
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INTELIGENTNÍCH SYSTÉMŮ**
**DEPARTMENT OF INTELLIGENT SYSTEMS**


# URČENÍ TYPU A SMĚRU ZBRANĚ V OBRAZOVÉ SCÉNĚ

**DETERMINATION OF GUN TYPE AND POSITION IN IMAGE SCENE**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **RÓBERT KOLCÚN**
**AUTHOR**

**VEDOUCÍ PRÁCE** **Prof. Ing., Dipl.-Ing. MARTIN DRAHANSKÝ, Ph.D.**
**SUPERVISOR**



**BRNO 2018**

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav inteligentních systémů Akademický rok 2017/2018

# Zadání bakalářské práce

Řešitel: **Kolcún Róbert**
Obor: Informační technologie
Téma: **Určení typu a směru zbraně v obrazové scéně**
**Determination of Gun Type and Position in Image Scene**
Kategorie: Umělá inteligence

Pokyny:

1. Prostudujte literaturu týkající se výskytu objektů v obrazu a seznamte se s algoritmy pro jejich detekci a rozpoznávání.
2. Navrhněte algoritmický postup pro stanovení typu zbraně (krátká, dlouhá, vč. příp. dalšího jemnějšího dělení) a jejího natočení ve scéně.
3. Postup navržený v předchozím bodu implementujte. Proveďte otestování Vašeho řešení.
4. Shrňte dosažené výsledky a diskutujte možnosti budoucího vývoje.

Literatura:

- Olmos R., Tabik S., Herrera S. *Automatic Handgun Detection Alarm in Videos Using Deep Learning*. Neurocomputing, 2017, DOI https://doi.org/10.1016/j.neucom.2017.05.012.
- Lai J., Maples S. *Developing a Real-Time Gun Detection Classifier*. Dostupný on-line: http://cs231n.stanford.edu/reports/2017/pdfs/716.pdf.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Drahanský Martin, prof. Ing., Dipl.-Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
*vedoucí ústavu*

# Abstrakt

Cieľom tejto práce je navrhnúť spôsob pre klasifikáciu typu zbrane do dvoch kategórií na krátke a dlhé, a určenie náklonu zbrane v obrazovej scéne. Zvolený problém som riešil pomocou niekoľkých postupov, použitím klasických postupov ako K-Nearest-Neighbour a SVM klasifikátora, a použitím konvolučných neurónových sietí. V práci bolo vytvorené porovnanie týchto postupov s výslednou presnosťou až 90%. Výsledky tejto práce umožnujú vybrať správny postup pre riešenie daného problému.



# Abstract

The main goal of this work is to design an approach for classifying guns into two categories, with short and long weapons, and determining a position of guns in the image scene. This problem was solved using the classical approach as K-Nearest-Neighbour and SVM classification and using convolutional neural networks. In this work was made a comparison of these approaches with a resulting accuracy up to 90%. With the results of this work, it is possible to choose a right approach to this problem.



# Kľúčové slová

zbraň, klasifikácia, pozícia, obrazová scéna, konvolučná neurónová sieť, svm, kmeans

# Keywords

gun, classification, position, image scene, convolutial neural network, svm, kmeans


# Citácia

KOLCÚN, Róbert. *Určení typu a směru zbraně v obrazové scéně.* Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing., Dipl.-Ing. Martin Drahanský, Ph.D.

# Určení typu a směru zbraně v obrazové scéně

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Profesora Martina Drahanského. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Róbert Kolcún
15. mája 2018

## Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce pánovi Profesorovy Martinovy Drahanskému za poskytnutú pomoc a konzultácia počas tvorby tejto práce.

# Obsah

# Kapitola 1

# Úvod

V dnešnej dobe sa so strelnými zbraňami stretávame pomerne často, či sú to už rôzne akčné počítačové hry, voľno časové aktivity ako airsoft či paintball, ale taktiež ich môžeme denno-denne vidieť na uliciach u príslušníkov bezpečnostých zložiek štátu. Z tohto hľadiska je bezpečnosť veľmi dôležitý prvok nášho života a zbrane k tomu v istej prieme prispievajú, rovnako ako v pozívnom ale aj v negatívnom smere.

Hlavne kvôli negatívnemu vplyvu zbraní sa bežpečnostné zložky snažia znižovať toto riziko a dopad na ľudské zdravie. Je možné s určitosťou povedať že technológie, ktoré by dokázali vyhľadávať zbrane u ľudí na uliciach alebo iných miestach, by určite pomohli zvýšiť bezpečnosť danej oblasti alebo krajiny. V tomto prípade dokážu počítačové technológie a počítačove videnie pomôcť.

Preto sa v tejto bakalárskej práci budeme konkrétne venovať určovaniu typu strelnej zbrane v dvoch kategóriách a to na krátke a dlhé zbrane, a určeniu ich náklonu v obrazovej scéne.

V prvej kapitole sa zameriame na rozdelenie zbraní podľa rôznych kritérií, následne si uvedieme niekoľko významných prístupov k detekcií a klasifikácií objektu v obraze, ktoré vrámci spracovania obrazu vytvorili istý prevrat. A uvedieme si niekoľko nástrojov ktoré takého spracovanie obrazu uľahčujú.

V druhej kapitole si následne opíšeme podrobný návrh postupov, ktoré v tejto práci budú implementované a otestované. Spôsoby a možnosti získavania dát pre spracovanie obrazu.

Ďalej sa v tretej kapitole budeme venovať presnému opisu implementácie navrhovaného riešenia.

A následne sa v posledných kapitolách pozrieme na výsledky, porovnáme navrhnuté riešenia a zhodnotíme možný budúci postup tejto práce.

# Kapitola 2

# Klasfikácia a detekcia zbraní v obraze

V tejto kapitole sa zoznámime so základnými pojmami, princípmi a spôsobmi, ktoré súvisia s problematikou tejto práce a ktoré budú ďalej využívané. Kapitola opisuje základné rozdelenia zbraní do viacerých kategórií, princíp spracovania digitálneho obrazu. Následne vysvetľuje z akých problémov pozostáva detekcia a rozpoznávanie objektov v obraze, opisuje populárne prístupy pre detekciu objektu. A podrobne popisuje rôzne spôsoby pre klasifikáciu objektov do dvoch alebo aj viacerých kategórií, ktoré sa v tejto práci budú používať.

## 2.1 Zbrane

Obvyklá definícia hovorí, že zbraň je nástroj, predmet, či dokonca celé zariadenie, ktoré je prispôsobené k vyvolaniu zranenia na živý organizmus alebo k ničeniu objektu [21]. Za prvé zbrane môžeme považovať kopije, ktoré používali ľudia pri love zvierat asi pred 400,000 rokmi [39].

Vo všeobecnosti môžeme zbrane rozdeliť podľa mnohých kritérií, napr. podľa zdroja energie použitej k vypudeniu projektilu zo zbrane, podľa konštrukcie a režimu streľby, ďalej z hľadiska postupu pri nabíjaní alebo podľa veku zbrane, na nové - slúžiace svojmu účelu a historické - ktoré sú už nespôsobilé k pôvodnemu účelu. My sa zameriame na 2 základné rozdelenia a to podľa toho, ako zbrane pôsobia na živú silu, delíme na [21]:

- **Strelné** - rozrušujú vzdialený cieľ, živý alebo neživý, prodstredníctvom dopadovej energie strely vypudenej zo zbrane,
- **Chladné** - účinkujú bodom alebo sekom naostrenej čepele, ktorá je vsadená do rukoväťe alebo je nasadená na tyč či porísko,
- **Úderné** - pôsobia na živý objekt tupým úderom svojej časti, ktorá býva spojená s vhodnou rukoväťou,

a podľa ovládateľnosti a možnosti prenášania ich delíme na [21]:

- **Ručné** strelné zbrane môže prenášať a ovládať jediná osoba. Sú ovládané buď jednou rukou - krátke zbrane - alebo oboma rukami - dlhé zbrane,
- **Lafetované** zbrane musia byť vzhľadom ku svojej hmotnosti a rozmerom umiestnené na zvláštnom podstavci - *lafete*. Takúto zbraň takisto väčšinou obsluhuje viac ľudí.

V tejto práci sa zameriame na strelné a ručné zbrane s rozdelením na krátke a dlhé.

## 2.2 Spracovanie obrazu

Pre popis obrázkov a ostatných signálov sú často používané matematické modely. Kde signál je funkcia závislá na určitých premenných s fyzikálnym významom, môže byť 1-dimenzionálna (napr. závislá na čase), 2-dimenzionálna (napr. obrázok závislý na 2 koordinátoch v ploche), 3-dimenzionálna (napr. popis pozície objektu v priestore), alebo aj viac-dimenzionálna [37].

Každý obraz môže byť definovaný ako spojitá funkcia s dvomi neznámymi $f(x, y)$ kde $x$ a $y$ sú súradnice v ploche. Tento spojitý obraz je digitalizovaný na tzv. vzorkovacích miestach. Tieto vzorkovacie miesta sú usporiadané v ploche, ich geometrický vzťah sa nazýva mriežka. Digitálny obraz je potom dátova štruktúra, ktorá je bežne reprezentovaná ako matica. Jeden bod v mriežke reprezentuje jeden element 2-dimenzionálneho obrazu nazývany pixel, v 3-dimenzionálnom obraze sa tento element nazýva voxel [37]. Pri viac-dimenzionálnych digitálnych obrazoch sa pri spracovaní obrazu používa vektor hodnôt(napr. RGB hodnoty obrazového bodu).

Oblasť spracovania digitálneho obrazu je v dnešnej dobe veľmi široká a nachádza uplatnenie vo viacerých oboroch. Môže sa využívať pri automatickej vizuálnej inšpekcii produktov, pre zaistenie vyššej produktivity a kvality výrobku v továrňach. Ďalej pri spracovaní snímkov z lietadiel alebo satelitov pre získanie dát o prírodných zdrojoch, ako napr. v poľnohospodárstve alebo lesníctve. Širokú aplikáciu má v medicíne pri obrázkoch získavaných pomocou röntgenových zariadení, CT a magnetickej rezonancie [40]. A v súčasnosti taktiež v automobilovom priemysle pri rozvíjajúcej sa oblati autonómneho riadenia automobilov.

## 2.3 Detekcia objektu v obraze

V prvom rade je potrebné povedať že detekcia objektu a klasifikácia v obraze je veľmi dobre známy problém v oblasti spracovania obrazu. V dnešnej dobe už niektoré modely prebehli svojou presnosťou a výkonom aj človeka [22].

Rozoberaním problému lokalizácie a správnej klasifikácie objektu v obraze, skončíme s potrebou detekcie a klasifikácie niekoľkých objektov súčasne v jednej scéne. Detekcia objektu v obraze je problém pozostavajúci z hľadania a klasifikovania variabilného počtu a rôznych typov objektov v obraze. Dôležitým rozdielom je "variabilná" časť. V porovnaní s klasifikáciou je výstup detekcie objektov rôznorodý svojou dĺžkou, keďže počet objektov v obraze sa môže meniť z obrázka na obrázok [34].

### 2.3.1 Prehľad existujúcich riešení

#### Klasický prístup

Aj keď v priebehu rokov bolo množstvo rôznych typov riešení, pre túto prácu je vhodné spomenúť 2 hlavné prístupy.

Prvý z nich je Viola-Jones spôsob, ktorý navrhol v roku 2001 Paul Viola a Michael Jones v práci [41]. Tento prístup je rýchly a relatívne jednoduchý, až natoľko, že sa tento algoritmus implementuje v kamerách s bodovým snímačom, ktorý umožňuje detekciu tvári v reálnom čase s malým množstvom potrebného výkonu. V jednoduchosti, algoritmus generuje rôzne, môže až tisíce, jednoduché binárne klasifikátory pomocou tzv. Haar funkcií.

Tieto klasifikátory sú kaskádovito zoradené a vyhodnocované v poradí podľa ich zložitosti, s využitím viacnásobného posuvného okna [eng. multi-scale sliding window]. V prípade negatívnej klasifikácie v ktorejkoľvek úrovni vyhodnocovania, je tento proces ukončený a pokračuje sa klasifikáciou nad ďalším podoknom (angl. *subwindow*) [41].

Druhý spôsob je klasifikácia pomocou histogramu orientovaných prechodov (angl. *histogram of oriented gradients*) a Support Vector Machine, ktorý bude podrobne rozobratý v kap. 2.4. Tento prístup takisto používa viacnásobné posuvné okno, avšak aj keď je lepší ako Viola-Jones, je oveľa pomalší [34].

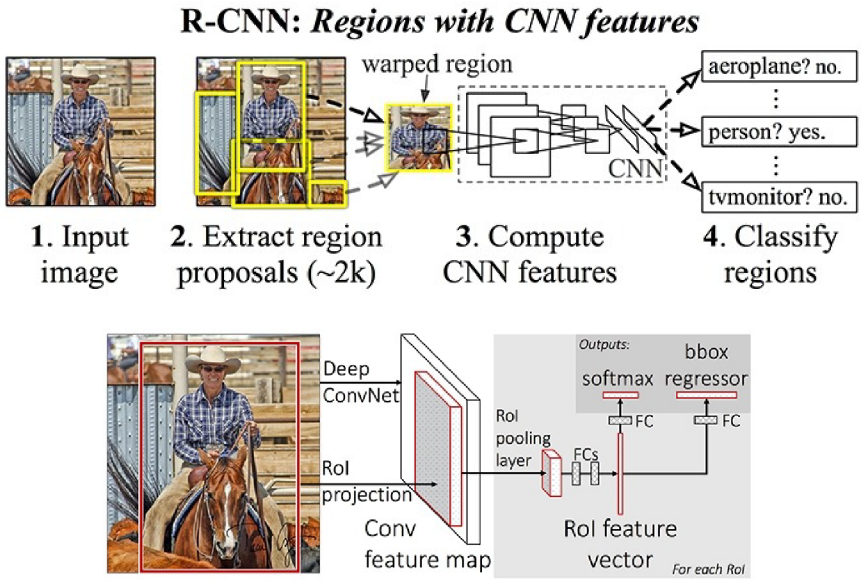
### Prístup pomocou hlbokých neurónových sietí

S príchodom neurónových sietí prišla aj veľká zmena v oblasti spracovania obrazu, preto je vhodné spomenúť niekoľko spôsob detekcie objektov ktoré stavajú na neurónových sieťach.

**R-CNN** je jeden z prvých riešení [24], v ktorom navrhovali 3 stupňový prístup:

- extrahovať možné objekty pomocou metódy regionálnych návrhov (angl. *region proposal*),
- extrahovať príznaky z každého regiónu pomocou konvolučných neurónových sietí,
- klasifikovať každý región pomocou Support Vector Machine.

**Fast R-CNN** je označenie pre prístup, ktorý vylepšil predchádzajúci R-CNN, základom je, že stavia už iba na využití konvolučných neurónových sietí. V práci použili dva prístupy k detekcii objektu, sliding window a region proposal. Výsledky ukázali, že prístup pomocou regionálnych návrhov (angl. *region proposals*) bol rýchlejší a dosiahli rýchlosť spracovania až 2 snímkov za sekundu [23].




Obr. 2.1: Porovnanie architektúr R-CNN(hore) a Fast R-CNN(dole) [34].

**YOLO** (angl. *You Only Look Once*), rieši problém detekcie objektu v obraze ako regresívny problém. Namiesto bežného postupu ako je najprv použitie techniky regionálnych

návrhov a následne klasifikovanie týchto regiónov. To vedie YOLO k veľmi rýchlemu spracovaniu v reálnom čase ale za cenu presnosti. Týmto spôsobom YOLO dokáže dosiahnuť 63.4% mAP, kde mAP je metrika pre určenie presnosti detekcie objektu v obraze [20], s 22 ms oneskorením [33].

### 2.3.2 Kĺzajúce okno

Kĺzajúce okno (angl. *Sliding window*) je jedna z metód pre detekciu objektov v obraze. Táto metóda je veľmi vyčerpávajúca pretože zakladá na veľkom počte kandidátnych okien, až $10^4$, vo vstupnom obrázku. Metóda prechádza vstupný obrázok na všetkých lokáciách s rôznou veľkosťou okna, následne sa na každom "okne" spúšťa klasifikátor. Nevýhodou tohto prístupu je jeho časová náročnosť, preto sa tento spôsob nedá použiť pri spracovaní obrazu v reálnom čase. Ale na druhú stranu metóda poskytuje dobrú presnosť pri kvalitnom klasifikátore [32].

### 2.3.3 Regionálne návrhy

Metóda regionálnych návrhov (angl. *region proposals*) na rozdiel od metódy kĺzajúceho okna, nepredpokladá za kandidátne okná všetky možnosti. Tieto kandidátne okná sú vybrané pomocou metód návrhu detekcie (angl. *detecion proposel methods*), konkrétne metódy a ich porovnanie je možné nájsť v článku *What makes for effective detection proposals?* [26]. Prvý model na detekciu objektov, ktorý použil konvolučné neurónové siete s touto metódou pre výber okien bol R-CNN [32].

## 2.4 Rozpoznávanie a klasifikácia

Ako bolo spomenuté v úvode v kapitole 2.3, klasifikácie je jeden z aspektov pri detekcii a rozpoznávaní objektov v obraze. V tejto podkapitole bude podrobne popísaných niekoľko algoritmov, pomocou ktorých je možné objekty z obrazu klasifikovať do kategórií.

### 2.4.1 K-Nearest-Neighbor

$k$-Nearest Neighbor je algoritmus, ktorý sa učí pod dozorom (angl. *supervised learning algorithm*) často používaný pri rozpoznávaní vzorov v klasifikácii, avšak je možné ho použiť aj pre odhad a predikciu [29]. Algoritmus je pamäťovo náročný [eng. memory-based] a nepotrebuje žiaden trénovací model. Pre jeho fungovanie nie je potrebný žiaden explicitný postup trénovania, okrem zberu vektorov príznakov s označeniami tried do ktorých patria.

Klasifikácia dát prebieha v 2 krokoch: nájdenie $k$ najbližších susedov, spomedzi trénovaných dát a vykonanie "väčšinového hlasovania" medzi nájdenými susedmi pre priradenie najčastejšie sa vyskytovaného označenia triedy.

Nech $\{(x_i, y_i); i = 1, 2, \ldots, n\}$ je množina trénovacích dát, kde $x_i$ je vektor príznakov a $y_i$ je názov triedy, do ktorej patrí vektor $x_i$. Predpokladáme, že každé $x_i$ je v určitom multidimenzionálnom priestore príznakov s metrikou $P$ a $y_i \in \{1, 2, \ldots, l\}$, kde $y_i$ je číslo zodpovedajúcej triedy. Cieľom je priradiť neoznačený vektor $x$ do zodpovedajúcej triedy z množiny $\{1, 2, \ldots, l\}$.

Najjednoduchšia verzia algoritmu $k$-NN je 1-NN, kde vektor $x$ je priradený najbližšiemu susedovi. To znamená, že ak $x_j$, kde $j \in \{1, 2, \ldots, n\}$, je najbližší k $x$ vo forme vzdialenosti $P$ [38]:

$$x_j = arg\ min_{\{x_i, 1 \leq i \leq n\}} P(x, x_i) \tag{2.1}$$

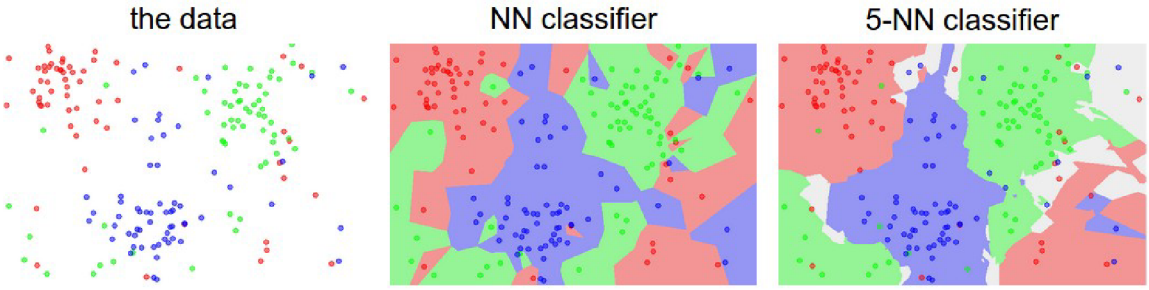
tak označenie triedy pre vektor $x$ je číslo $y_i$.

Pre formu algoritmu $k$-NN, kde $k > 1$ je postup podobný, ale priradenie označenia triedy pre $x$ je na základe najčastejšie sa vyskytovaného označenia triedy spomedzi $k$ najbližších susedov z trénovacích bodov $x_i$, kde $k$ je užívateľom definovaná konštanta [38].

Najbežnejší výpočet pre vzdialenosť bodov je pomocou Euklidovskej vzdialenosti. Táto vzdielnosť $d_{a,b}$ je medzi dvoma $J$-dimenzionálnymi vektormi $a$ a $b$ vyjadrená ako [38]:

$$d_{a,b} = \sqrt{\sum_{j=1}^{J} (a_j - b_j)^2} \tag{2.2}$$

Na obrázku 3.3 je zobrazený rozdiel medzi 1-NN a 5-NN algoritmom pre klasifikáciou, použitím 2-dimenzionálnych bodov a 3 tried dát (červená, modrá, zelená). Farebné regióny vyznačujú rozhodovacie hranice klasifikátora, ktorý využíva Euklidovskú vzdielnosť. Biele oblasti ukazujú body, ktoré sú nejednoznačne klasifikované (to znamená, že hodnotenie triedy je viazané aspoň na dve triedy). V ukážke je vidno, že v prípade 1-NN klasifikátora, niektoré body vytvárajú "malé ostrovy" (napr. zelený bod v strede mraku medzi modrými bodmi), zatiaľ čo 5-NN klasifikátor vyhladuje tieto nerovnomernosti, a pravdepodobne vedie k lepšiemu zovšeobecneniu nad testovacími údajmi.




Obr. 2.2: Porovnanie k-NN klasifikátorov [8].

### 2.4.2 Support Vector Machines

Support Vector Machines sú metódy učenia používané pre binárnu klasifikáciu. Základnou myšlienkou je nájdenie hyper–roviny, ktorá perfektne oddelí $d$–dimenzionálne dáta do dvoch tried [16]. SVM sa snaží maximalizovať vzdialenosť medzi rozdeľujúcou rovinou a dátami nachádzajúcimi sa v každej z 2 polrovín [28]. Avšak keďže vstupné dáta väčšinou nie sú lineárne separovatelné, SVM predstavujú pojem *kernel induced feature space*, ktorý prevádza dáta do vyššieho dimenzionálneho priestoru, kde sú dáta oddeliteľné.[16].

Ak trénovacie dáta sú lineárne separovatelné, tak existuje dvojica $(w, b)$, kde $w$ je váhový vektor, $b$ je predpoveď (alebo $-b$ je prahová hodnota) a $x_i$ vstupná hodnota, ako [28]:

$$w^T * x_i + b \geq 1, \; pre \; x_i \in P \tag{2.3}$$

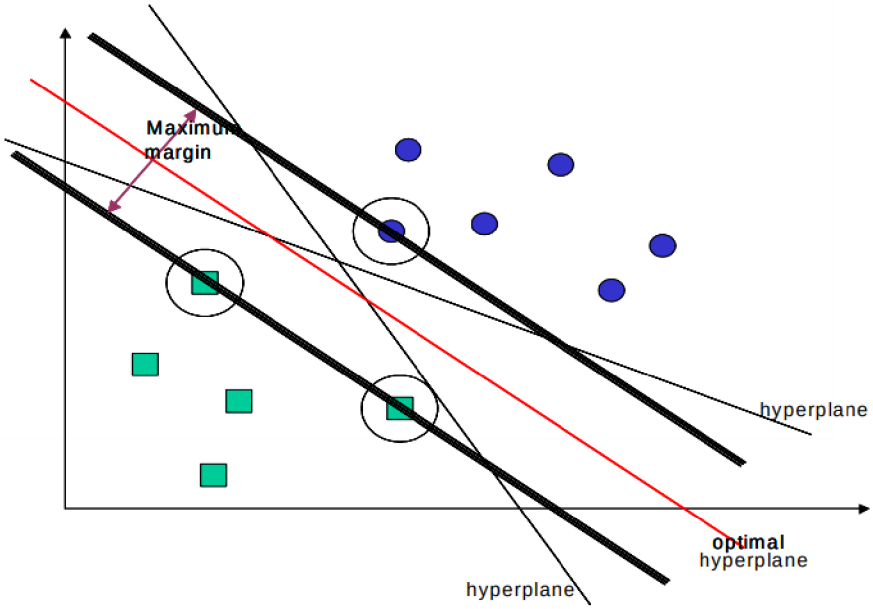
$$w^T * x_i + b \leq -1, \; pre \; x_i \in N \tag{2.4}$$

s rozhodovacím pravidlom, kde funkcia *sgn* je funkcia znamienka (angl. *sign function*).

$$f_{w,b}(x) = sgn(w^T x + b) \tag{2.5}$$

V tomto prípade, keď je možné lineárne rozdeliť dve triedy, tak optimálna hyper–rovina pre rozdelenie môže byť nájdena, ako minimalizácia kvadratickej formy rozdeľujúcej hyper–roviny

V tomto prípade lineárneho rozdelenia dát, pri nájdení optimálnej rozdeľujúcej hyper–roviny, dátove body, ktoré ležia na jej okraji sa nazývajú podporné vektorové body (angl. *support vector points*) a riešenie je reprezentované ako lineárna kombinácia iba 3 týchto bodov (viď. obrázok 2.3 ). Ostatné body sú ignorované [28].



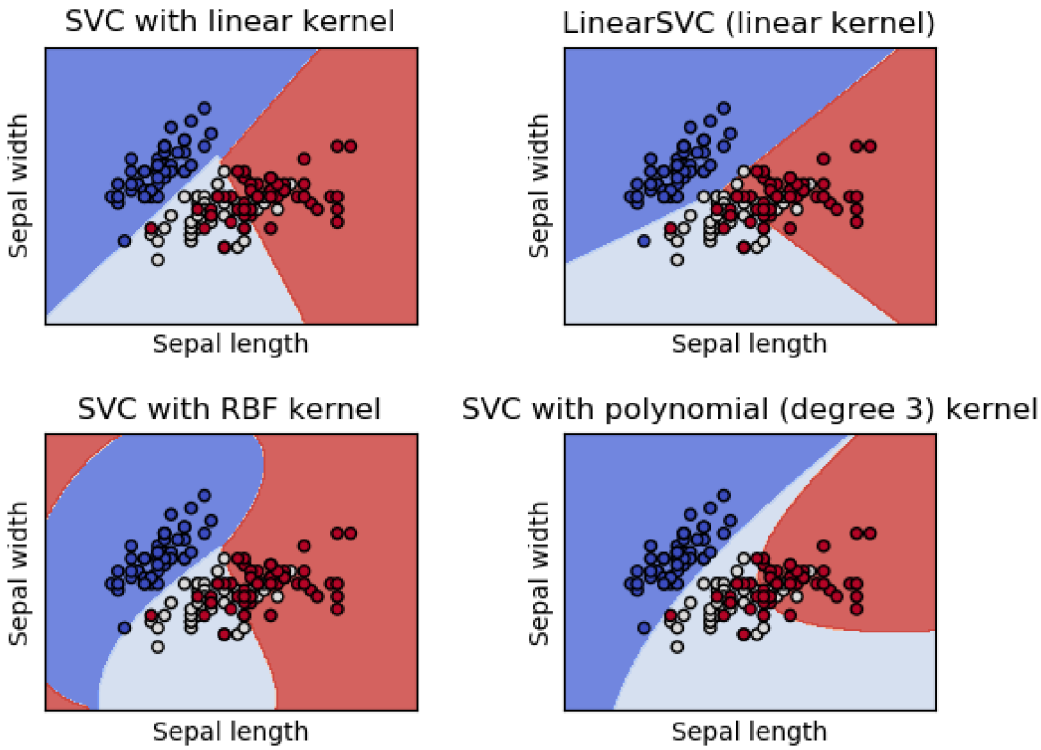

Obr. 2.3: Maximálne rozpätie [28].

Avšak väčsina reálnych problémov zahŕňa nelineárne rozdelenie dát pre, ktoré neexistuje žiadna hyper–rovina, ktorá by úspešne rozdelila trénovacie dáta. Riešenie tohto problému je mapovať dáta do vyššie–dimenzionálneho priestoru a definovať tam rozdeľujúcu hyper–rovinu. Tento vyšší-dimenzionálny priestor je tzv. transformovaný priestor príznakov (angl. *transformed feature space*), ktorý je určený ako opak ku vstupnému priestoru (angl. *input space*) obsahujúcemu trénovacie dáta [28].

Pri vhodne zvolenom transformovanom priestore príznakov dostatočnej veľkosti, môžu byť všetky trénovacie dáta rozdelitelné. Lineárne rozdelenie v tomto priestore zodpovedá nelineárnemu rozdeleniu v pôvodnom vstupnom priestore.

Jadrá(angl. *kernels*) sú špeciálnou triedou funkcií, ktoré dovoľujú, aby sa táto reprezentácia dát vypočítala priamo vo funkčnom priestore (angl. *feature space*) bez toho, aby sa vykonalo vyššie popísané mapovanie do určitého Hilbertovho priestora [18]. Po vytvorení správnej hyper–roviny sa funkcia jadra použije na mapovanie nových bodov do funkčného priestoru pre klasifikáciu [28].

Voľba správnej funkcie jadra je veľmi dôležitá, keďže definujú transformovaný priestor príznakov v ktorom budú trénovacie dáta klasifikované. Genton [17] popísal niekoľko tried jadier, avšak neadresoval otázku, ktorá trieda je najvhodnejšie pre daný problém.




Obr. 2.4: Porovnanie SVM klasifikátora s použitím rôznym jadier [10].

### 2.4.3 Neurónové siete

Neurónová sieť (angl. *neural network*) (NN) je masívne paralelný procesor, ktorý má sklon k uchovávaniu experimentálnych vedomostí a ich ďalšieho využívania. Napodobňuje ľudský mozog v dvoch aspektoch [36]:

- poznatky sú zbierané v neurónovej sietí počas učenia,
- medzineurónové spojenia (synaptické váhy - SV) sú využívané na ukladanie vedomostí.

Toto je jedna z definícii neurónových sietí, akceptovaná komunitou. Je zrejmé, že inšpirácia ku vzniku NN prišla z biologických systémov. Na prvý dojem vysoko abstraktná disciplína nachádza množstvo aplikácii v praxi a stáva sa prostriedkom pre riešenie problémov v širokom spektre odborných oblastí [36].

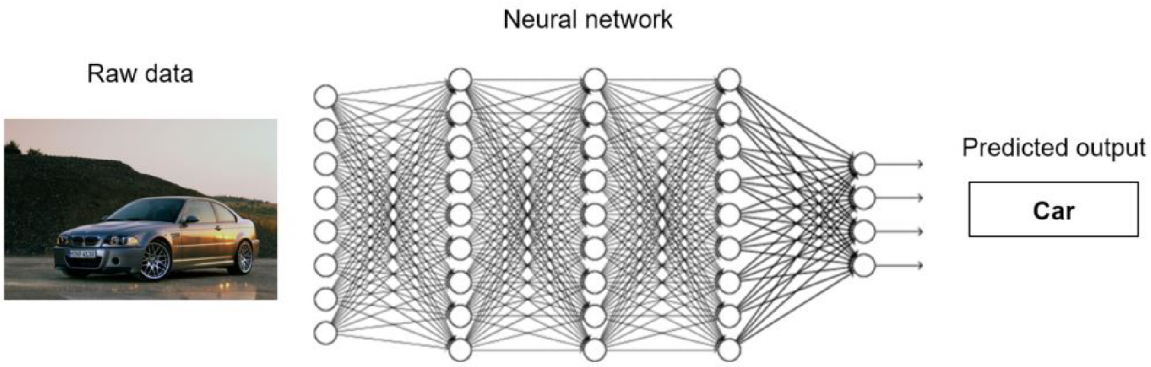
Rôzne architektúry umelých neurónových sietí (angl. *artificial neural network*) sa používajú pre riešenie rôznych úloh. Konvolučné a rekurentné NN sú dve z najúspešnejších a sú z veľkej časti zodpovedné za nedávnu revolúciu umelej inteligencie [35].

Vo všeobecnosti môžeme vymenovať nasledovné oblasti využitia neurónových sietí [36]:

- klasifikácie do tried, klasifikácia situácií,
- riešenie predikčných problémov,
- problémy riadenia procesov,

- tranformácia signálov.

Neurónové siete sú usporiadané do vnútorne prepojených vrstiev umelých neurónov. Jednoducho povedané, každá vrstva berie výstup z prechádzajucej vrstvy, aplikuje transformácie a výsledok pošle na vstup ďalšej vrstve. Prvá vrstva vstupov je prepojená na vstupné dáta, ktoré sa majú spracovať a posledná vrstva je akýkoľvek výstup, ktorý chceme predpovedať [35] (viď. obrázok 2.5).

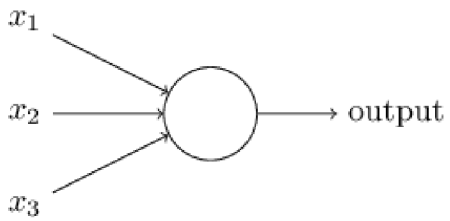



Obr. 2.5: Jednoduchá architektúra NN s 1 vstupnou, 3 skrytími a 1 výstupou vrstvou [35].

### Perceptron

Pre pochopenie fungovania neurónovej siete je potrebné najprv pochopiť umelý neurón, zvaný perceptron. Perceptron vynašiel vedec Frank Rosenblatt v 1950 až 1960 roku, inšpirovaný predchádzajucov prácov Warren McCulloch a Walter Pitts. Dnes sa bežne používa iný model umelého neurónu tzv. sigmoid neurón [31].

V jednoduchosti, perceptron zoberie niekoľko vstupov, $x_1, x_2, \ldots$ a reprodukuje ich na jediný binárny výstup.




Obr. 2.6: Jednoduchý príklad perceptronu [31].

Rosenblatt navrhol jednoduché pravidlo pre výpočet výstupu. Zaviedol váhy (angl. *weights*), $w_1, w_2, \ldots$, reálne čisla ktoré vyjadrujú dôležitosť príslušných vstupov vzhľadom na výstupy. Výstup neurónu, 0 alebo 1, sa určuje podľa toho či vážena suma $\sum_j w_j x_j$ je menšia alebo väčsia ako určitá prahová hodnota. Matematické vyjadrenie aktivačnej funkcie perceptronu by potom bolo [31]:

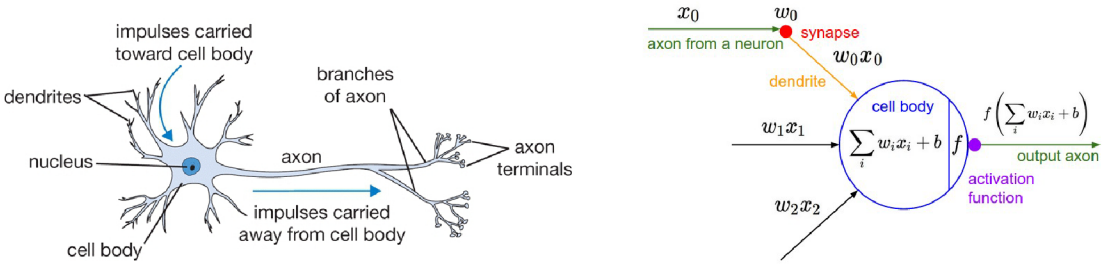
$$output = 0, \; ak \sum_j w_j x_j \leq threshold \tag{2.6}$$

$$output = 1,\ ak\ \sum_j w_j x_j > threshold \tag{2.7}$$

Matematický model perceptronu je možné zjednodušiť a to prepísaním formuly $\sum_j w_j x_j$ na skalárny súčin (angl. *dot product*), $w * x \equiv \sum_j w_j x_j$, kde $w$ a $x$ sú vektory váh a vstupov. Druhá zmena je presun prahovej hodnoty na opačnú stranu nerovnice, a nahradiť ju tzv. perceptron bias, $b \equiv -threshold$. Použitím týchto úprav bude model vyzerať nasledovne [31]:

$$output = 0,\ ak\ w * x + b \leq 0 \tag{2.8}$$

$$output = 1,\ ak\ w * x + b > 0 \tag{2.9}$$




Obr. 2.7: biologický neurón(vľavo) a jeho matematický model(vpravo) [4].

### Sigmoid

Sigmoid je jedna z aktivačných funkcií používaná v neurónových sieťach, jej matematická formula je

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.10}$$

, $\sigma$ je matematická sigmoida a $x$ je vstupná hodnota funkcie, zobrazená na obrázku 2.8 vľavo.

Funkcia zoberie skutočné číslo a "stlačí" ho do rozmedzia 0 až 1. Kde veľké záporné čísla nadobudnú hodnotu 0 a veľké kladné čísla hodnotu 1. Funkcia sigmoid sa historicky často používala pretože má peknú interpretáciu pre spúštanie "výstrelu" neurónu, od nevypálenia (0) až po úplne nasýtenie strely pri predpokladanej maximálnej frekvencii (1). V praxi, sa sigmoid používa už len zriedka. Pretože má dve hlavné nevýhody [4].

- **Nasýtenie sigmoidu** - veľmi nežiadnúca vlasnoť sigmoid neurónu je, že keď sa aktivácia neurónu nasýti na obidvoch koncoch 0 alebo 1, sklon (angl. *gradient*) v týchto oblastiach je takmer nulový. Tento sklon sa používa pri spätnej propagácií (angl. *backpropagation*) v neurónových sieťach. Preto ak je miestny gradient veľmi malý, takmer žiaden signál neurónom nepreteká k jeho váham a rekurzívne k jeho dátam. Rovnako ak pri inicializácií sú váhy príliš veľké, väčsina neurónov by bola nasýtených a sieť by sa sotva niečo učila.

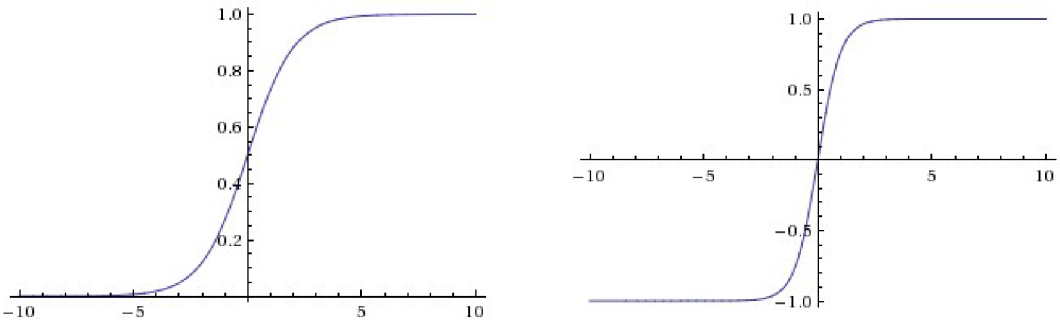
- **Výstupy nie sú centrované na nulu** - to má dôsledky na dynamiku pri klesaní gradient-u, pretože ak sú údaje prichádzajúce do neurónu vždy kladné, potom gradient na váhach $w$ nadobudne, počas spätnej propagáciie, všetky hodnoty pozitívne alebo negatívne (v závisloti na gradient-e celého výrazu). To by mohlo viesť k nežiadúcej cik-cakovitej dynamike pri aktualizáciach gradient-ov pre váhy.

### Tanh

Funkcia tanh je dalšou z aktivačných funkcií neurónu je zobrazená na obrázku 2.8 vpravo. Tanh zoberie skutočné číslo a stlačí ho do rozmedzia -1 až 1. Pracuje podobne ako sigmoid. Matematický zápis je [4]:

$$tanh(x) = 2\sigma(2x) - 1 \tag{2.11}$$

Keďže netrpí nevýhodami ako sigmoid tak v praxi je preferovanejší.




Obr. 2.8: Vľavo: Sigmoid nelineárne stlačenie čísel do rozsahu [0,1], Vpravo: Tanh nelineárne stlačenie čísel do rozsahu [-1,1] [4].

### ReLU a Leaky ReLU

V praxi existuje niekoľko dalších aktivačných funkcií, každá je použiteľná pre riešenie iného problému, čiže každá má svoje výhody ale aj nevýhody. Ako príklad môžeme uviesť ReLU (Rectified Linear Unit)

$$f(x) = max(0, x) \tag{2.12}$$

alebo jeho obdobu Leaky ReLU, ktorá sa snaží vyriešiť nevýhody ReLU.

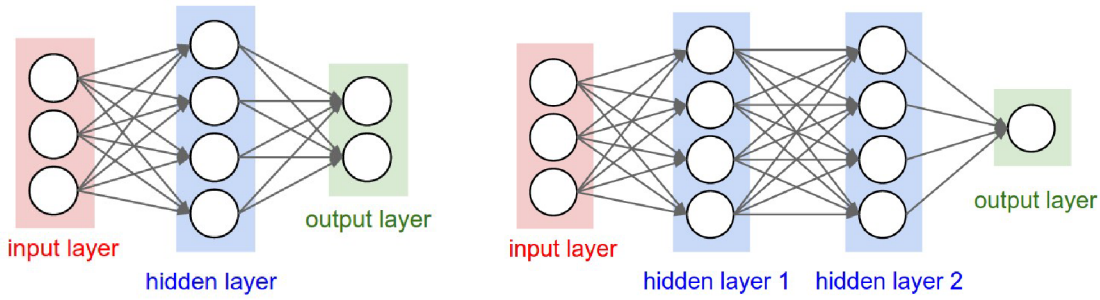
$$f(x) = x \quad ak\ x > 0 \tag{2.13}$$

$$f(x) = ax \quad ak\ x <= 0 \tag{2.14}$$

kde $a$ je malá konštanta (napr. 0.01) a $x$ vstupná hodnota [4].

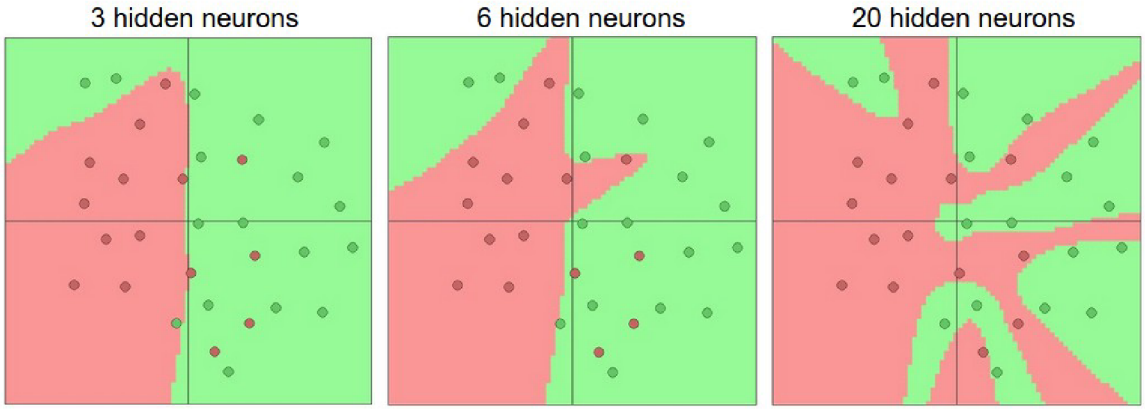
### Architektúra neurónovej siete

Ako bolo spomínané už vyššie, neurónová sieť je modelovaná ako kolekcia neurónov, ktoré sú prepojené v acyklickom grafe. Modeli neurónových sietí sú často organizované do odlišných vrstiev neurónov. Pre bežné neurónové siete je najbežnejšou vrstvou tzv. plne-prepojená vrstva [eng. fully-connected layer], v ktorej sú neuróny medzi dvoma priľahlými vrstvami plne párovo prepojené, ale neuróny v jednej vrstve nemajú medzi sebou žiadne spojenia. Na obrázku 2.9 nižšie sú uvedené dve topológie, ktoré využívajú plne-prepojené vrstvy [4]:




Obr. 2.9: Vľavo: 2-vrstvová neurónová sieť, Vpravo: 3-vrstvová neurónová sieť [4].

Vačšie neurónové siete dokážu reprezentovať komplikovanejšie funkcie. Na obrázku 2.10 je vidieť rozdiel klasifikácie dát do 2 tried (červená, zelená) použitím rôzneho počtu vrstiev v neurónovej sietí.




Obr. 2.10: Zobrazenie klasifikácie dát rôzne veľkými neurónovými sieťami [4].

### 2.4.4 Konvolučné neurónové siete

Konvolučné neurónové siete (angl. *convolutional neural networks*) (CNNs), sú špeciálnym prípadom neurónových sietí pre spracovanie dát, ktoré majú známu topológiu podobnú mriežke. Pre príklad je možné uviesť 2-dimenzionálne dáta ako sieť pixelov pri spracovaní obrázkov. Názov týchto sietí *konvolučné* indikuje používanie matematickej operácie zvanej konvolúcia (angl. *convolution*) [25].

#### Konvolúcia

Vo svojej najvšeobecnejšej podobe, konvolúcia je operácia dvoch funkcií s reálnymi hodnotami argumentov. Operácia konvolúcie je typicky označovaná hviezdičkou:

$$s(t) = (x * w)(t) \tag{2.15}$$

V terminilógií konvolučnej siete sa prvý argument (v tomto príklade funkcie $x$) často označuje ako vstup a druhý argument (v tomto príklade funkcia $w$) ako jadro (angl. *kernel*), v čase $t$. Výstup je niekedy označovaný ako mapa príznakov.

Bežne konvolúciu používame na viac ako jednej osi súčasne. Pri použití 2-dimenzionálneho obrázka $I$ ako náš vstup, budeme chcieť použiť aj 2-dimenzionálne jadro $K$, kde $m$ a $n$ sú

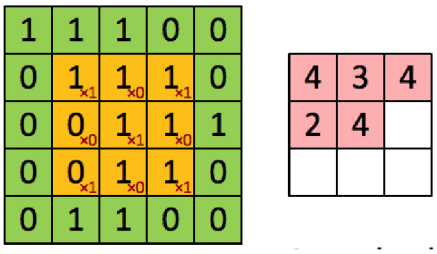
veľkosti vstupov [25].

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.16}$$

Avšak množstvo knižníc, ktoré implementujú neurónové siete, používajú tzv. cross-correlation funkciu [25].

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, i+n)K(m,n) \tag{2.17}$$

Hlavným cieľom konvolúcie v konvolučných neurónových sieťach je extrakcia príznakov zo vstupného obrázka. V jednoduchosti je možné si predstaviť 2-dimenzionálne dáta (obrázok) o rozmeroch 5x5 pixelov, ktorých hodnoty sú 0 alebo 1. Jadro, alebo môžeme ho nazvať aj filter o veľkosti napr. 3x3, prechádza postupne celý vstupný obrázok s krokom 1 a ráta hodnotu výstupneho pixelu na základe svojich hodnôt, ktoré obsahuje. Výsledkom je potom mapa príznakov (angl. *Feature map*) označovaná aj ako aktivačná mapa (angl. *Activation map*) o veľkosti 3x3 pixeli [12].




Obr. 2.11: Konvolúcie v CNN, zelená matica je vstupný obrázok a žltá matica je filter(vľavo), extrahované príznaky(vpravo) [12].

Veľkosť aktivačnej mapy je možné nastaviť pomocou 3 parametrov.

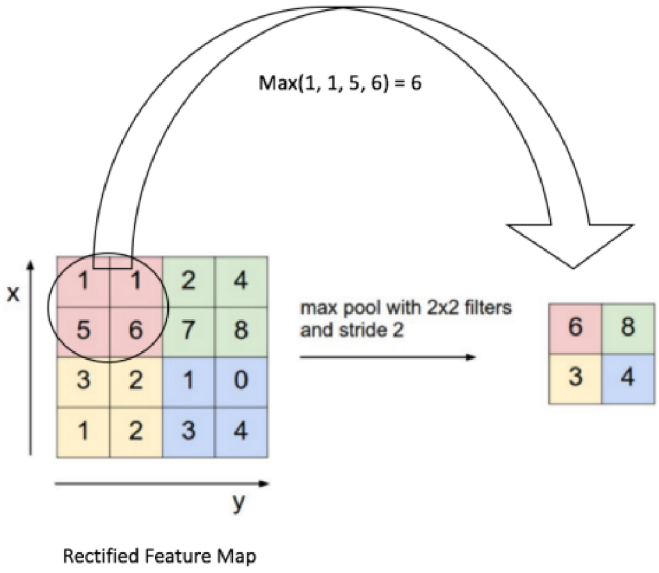
- **Hĺbka** (angl. *Depth*) - veľkosť hĺbky zodpovedá počtu použitých filtrov v konvolučnej vrstve siete.
- **Krok** (angl. *Stride*) - počet pixelov o ktorý sa filter posúva počas výpočtu aktivačnej mapy. Použitím väčších skokov bude výsledná aktivačná mapa menšia.
- **Nulový doplnok** [eng. Zero-padding] - občas je vhodné vložiť nulový doplnok okolo vstupného obrázka. Tento spôsob nám umožňuje ovládať veľkosť výstupnej aktivačnej mapy.

### Zlučovanie

Zlučovanie (angl. *Pooling*) je další krok, ktorý sa vykonáva v CNN, tento krok je vykonaný v Pooling vrstvách siete. Jeho funkciou je znižovanie dimenzionality vstupnej aktivačnej mapy, ale so snahou zachovať dôležitú informáciu, ktorá táto mapa obsahuje. Existuje niekoľko rôznych typov zlučovania napr. maximum, priemer alebo suma.

Pre tento krok je potrebné definovať veľkosť okna (napr. 2x2 pixely) a krok o, ktorý sa bude toto okno posúvať. V prípade typu Max Pooling s veľkosťou filtra 2x2 sa zoberú 4 hodnoty z aktivačnej mapy a vyberie sa z nich maximálna, podobným spôsobom fungujú

aj ostatné typy zlučovania, ale výsledná hodnota je napr. priemer, alebo suma týchto 4 hodnôt. V praxi sa ukázalo, že najvýhodnejší je výber maxima [12].




Obr. 2.12: Príklad použitia Max Pooling s filtrom 2x2 a krokom 2 [12].

### Architektúra

CNN využívajú skutočnosti, že vstup pozostáva z obrázkov. Na rozdiel od obyčajnej neurónovej siete majú vrstvy CNN neuróny usporiadané v troch rozmeroch: šírka, výška, hĺbka [eng. width, height, depth]. Hĺbka odkazuje na 3.dimenziu aktivačného zväzku (angl. *activation volume*), nie na celkovú hĺbku neurónovej siete. Neuróny vo vrtsve budú prepojené iba k malej oblasti vrstvy pred ňou, namiesto plného prepojenia [6].

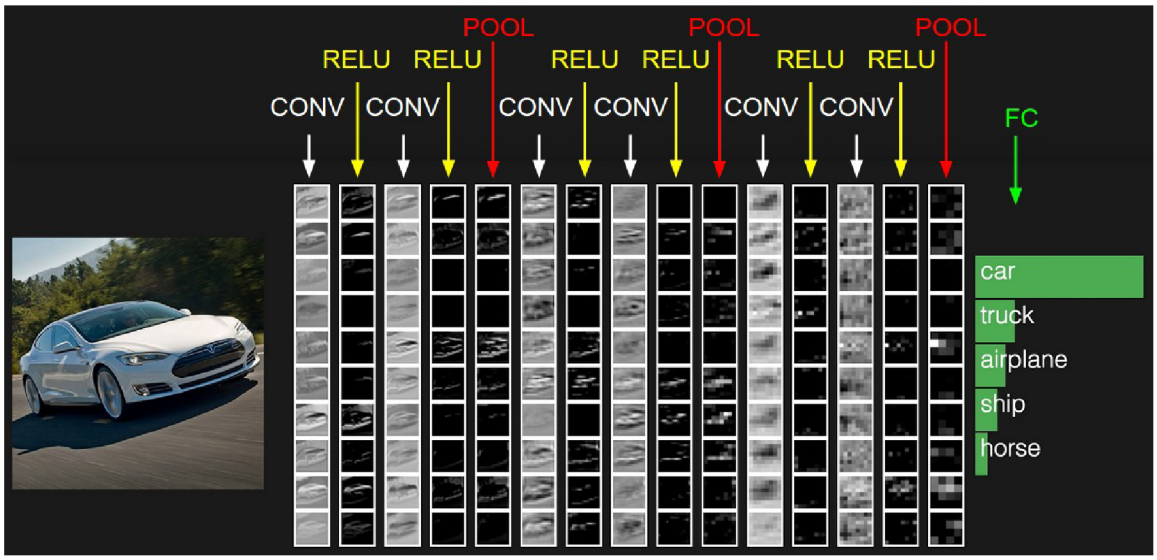
Pre stavbu CNN sa využívajú 3 základné typy vrstiev a 1 podporná vrstva.

- **Convolutional layer** - konvolučná vrstva je hlavný stavebný blok CNN, ktorá vykonáva väčšinu výpočtov. Parametre tejto vrstvy pozostávajú zo súboru filtrov, ktoré sa dokážu učiť. Takto naučené filtre sa aktivujú, keď uvidia určitý typ vizuálneho prvku, napr. hranu určitej orientácie, alebo škvrnu určitej farby a pod..

- **Pooling layer** - táto vrstva je obvykle vkladaná ako spojenie medzi dvoma konvolučnímy vrstvami. Jej funkciou je postupné znižovanie priestorovej veľkosti reprezentovaného vstupu, pre zníženie množstva parametrov a výpočtov v sietí, a teda aj kontroly pretrénovania.

- **Fully-Connected layer** - rovnako ako v obvyklých neurónových sieťach, je to vrstva kde všetky neuróny sú prepojené s predchádzajucou vrstvou.

- **Dropout layer** - cieľom tejto vrstvy je prevencia k pretrénovaniu siete, podľa nastavenia sa určité percento neurónov, ktoré sa počas trénovanie zahadzuje. Táto vrstva tlačí neúronovú sieť k tomu, aby sa učila na komplexnejších príznakoch, implementovaním tejto vrstvy do siete sa približne zdvojnásobí počet interácií trénovania. Avšak trénovaci čas pre každú iteráciu je kratší.




Obr. 2.13: Príklad architektúry konvolučnej neurónovej siete [6].

### 2.4.5 Populárne architektúry

Keďže konvolučné neurónové siete už existujú niekoľko rokov a komunita sa snaží o ich neustále zlepšovanie, tak sú organizované celosvetové súťaže na, ktorých sa niekedy objaví architektúra siete, ktorá dosahuje veľmi dobré výsledky a tak je možné ju použiť ako odporúčania a vychádzať z nej pri tvorbe vlastnej architektúry.

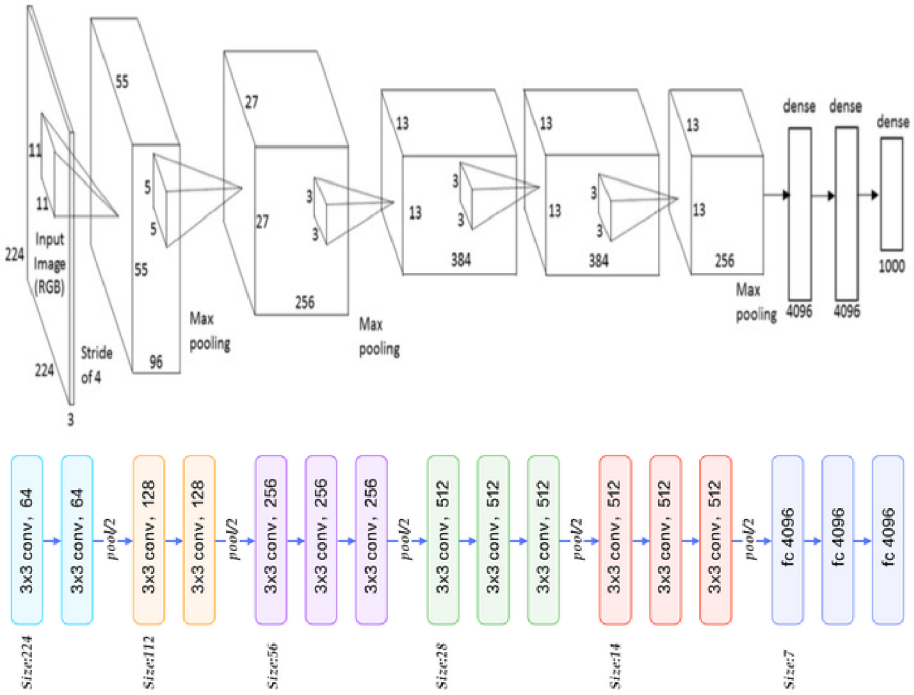
**AlexNet** (2012) je jedna z prvých hlbokých CNN. Táto sieť dala základ pre stavbu budúcich architektúr, zároveň bola použitá pre výhru v súťaži 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). V porovnaní s modernými architektúrami bola táto veľmi jednoduchá, tvorilo ju 5 konvolučných vrstiev, pooling vrstviev, dropout vrstiev a 3 plne prepojených vrstiev. Sieť bola použitá pre klasifikáciu objektov do 1000 kategórií. Vychádzala z niekoľkých hlavných princípov [19]:

- bola trénovaná na ImageNet[1] databázovej sade, ktorá obsahuje cez 15 miliónov obrázkov vo viac ako 22000 kategóriách.
- Použila ReLU nelineárnu aktivačnú funkciu (objavil sa kratší čas trénovania, keďže výpočet ReLU je niekoľko krát rýchlejši ako oproti v tej dobe bežne používanej Tanh).
- Bola použitá technika augmentácie dát, ktorá pozostávala z rôznych transformácii obrázkov ako napr. horizontálny odraz.
- Implementovala Dropout vrstvy pre zníženie problému spojeným s pretrénovaním neúronovej siete.
- Trénovanie prebiehalo na dvoch GTX 580 grafických kartách, počas 5 až 6 dní.

**VGG Net** (2014) je možné nájsť v niekoľkých variáciách, ako VGG16 alebo VGG19, číslo v názve udáva súčet konvolučných a plne prepojených vrstiev. Veľkosti filtrov v konvolučných vrstvách sú o rozmeroch 3x3 oproti AlexNet, kde ich veľkosť bola 11x11. Avšak

[1] http://www.image-net.org/

základná myšlienka tejto architektúry je v hĺbke a jednoduchosti, keďže sa zakladá na veľkom množstve konvolučných vrstiev, z čoho vyplýva veľké množstvo filtrov a tým pádom zväčšuje svoj výstup do hĺbky. Aj keď táto architektúra nebola víťaznou v roku 2014 na ILSVRC, ukázala základnú myšlienku, že sieť môže byť jednoduchá, ale musí byť hlboká [19].




Obr. 2.14: Porovnanie architektúr AlexNet(hore) [7] a VGG16(dole) [14].

## 2.5 Nástroje pre tvorbu klasifikátorov a neurónových sietí

V oblasti klasifikácie, či už pomocou klasických prístupov, alebo pomocou neurónových sietí, existuje množstvo nástrojov, ktoré implementujú tieto techniky a sú voľné dostupné na použitie. Technické giganty ako Google, Amazon, Facebook alebo Microsoft sú firmy, ktoré vkladajú veľké investície do tejto oblasti. Vedú buď svoj vlastný vývoj, alebo získavajú/podporujú niektoré existujúce riešenia. Keďže existuje veľké množstvo týchto nástrojov, tak v kapitole budú spomenuté iba niektoré, ktoré sa radia medzi najpopulárnejšie [5].

### Theano

Theano je jeden z prvých nástrojov, vytvoril ho Yoshua Bengio spolu s výskumným tímom na University of Montreal v roku 2007. Bol prvý široko používaný nástroj pre strojové učenie. Theano je Python knižnica, extrémne rýchla a výkonná, ale kritizovaná za to, že je nízko úrovňovým nástrojom. Tím, ktorý stojí za touto knižnicou oznámil v roku 2017, že po vydaný poslednej verzie v roku 2018 už vývoj nebude pokračovať [5].

### TensorFlow

TensorFlow je softvérová knižnica s otvoreným zdrojovým kódom (angl. *open source library*) pre numerické výpočty pomocou dátových vývojových diagramov (angl. *data flow graphs*). Uzly v grafe reprezentujú matematické operácie, zatiaľ čo hrany grafu reprezentujú multidimenzionálne dátové polia (tensors), ktoré medzi sebou kominukujú. Flexibilná architektúra umožňuje nasadenie na viacerých CPU alebo GPU, serveroch alebo aj mobilných zariadeniach. TensorFlow bol pôvodne vyvinutý na účely výskumu strojového učenia a výskumu hlbokých neurónových sietí [11].

Aktuálne je TensorFlow najviac používaný systém pre trénovanie hlbokých neurónových sietí. Stojí za ním veľká komunita technických firiem, odborníkov a technologických nadšencov z celého sveta, aj keď bola kritizovaná za jej prílišnú komplexnosť. Ďalšou bežnou kritikou je, že podľa mnohých odborníkov je oveľa pomalšia v porovnaní s inými knižnicami [5].

### Keras

Keďže vysoká všeobecnosť knižice TensorFlow pre jej širokú aplikáciu, robí jej použitie pre tvorbu neurónových sietí komplikovanejšiu. Keras sa v tomto smere snaží využívať TensorFlow ako svoj backend a tvorbu neurónových sietí zjednodušiť. Za jeho backend je možné použiť aj CNTK alebo Theano. Táto implementácia knižnice Keras robí experimentovanie s neurónovými sieťami jednoduchšie a rýchlejšie. Aj keď sa snaží implementáciu zjednodušiť, stále si zachováva modularitu a preto je možné dostatočne dobré modely neurónových sietí upravovať a prispôsobovať pre riešienie rôznych problémov [9].

### PyTorch a Torch

PyTorch je python implementácia nástroja Torch, ktorý bol vydaný spoločnosťou Facebook v roku 2017. Používa dynamické výpočtové grafy (angl. *dynamic computational graphs*), ktoré významne prispievajú, k analýze neštrukturovaných údajov. PyTorch si upravil alokátor GPU, ktorý umožňuje, aby modely neurónových sietí boli viac pamäťovo efektívne. Niektoré z hlavných nevýhod sú, že nástroj je stále v porovnateľne novej beta verzií a nemá dostatočne veľkú podporu komunity [5].

### Caffe a Caffe2

Caffe je dalši z nástrojov pre tvorbu neurónových sietí, avšak jeho použitie je priamo mierené pre spracovanie obrazu a nie na iné aplikácie, ako spracovanie textu alebo zvuku [2]. Za pritiotu si dáva rýchlosť a modularitu. Bol vyvinutý Berkley Artificial Intelligence Research. Pre veľkú popularitu Caffe sa Facebook rozhodol vydať Caffe2 v roku 2017. Kde Caffe2 ponúka užívateľovi použiť predtrénované modely pre rýchlu tvorbu demo aplikácií [5].

### Scikit-learn a Scikit-image

Scikit-learn je vysoko úrovňová knižnica navrhnutá pre algoritmy strojového učenia pod dozorom, alebo bez dozoru. Ako jedna zo zložiek vedeckého ekosystému jazyka Python, je postavená na knižniciach NumPy a SciPy, kde každá z nich je zodpovedná za úlohy z vedeckej oblasti spracovania dát na nízkej úrovni. Obsahuje množstvo algoritmov pre klasifikáciu, regresiu alebo zhlukovanie dát [3].

Sciki-image je voľne dostupná knižnica, ktorá obsahuje kolekciu algoritmov pre spracovanie obrázkov, určená pre SciPy. Je písaná v jazyku Python, knižnica je vývíjaná SciPy komunitou [42].

## 2.6 Predspracovanie obrazu

Keďže riešenie bude obsahovať aj klasické prístupy pre klasifikáciu nielen riešenie pomocou konvolučných neurónových sietí. Tak je dôležité pred klasifikáciou vhodne upraviť resp. extrahovať z obrázka vhodné príznaky pre ich lepšiu klasifikáciu.

### Rovnomerný pomer strán

Pre spracovanie obrázkov pomocou neurónových sietí je potrebné zabezpečiť, aby každý obrázok mal rovnakú veľkosť a rovnaký pomer strán. Pretože väčšina modelov neurónovej siete predpokladá štvorcový vstupný obraz [15].

### Šedotónový obraz

Ďalšou možnou technikou je vytvorenie šedotónového obrazu, kde všetky 3 zložky farby (RGB) zlúčime do jednej, šedotónovej. Tento prevod sa vykoná výpočtom podľa vzorca:

$$Y = 0.2125 * R + 0.7154 * G + 0.0721 * B[42] \tag{2.18}$$

,kde $Y$ je výsledná hodnota sivého pixela, $R$ je hodnota červenej zložky pixela, $G$ je zelená zložka pixela a $B$ je modrá zložka pixela v RGB [15].

### Rožšírenie vstupných dát

Ďalšou bežnou technikou predspracovanie dát je rožšírenie existujúceho súboru dát s rôznymi variáciami vstupných obrázkov. Tieto variácie môžu zahŕňať zväčšovanie, zmenšovanie, rotačné zmeny a iné transformácie obrázkov. Tento postup znižuje šancu, že neurónová sieť rozpozná nežiadúce charakteristiky v množine vstupných dát a taktiež sa zabezpečí zovšeobecnenie klasifikácie resp. rozpoznávania objektu [15].

### Normalizacia

Hodnoty pixelov sú väčšinou v rozpätí 0-255. Používať tieto hodnoty na vstupe priamo do siete môže viesť k pretečeniu číselných premenných. Taktiež sa ukázalo, že výber niektorých aktivačných funkcií nie je kompatibilný s takýmito vstupmi. Nevhodný vstup môže viesť k nesprávnemu učeniu siete, preto je vhodné tieto dáta normalizovať do rozpätia 0-1 [13].

### Histogram orientovaných prechodov

Základnou myšlienkou za dekriptorom histogramu orientovaných prechodov (angl. *Histogram of oriented gradients descriptor*) je to, že lokálny vzhľad objektu a tvar v obraze, môžu byť popísané distribúciou intenzity gradientov alebo smerov hrán.

Obraz je rozdelený na malé spojené oblasti nazývané bunky (angl. *cells*), a pre pixely v každej bunke sa vytvára histogram orientovaných prechodov. Deskriptor je spojenie týchto histogramov. Pre zvýšenie presnosti môžu byť lokálne histogramy kontrastovonormalizované výpočtom miery intenzity vo väčšej oblasti obrazu nazývanej blok a potom

pomocou tejto hodnoty normalizovať všetky bunky v rámci bloku. Táto normalizácia má za následok lepšiu invariáciu voči zmenám v osvetlení a tieňovaní v obraze [30].

## 2.7 Zhrnutie

V úvode kapitola zhrnula základné kategórie do ktorých sa rozdeľujú zbrane podľa toho ako pôsobia na živú silu a podľa ich ovládateľnosti 2.1. Následne bolo priblížené digitalizovanie obrazu, spôsob jeho reprezentácie ako matica pixelov a oblasti, v ktorých je spracovanie obrazu často využívané 2.2.

V kapitole 2.3 bolí vysvetlené základné časti z ktorých pozostáva detekcia objektu v obraze a vytvorený prehľad prístupov od klasických až po niekoľko prístupov, ktoré využívajú neurónové siete. Kapitola taktiež obsahuje krátke vysvetlenie dvoch algoritmov ktoré sa používaju pri dektekcií objektu v obraze a to kĺzajúce okno 2.3.2 a regionálne návrhy 2.3.3.

Ďalšia podkapitola 2.4 približuje a podrobne opisuje dva algoritmy, ktoré možno zaradiť do klasikého prístupu v rámci problematiky klasifikácie objektov, K-Nearest-Neighbour a Support Vector Machine. Táto podkapitola postupne vysvetľuje základnú myšlienku a funkciu neurónových sietí 2.4.3, od ich základného stavebného komponentu umelého neurónu resp. perceptronu, cez porovnanie aktivačných funkcií, až po základ architektúry neurónovej siete.

Následne sú spomenuté konvolučné neurónové siete 2.4.4 ako špeciálny prípad neurónových sietí a ich základné operácie ako konvolúcia a zlučovanie, vysvetlenie architektúry a význam každej zo štyroch vrstiev z ktorých konvolučne neurónové siete pozostávajú. V závere tejto podkapitoly sú popísané dve architektúry, ktoré priviedli istú zmenu pri tvorbe sietí, a je možné z nich vychádzať pri tvorbe vlastnej architektúry.

V kapitole 2.5 je vytvorený prehľad populárnych knižníc a systémov, ktoré je možné použiť či už pri tvorbe klasických klasifikátorov, alebo pri tvorbe neurónových sietí pre spracovanie textu, zvuku alebo obrazu.

Záver kapitoly je venovaný niekoľkým základným technikám predspracovanie obrazu 2.6.

# Kapitola 3

# Návrh riešenia

Kapitola bude zahŕnať použitý software pre tvorbu klasifikátorov a neurónových sietí, špecifikáciu hardwaru na ktorom bude trénovanie priebiehať. Ďalej priblíži zbroje a spôsoby pre získanie, spracovanie a rozdelenie dát zbraní. Následne sa bude venovať celkovému návrhu postupu, od predspracovania obrazu až po zber výsledkov, pre určenie typu zbrane a jej náklonu. Celá kapitola bude vychádzať z informácií, ktoré boli spomenuté v kapitole 2.

## 3.1 Použitý software a hardware

Prvým bodom pri návrhu riešenia je dôležité vybrať správny programovací jazyk a nástroje, ktoré sú pre riešenie daného problému vhodné a ktoré môžu uľahčiť aj celkovú implementáciu.

V kapitole 2.5 bol vytvorený základný prehľad populárnych nástrojov pre tvorbu klasifikátorov, neurónových sietí a predspracovanie obrazu. Vzhľadom na porovnanie týchto nástrojov, bude pre túto prácu použitý programovací jazyk Python, knižnica Keras pre tvorbu konvolučných neurónových sietí, ktorá ako svoj backend bude využívať knižnicu TensorFlow. A následne pre klasický prístup ku klasifikácii obrázkov bude použitá knižnica Scikit-learn a Scikit-image pre implementáciu predspracovania obrazu.

Keďže trénovanie klasifikátorov a neurónových sietí je výpočtovo náročná operácia, je vhodné použiť výkonný hardware a akceleráciu výpočtov na GPU. Pre túto prácu bude použitý počítač so štvorjadrovím procesorom i7-6700HQ, veľkosťou operačnej pamäte 16 GB a grafickou kartou Nvidia GeForce 940MX. Na použitom počítači bude nainštalovaný 64-bitový operačný systém Fedora 27.

## 3.2 Klasifikácia typu zbrane

Jedným z cieľov tejto práce je klasifikovať typy zbraní do 2 kategórií, a to na krátke a dlhé zbrane. Táto klasifikácia môže prebiehať niekoľkými spôsobmi, prehľad týchto prístupov bol zhrnutý v kapitolách 2.3 a 2.4. Pre klasifikáciu v tejto práci bude použitých niekoľko z týchto prístupov a vo výsledku budú porovnané, ktorý dosiahol najlepšie výsledky.

Prvé riešenie bude spočívať v klasickom prístupe, ktoré pozostáva z predspracovania vstupných dát pomocou prekonvertovania dát do šedotónového obrazu a histogramu orientovaných prechodov (viď. 2.6). Na tieto predspracované dáta bude v jednom z riešení použitý K-Nearest-Neighbor klasfikátor, pre druhé riešenie bude použitý SVM klasifikátor,

testovanie a porovnávanie výsledkov môže prebiehať s rôznymi konfiguráciami týchto klasifikátorov.

- Pre **K-Nearest-Neighbor**, knižnica scikit-learn poskytuje 2 rôzne implementácie tohto klasifikátora. Trieda *KNeighborsClassifier* klasifikuje na základe $k$ najbližších susedov, kde $k$ je celé číslo špecifikované užívateľom.

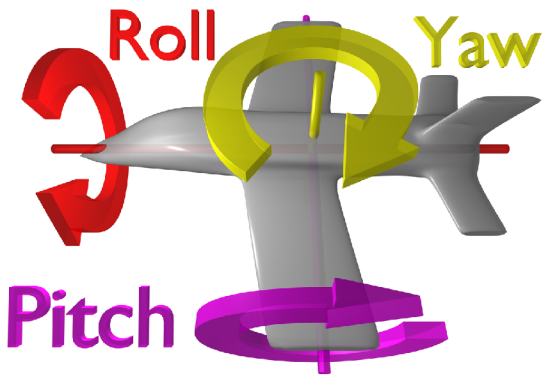
  Druhá implementácia je trieda *RadiusNeighborsClassifier*, ktorá klasifikuje na základe počtu susedov v rámci pevného polomeru $r$ každého trénovacieho bodu, kde $r$ je hodnota s pohyblivou desatinnou čiarkou určená užívateľom[1].

- Pre **Support Vector Machines**, scikit-learn obsahuje 3 triedy *SVC*, *NuSVC* a *LinearSVC* pre viac-triednu klasifikáciu s možnosťou použitia rôznych typov jadier [2].

Pre dalšie riešenie klasfikácie zbraní budú použité dve konvolučné neurónové siete, ktorých výsledky budú porovnané, všeobecná architektúra sietí je opísaná v kapitole 3.4. Výsledok poslednej vrstvy softmax klasfikatóra budú 2 výstupy, ktoré budú určovať typ zbrane. Predspracovanie vstupných dát bude pozostávať z normalizácie hôdnot RGB zložiek pixelov, nastavenia rovnomernej veľkosti strán obrázka (viď. 2.6) a následnej augmentácií dát pre zväčšenie počtu vstupných dát (viď. 3.5.2).

## 3.3 Určenie náklonu zbrane

Druhý z cieľov tejto práce je určenie náklonu zbrane v obraze. Tento náklon bude určený v 3 osiach. Dôležité je spomenúť, že názvy osí sú pomenované podľa tých, ktoré sa používajú v letectve, viď. obrázok 3.1.




Obr. 3.1: Mená osí pre letectvo [1].

Tento problém je možné preniesť do klasifikácie, preto pre riešenie tohto problému budú použité konvolučné neurónové siete, ktorých všeobecná architektúra je popísaná v 3.4. Výsledok poslednej vrstvy softmax klasfikátora bude 72 výstupov, ktoré budú určovať o aký uhol je zbraň natočená. Každá zo 72 kategórií bude zastupovať rozpätie 5 stupňov, čiže celkovo sa určí náklon zbrane v celom rozsahu od 0 do 360 stupňov.

[1] http://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification
[2] http://scikit-learn.org/stable/modules/svm.html#custom-kernels

Postup predspracovania obrazu bude rovnaký ako pri klasfikácií zbraní pomocou konvolučných neurónových sietí, normalizácia, úprava rozmeru vstupu na štvorec a augmentácia dát (viď. 3.5.2) Pre každú os bude natrénovaná samostatná konvolučná neurónová sieť, výsledok bude teda obsahovať 3 natrénované modely.

### 3.3.1 Odchýlka chyby

Knižnica Keras obsahuje funkciu *categorical_accuracy*[3] pre určenie presnosti siete pri klasifikovaní do viacerých kategórií. Čo je možné použiť, avšak pre určenie náklonu by bola vhodná iná metrika určovania presnosti siete. Preto bude implementovaná vlastná funkcia *angle_error*, ktorá bude počítať presnosť siete podľa priemerného rozdielu medzi skutočnými a predpovedanými uhlami.

## 3.4 Konvolučné neurónové siete

### 3.4.1 Nastavenie parametrov

Ako bolo spomínané v kapitole 2.4.4 je niekoľko parametrov, ktoré je potrebné nastaviť v konvolučných a pooling vrstvách. Pre správne fungovanie siete je potrebné nastaviť aj správne hodnoty týchto parametrov, preto v navrhovaných architektúrach budú hodnoty parametrov nastavené na tie najpoužívanejšie.

Veľkosť výstupu a správnosť nastavenia parametrov v konvolučnej vrstve je možné vypočítať pomocou vzťahu:

$$\frac{(W - F + 2 * P)}{S} + 1 \tag{3.1}$$

Kde $W$ je veľkosť vstupných dát, $F$ je veľkosť filtra, $P$ je nastavenie zarovnania, v prípade nulového zarovnania je hodnota 1 a $S$ je veľkosť kroku. Pre správné nastavenie musí byť výsledná hodnota celé číslo, kde táto výsledná hodnota udáva aj veľkosť výstupu. Najpoužívanejšie hodnoty parametrov sú: $F = 3, S = 1, P = 1$ a vstup $W$ o hodnote, ktorá je deliteľná číslom 2 [6].

Funkciou pooling vrstvy je znižovanie dimenzionality a zároveň ponechanie dôležitých informácií, ako najpoužívanejšie hodnoty parametrov sú veľkosť filtra 2x2 s krokom 2 pri použití MaxPooling vrstvy [6].

### 3.4.2 Návrh architektúr

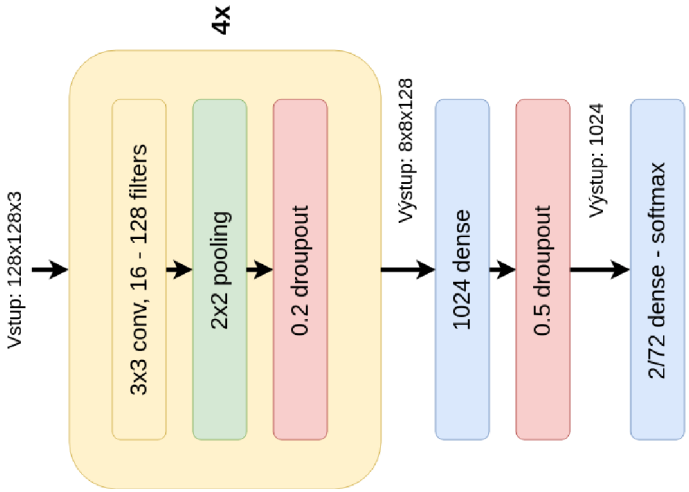
Prvý model siete je inšpirovaný architektúrou AlexNet (viď. 2.4.5). Model obsahuje celkovo 15 vrstiev, z ktorých 4 je konvolučných, 4 max pooling, 5 dropout a 2 dense vrstvy. Veľkosť vstupných dát do prvej vrstvy je 128x128x3. Vo všetkých konvolučných vrstvách sú použité filtre o veľkosti 3x3 s krokom 1 a použitím nulového zarovnania.

V modeli sa každou konvolučnou vrstvou zdvojnásobuje počet filtrov, z počiatočných 16 až na 128 v poslednej vrstve. Pooling vrstvy sú typu max, veľkosť filtra je 2x2 s posunom 2 po každej osi. Za každou pooling vrstvou sa nachádza Droupout vrstva s nastavením 0.2, čiže 20 percent náhodných prepojení sa ignoruje.

Po štyroch blokoch konvolučnej, pooling a droupout vrstvy nasleduje dense vrstva s počtom prepojení 1024 a dropout vrstva s nastavním 0.5. Ako posledná je dense vrstva s 2

[3] https://keras.io/metrics/

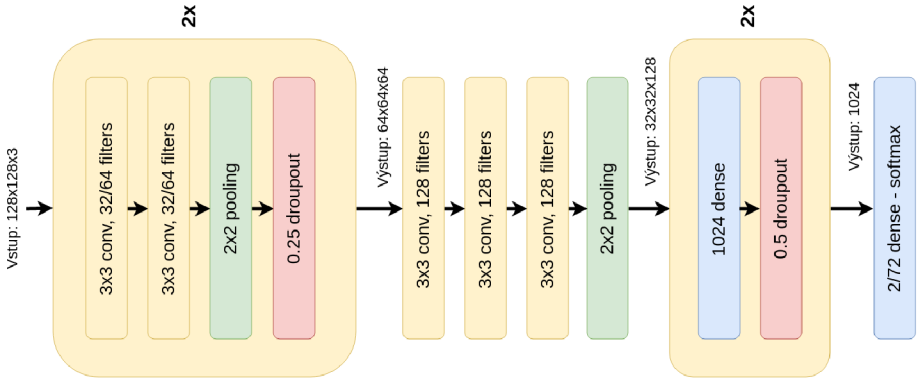
alebo 72 prepojeniami a softmax klasifikátorom, počet výstupov závísí od toho, či určujeme typ alebo náklon zbrane. V celej sieti sú použité ReLu aktivačné funkcie.




Obr. 3.2: AlexNet-Like navrhovaná architektúra.

Druhý navrhovaný model je inšpirovaný architektúrou VGG sietí (viď. 2.4.5). Vzhľadom na možnosti výkonu na ktorom bude trénovanie prebiehať, je navrhovaná sieť o dva bloky vrstviev menšia a taktiež konvolučne vrstvy obsahujú menej filtrov.

Celkovo sieť obsahuje 2 bloky obsahujúce 2 konvolučné vrstvy s počtom filtrov 32 a 64, pooling a droupout vrstvu s 20% ignorovaním prepojení. Ďalej nasledujú 3 konvolučné vrstvy so 128 filtrami a pooling vrstva. Ako posledné sú 2 bloky obsahujúce dense vrstvu s počtom prepojení 2048 a dropout vrstvu s ignorovaním nastaveným na 50%. Posledná výstupná vrstva obsahuje 2 alebo 72 prepojení so softmax klasfikátorom. Každá konvolučná vrstva obsahuje filtre o veľkosti 3x3, krokom 1 a s použitím nulového doplnku. Pooling vrstvy sú typu max, veľkosť filtra je 2x2 s posunom 2 po každej osi. V celej sieti je použitá ReLu aktivačná funkcia.




Obr. 3.3: VGG-Like navrhovaná architektúra.

### 3.4.3 Hodnotenie presnosti modelov

Hodnotenie presnosti modelov bude prebiehať pomocou dvoch metrík.

Prvá z metrík je tzv. chybová alebo tiež kontigenčná matica. Každý stĺpec v matici predstavuje klasifikované triedy a jednotlivé riadky predstavujú správne triedy. Tabuľka 3.1 zobrazuje túto maticu. Hodnota TP označuje počet správne klasifikovaných obrázkov triedy true, hodnota FP označuje počet nesprávne klasfikovaných obrázkov triedy true. Hodnota TN označuje počet správne klasifikovaných obrázkov triedy false, hodnota FN označuje počet nesprávne klasifikovaných obrázkov triedy false [27].

| | | Klasifikované hodnoty | |
|---|---|---|---|
| | | Trieda false | Trieda true |
| Správne hodnoty | Trieda false | TN (True Negative) | FP (False Positive) |
| | Trieda true | FN (False Negative) | TP (True Positive) |

Tabuľka 3.1: Chybová matica

Pre prípad tejto práce si môžeme previesť triedu true na triedu krátke zbrane a triedu false na triedu dlhé zbrane. Ako hodnotenie presnosti bude použitá metrika úspešnosť (angl. *Accuracy*).

Úspešnosť - táto hodnota určuje ako často klasfikátor správne klasifikoval daný obrázok, počíta sa ako:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.2}$$

Ako druhá metrika, pre určenie presnosti modelov ktoré určujú náklon zbrane, bude implementovaná funkcia *angle_error*, ako bolo opísané v 3.3.1. Hodnotenie funguje tak, že je nastavená prahová hodnota uhla podľa ktorej sa určuje, či daná predpoveď siete bola správna alebo nie. Správne určenie je vypočítané, ako rozdiel medzi skutočným uhlom a predpovedaným uhol pomocou natrenovaného modelu, ak je rozdiel menší ako prahová hodnota, tak predikcia sa považuje za správnu v opačnom prípade za nesprávnu.

## 3.5 Trénovacia databáza zbraní

Pre správne trénovanie klasifikátorov a neurónových sietí je dôležité mať dáta v dostatočnom počte a správne označené. Trénovacia databáza je získaná z niekoľkých zdrojov.

- **IMFDB**[4] - je databáza záberov z filmov, v ktorých sa nachádzajú zbrane. Obsahuje nie len celkové scény, ale aj samostatné obrázky zbraní, ktoré sa v danej scéne nachádzajú. Pre túto prácu je potrebné, aby daný obrázok obsahoval iba zbraň, preto je potrebné obrázky z tejto databázy ručne odfiltrovať na samotné zbrane a scény so zbraňami. Následne ich potom zaradiť do správnej kategórie na krátke a dlhé.
- **ImageNet**[5] - je databáza obrázkov, ktorá obsahuje viac ako 14 miliónov obrázkov vo viac ako 21000 kategóriách. Výhodou tejto databázy je, že obrázky už patria do určenej kategórie a tak nie je potrebné ich ručne prechádzať a kategorizovať.
- **Google**[6] - pre doplnenie a zväčšenie počtu obrázkov je možné použiť google vyhľadávanie. Tak ako v prípade IMFDB bude potrebné obrázky ručne kategorizovať.

[4] http://www.imfdb.org/wiki/Main_Page
[5] http://www.image-net.org/
[6] http://www.google.com

- **Free3D**[7] - databáza voľne dostupných 3D modelov zbraní s textúrami v rôznych formátoch.

Databáza zbraní z prvých 3 zdrojov je možné použiť na klasficikáciu zbraní do 2 kategórií na krátke a dlhé zbrane. Pre trénovanie neurónových sietí na určenie náklonu zbrane v obraze bude potrebné použiť 3D modely zbraní a následne pomocou nich vygenerovať obrázky zbraní v požadovaných natočeniach zbrane v obraze.

### 3.5.1 Generovanie dát z 3D modelov

Generovanie obrázkov pre určenie náklonu zbrane v scéne bude prebiehať pomocou databázy 3D modelov zo zdroja Free3D. Následne sa použije software SYDAGenerator[8] na generovanie obrázkov z 3D modelov, do ktorého je potrebné vložiť 3D model vo formáte g3db a pozadie scény.

Keďže 3D modely, ktoré budú použité nie sú v požadovanom formáte, je potrebné ich prekonvertovať, to je možné urobiť pomocou nástroja **fbx-conv**[9]. Tento nástroj podporuje konvertovanie 3D modelov z formátu *fbx*, alebo *obj* do požadovaného formátu `g3db`.

### 3.5.2 Augmentácia obrázkov

Ako bolo opísané v kapitole 2.6 jedna z možností zväčšenia počtu vstupných dát a zlepšenia generalizácie pri klasifikácii je augmentácia vstupných dát. V tejto práci môžeme rozdeliť augmentáciu dát do dvoch skupín, prvý druh augmentácie bude prebiehať pre trénovanie klasifikátorov, ktoré určujú typ zbrane, druhý druh augmentácie bude prebiehať pri obrázkoch určených pre náklon zbrane.

Pri augmentácií dát pre klasifikovanie dvoch tried, na krátke a dlhé zbrane, prebehne niekoľko transfomácií obrázka.

- **Rotácia** - obrázky budú rotované o náhodný uhol v rozmedzí 0-180 stupňov.
- **Preklopenie** - vertikálne alebo horizonálne preklopenie obrázka.
- **Posun** - posun obrázka po x-ovej a y-ovej osi.
- **Výplň** - pri rotácií alebo posune obrázka môže vzniknúť čierna plocha pixelov, tieto čierne pixely je potrebné vyplniť. Tieto čierne pixely budú doplnené hraničným pixelom obrázka, ktorý bude nakopírovaný až po nový okraj.

Obr. 3.4: Pôvodny obrázok (vľavo), augmentovaný obrázok (vpravo)

[7] https://free3d.com/3d-models/weapons
[8] http://www.fit.vutbr.cz/~igoldmann/app/SYDAGenerator
[9] https://github.com/libgdx/fbx-conv

Druhý typ augmentácie dát prebieha pri obrázkoch, ktoré boli vygenerované pomocou 3D modelu. Keďže nie je potrebné generovať všetky polohy otočenia od 0 po 360 stupňov, stačí vygenerovať dáta v polohách od 0 po 90 stupňov a od 270 do 359 stupňov. Zvyšné polohy je možné dogenerovať pomocou transformácie obrázka, pomocou vertikálneho alebo hotizontálneho preklopenia. Avšak v tomto prípade je potrebné po transformácii obrázka vyrátať novú hodnotu uhla transformovaného obrázka, to je možné dosiahnuť pomocou niekoľkých jednoduchých vzorcov.

**Roll**, pri natočení modelu v ose roll, je potrebné aplikovať horizontálne preklopenie obrázka. Následne uhol transformovaného obrázka $trans\_angle$ bude vypočítaný z pôvodného uhla $angle$. V rozsahu $270 \leq angle \leq 359$ podľa:

$$trans\_angle = -angle + 540 \tag{3.3}$$

a v rozsahu $0 \leq angle \leq 90$ podľa:

$$trans\_angle = abs(angle - 180) \tag{3.4}$$

Obr. 3.5: Natočenie zbrane v ose roll, 300 stupňov(vľavo), 0 stupňov(v strede), 70 stupňov(vpravo)

**Yaw**, pri natočení v ose yaw, sa aplikuje vertikálne preklopenia obrázka. Uhol transformovaného obrázka sa vypočíta v rozsahu $270 \leq angle \leq 359$ podľa:

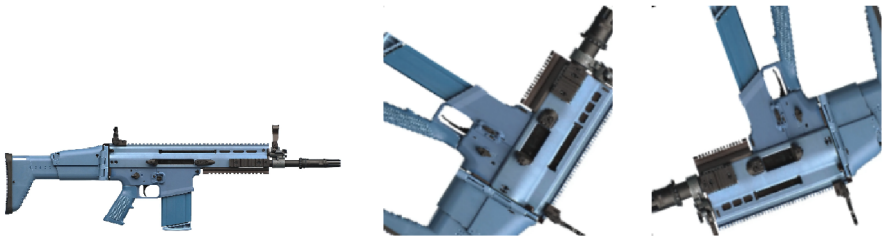
$$trans\_angle = abs(angle - 540) \tag{3.5}$$

a v rozsahu $0 \leq angle \leq 90$ podľa:

$$trans\_angle = abs(angle - 180) \tag{3.6}$$

Obr. 3.6: Natočenie zbrane v ose yaw, 300 stupňov(vľavo), 0 stupňov(v strede), 75 stupňov(vpravo)

**Pitch**, v ose pitch postačuje vygenerovať obrázok zbrane s náklonom 0 stupňov so smerovaním hlavne zbrane doprava. Počas augmentácie dát sa môže aplikovať horizontálne preklopenie a následne sa tento vstupný obrázok otočí o náhodný uhol. Preto nie je potrebný žiaden další prepočet uhla.



Obr. 3.7: Natočenie zbrane v ose pitch, 0 stupňov(vľavo), 60 stupňov(v strede), 200 stupňov(vpravo)

## 3.6 Zhrnutie navrhovaných postupov

Kapitola zhrnula software, ktorý sa bude používať pre tvorbu a trénovanie klasifikátorov a neurónových sietí. Taktiež obsahuje popis parametrov zariadenia, na ktorom toto trénovanie bude prebiehať.

Následne v kapitole 3.2 je opísaný navrhovaný spôsob pre klasifikáciu typu zbrane do 2 tried, na krátku a dlhú, približuje programové triedy, ktoré je možné pre tento problém použiť z knižnice scikit-learn. A predstavuje celkový postup klasfikácie od predspracovania obrazu až po konečný výsledok.

Kapitola 3.3 opisuje navrhovaný postup riešenia pre určenie typu zbrane pomocou konvolučných neurónových sietí, vysvetľuje terminológiu pre použité názvy osí rotácie a v 3.3.1 približuje navrhovanú funkciu pre lepšie určenie presnosti siete pri určovaní náklonu zbrane.

Celá kapitola 3.4 je venová podrobnému opisu dvoch navrhovaných architektúr konvolučných neurónových sietí a vysvetleniu všetkých zvolených hodnôt parametrov siete.

V závere sú spomenuté zdroje, z ktorých je možné čerpať obrázky a 3D modely zbraní, je opísaný postup generovanie obrázkov z 3D modelov a ich úprava pre účely tejto práce. Posledná podkapitola 3.5.2 obsahuje zoznam transformácii obrazu pre augmentáciu dát. Opisuje 2 druhy augmentácie použitých v tejto práci, kde prvý je zameraný na dáta určené pre klasifikáciu zbraní do 2 tried a druhý pre dáta určené na stanovenie náklonu zbrane v obraze.

# Kapitola 4

# Implementácia a testovanie

V tejto kapitole bude opísaná implementácia a rozdelenie komponentov vzhľadom na navrhovaný postup. Kapitola priblíži počty dát pre trénovanie modelov, tvorbu modelov ich trénovanie, ukladanie a meranie ich úspešnoti.

## 4.1 Trénovacie dáta

### 4.1.1 Zber dát

Pre trénovanie modelov boli použité obrázky zo zdrojov spomenutých v kapitole 3.5. Tabuľka nižšie uvádza podrobné počty obrázkov z daných zdrojov, ktoré boli použité pre určenie typu zbrane v 2 kategóriách.

| Zdroj | Počet krátkych zbraní | Počet dlhých zbraní |
|---|---|---|
| IMFDB | 647 | 670 |
| ImageNet | 730 | 94 |
| Google vyhľadávanie | 0 | 86 |
| SPOLU | 1377 | 850 |

Tabuľka 4.1: Podrobné počty trénovacích dát.

Vo všeobecnosti je jednoduchšie nájsť obrázky, ktoré obsahujú krátke zbrane (pištole) ako dlhé zbrane. Databáza ImageNet obsahovala vyše 3000 odkazov na obrázky, avšak veľká časť z nich bola chybná alebo daný súbor už neexistoval.

Ako bolo spomínané v 3.5.1, obrázky ktoré boli použité na trénovanie modelov pre určenie náklonu zbrane bolo potrebné vygenerovať pomocou 3D modelov. Pre toto generovanie bolo použitých desať pozadí scény a päť 3D modelov z toho tri modely boli dlhé zbrane a dve modely krátkych zbraní.


Obr. 4.1: Program pre generovanie obrázkov z 3D modelov.

Niektoré použité modely mali nastavený zlý bod otáčania a taktiež boli veľmi malé. Preto, ako je vidieť na obrázku 4.2, museli byť výsledne obrázky ešte orezané aby sa v nich nachádzala iba zbraň bez nepotrebného okolia.

Obr. 4.2: Výsledný vygenerovaný obrázok po orezaní okolia.

Tabuľka nižšie uvádza presné počty vygenerovaných obrázkov.

| Typ osi otáčania | Počet obrázkov |
|---|---|
| pitch | 1480 |
| roll | 4450 |
| yaw | 4584 |

Tabuľka 4.2: Podrobné počty trénovacích dát.

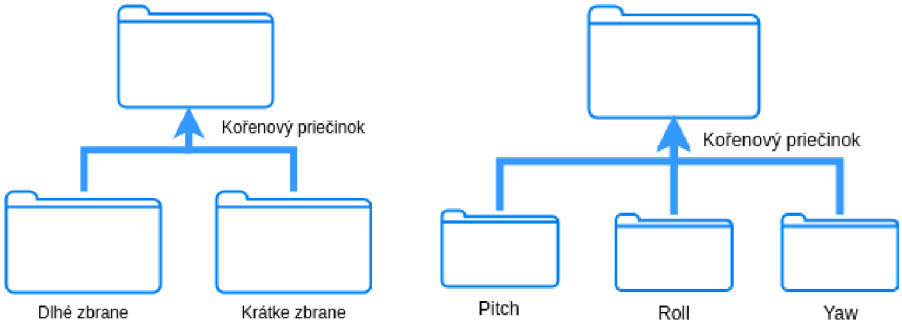
Pre os otáčania pitch boli vybrané obrázky z datasetu určeného pre klasifikáciu typu zbrane. Pre zvyšné 2 osi boli obrázky generované, počty sa líša kvôli niektorým chybným obrázkom, ktoré museli byť vymazané.

### 4.1.2 Načítavanie dát

Pre načítavanie dát je implementovaná trieda *DataLoader* v scripte *loader.py*. Trieda obsahuje niekoľko funkcií pre načítanie týchto dát, tieto funkcie vrátia pole načítaných obrázkov, pole indexov označení (angl. *labels*) obrázkov a slovník, ktorý obsahuje index a názov označenia, napr. {“kratka zbran”: 0, “dlha zbran”: 1}.

Prvá z funkcií načítava dáta z priečinka, kde označenie obrázkov, či je to krátka alebo dlhá zbraň, je zabezpečené podľa názvov podpriečinkov. Pre obrázky o náklone zbrane sa označenie osi berie tiež z názvu podpriečinka, avšak presné označenie o koľko stupňov je model v obrázku otočený sa získa z názvu obrázka, ktorý musí spĺnať presný formát: *p0.0_y0.0_r0.0-typzbrane.jpg*. Kde *p0.0* označuje hodnotu náklonu v osi pitch, *y0.0* hodnotu náklonu v osi yaw, *r0.0* v osi roll a *typzbrane* môže byť akýkoľvek reťazec. Formát vstupných obrázkov môže byť *.jpg* alebo *.jpeg*.

Správnu hierarchiu priečinkov je vidieť na obrázku 4.3.




Obr. 4.3: Hierarchia priečinkov pre správne označenie vstupných obrázkov.

Druhá z funkcií umožňuje načítať zoznam obrázkov z textového súboru, kde sa na každom riadku nachádza práve jedna cesta k požadovanému obrázku. Spôsob označovania obrázkov je rovnaký ako v prvom prípade, preto je taktiež potrebné dodržať správnu hierachiu priečinkov.

### 4.1.3 Predspracovanie dát

Pre predspracovanie obrazu sú implementované 2 triedy *Preprocessor* a *Preprocessing* v scripte *preprocessing.py*. Trieda *Preprocessor* obsahuje list, do ktorého je možné pridávať funkcie, ktoré sú implementované v triede *Preprocessing*. Po zavolaní funkcie *apply(input_data)* z triedy *Preprocessor*, sa tieto funkcie v poradí, v akom boli pridané do listu zavolajú a prebehne predspracovanie vstupných dát. Každá z možností predspracovanie obrazu, ktorá bola uvedená v 2.6, je implementovaná v triede *Preprocessing*, v samostatnej funkcii, pomocou knižnice scikit-image.

Pre augmentáciu dát určených na klasfikáciu typu zbrane je použitá trieda *ImageDataGenerator* z knižnice Keras. Parametre tohto generátora sú nastavené na: horizonálne a vertikálne preklopenia obrázka, posun obrázka až o 15% jeho dĺžky alebo šírky, rotácia o ná-

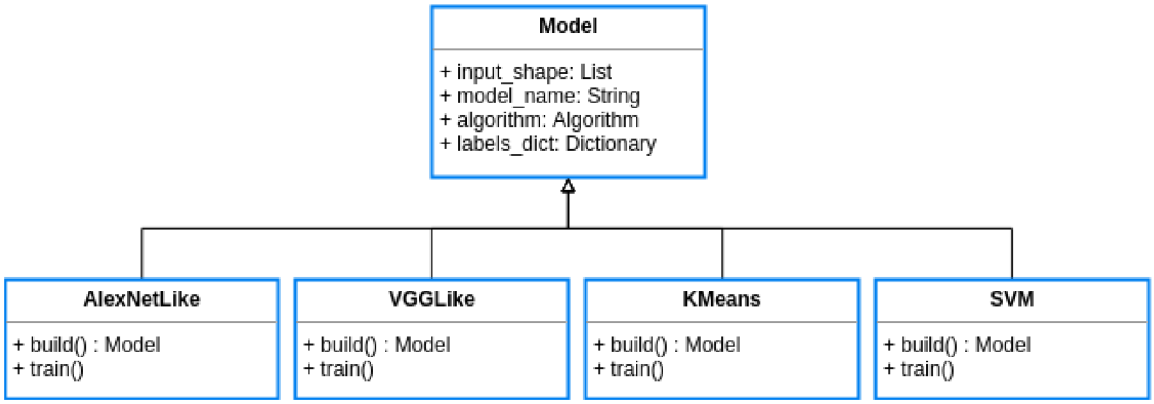
hodný uhol v rozsahu 0 až 180 stupňov a doplnenie čiernych pixelov pomocou nastavenia "nearest".

Pre augmentáciu dát, pre určenie náklonu zbrane je implementovaná samostatná trieda *AngleGenerator*, ktorá zabezpečuje správne dogenerovanie vstupných dát pomocou horizontálneho alebo vertikálneho preklopenia obrázka a správneho výpočtu uhla natočenia (viď. 3.5.2).

## 4.2 Modely

Všetky modely pre trénovanie sú implementované v scripte *models.py*. Existuje jedna hlavná trieda *class Model*, ktorá obsahuje všetky základné atributy potrebné pre modely, ako napr. meno modelu, vstupný rozmer dát, zvolený algoritmus trénovania a slovník, ktorý obsahuje indexi a názvy označení obrázkov.

Každý model následne dedí z tejto hlavnej triedy a implementuje v sebe dve funkcie *build*, ktorá zabezpečuje vytvorenie modelu a funkciu *train*, ktorá sa volá pri spustení trénovania daného modelu.




Obr. 4.4: Hierarchia dedenia tried modelov.

Konvolučné neurónové siete sú implementované pomocou triedy *Sequencia* z knižnice Keras. Pre klasifikátor K-Nearest-Neighbor je použitá trieda *KNeighborsClassifier* a trieda *SVC* pre klasifikátor SVM z knižnice scikit-learn.

### 4.2.1 Ukladanie a načítanie modelu

Pre ukladanie natrénovaných modelov je implemtovaná funkcia *save_model()* v triede *DataSaver* v scripte *loader.py*. Pri ukladaní modelu sa ukladá model v binárnej forme, ukladá sa konfiguračný súbor *settings.json*, ktorý obsahuje informácie ako veľkosť vstupných dát, použité funkcie na predspracovanie dát, meno modelu, typ algoritmu učenia, zoznam indexov označenia dát a dalšie informácie špecifické pre daný typ modelu. Uložením konvolučnej neurónovej siete sa ukladá graf priebehu trénovania.

Naopak pre načítanie modelov je implementovaná funkcia *load_model_data()* v triede *DataLoader* v scripte *loader.py*, ktorá načíta binárnu podobu modelu a všetky informácie z konfiguračného súboru *settings.json*.

### 4.2.2 Trénovanie modelov

Pre trénovanie modelov sú v rámci implementácie použité všetky triedy, ktoré boli podrobne opísané vyššie. Proces trénovania prebieha v niekoľkých krokoch.

1. Načítanie vstupných dát.
2. Predspracovanie načítaných dát.
3. Rozdelenie predspracovaných dát do viacerých skupín.


Obr. 4.5: Rozdelenie dát do skupín, na 3 skupiny pre CNN a na 2 skupiny pre K-means a SVM klasifikátor.

4. Vytvorenie modelu.
5. Spustenie trénovania vytvoreného modelu.
6. Uloženie natrénovaného modelu a konfiguračného súboru.
7. Testovanie presnosti modelu pomocou testovacích dát.

Pre uľahčenie trénovania je implementovaný script *train.py*, ktorý požaduje niekoľko argumentov na vstupe. Zoznam vstupných argumentov a ich význam, argumenty v hranatých zátvorkách sú voliteľné:

- –**model** cesta k priečinku, do ktorého bude natrénovaný model uložený.
- –**dataset** cesta k priečinku, z ktorého budú načítane vstupné dáta.
- –**alg** voľba modelu, ktorý bude trénovaný. Tento argument má niekoľko možností. Možnosť “cnnc” pre trénovanie konvolučnej neurónovej siete určenej na klasifikáciu zbrane, “cnna” pre trénovanie konvolučnej neurónovej siete pre určenie náklonu zbrane a “svm” alebo “kmeans” pre trénovanie SVM alebo K-nearest-neighbor klasifikátora na určenie typu zbrane.
- [–**ep**] pri konvolučných neurónových sieťach určuje počet epóch, pri jeho nezadaní je použitá základná hodnota 45.
- [–**bs**] určuje veľkosť batch-size pri trénovaní konvolučných neurónových sietí, jeho základná hodnota je 16.
- [–**rt**] je argument, ktorý je potrebné zadať pri trénovani sietí na určenie náklonu zbrane, jeho hodnota určuje uhol náklonu, na ktorý sa sieť bude učiť, možné hodnoty argumentu sú: “yaw”, “roll”, “pitch”.

### 4.2.3 Presnosť modelov

Pre celkové hodnotenie presnosti modelov bola implementovaná funkcia *evaluate_model* v scripte *evaluation.py*. Hodnotí presnosť modelov podľa metrík spomenutých v 3.4.3.

V prípade určenia typu zbrane sa používa kontigenčná matica a hodnota celkovej úspešnosti modela. Táto metrika je použitá aj počas trénovania CNN na predbežné určenie presnosti a taktiež pri koncovom testovaní modelov.

Avšak v prípade CNN, ktoré určujú náklon zbrane je potrebné túto metriku trocha pozmeniť. Ako je opísané v 3.3.1, v scripte *base.py* je implementovaná funkcia *angle_error*, ktorá ráta priemerný rozdiel medzi skutočnymi a predpovedanými uhlami siete. Táto funkcia je použitá ako metrika počas trénovania CNN a pri koncovom testovaní presnosti siete. Následne po určení, či je predikcia správna alebo nie, pomocou prahovej hodnoty, sa použije výpočet úspešnosti z hodnôt v chybovej matici.

## 4.3 Testovanie

Pre testovanie riešenia tejto práce sa využíva funkcia zodpovedná za evaluáciu modelov *evaluate_model* v scripte *evaluation.py* podľa metrík, ktoré boli spomenuté už vyššie. Testovanie modelov prebieha ako určovanie presnosti týchto natrénovaných modelov.

```
[INFO] Accuracy: 61.18%
  [INFO] long_weapon  26%     (32/124)
  [INFO] short_weapon 95%     (124/131)
```

Obr. 4.6: Príklad výstupu po testovaní modelu určujúceho typ zbrane.

```
[INFO] Accuracy: 2.47% Threshold:5
  [INFO] Correct: 17
  [INFO] Wrong: 671
```

Obr. 4.7: Príklad výstupu po testovaní modelu určujúceho náklon zbrane.

## 4.4 Záver implementácie

Kapitola v úvode uvádza presné počty dát, podľa zdrojov, ktoré sú použité pre tŕenovanie modelov. Vysvetľuje postup generovania obrázkov z 3D modelov. Ďalej sa venuje možnostiam načítavania dát zo súboru alebo priečinkov pri dodržaní potrebnej hierarchie priečinkov a názvov súborov. Približuje triedy, ktoré implementujú predspracovanie vstupných dát, tvorbu modelov, uvádza informácie, ktoré sú uložené pri ukladaní modelov a spôsob ich načítania. Nasleduje celý postup trénovania modelov, opis scriptu, ktorý toto trénovanie uľahčuje a vysvetľuje metriky, podľa ktorých sa hodnotí presnosť modelov.

# Kapitola 5

# Výsledky

Pre porovnanie a určenie najlepšieho postupu pre klasifikáciu zbrane a určenie náklonu zbrane v scéne, bude kapitola zahŕňať všetky výsledky navrhovaných postupov, ich popis a finálne vyhodnotenie.

## 5.1 Určenie typu zbrane

Pre určenie typu zbrane v dvoch kategóriách a to na krátke a dlhé zbrane, boli navrhnuté dva klasické prístupy K-Nearest-Neighbor a SVM klasifikátor a dve architektúry pre konvolučné neurónové siete.

### 5.1.1 Klasický prístup

Pre trénovanie K-Nearest-Neighbor a SVM boli načítané všetky vstupné trénovancie dáta (1377 pre krátke zbrane a 850 pre dlhé zbrane), následne boli tieto dve skupiny zarovnané na rovnaký počet obrázkov pre každú kategórií na 850 obrázkov. Veľkosť tejto vstupnej databázy bola, pomocou augmentácie dát, zväčšená päťnásobne a rozdelená na trénovacie a testovacie dáta. Tieto augmentované dáta prešli predspracovaním, ktoré pozostáva z normalizácie, šedotónovej, a hoghovej transformácie. Na trénovanie bolo použitých 5950 obrázkov, na testovanie resp. evaluáciu modelu 2550 obrázkov.

#### SVM

Trénovanie SVM klasifikátora prebiehalo so základnými nastaveniami, ktoré nastavuje scikit-learn knižnica pre triedu $SVC$[1]. Avšak pre porovnanie rôznych výsledkov bol pre dalšie trénovanie ešte zmenený kernel v SVM klasifikátore na "linear".

Štatistiky nižšie zobrazujú výsledky týchto dvoch SVM klasifikátorov, tabuľky vychádzajú z chybovej matice avšak zobrazujú ešte aj úspešnosť určenia v danej kategórií, ktorá je vyrátana ako

$$uspesnost = \frac{spravne\_urcene}{spravne\_urcene + nespravne\_urcene} * 100$$

Výsledky ďalej ešte uvádzajú celkový výpočet úspešnosti.

[1] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 761 | 536 | **61%** |
| | Krátke zbrane | 493 | 760 | **59%** |

Tabuľka 5.1: Výsledky klasifikácie pre SVM so základnými nastaveniami.

Celková úspešnosť je 59.65% (viď. 3.4.3).

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 714 | 540 | **57%** |
| | Krátke zbrane | 498 | 798 | **62%** |

Tabuľka 5.2: Výsledky klasifikácie pre SVM s “linear” kernelom.

Celková úspešnosť je 59.29%.

#### K-Nearest-Neighbor

Trénovanie K-Nearest-Neighbor klasifikátora prebiehalo dvakrát, s prvým nastavením $k = 5$, kde $k$ je celé číslo, ktoré určuje počet najbližších susedov na základné, ktorých sa určuje kategória. A s nastavením $k = 1$.

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 880 | 374 | **70%** |
| | Krátke zbrane | 411 | 885 | **68%** |

Tabuľka 5.3: Výsledky klasifikácie pre K-Nearest-Neighbor s $k = 5$.

Celková úspešnosť je 69.22%.

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 881 | 373 | **70%** |
| | Krátke zbrane | 405 | 891 | **69%** |

Tabuľka 5.4: Výsledky klasifikácie pre K-Nearest-Neighbor s $k = 1$.

Celková úspešnosť je 69.49%.

### 5.1.2 Konvolučné neurónové siete

Pre určenie typu zbrane pomocou konvolučných neurónových sietí, boli použité dve navrhované architektúry (viď. 3.4.2). Presné nastavenia a parametre siete boli popísané v návrhu. Počet a rozdelenie vstupných dát na rovnaký pomer je totožný s uvedeným pri výsledkoch v klasickom prístupe, augmentácia dát prebiehala vždy počas každej epochy trénovania. Celkovo bolo na trénovanie použitých 1190 obrázkov, na validáciu 255 a na testovanie tak isto 255 obrázkov.

### AlexNetLike

Prvá archiktúra bola inšpirovaná AlexNet architektúrov, jej trénovanie prebiehalo v 50 epochách. Kedže, ako je vidno na obrázku 5.1, od 40.epochy už veľké zlepšenie neprebiehalo, preto nebolo potrebné trénovať sieť ďalej.


Obr. 5.1: Priebeh trénovania siete AlexNetLike.

Kde *acc* označuje celkovú presnosť siete na trénovacích dátach, *val_acc* presnosť siete na validačných dátach, *train_loss* a *val_loss* je tzv. *loss function* alebo označovaná aj ako *cost function*, ktorá v jednoduchosti určuje nepresnosť danej siete, to znamená, že čím nižšia hodnota, tým by mala sieť dosahovať lepšie výsledky.

Výsledná presnosť na testovacích dátach je v tabuľke 5.5.

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 117 | 7 | **94%** |
| | Krátke zbrane | 95 | 36 | **73%** |

Tabuľka 5.5: Výsledky natrénovanej siete AlexNetLike.

Celková úspešnosť je 83.14%.

### VGGLike

Druhá architektúra bola inšpirovaná VGGLike architektúrov, jej trénovanie priebehalo v 40 epochách.


Obr. 5.2: Priebeh trénovania siete VGGLike.

Počiatočná hodnota *train_loss* bola veľmi vysoká, graf je následkom toho veľmi skreslený a nie je možné presne vidieť postup trénovanie, avšak kompletné výsledky nad testovacími dátami sú uvedené v tabuľke 5.6.

| | | Klasifikované hodnoty | | |
|---|---|---|---|---|
| | | Dlhé zbrane | Krátke zbrane | Úspešnosť |
| Správne hodnoty | Dlhé zbrane | 108 | 26 | **87%** |
| | Krátke zbrane | 63 | 68 | **48%** |

Tabuľka 5.6: Výsledky natrénovanej siete VGGLike.

Celková presnosť siete je 67.06%.

## 5.2 Určenie náklonu zbrane

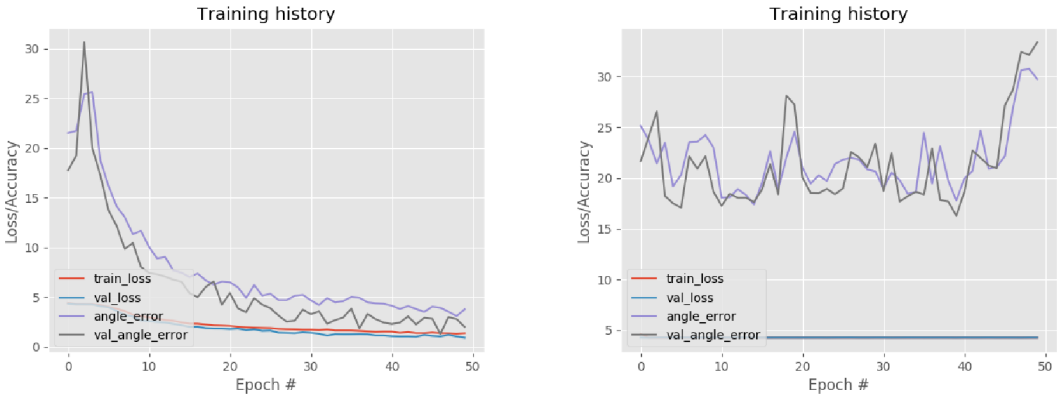
Pre určenie náklonu zbrane v troch osách boli navrhnuté dve architektúry konvolučných neurónových sietí. Každá z týchto sietí bola natrénovaná tri-krát kvôli trom osiam rotácie. Nastavenia sietí boli opísané v návrhu.

Pre trénovanie v osi pitch bolo použitých 1036 obrázkov, na validáciu 222 a taktiež 222 pre testovanie presnosti siete. V osi roll bolo na trénovanie použitých 3315 obrázkov, na validáciu 667 a 668 pre testovanie. Nakoniec pre os yaw bolo použitých 3208 obrázkov na trénovanie a po 688 na validáciu a testovanie. Všetky tieto obrázky prešli augmentáciou dát, ako je opísané v 3.5.2.

### Pitch

Siete boli trénovane v 50 epochách. Priebeh trénovania obidvoch sietí pre os Pitch je vidieť na obrázku 5.3. Kde pre presnosť týchto sietí je použitá metrika opísaná v 4.2.3. Hodnoty

*angle_error* zobrazujú celkovú presnosť siete vzhľadom na priemernú odchýlku uhla pre trénovacie dáta, *val_angle_error* určuje túto odchýlku pre validačné dáta.




Obr. 5.3: Priebeh trénovania sietí AlexNetLike(vľavo) a VGGLike(vpravo).

Výsledná presnosť sietí je uvedená v tabuľkách 5.7 a 5.8 pre dve prahové hodnoty 5 a 10.

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 189 | 33 | **85.14%** |
| 10 | 205 | 17 | **92.34%** |

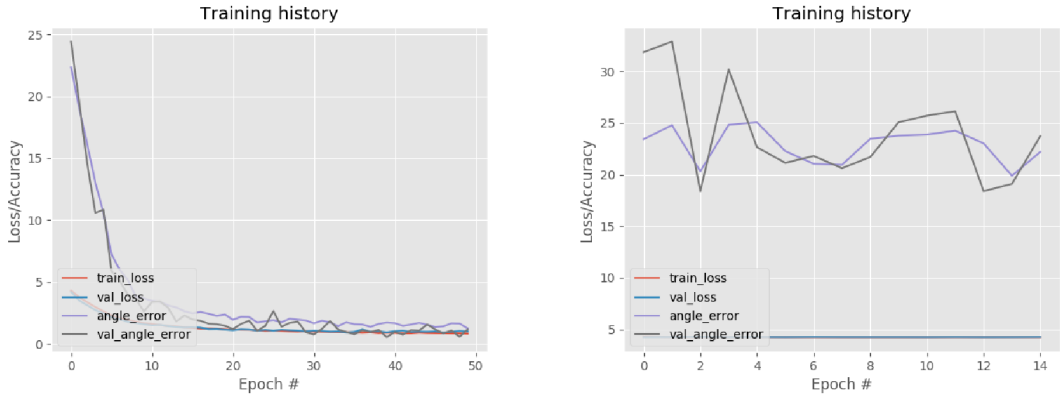
Tabuľka 5.7: Výsledky natrénovanej siete AlexNetLike pre os Pitch.

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 9 | 213 | **4.05%** |
| 10 | 17 | 205 | **7.66%** |

Tabuľka 5.8: Výsledky natrénovanej siete VGGLike pre os Pitch.

### Roll

V prípade AlexNetLike architektúry prebiehalo trénovanie rovnako ako v prípade osi Pitch, avšak pri sietí VGGLike bolo potrebné znížiť počet interácií trénovania kvôli dlhej dobe spracovania jednej interácie, kedže trénovanie prebiehalo na väčšom počte dát oproti trénovaniu v ose Pitch. Aj z priebehu trénovanie zobrazeného na obrázku 5.4 je možné vidieť že sieť veľmi nenapreduje k lepším výsledkom.




Obr. 5.4: Priebeh trénovania sietí AlexNetLike(vľavo) a VGGLike(vpravo).

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 608 | 60 | **91.02%** |
| 10 | 638 | 30 | **95.51%** |

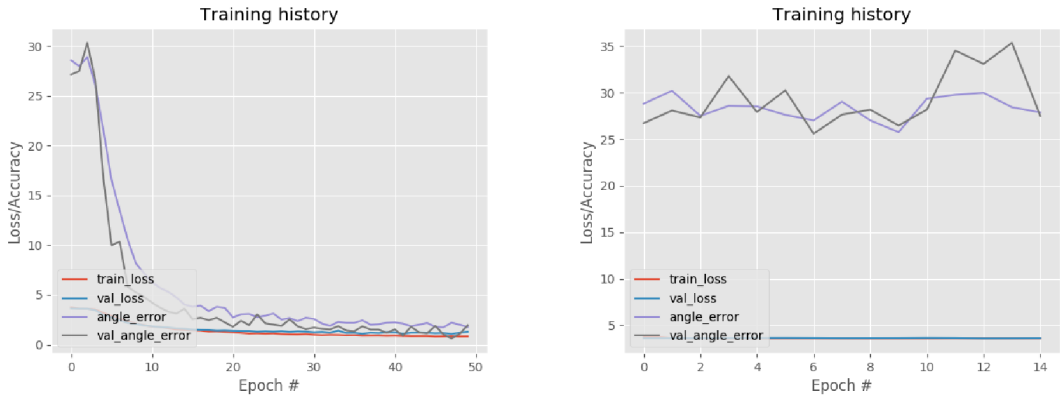
Tabuľka 5.9: Výsledky natrénovanej siete AlexNetLike pre os Roll.

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 25 | 643 | **3.74%** |
| 10 | 36 | 632 | **5.39%** |

Tabuľka 5.10: Výsledky natrénovanej siete VGGLike pre os Roll.

### Yaw

Trénovanie pre os Yaw prebiehalo rovnako ako v prípade osi Roll, počet epóch pre sieť VGGLike bol rovnako skrátený na 15 kvôli časovej náročnosti trénovania.




Obr. 5.5: Priebeh trénovania sietí AlexNetLike(vľavo) a VGGLike(vpravo).

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 342 | 346 | **49.71%** |
| 10 | 376 | 312 | **54.65%** |

Tabuľka 5.11: Výsledky natrénovanej siete AlexNetLike pre os Yaw.

| Prahová hodnota | V prahovej hodnote | Mimo prahovej hodnoty | Úspešnosť |
|---|---|---|---|
| 5 | 26 | 662 | **3.78%** |
| 10 | 41 | 647 | **5.96%** |

Tabuľka 5.12: Výsledky natrénovanej siete VGGLike pre os Yaw.

## 5.3 Zhodnotenie výsledkov

V tabuľke 5.13 je uvedené celkové porovnanie výsledkov pre určenie typu zbrane. Z týchto výsledkov môžeme vidieť že navrhovaná architektúra AlexNetLike dosiahla najlepšie čiastočné ale hlavne aj celkové výsledky až 83.14%. Farebne sú vyznačené najlepšie (zelené) a najhoršie (červené) výsledky pre jednotlivé presnosti.

| Názov siete | Dlhá zbraň - presnosť | Krátka zbraň - presnosť | Celková presnosť |
|---|---|---|---|
| SVM | 61% | 59% | **59.65%** |
| SVM - linear | 57% | 62% | **59.29%** |
| KMeans - 1 | 70% | 69% | **69.49%** |
| KMenas - 5 | 70% | 68% | **69.22%** |
| AlexNetLike | 94% | 73% | **83.14%** |
| VGGNetLike | 87% | 48% | **67.06%** |

Tabuľka 5.13: Celkové porovnanie výsledkov pre určenie typu zbrane.

Ďalej tabuľka 5.14 zobrazuje výsledky pre jednotlivé osi a dve navrhované architektúry. Je jasne vidieť, že architektúra AlexNetLike rádovo predskočila úspešnosťou druhú navrhovanú architektúru. V hlavičke tabuľky je uvedené $p$, ktoré označuje hodnotu prahovej hodnoty pre určenie správnej predikcie. Farebne sú znova vyznačené najlepšie a najhoršie výsledky.

| Os rotácie | Názov siete | Presnosť pre p=5 | Presnosť pre p=10 |
|---|---|---|---|
| Pitch | AlexNetLike | **85.14%** | **92.34%** |
| | VGGLike | **4.05%** | **7.66%** |
| Roll | AlexNetLike | **91.02%** | **95.51%** |
| | VGGLike | **3.74%** | **5.39%** |
| Yaw | AlexNetLike | **49.71%** | **54.65%** |
| | VGGLike | **3.78%** | **5.96%** |

Tabuľka 5.14: Súhrné porovnanie dosiahnutých výsledkov pre určenie náklonu zbrane.

V závere je možné konštatovať že pre riešené problémy sú vhodnejšie menšie konvolučné neurónové siete na rozdiel od tých hlbokých, kedže veľkosť trénovacích dát je veľmi malá, rádovo len v pár tisíckach.

Výsledkom tohto porovnania sú štyri najlepšie modely, AlexNetLike pre určenie typu zbrane a tri AlexNetLike modely pre určenie náklonu zbrane, tieto štyri modeli je možné použiť pre celkovú predikciu pomocou scriptu *predict.py*.

Kde ako vstupné argumenty je potrebné vložiť cestu k týmto štyrom modelom, mená argumentov sú *–class* pre model určujúci typ zbrane, *–anglep* pre model určujúci náklon v ose pitch, *–angler* pre os roll, *–angley* pre os yaw a argument *–image* ktorého hodnota je cesta k obrázku použitého na predikciu.

## 5.4 Možnosti budúceho vývoja

Odbor počítačového videnia je veľmi široký a možnosti pre spracovania obrazu je mnoho. Táto práca sa zamerala a zhrnula iba niekoľko základných postupov. Preto by ďalšie rozšírenie tejto práca bolo určite možné a prínosné pre nájdenie lepšieho riešenia, pre problém ktorý táto práca rieši.

Ako jeden z dalších postupov by bolo možné využiť už natrénované hlboké konvolučné neurónové siete a porovnať ich výsledky. Za další z bodov budúceho vývoja určite stojí aj rožšírenie a prepracovanie generovania vstupných dát, aby bolo možné pracovať s väčšími dátovými sadami. Taktiež pre klasifikáciu typov zbraní by bolo vhodné rozlišovať nie len kategórie na krátke a dlhé, ale napríklad klasifikácia na automatické a poloautomatické zbrane, či jemnejšie rozdelenie pre klasifikovanie presného typu zbrane.

Pre budúci rozvoj práce určite existuje niekoľko smerov, ktorými môže tento vývoj pokračovať.

# Kapitola 6

# Záver

V práci bolo navrhnutých niekoľko spôsobov pre klasifikovanie typu a určenie náklonu zbrane, od klasických prístupov, ktoré využívaju tranformáciu obrazu do šedotónového obrazu a hoghovu transformáciu obrazu pomocou klasifikatórov ako K-Nearest-Neighbour a Support Vector Machine.

Až po využitie moderných spôsobov spracovania obrazu, ako sú konvolučné neurónové siete, kde boli navrhnuté dve architektúry týchto sietí.

Dosiahnuté výsledky boli porovnané a následne vybrané najlepšie riešenie s presnosťou klasifikácia zbrane až 83.14% pomocou konvolučnej neurónovej siete. Taktiež boli dosiahnuté vysoké percentá úspešnosti pre určenie náklonu zbrane použitím konvolučnej neurónovej siete od najhoršieho výsledku 54.64% v osi yaw až po presnosť 92.34% v osi pitch a 95.51% v osi roll.

Pri klasifikácii krátkych zbraní ktoré obsahovali tlmič bolo zistené nesprávne určovanie kategórie, avšak po konzultáciach s vedúcim práce sme sa rozhodli tento problém neriešiť a tak tento problém môže byť jeden z bodov budúceho vývoja tejto práce. Taktiež pre budúci vývoj práce by bolo možné rožšíriť veľkosť vstupnej databázy dát a využiť pre klasifikáciu už predtrénované hlboké konvolučné neurónové siete.

# Literatúra


[1] Aircraft principal axes. online, [Online; navštívené 09.05.2018].
URL https://en.wikipedia.org/wiki/Aircraft_principal_axes

[2] Battle of the Deep Learning frameworks. online, [Online; navštívené 03.05.2018].
URL https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

[3] Choosing an Open Source Machine Learning Library. online, [Online; navštívené 04.05.2018].
URL https://www.altexsoft.com/blog/datascience/choosing-an-open-source-machine-learning-framework-tensorflow-theano-torch-scikit-learn-caffe/

[4] Commonly used activation functions. online, [Online; navštívené 07.01.2018].
URL http://cs231n.github.io/neural-networks-1/

[5] A comparison of deep learning frameworks. online, [Online; navštívené 03.05.2018].
URL https://www.exastax.com/deep-learning/a-comparison-of-deep-learning-frameworks/

[6] Convolutional Neural Networks (CNNs / ConvNets). online, [Online; navštívené 09.01.2018].
URL http://cs231n.github.io/convolutional-networks/

[7] An illustration of the architecture of AlexNet CNN. online, [Online; navštívené 02.05.2018].
URL https://www.researchgate.net/figure/An-illustration-of-the-architecture-of-AlexNet-CNN-14_fig4_312188377

[8] Image Classification. online, [Online; navštívené 06.01.2018].
URL http://cs231n.github.io/classification/

[9] Keras: The Python Deep Learning library. online, [Online; navštívené 09.01.2018].
URL https://keras.io/

[10] Support Vector Machines. online, [Online; navštívené 07.01.2018].
URL http://scikit-learn.org/stable/modules/svm.html#classification

[11] TensorFlow. online, [Online; navštívené 09.01.2018].
URL https://www.tensorflow.org/

[12] Use of convolutional neural network for image classification. online, [Online; navštívené 02.05.2018].

URL https://www.apsl.net/blog/2017/11/20/use-convolutional-neural-network-image-classification/

[13] What are some ways of pre-procesing images before applying convolutional neural networks for the task of image classification? online, [Online; navštívené 04.05.2018].
URL https://www.quora.com/What-are-some-ways-of-pre-procesing-images-before-applying-convolutional-neural-networks-for-the-task-of-image-classification

[14] What is the VGG neural network? online, [Online; navštívené 02.05.2018].
URL https://www.quora.com/What-is-the-VGG-neural-network

[15] B, N.: Image Data Pre-Processing for Neural Networks. online, [Online; navštívené 10.01.2018].
URL https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258

[16] Boswell, D.: Introduction to Support Vector Machines. 2002.

[17] Chen, Y.; Li, S.: Classes of Kernels for Machine Learning: A Statistics Perspective. *Journal of Machine Learning Research 2*, 2001: s. 299–312.

[18] Chen, Y.; Li, S.: A Brief Introduction to Hilbert Space. 2016.

[19] Deshpande, A.: The 9 Deep Learning Papers You Need To Know About. online, [Online; navštívené 02.05.2018].
URL https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

[20] Everingham, M.; Van Gool, L.; Williams, C. K. I.; aj.: The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, ročník 88, č. 2, Jun 2010: s. 303–338, ISSN 1573-1405, doi:10.1007/s11263-009-0275-4.
URL https://doi.org/10.1007/s11263-009-0275-4

[21] Faktor, Z.: *Střelné zbraně*. Praha: Magnet-Press, 1995, ISBN 80-85847-46-9.

[22] Geirhos, R.; Janssen, D. H. J.; Schütt, H. H.; aj.: Comparing deep neural networks against humans: object recognition when the signal gets weaker. 2017.

[23] Girshick, R.: Fast R-CNN. *Microsoft Research*, 2015.

[24] Girshick, R.; Donahue, J.; Darrell, T.; aj.: Rich feature hierarchies for accurate object detection and semantic segmentation. *UC Berkeley*, 2014.

[25] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[26] Hosang, J.; Benenson, R.; Dollár, P.; aj.: What makes for effective detection proposals? *IEEE transactions on pattern analysis and machine intelligence*, 2016: s. 814–830.

[27] Kevin, M.: Simple guide to confusion matrix terminology. online, [Online; navštívené 07.05.2018].
URL
http://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

[28] Kotsiantis, S.: Supervised Machine Learning: A Review of Classification Techniques. *Department of Computer Science and Technology*, 2007: s. 249–268.

[29] Larose, D. T.: *Discovering Knowledge in Data: An Introduction to Data Mining.* Wiley, 2005, ISBN 978-0471666578.

[30] Navneet, D.; Bill, T.: Histograms of Oriented Gradients for Human Detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, Washington, DC, USA: IEEE Computer Society, 2005, ISBN 0-7695-2372-2, s. 886–893, doi:10.1109/CVPR.2005.177.
URL http://dx.doi.org/10.1109/CVPR.2005.177

[31] Nielsen, M.: Using neural nets to recognize handwritten digits. online, [Online; navštívené 07.01.2018].
URL http://neuralnetworksanddeeplearning.com/chap1.html

[32] Olmos, R.; Tabik, S.; Herrera, F.: Automatic handgun detection alarm in videos using deep learning. *Neurocomputing*, ročník 275, 2018: s. 66 – 72, ISSN 0925-2312, doi:https://doi.org/10.1016/j.neucom.2017.05.012.
URL http://www.sciencedirect.com/science/article/pii/S0925231217308196

[33] Redmon, J.; Divvala, S. K.; Girshick, R. B.; aj.: You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016: s. 779–788.

[34] Rey, J.: Object detection: an overview in the age of Deep Learning. online, [Online; navštívené 01.05.2018].
URL https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/

[35] Saez, D.: Correcting Image Orientation Using Convolutional Neural Networks. online, [Online; navštívené 07.01.2018].
URL https://d4nst.github.io/2017/01/12/image-orientation/

[36] Sinčák, P.; Andrejková, G.: Neurónové siete: Inžiniersky prístup. online, [Online; navštívené 07.01.2018].
URL http://neuron-ai.tuke.sk/cig/source/publications/books/NS1/html/node8.html

[37] Sonka, M.; Hlavac, V.; Boyle, R.: *Image processing, analysis, and machine vision.* Thomson, international student ed vydání, 2007, ISBN 978-0495082521.

[38] Suchacka, G.; Skolimowska-Kulig, M.; Potempa, A.: A k-Nearest Neighbors Method for Classifying User Sessions in E-Commerce Scenario. *Journal of Telecommunications & Information Technology*, 2015: s. 64–69, ISSN 1509-4553.

[39] Thieme, H.: Lower Palaeolithic hunting spears from Germany. *A natureresearch*, 1996: str. 807–810, ISSN 1476-4687.

[40] Tinku Acharya, A. K. R.: *Image Processing: Principles and Applications.* Wiley-Interscience, 2005, ISBN 978-0471719984.

[41] Viola, P. A.; Jones, M. J.: Robust Real-Time Face Detection. *International Journal of Computer Vision*, ročník 57, 2001: s. 137–154.

[42] van der Walt, S.; Schönberger, J. L.; Nunez-Iglesias, J.; aj.: scikit-image: image processing in Python. *PeerJ*, ročník 2, 6 2014: str. e453, ISSN 2167-8359, doi:10.7717/peerj.453.
URL http://dx.doi.org/10.7717/peerj.453