10, doi:10.23919/IRS.2019.8768156, iSSN: 2155-5753.

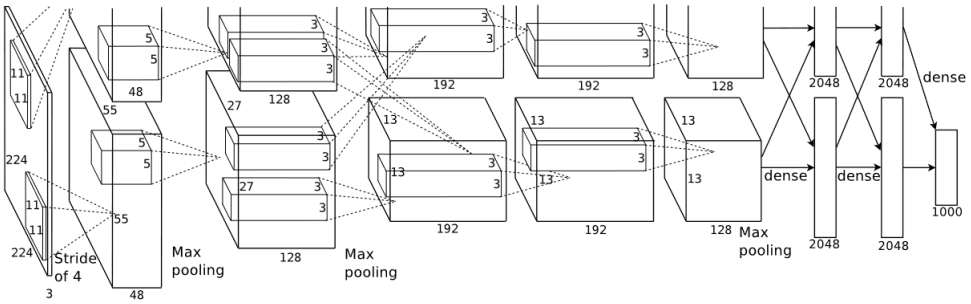
[30] Hoermann, S.; Stumper, D.; Dietmayer, K. Probabilistic Long-Term Prediction for Autonomous Vehicles. June 2017, doi:10.1109/IVS.2017.7995726.

[31] Zheng, R.; Liu, C.; Guo, Q. A decision-making method for autonomous vehicles based on simulation and reinforcement learning. In *2013 International Conference on Machine Learning and Cybernetics*, volume 01, July 2013, pp. 362–369, doi:10.1109/ICMLC.2013.6890495, iSSN: 2160-1348.

[32] Tammina, S. Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images. *International Journal of Scientific and Research Publications (IJSRP)*, volume 9, no. 10, Oct. 2019: p. p9420, ISSN 2250-3153, doi:10.29322/IJSRP.9.10.2019.p9420. Available from: http://www.ijsrp.org/research-paper-1019.php?rp=P949194

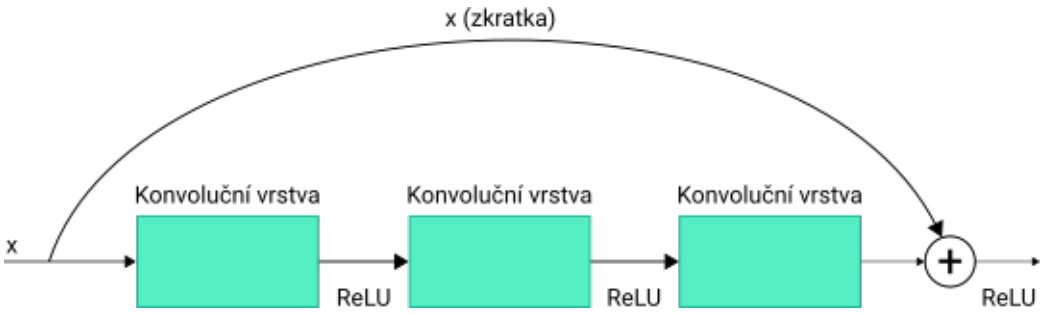
Přílohy

A Modely



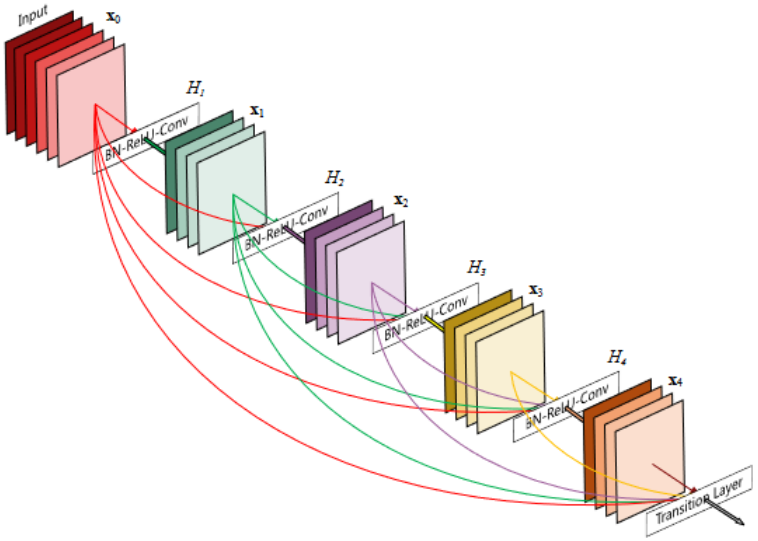
Obrázek 12: Achitektura konvoluční neuronové sítě AlexNet

Zdroj: převzato z [14]



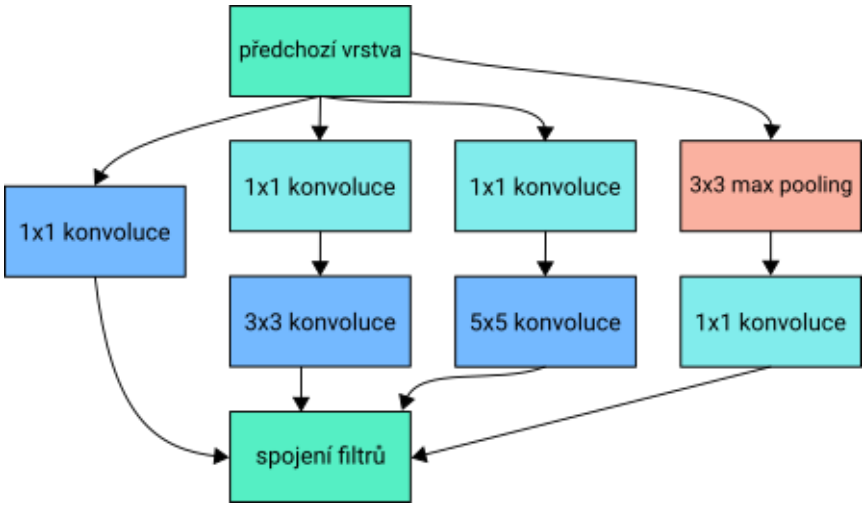
Obrázek 13: Identický reziduální blok konvoluční neuronové sítě ResNet. Výstupem zkratky i hlavní cesty je výstup o stejné dimenzi.

Zdroj: podle [15]



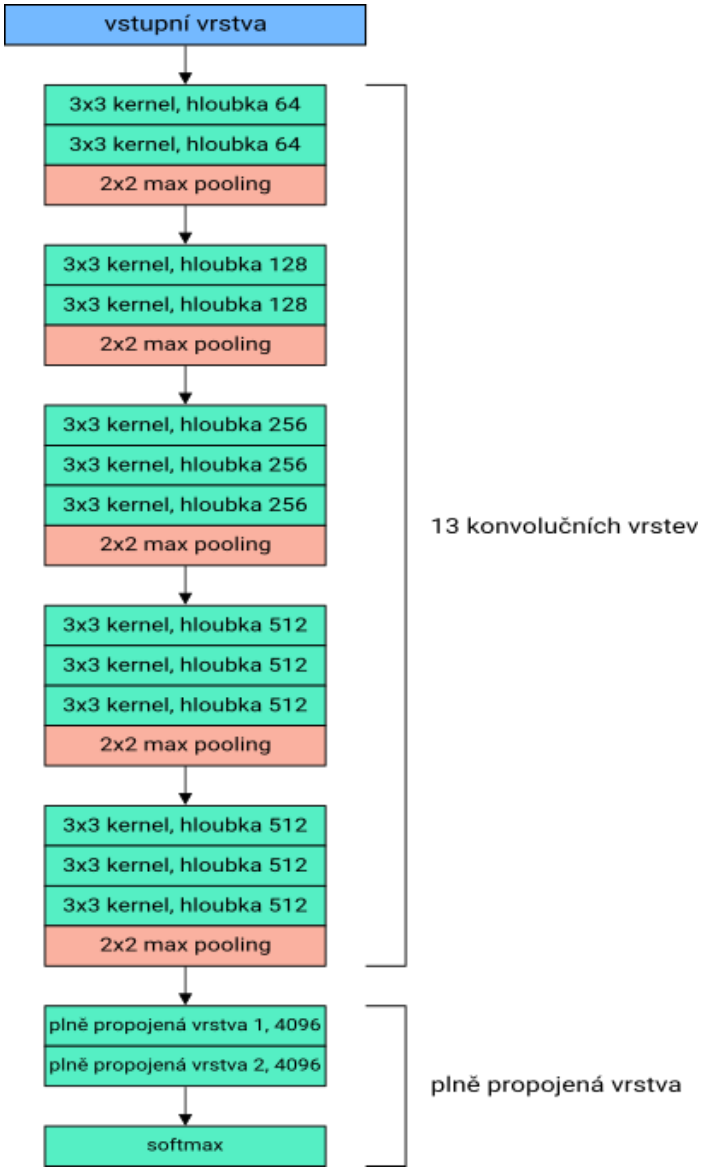
Obrázek 14: Dense blok s pěti vrstvami konvoluční neuronové sítě DenseNet

Zdroj: převzato z [16]



Obrázek 15: Inception blok s redukcí dimenze

Zdroj: podle [17]



Obrázek 16: Architektura VGG16

Zdroj: podle [32]

B Dataset

Třída	Název značky	Obrázek značky
0	Nejvyšší dovolená rychlost 20km/h	
1	Nejvyšší dovolená rychlost 30km/h	
2	Nejvyšší dovolená rychlost 50km/h	
3	Nejvyšší dovolená rychlost 60km/h	
4	Nejvyšší dovolená rychlost 70km/h	
5	Nejvyšší dovolená rychlost 80km/h	
6	Konec nejvyšší dovolené rychlosti 80km/h	
7	Nejvyšší dovolená rychlost 100km/h	
8	Nejvyšší dovolená rychlost 120km/h	
9	Zákaz předjíždění	
10	Zákaz předjíždění pro nákladní automobily	
11	Křižovatka s vedlejší pozemní komunikací	
12	Hlavní pozemní komunikace	
13	Dej přednost v jízdě!	
14	Stůj, dej přednost v jízdě!	
15	Zákaz vjezdu všech vozidel (v obou směrech)	
16	Zákaz vjezdu nákladních automobilů	
17	Zákaz vjezdu všech vozidel	
18	Jiné nebezpečí	
19	Zatáčka vlevo	

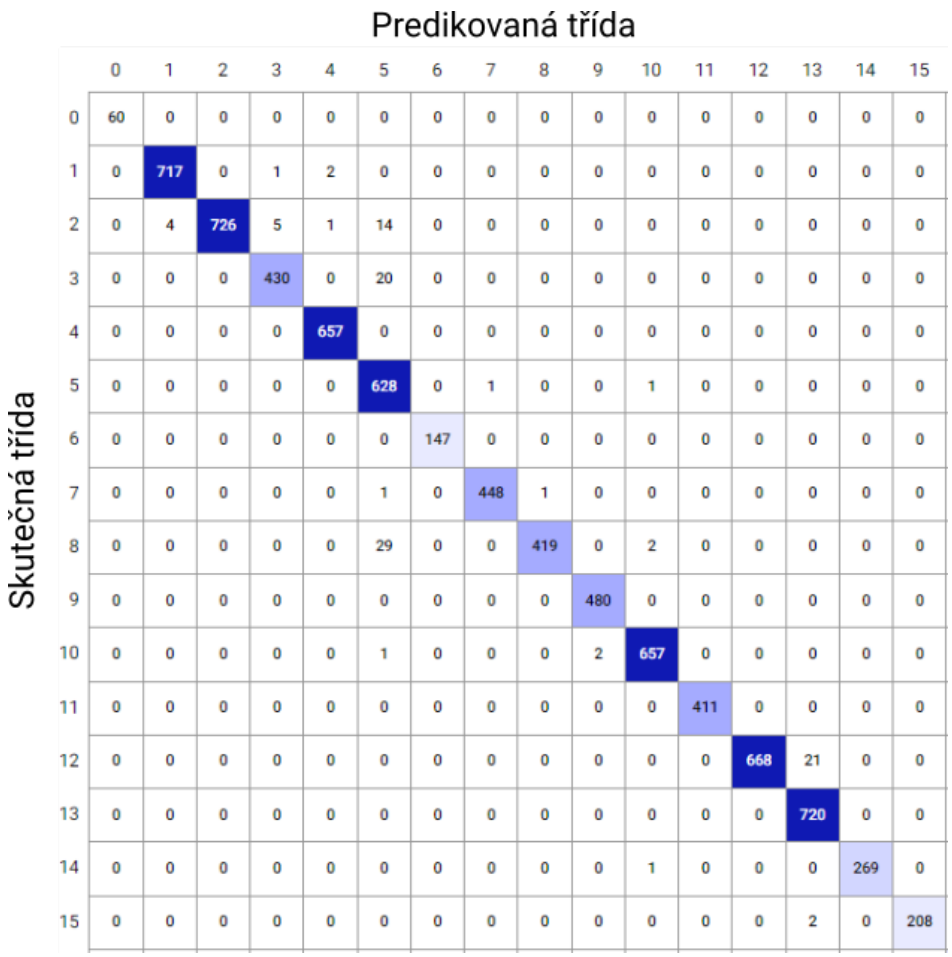
20	Zatáčka vpravo	
21	Dvojitá zatáčka, první vlevo	
22	Nerovnost vozovky	
23	Nebezpečí smyku	
24	Zúžená vozovka (z jedné strany)	
25	Práce na silnici	
26	Světelné signály	
27	Přechod pro chodce	
28	Děti	
29	Cyklisté	
30	Náledí	
31	Zvěř	
32	Konec všech zákazů	
33	Přikázaný směr jízdy vpravo	
34	Přikázaný směr jízdy vlevo	
35	Přikázaný směr jízdy přímo	
36	Přikázaný směr jízdy přímo a vpravo	
37	Přikázaný směr jízdy přímo a vlevo	
38	Přikázaný směr objíždění vpravo	
39	Přikázaný směr objíždění vlevo	
40	Kruhový objezd	
41	Konec zákazu předjíždění	

42	Konec zákazu předjíždění pro nákladní automobily	

Tabulka 7: Tabulka všech tříd a jejich názvů a obrázků značek v datasetu dopravních značek

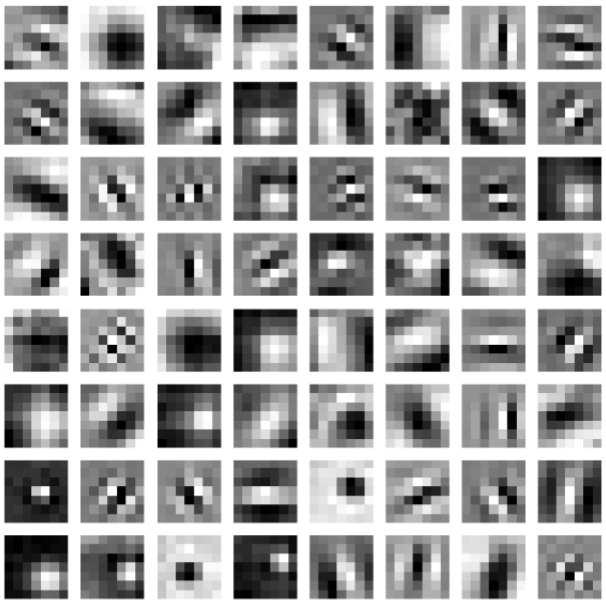
C Experimenty

C.1 ResNet50



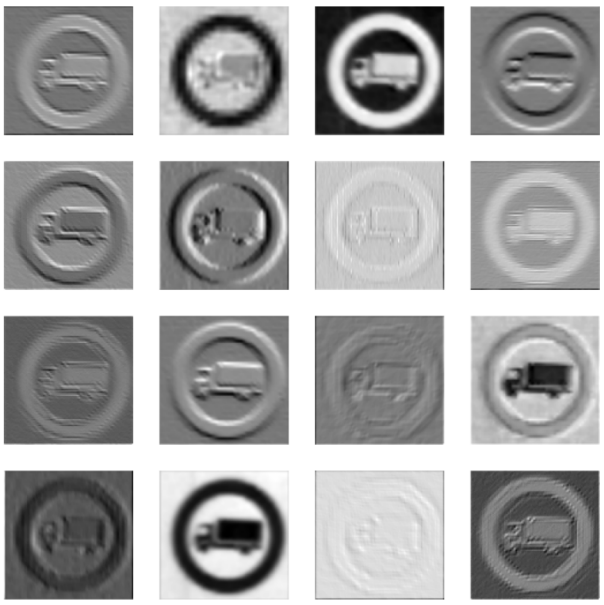
Obrázek 17: Matice záměn nejlepšího natrénovaného modelu ResNet50 (experiment 4) — prvních 16 tříd. Barevná škála značí počet testovacích dat v dané třídě (tmavě modrá indikuje více testovacích dat v dané třídě než světle modrá).

Zdroj: generováno platformou Comet.ml `https://comet.ml/`



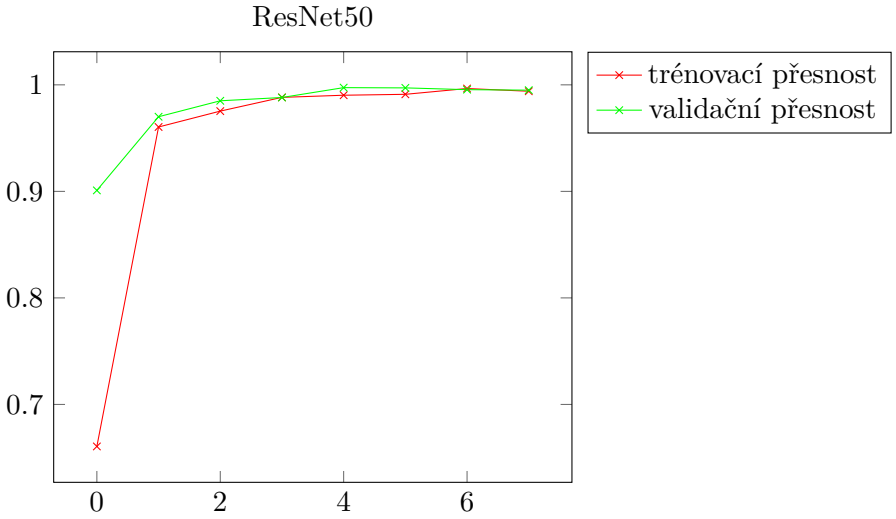
Obrázek 18: Ukázka konvolučního filtru z první konvoluční vrstvy modelu ResNet50

Zdroj: vlastní aplikace



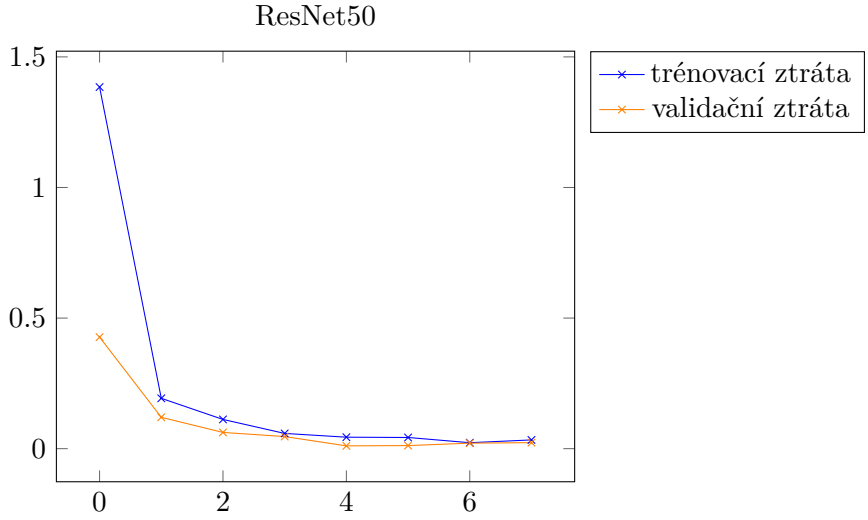
Obrázek 19: Ukázka 2D aktivační mapy ResNet50 po průchodu první konvoluční vrstvou (pouze průchod prvních 16 kernelů)

Zdroj: vlastní aplikace



Graf 10: Vývoj trénovací a validační přesnosti nejlepšího natrénovaného modelu ResNet50 (experiment 4). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje přesnost.

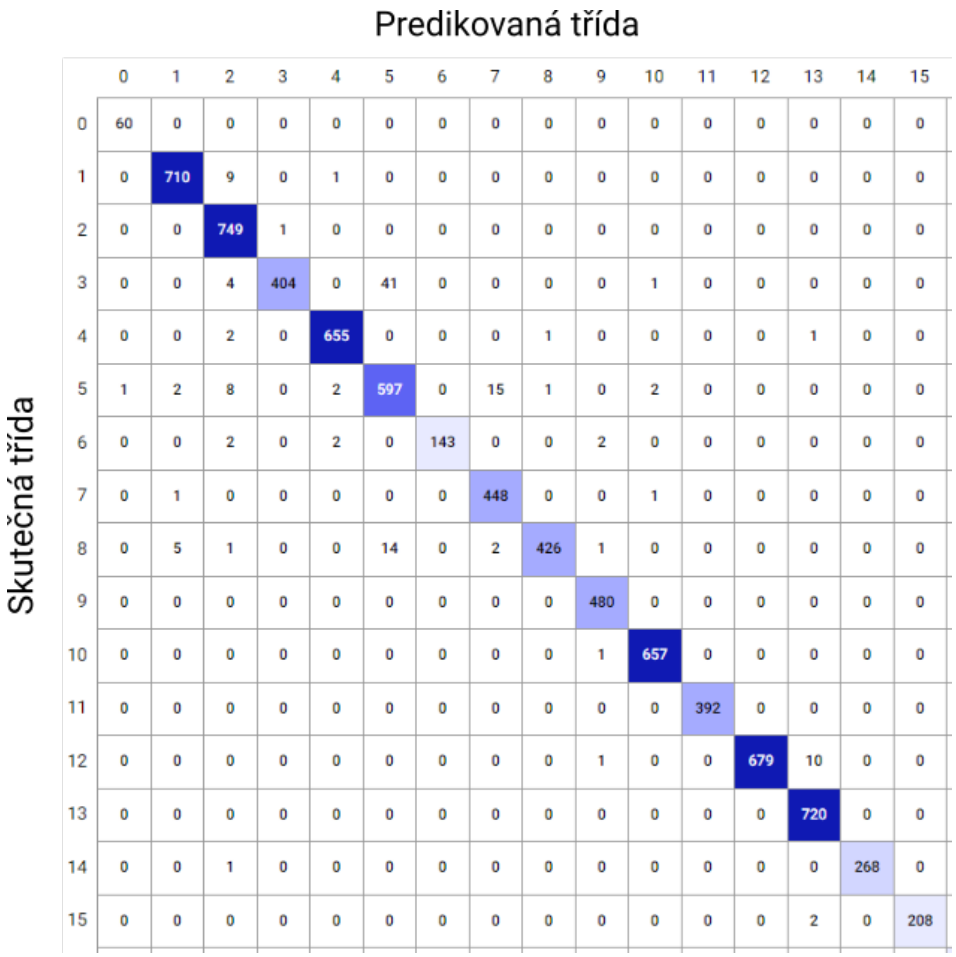
Zdroj: vlastní zpracování



Graf 11: Vývoj trénovací a validační ztráty nejlepšího natrénovaného modelu ResNet50 (experiment 4). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje ztrátu.

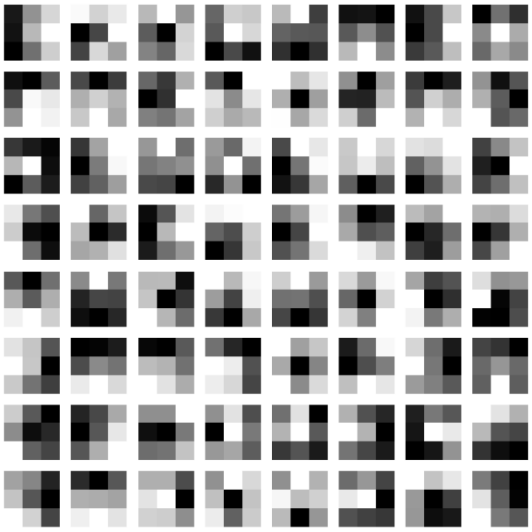
Zdroj: vlastní zpracování

C.2 VGG16



Obrázek 20: Matice záměn nejlepšího natrénovaného modelu VGG16 — prvních 16 tříd. Barevná škála značí počet testovacích dat v dané třídě (tmavě modrá indikuje více testovacích dat v dané třídě než světle modrá).

Zdroj: generováno platformou Comet.ml `https://comet.ml/`

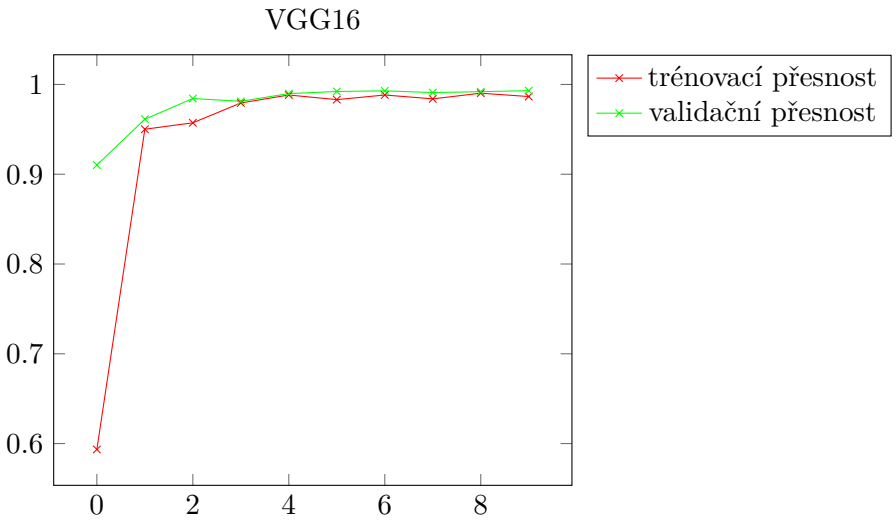


Obrázek 21: Ukázka konvolučního filtru z první konvoluční vrstvy modelu VGG16

Zdroj: vlastní aplikace

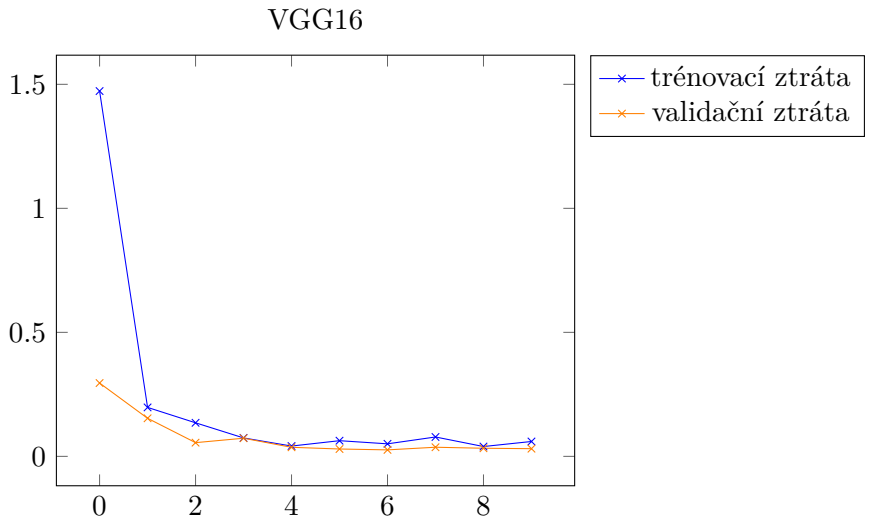
Obrázek 22: Ukázka 2D aktivační mapy VGG16 po průchodu první konvoluční vrstvou (pouze průchod prvních 16 kernelů)

Zdroj: vlastní aplikace



Graf 12: Vývoj trénovací a validační přesnosti nejlepšího natrénovaného modelu VGG16 (experiment 3). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje přesnost.

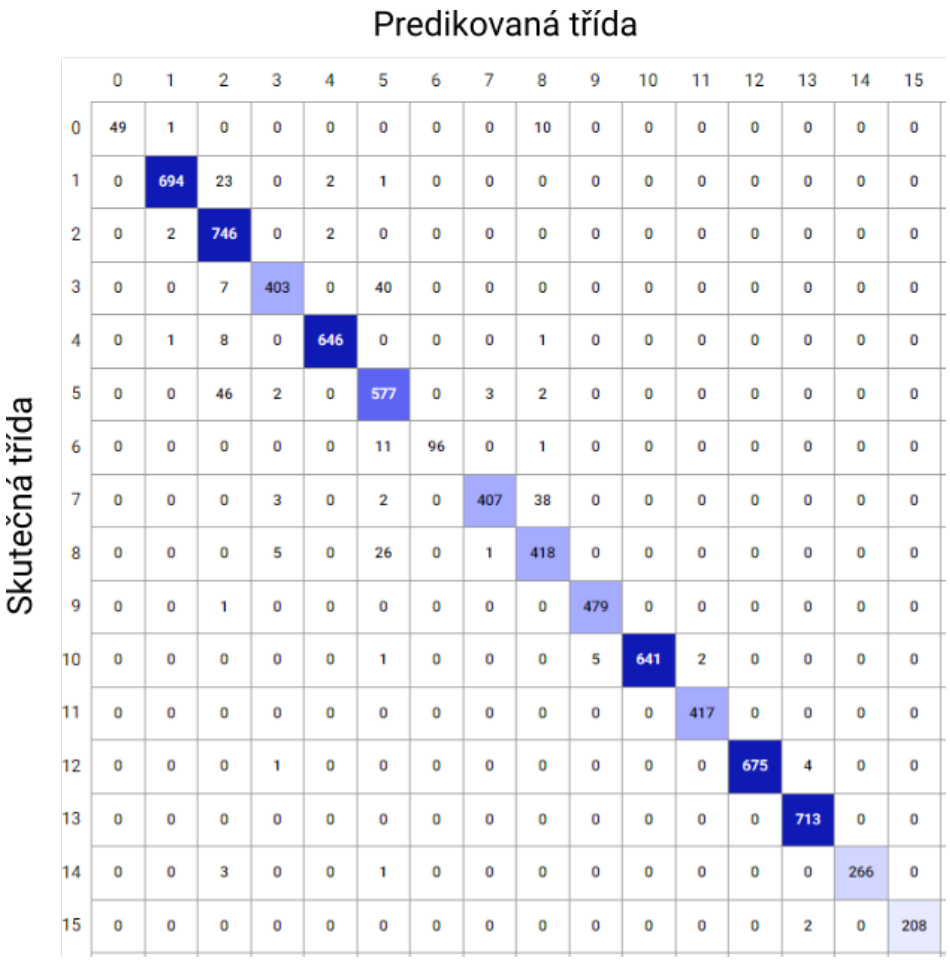
Zdroj: vlastní zpracování



Graf 13: Vývoj trénovací a validační ztráty nejlepšího natrénovaného modelu VGG16 (experiment 3). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje ztrátu.

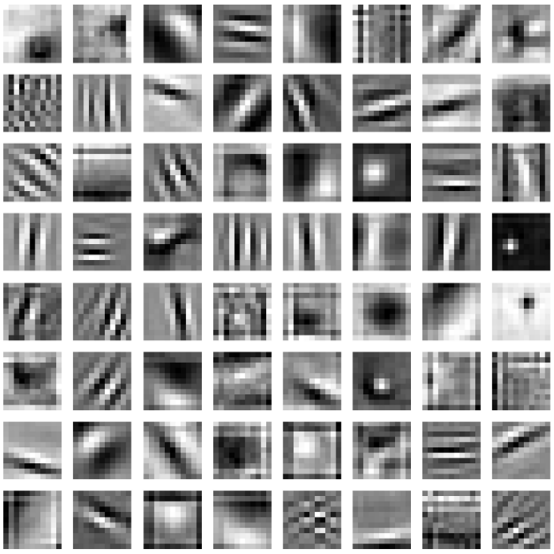
Zdroj: vlastní zpracování

C.3 AlexNet



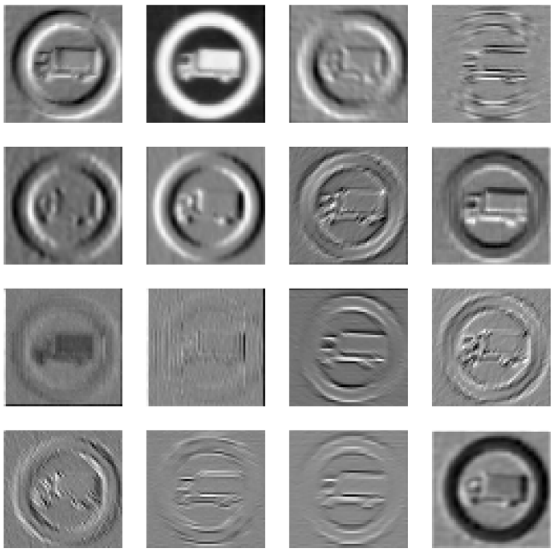
Obrázek 23: Matice záměn nejlepšího natrénovaného modelu AlexNet (experiment 1) — prvních 16 tříd. Barevná škála značí počet testovacích dat v dané třídě (tmavě modrá indikuje více testovacích dat v dané třídě než světle modrá).

Zdroj: generováno platformou Comet.ml `https://comet.ml/`



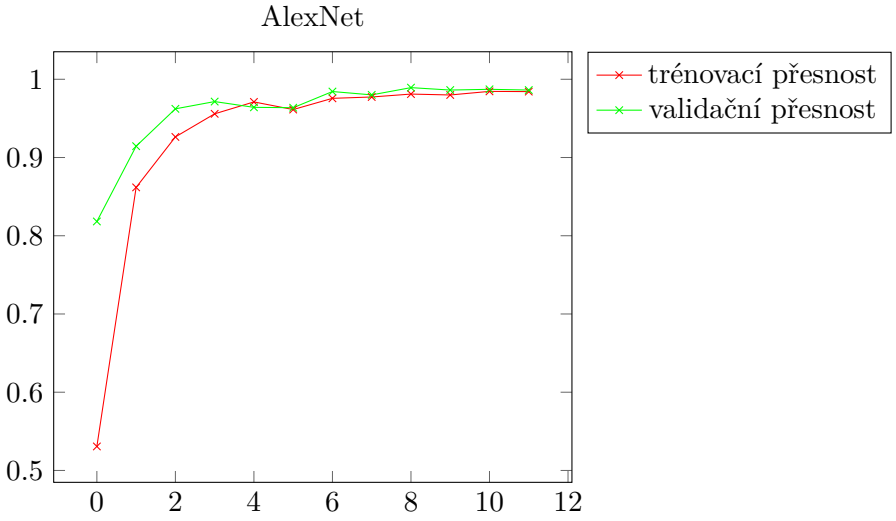
Obrázek 24: Ukázka konvolučního filtru z první konvoluční vrstvy modelu AlexNet

Zdroj: vlastní aplikace



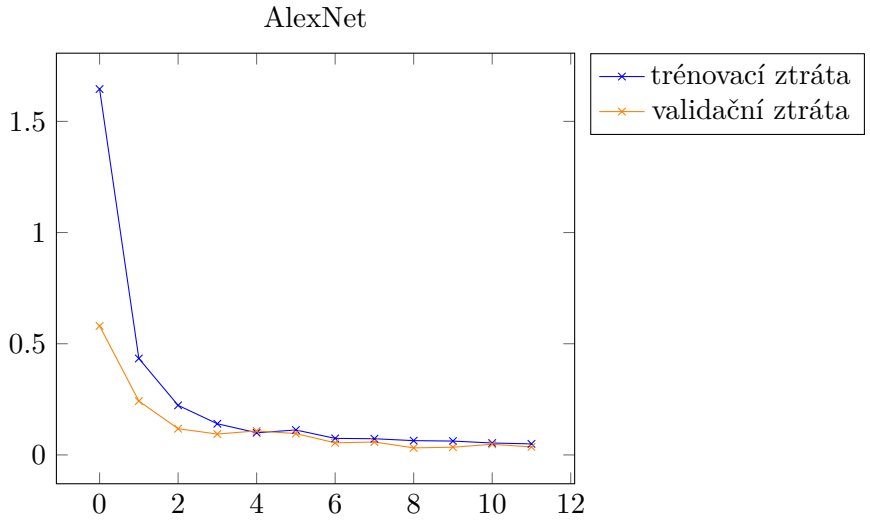
Obrázek 25: Ukázka 2D aktivační mapy AlexNet po průchodu první konvoluční vrstvou (pouze průchod prvních 16 kernelů)

Zdroj: vlastní aplikace



Graf 14: Vývoj trénovací a validační přesnosti nejlepšího natrénovaného modelu AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje přesnost.

Zdroj: vlastní zpracování

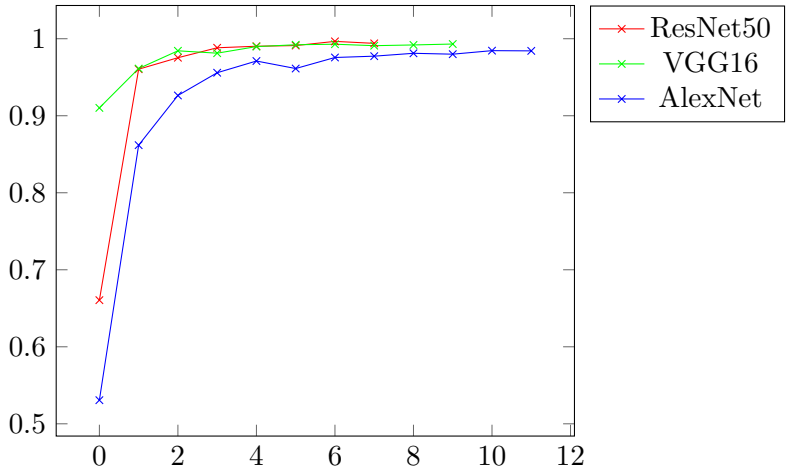


Graf 15: Vývoj trénovací a validační ztráty nejlepšího natrénovaného modelu AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje ztrátu.

Zdroj: vlastní zpracování

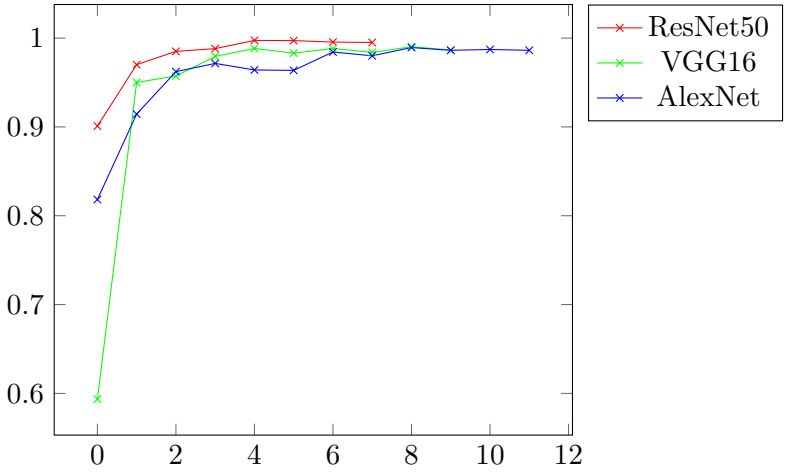
C.4 Nejlepší experimenty všech modelů

Porovnání trénovací přesnosti nejlepších experimentů



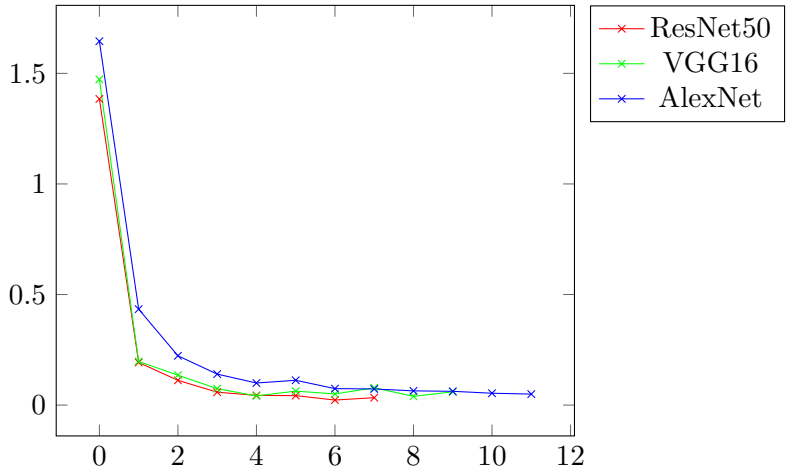
Graf 16: Vývoj trénovací přesnosti nejlepších natrénovaných modelů ResNet50 (experiment 4), VGG16 (experiment 3) a AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje trénovací přesnost.

Zdroj: vlastní zpracování



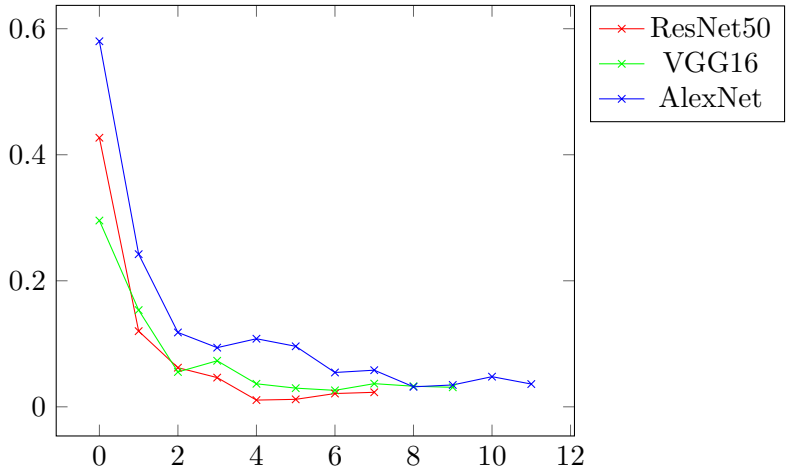
Graf 17: Vývoj validační přesnosti nejlepších natrénovaných modelů ResNet50 (experiment 4), VGG16 (experiment 3) a AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje validační přesnost.

Zdroj: vlastní zpracování



Graf 18: Vývoj trénovací ztráty nejlepších natrénovaných modelů ResNet50 (experiment 4), VGG16 (experiment 3) a AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje trénovací ztrátu.

Zdroj: vlastní zpracování



Graf 19: Vývoj validační ztráty nejlepších natrénovaných modelů ResNet50 (experiment 4), VGG16 (experiment 3) a AlexNet (experiment 1). Osa x reprezentuje epochu (první epocha začíná nulou) a osa y reprezentuje validační ztrátu.

Zdroj: vlastní zpracování

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Dennis Tschamler**
Studium:	I2000056
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Aplikace konvolučních neuronových sítí pro autonomní vozidla.**
Název diplomové práce AJ:	Application of convolutional neural networks for autonomous vehicles.

Cíl, metody, literatura, předpoklady:

Rámcová osnova práce

1. Úvod
2. Umělé neuronové sítě
3. Konvoluční neuronové sítě
4. Využití konvolučních neuronových sítí pro autonomní vozidla
5. Metodika – testování konvolučních neuronových sítí pro autonomní vozidla
6. Výsledky a diskuze
7. Závěr
8. Seznam použité literatury
9. Přílohy

- Java Deep Learning Cookbook: Train neural networks for classification, NLP, and reinforcement learning using Deeplearning4j (2019) - Rahul Raj (ISBN-13: 978-1788995207)
- A Guide to Convolutional Neural Networks for Computer Vision (2018) - Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah (ISBN-13: 978-1681730219)
- Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection (2019) - Umberto Michelucci (ISBN-13: 978-1484249758)

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	prof. RNDr. PhDr. Antonín Slabý, CSc.

Datum zadání závěrečné práce: 9.9.2021