VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

URČENÍ NATOČENÍ HLAVY NA SNÍMKU NEURONOVOU SÍTÍ

HEAD POSE ESTIMATION IN AN IMAGE BY A NEURAL NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **LUKÁŠ RYBNIKÁR**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. FILIP ORSÁG, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Rybnikár Lukáš**
Program: Informační technologie
Název: **Určení natočení hlavy na snímku neuronovou sítí**
Head Pose Estimation in an Image by a Neural Network
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte algoritmy zpracování obrazu a základy neuronových sítí (zaměřte se na konvoluční neuronové sítě) a odpovídající softwarové nástroje (knihovnu OpenCV a framework Tensorflow).
2. Navrhněte aplikaci, která využije algoritmů zpracování obrazu a klasifikátoru v podobě neuronové sítě za účelem přibližného určení úhlů natočení obličeje nalezeného ve snímku.
3. Navrženou aplikaci implementujte v libovolném programovacím jazyce.
4. Připravte si množinu trénovacích a testovacích dat pomocí poskytnutého generátoru a proveďte experimenty.
5. Zhodnoťte, jak odpovídá určení natočení hlavy ve snímku skutečnosti a jaké jsou slabé a silné stránky navrženého řešení.

Literatura:

- SZEGEDY, Christian, et al. Going deeper with convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015. p. 1-9.
- VU, Tuan-Hung; OSOKIN, Anton; LAPTEV, Ivan. Context-aware CNNs for person head detection. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015. p. 2893-2901.
- KHAN, Salman; RAHMANI, Hossein; SHAH, Syed Afaq Ali; BENNAMOUN, Mohammed. A Guide to Convolutional Neural Networks for Computer Vision. Morgan & Claypool Publishers. 2018. ISBN: 978-1681730219.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Orság Filip, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 1. listopadu 2018

Abstrakt

Umelé neurónové siete nie sú novinkou, ale posledné roky zaznamenali na populárnosti a dostali sa viac do povedomia širokej verejnosti. Táto bakalárska práca sa zameriava na zistenie uhlu, pod ktorým sa nachádza hlava človeka na obrázku pomocou konvolučných neurónových sietí. Oblastí, v ktorých možno neurónové siete využiť, je veľa a posledné roky máme k dispozícií hardvér, ktorý nám umožňuje tieto siete učiť v bežne prístupných podmienkach. V teoretickej časti sa zoznámime s tým čo sú to neurónové siete, ako fungujú, ako sa delia a popíšeme si konvolučné neurónové siete. V praktickej časti sa zoznámime s nástrojmi, ktoré budeme používať, začneme experimentovať, zisťovať vhodnú konfiguráciu neurónovej siete a pokúsime sa získať čo najlepší výsledok.

Abstract

Artificial neural networks are not a novelty, but their recent rise in popularity is noticeable as well as their gain of attention from the masses. This bachelor thesis focuses on the head pose estimation in an image using the convolution neural networks. The fields of use of neural networks are vast and during last years strong enough hardware has been developed to allow us to train these networks under commonly accessible conditions. In theoretical part there are neural networks introduced with an explanation of what they are, how they work, how they are divided followed by a detailed description of convolutional neural networks. In the practical part the necessary tools used for development needed to perform experiments, such as determining appropriate configuration for neural network and optimization to get the best results possible, are described.

Kľúčové slová

Detekcia tváre, Natočenie tváre, Konvolučná neurónová sieť, Strojové učenie, Umelá neurónová sieť, Neurón, Perceptron, Umelá inteligencia

Keywords

Face detection, Head pose, Convolution neural network, Machine learning, Artificial neural network, Neuron, Perceptron, Artificial Intelligence

Citácia

RYBNIKÁR, Lukáš. *Určení natočení hlavy na snímku neuronovou sítí*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Filip Orság, Ph.D.

Určení natočení hlavy na snímku neuronovou sítí

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pana Ing. Filipa Orsága, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Lukáš Rybnikár
15. mája 2019

Poďakovanie

Týmto by som chcel poďakovať vedúcemu Ing. Filip Orság, Ph.D. za odborné rady pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

V dnešnej dobe počujete o umelej inteligencií, strojovom učení a umelých neurónových sietiach čoraz častejšie. Umelé neurónové siete sa dostávajú do viac a viac oblastí a ľudia sa s nimi stretávajú dennodenne. Majú čoraz väčší vplyv na náš každodenný život aj keď si to mnohý neuvedomujú.

Spracovanie obrazu je v dnes každodenná záležitosť, či už sa jedná o spracovanie videa v reálnom čase, hľadanie spoločných znakov v snímkoch alebo detekcia objektov a ich vlastností. A práve na tieto veci sa veľmi hodia neurónové siete, ktoré sú vďaka svojej univerzálnosti, abstrakcií a vlastnostiam pri učení veľmi vhodným nástrojom. Vzhľadom na zvyšujúci sa výkon osobných počítačov je možné si vytvárať a učiť čoraz komplexnejšie neurónové siete aj v domácom prostredí.

Cieľom tejto práce je pomocou obrázku zistiť v akej pozícií je hlava, ktorá je na ňom zobrazená. K tomuto účelu budú použité konvolučné neurónové siete, ktoré sú na spracovanie obrazu špecializované. Existuje veľká škála už existujúcich modelov, naučených sieti, ktoré sú dostupné ale všetky sú obmedzené na to že sa učili na obmedzenej množine dát, ktorá je častokrát veľmi špecifická. Táto práca bude zameraná na vlastnú sieť, jej zdokonalovanie a hľadanie optimálneho riešenia, ktoré bude vhodné práve pre daný problém.

Na začiatku budú predstavené neurónové siete, to ako vznikli, ich podobnosť s reálnymi neurónovými sieťami, to ako fungujú, na čo sa dajú využívať a to kde sa skrývajú ich problémy. Rozoberieme si podrobnejšie konvolučné neurónové a ich využitie pre túto prácu.

V ďalšej časti bude zanalyzovaný problém, vybrané vhodné nástroje pre jeho riešenie. Navrhne sa postup a riešenie, to aké dáta použiť a akým spôsobom ich spracovať, bude vytvorená neurónová sieť a pomocou experimentov bude upravovaná s myšlienkou získať čo najlepší výsledok.

Na záver budú zhodnotené výsledky, porovnanie chovanie sa siete nad rôznymi setmi dát a zhodnotenie do akej miery sa podarilo uspieť a kde sú nedostatky daného riešenia a návrh možných vylepšení.

Kapitola 2

Neuronové siete

Keďže táto bakalárska práca pracuje s neurónovými sieťami tak si v tejto kapitole objasníme čo sú umelé neuronové siete. V informatike, tak ako aj v iných vedných smeroch sa mnohé algoritmy a riešenia istých problémov inšpirujú v prírode. Umelé neuronové siete boli inšpirované nervovým systémom ktorý môžeme nájsť u živočíchov v prírode. V oboch prípadoch sú základné stavebné jednotky(neuróny) poskladané do zložitejších štruktúr.

Keďže prirodzené neuronové siete sú vytvorené evolúciou, tým pádom by mali byť optimalizované ale evolúcia často vyberá prvé riešenie ktoré je lepšie ako to súčasné a nepokúša sa nájsť lepšie riešenie. Ďalší problém je v tom že doteraz nie je presne objasnené ako presne fungujú prírodné neuróny a neuronové siete. Preto sa používa v informatike len aproximácia týchto sietí ktorá sa dá popísať matematicky, jej umelé neuróny sa skladajú z tréningových parametrov a dôležitým faktorom je aj to aby bola opísateľná digitálne a mohli ich spracúvať počítače.

2.1 História neurónových sietí

História[**?**] umelých neurónových sieti siaha do roku 1943 kedy Warren McCulloch a Walrterom Pittseom vytvorili prvý umelý model neurónu kde ukázali ako sú efektívne využiteľné neuronové siete v oblasti Boolovských funkcii. Ich prácu posunul ďalej Donald Hebb ktorý v knihe „Organization of Behavior“ popísal pravidlá učenia neurónových sietí.

V 1958 Frank Rosenblatt vytvoril prvý model perceptrónu. Bol to prvý model neurónovej siete ktorý využíval jeden umelý neurón.

Ďalšími veľkými osobnosťami neurónových sietí boli Marvin Minsky a Seymour Papert ktorý roku 1969 prišli s knihou „Perceptron“ na základe čoho vznikli debaty ohľadne toho ako sú neuronové siete nepoužiteľné na riešenie problémov lebo sa nevedeli vysporiadať napríklad s funkciou XOR.

Roku 1982 John Hopefeld prišiel s algoritmom na spätné šírenie(Back-Propagation) určeným pre učenie viacvrstvových neurónových sietí. Následne v roku 1986 prišli Rumelhart, Hinton a Williams s pravidlom pre učenie neurónových sietí pomocou Back-Propagation ktorý sa používa v mnohých sietiach dodnes a umožnili použitie neurónových sietí v množstve aplikácií na spracovanie jazyka, signálov, objektov, atď.

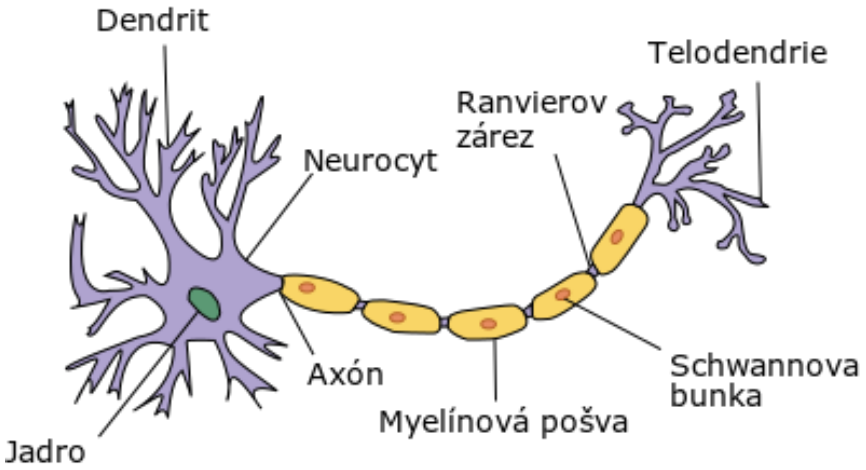
V súčasnej dobe sa využívajú rôzne druhy neurónových sietí ktoré sa líšia variabilným počtom vrstiev a možnosťami učenia. Rovnako vzniká aj hardvér ktorý je na počítanie týchto sieti optimalizovaný.

2.2 Neurón

Neurón, známy aj ako nervová bunka je základnou stavebnou časťou nervovej sústavy. Každá nervová sústava je jedinečná ako napríklad otlačky prstov. Biologické neuronové siete poslúžili ako vzor pre tie umelé ktoré sa používajú v informatike. V tejto podkapitole je vysvetlené ako tieto neuróny fungujú a ako jedny s druhými súvisia.

Biologický neurón

Neuróny[8] sú bunky ktoré sú schopné pomocou chemických reakcií vytvárať slabé elektrické impulzy a tým sa dorozumievať s okolitými neurónmi. Je tvorený telom, nazývaným aj soma, ktoré obsahuje jadro. Z tela vybiehajú kratšie výbežky nazývané dendrity slúžiace ako vstupy do neurónu a jeden dlhší výbežok nazývaný axón alebo neurit ktorý sa na konci pripája na dendrit iného neurónu. Toto spojenie nazývame synapsia. Neurón môže mať vstup až z niekoľkých tisícov neurónov. V jadre sa vyhodnocujú všetky podnety ktoré prichádzajú cez dendrity. V prípade že hodnota presiahne určitý prah významnosti tak sa aktivuje synapsia a prenesie informáciu o prekročení prahu na všetky ďalšie elektróny ktoré sú na neho napojené. Každý neurón môže mať exhibičné, tie ktoré zvyšujú hodnotu, alebo inhibičné, tie ktoré túto hodnotu znižujú. Tak isto má každý vstup inú váhovú hodnotu pre daný neurón. Tieto prepojenia vytvárajú neurónovú sieť. Tieto vlastnosti poslúžili ako vzor pre umelý neurón a ich neuronové siete ktoré sa používajú v dnešnej dobe.

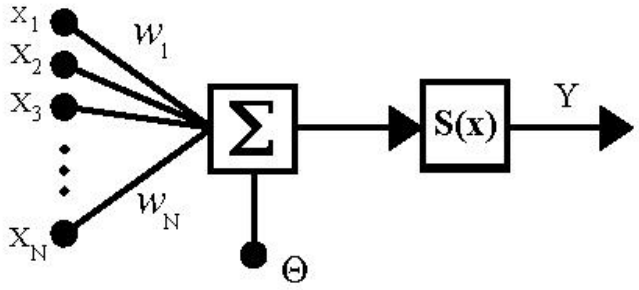


Obr. 2.1: Biologický neurón[1]

[1] https://sk.wikipedia.org/wiki/Neurón

Umelý neurón

Umelý neurón[5], tiež známi aj ako logický je abstrakcia toho biologického. Nie je možné vytvoriť presný model keďže tie biologické sú komplexné a doposiaľ nie je presne vysvetlený spôsob ako spracúvajú informácie a tým pádom ho nie je možné popísať matematicky a následne spracovať pomocou počítača. Neurón je binárny, takže jeho stav môže byť 1 alebo 0. Biolgické dendrity sú popísané ako vektorový vstup $\vec{x} = (x_1 .. x_i)$. Vektor môže byť inhibičný alebo exhibičný, podľa toho pridáva alebo uberá na váhe. Ďalej sú tieto vstupy ohodnotené váhami ktoré tvoria vektor $\vec{w} = (w_1 .. w_i)$. Telo tvorí funkcia $f(n)$ ktorá sa často označuje aj ako prenosová alebo aktivačná. Prahový koeficient, nazývaný aj bias, ktorý označuje kedy sa aktivuje daný neurón značíme ako θ(theta). Axóny ktoré tvoria výstup z neurónu označujeme ako y.



Obr. 2.2: Model umelého neurónu[2].

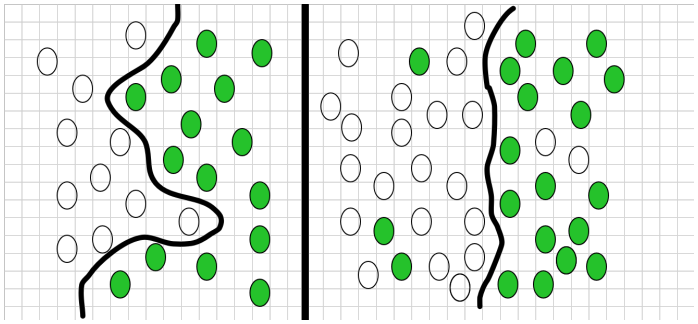
Tento model ktorý je vidieť na obrázku ide teda popísať vzorcom 2.1. Ide o matematický popis perceptŕonu od Rosenblatta obohatenú o prah. V jeho v verzii sa neurón aktivuje vo chvíli keď keď nadobudne kladných hodnôt, hodnota prahu je teda 0 a preto sa neuvádza. Takáto funkcia sa nazýva Heavisideova funkcia 2.2 s prahom 0 nazývaná aj skoková funkcia.

$$y = f(\sum_{i=1}^{n} x_i w_i + \theta) \tag{2.1}$$

$$f(x) = \begin{cases} 0 & x < \theta \\ 1 & \theta \leq x \end{cases} \tag{2.2}$$

[2] https://cs.wikipedia.org/wiki/Umělá_neuronová_síť

V praxi sa používajú rôzne druhy prenosových funkcií, často sa v neurónových sietiach používajú ich kombinácie podľa funkcionality daného neurónu. Limitáciou perceptónu je to, že dokáže vyhodnotiť len lineárne oddeliteľné vstupy keďže ide o binárny klasifikátor.



Obr. 2.3: Znázornenie rozdelenia pomocou binárneho klasifikátora.

Najjednoduchšou neskokovou funkciou je lineárna funkcia 2.3 ktorá premieta vyhodnotený vstup priamo na výstup. Najčastejšie sa však používa prenosová funkcia v podobe hyperbolického tangenu 2.4, Gausovej 2.5 alebo sigmodialnej 4.4 funkcie. Tieto funkcie majú spojité derivácie v každom bode. Táto vlasnosť je potrebná pri učení neurónovej siete so spätným šírením chyby. V praxi sa veľmi často používa ReLU(Rectified linear unit) 2.7, alebo komplikovanejšia funkcia softmax 2.8 a ich upravené verzie.

$$f(x) = x \tag{2.3}$$

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.4}$$

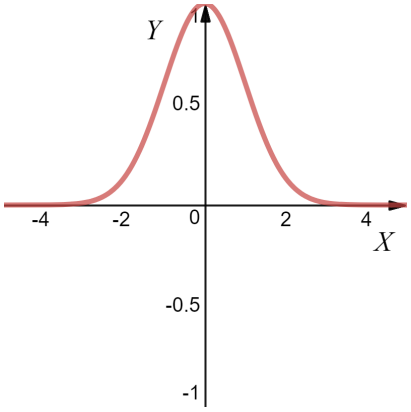
$$f(x) = e^{-x^2} \tag{2.5}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

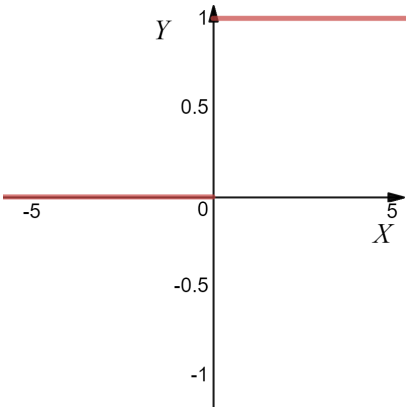
$$f(x) = \begin{cases} 0 & x < 0 \\ x & x > 0 \end{cases} \tag{2.7}$$

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{2.8}$$

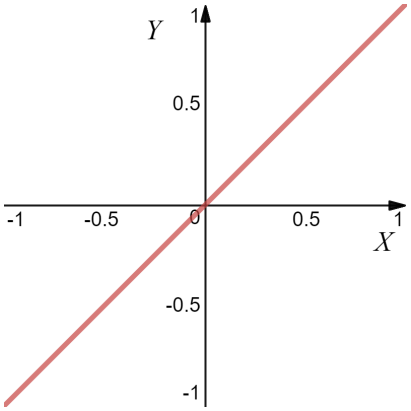
Grafy niektorých aktivačných funkcií:



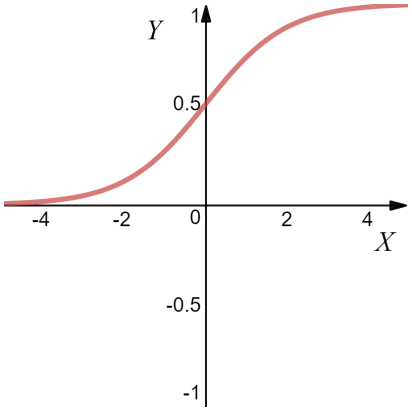
(a) Graf Guassovej funkcie.



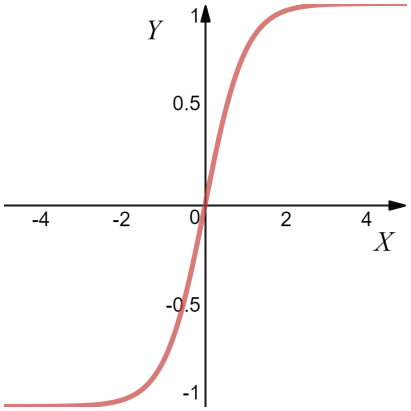
(b) Graf Heavisideovej funkcie.



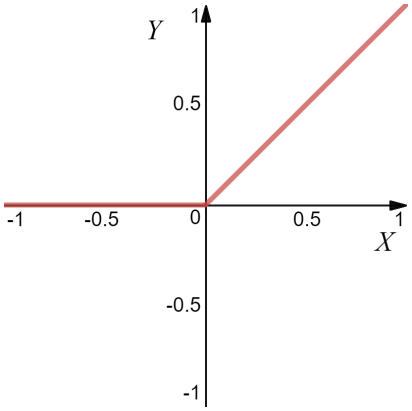
(c) Graf linernej funkcie.



(d) Graf sigmodialnej funkcie.



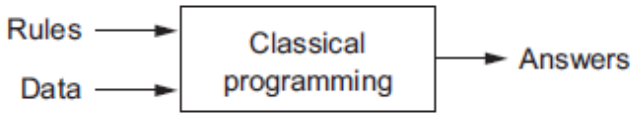
(e) Graf hyperbolického tangensu.



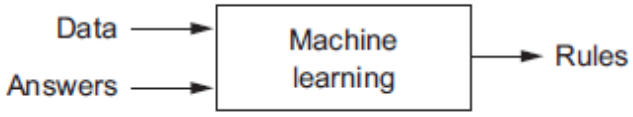
(f) Graf ReLU.

2.3 Umelá neurónová sieť

Teraz keď sme si objasnili ako funguje umelý neurón môžeme vysvetliť pojem neurónová umelá sieť[2]. Každá umelá sieť sa skladá zo vzájomne prepojených umelých neurónov tak, že neurón zoberie vstupné dáta, tie spracuje a jeho výstup sa stáva vstupom pre všetky ďalšie neuróny, ktoré sú na neho napojené a ďalej spracúvajú. To ako sú tieto neuróny medzi sebou prepojené určuje topológiu danej siete. Táto topológia je ohodnotená orientovaným grafom. Rozdiel medzi klasickým programovaním a strojovým učením pomocou neurónovej siete je v tom, že pri klasickom programovaní máme na vstupe pravidlá a dáta, tie sa spracujú a získame odpoveď. V prípade strojového učenia máme na v stupe dáta a odpovede a po spracovaní získame pravidlá, ktoré môžeme aplikovať na iné dáta.



Obr. 2.5: Obrázok znázorňujúci priebeh klasického programovania [2].

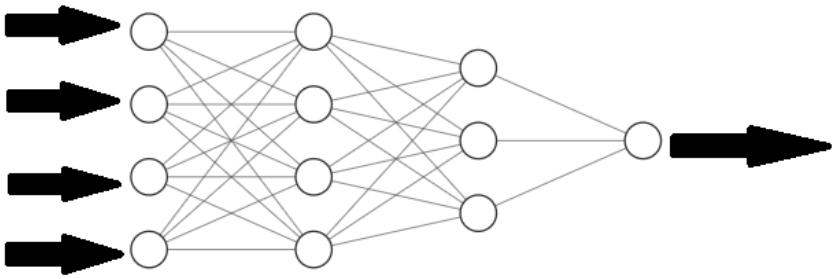


Obr. 2.6: Obrázok znázorňujúci ako prebieha strojové učenie [2].

Je viacero spôsobov ako sa dajú neuronové siete rozdeliť. Napríklad podľa určenia, podľa toho či využívajú len súčasnú vrstvu alebo aj iné, podľa topológie, podľa výstupnej vrstvy atď. Najjednoduchším modelom je práve perceptrón, ktorý dokáže určiť len 2 triedy a to často nestačí. Tento model patrí do skupiny jednovrstvových modelov. Neuróny v každej umelej neurónovej sieti môžu byť pospájané do väčších celkov, ktoré nazývame vrstvy neurónovej siete. Ak je použitá viac ako jedna vrstva označuje sa táto topológia ako viacvrstvová.

Dopredné siete

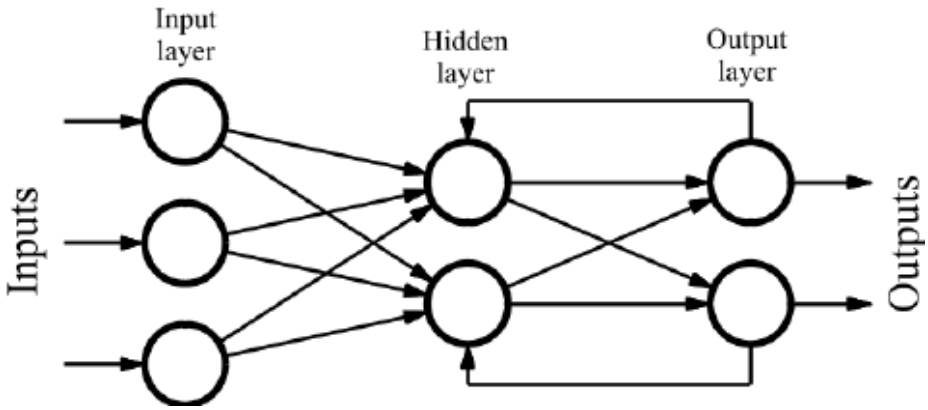
Dopredné(Feed-forward)[11], označované ako FNN siete, sú siete kde signál spracovania informácie putuje len jedným smerom a to od vstupu po výstup zo siete. Takže v prípade týchto sietí výstup z danej siete už nemôže ovplyvniť danú vrstvu ale len tie nasledujúce. Tieto siete sú vhodné pre prípady kde na základe vstupných premenných chceme zistiť ako ovplyvnia premenné na výstupe zo siete. Viac vrstevné Feed-forward siete, tiež nazývané aj viac vrstevný perceptrón sú najčastejšie používané siete v neurónových sietiach. Tento typ neurónových sietí sa používa často u rozpoznávaní vzorov na základe vstupu.



Obr. 2.7: Príklad doprednej siete.

Rekurentné siete

Označované ako RNN[12] alebo nazývané tiež aj siete s odozvou, sú siete, ktoré sú omnoho náročnejšie na zostrojenie a sú dynamické. Narozdiel od Feed-Forward sietí môžu v rekurentných sietiach putovať signály všetkými smermi a tak môže výstup z danej vrstvy ovplyvniť neuróny v tej istej vrstve alebo vo vrstvách pred ňou. Toto správanie je niekedy potrebné, dokáže riešiť veľa komplexných problémov ale tieto siete bývajú často veľmi komplikované a často sa veľmi náročne trénujú. Tým ako sú tieto siete dynamické sa ich stav mení až do bodu kedy nájdu rovnováhu a neuróny už naďalej nemenia hodnotu svojich výstupov a ostanú v rovnováhe až pokiaľ do siete nepríde nový vstup. Tieto siete sú veľmi vhodné pri spracúvaní prirodzeného jazyka ako písaného tak aj hovoreného, na rôzne predpovede ako napríklad spracovanie dát na burze a všeobecne na úlohy kde zohráva nejakú úlohu čas. Nie sú veľmi vhodné pre spracovanie obrazu.



Obr. 2.8: Príklad rekurentnej siete prebraný z [3]

[3] https://www.researchgate.net/figure/Graph-of-a-recurrent-neural-network_fig3_234055140

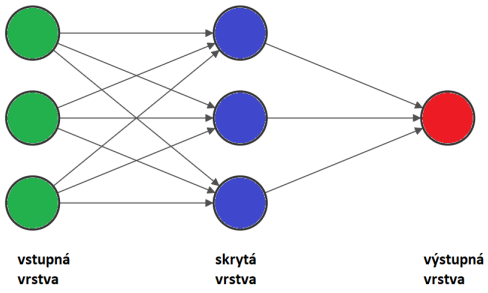
Konvolučné neurónové siete

Konvolučné neurónové siete(Convolutional Neural Networks), tiež označované ako CNN boli navrhnuté aby spracovávali obrázky na vstupe a tie premietali do výstupnej premmennej. Tento model má výhodu že si dokáže vnútorne vytvoriť dvoj dimenzionálnu reprezentáciu obrázka a vďaka tomu lepšie porozumieť škálovaniu a pozícií objektov na obrázku. Tento typ neurónovej siete tiež možno využiť aj na riešenie iných problémov ako obrázky, napríklad pri spracúvaní textu. Viac o konvolučných neurónových sietiach si rozoberieme v samostatnej kapitole keďže tieto siete sú objektom tejto práce.

V praxi sa často stretávame s takzvanými hybridnými sieťami, ktoré sa používajú veľmi často a striedajú vrstvy jednotlivých modelov.

2.3.1 Viacvrstvové neurónové siete

Ako už vyplýva z názvu, viacvrstvové neurónové[11] siete sa skladajú z viacerých vrstiev, ktoré sú poskladané za seba. Prvou vrstvou je vždy vstupná vrstva, ktorá z pravidla príma dáta vo forme tenzoru n-tého radu. Poslednou vrstvou je vždy takzvaná výstupná vrstva, ktorá obsahuje aktivačnú funkciu, ktorá nám určí typ výstupu z danej siete. Medzi týmito dvomi sa nachádzajú skryté vrstvi. Každá zo skrytých vrstiev môže mať iný účel ktorým obohacuje funkcionalitu neurónovej siete. Siete obsahujúce viac ako jednu skrytú vrstvu sa často označujú sa aj ako siete s hlbokým učením(Deep Learning).



Obr. 2.9: Znázornenie rozdelenia pomocou binárneho klasifikátora.

2.3.2 Topológie neurónových sietí

Ďalším rozdelením je rozdelenie podľa topológie neurónovej siete. Toto rozdelenie určuje to ako sú jednotlivé neuróny prepojené. Inými slovami môžeme topológiu neurónovej siete chápať ako vzťah medzi neurónmi pomocou ich prepojenia. Toto je veľmi dôležité pre funkcionalitu, spôsob učenia, ktorým sa sieť učí a jej výkonnosť.Samotná topológia sa dá rozdeliť na 2 zložky.

Prvou je neurónový rámec. Väčšina neurónových sietí, vrátane tých biologických má vrstevnú topológiu. A aj tie, ktoré nemajú vrstevnú sa často zobrazujú ako vrstvené. Napríklad v sieť kde je vstupný neurón zároveň aj výstupným sa často prezentujú ako jednovrstvové siete. Na úrovni rámca sa samotné neuróny považujú za abstraktné entity a nie je medzi nimi žiaden rozdiel. Rámec neurónovej siete môže byť popísaný podla počtu vrstiev a počtu

neurónov, ktoré každá z nich obsahuje.
(možno obrázok na rámec a číslovanie)
Druhou zložkou topológie je štruktúra prepojení. Tá určuje spôsob akým sú neuróny medzi sebou neuróny prepojené. Prepojenia možno rozdeliť na:

- *Medzi vrstevné pripojenie* - Neuróny sú prepojené s neurónmi susedných vrstiev.
- *Vnútro vrstvené pripojenie* - Neuróny sú prepojené s neurónmi tej istej vrstvy.
- *Vlastné pripojenie* - Výstup z neurónu je pripojený na vstup toho istého neurónu.
- *Skokové pripojenie* - Pripojené neuróny nie sú v dvoch susediacich vrstvách

Plne prepojená topológia

Medzi najjednoduchšie topológie patria takzvané plne prepojené topológie. Tieto topológie obsahujú vrstvy, ktoré majú takzvané plné medzi vrstevné prepojenie. Takže v každej vrstve sú prítomné všetky možné medzi vrstevné prepojenia.

2.3.3 Výstupy

Každá neurónová sieť má aspoň jednu výstupnú premennú, ktorá je výsledkom spracovania vstupu v neurónovej sieti. Tieto výstupy môžeme rozdeliť do troch základných kategórií podľa ktorých sa dajú deliť umelé neurónové siete[5]. Jedná sa o klasifikáciu, predpoveď a zhlukovanie.

Klasifikacia

Pri tomto type neurónovej siete je veľmi dôležité mať kvalitný dataset na, ktorom sa naša sieť učí. Autor siete potrebuje najprv pretransformovať všetky informácie do tvorby datasetu, musí každý vstup popísať, pridať anotácie, takzvané "labels". Takže pre každý vstup musí mať 1 až n potrebných anotácií, ktoré hovoria danej sieti čo je na na tomto vstupe. Tento typ siete sa využíva ak dopredu vieme množinu všetkých výstupov, ktoré budeme hľadať a chceme len rozoznať, ktorý z týchto výstupov je na danom vstupe. Klasifikácia sa môže použiť napríklad pri:

- rozoznávania tvári na obrázku, ich emocií alebo pohlavia
- vyhľadávania objektov na obrázku, napríklad značky, osoby, autá.
- vyhľadávania gest vo videu
- Detekcia hlasu, rozoznávanie majiteľa hlasu, preklad hovoreného slova na písaný text, zistenie nálady v hlase.
- rozdeľovanie textu na kategórie, napríklad detekcia spamu v emailoch
- veľa ďalších

Regresia

Regresia slúži na predpovedanie hodnôt na základe toho čo sa sieť naučila. Rovnako ako pri klasifikácii sa sieť učí pomocou anotácií ale jej výstup nie je len z množiny anotácií, ktoré jej boli poskytnuté. Sieť sa naučí na základe tréningových dát aký výsledok je očakávaný a je schopná vygenerovať hodnotu, ktorú predtým nevidela len na základe poznatkov z dát na ktorých sa učila. Regresiu možno využiť pri:

- predpovede toho že niekde niečo zlyhá
- zdravotné problémy na základne výsledkov vyšetrenia
- analýza webov, ich návštevnosti a tým prispôsobiť ponuku
- využitie na burze
- Odhad ceny objektu na základe základných informácií
- a ďalších

Zhlukovanie

Tiež nazývané aj clustering je prípad neurónovej siete, ktorá nepotrebuje anotácie na to aby sa učila. Triedi vstupy na základe podobností do zhlukov(clusterov). Väčšina dát, ktoré sú k dispozícií nemajú anotáciu a tak je zhlukovanie veľmi dobré pre ich spracovanie. V strojovom učení často platí pravidlo že čím viac má algoritmus dát na učenie tým sa stáva presnejším. Tieto algoritmy používajú takzvané učenie bez učiteľa. Zhlukovanie možno využiť pri:

- porovnávaní, hľadaní podobností v dokumentoch, obrázkoch, zvukových stopách
- hľadanie anomálií a iného neštandardného správania ako vedľajší efekt pri hľadaní podobností.

2.3.4 Učenie neurónovej siete

Jedna z hlavých vlastností neurónovej siete je vlastnosť sa učiť a tak sa má vlastnosť stále sa zlepšovať a hľadať lepšie a lepšie riešenia pre daný problém na daných tréningových dátach. Sieť sa pri učení v každom kroku mení hodnoty váh v neurónoch a tým sa snaží získať lepší a lepší výsledok. Učenie neurónových sieti môžeme rozdeliť do dvoch hlavných kategórií a to učenia s učiteľom a učenia bez učiteľa.

Učenie s učiteľom

Známe aj ako supervised learning[6] sa snaží znížiť rozdiel medzi výstupom zo siete, ktorý bol dosiahnutý a očakávaným výstupom pre daný vstup. K tomu sa používajú vyššie spomínané anotácie. Sieti je predaný očakávaný vstup a očakávaný výstup a ona na základe toho upravuje hodnoty váh jednotlivých neurónov každým krokom iterácie a snaží sa prísť k najlepšiemu výsledku. Po každej iterácií sa sieť testuje na takzvaných testovacích dátach, ktoré obsahujú rovnaké anotácie ako trénovacie dáta ale nie je na nich prevádzané samostatné trénovanie. Zistené výsledky sa predávajú ďalšiemu kroku učenia na základe ktorých sa sieť prispôsobí. Testovacie dáta sú často z rovnakej množiny ako trénovacie dáta len sú

pred tréningom oddelené a použité pre validáciu danej siete. Obvykle to býva 80% pre tréning a 20% pre validáciu. V niektorých prípadoch sa na validáciu môže použiť úplne iný, nezávislý set.

K zisťovaniu odchýlky od očakávaného výstupu sa využívajú takzvané chybové funkcie, ktoré počítajú odchýlku na a tú predávajú sieti aby vedela či sa zlepšila alebo daná iterácia nebola vhodná. Najznámejším algoritmom pre učenie s učiteľom je spätné šírenie chyby, ktorý využíva informácie z chybovej funkcie tak aby sa dala na každý neurón aplikovať optimalizácia.

V ideálnom stave sa sieť naučí generalizovať a je schopná vyvodiť správny výsledok aj pre vstup, ktorý jej nikdy nebol poskytnutý. Takáto sieť je vo väčšine prípadov cieľom učenia.

V praxi ale často tento prípad nenastane, aspoň nie hneď pri prvých pokusoch alebo v prípadoch keď sú trénovacie dáta/sieť zle postavená a dochádza k problému kde sa sieť naučí perfektne fungovať na naučených dátach ale nedokáže fungovať na neznámych vstupoch a tým príde o vlastnosť generalizácie, tento problém sa nazýva overfitting.

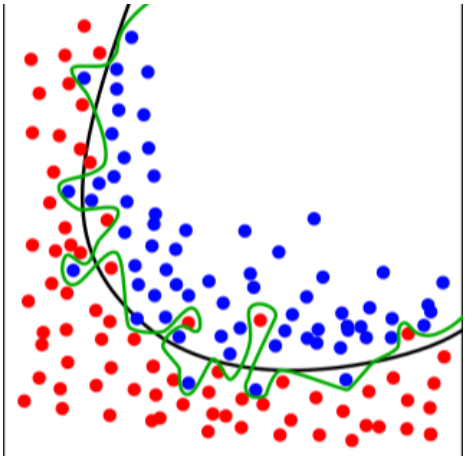
V opačnom prípade, ak máme príliš rozdielne dáta pre tréning tak sa sieť nedokáže učiť a nie je schopná sa naučiť vzťahy medzi vstupom a očakávaným výstupom. Tento problém sa nazýva under-fitting.

Tieto problémy sa dajú často zistiť z grafov učenia a ich riešenie je často v zmene trenovacích dát.

Známe aj ako Unsupervised learning narozdiel od učenia s učiteľom nepotrebuje hodnoty, ktoré sú očakávané na výstupe. Toto učenie pomocou svojich metód spracuje vstupný set dát a snaží sa ich na základe vzorov, štruktúr a podobností rozdeliť na skupiny(zhluky) na základe čoho upravuje váhy v jednotlivých neurónoch. Tento štýl učenia môže byť veľmi nepresný keďže sa nepoužíva žiadna chybová funkcia a ani sieť sama nevie aký výsledok je očakávaný. Ale na rozdiel od učenia s učiteľom má toto učenie menšie nároky na vstupné dáta a tým pádom môže využívať väčšie spektrum vstupných dát, ktoré nepotrebujú tak náročnú prípravu.

Overfitting

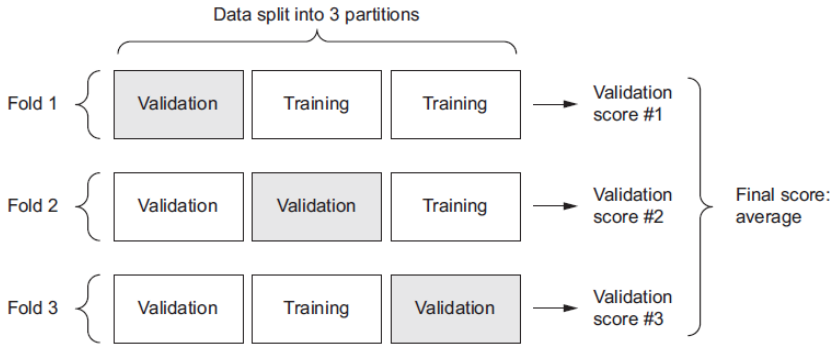
Overfitting[10] je veľmi častý a nežiadaný stav pri trénovaní neurónovej siete. Nastáva v prípade že sa sieť takzvane "preučí" a nedokáže sa generalizovať na vstupy na ktorých nebola učená. V tomto prípade sieť dosahuje dobré výsledky len na tréningových dátach. Overfitting sa často odhalí po testoch naučenej neurónovej siete na dátach, ktoré jej predtým neboli poskytnuté pri učení kde dosahuje omnoho horšie výsledky ako na dátach, ktoré jej boli poskytnuté pri učení.



Obr. 2.10: Čierna(konzistentná) čiara ukazuje ideálny priebeh učenia. Zelená(skákajúca) znázorňuje overfitting. [4]

Riešením problému overfitting je niekoľko. Po detekcií tohto problému sa môžu využiť niektoré z nasledujúcich možností.

Cross-validation je veľmi efektívna metrika použiteľná na prevenciu proti overfittingu. Pri tejto metóde sa rozdelia trénovacie dáta do viacerých skupín a podľa vybraného modelu cross-validation sa použijú. Samotná cross-validation má množstvo modelov a ich úprav, ktoré sa dajú použiť. Medzi veľmi populárne patrí kFold model kde sa rozdelia trénovacie dáta do k skupín a vždy sa použije k-1 skupín pre tréning a 1 skupina na validáciu daného setu. Postupne sa iteruje až pokiaľ sa ako validačná skupina nepoužijú všetky skupiny.



Obr. 2.11: Diagram znázorňujúci kFold cross-validation. Prebrané z [2].

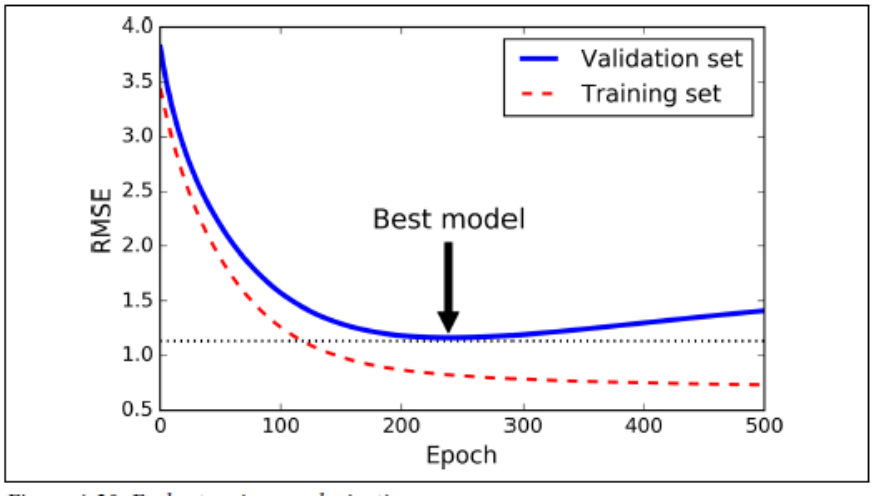
Trénovanie na viacerých dátach je tiež možnosť ako predísť overfitting problémom. Toto riešenie nemusí vždy pomôcť, záleží od konkrétneho problému, ktorý daná neurónová sieť rieši a taktiež nové dáta prinesú väčšiu hardvérovú náročnosť na naučenie siete.

Zjednodušenie môže viesť tiež k prevencií pred overfitting. Napríklad úprava algoritmu alebo spôsob spracovania vstupných dát môže znížiť overfitting problém pri učení neurónovej siete.

Early stopping je možné využiť ak sa model učí iteratívne. Vtedy sa dá porovnávať výsledky jednotlyvých iterácií a vo chvíli keď sa začnú zhoršovať výsledky na validačných

[4] https://en.wikipedia.org/wiki/Overfitting

dátach aj napriek tomu že sa zlepšujú na trénovaných dátach je sieť v bode kedy by sa mala zastaviť aby nezačala generalizovať a nenastal overfitting.



Obr. 2.12: Graf znázorňujúci early stopping riešenie. Prebrané z [4].

Under-fitting

Problém Under-fitting nastáva často vtedy keď je model neurónovej siete príliš jednoduchý alebo dataset z ktorého si vyberá sieť dáta na trénovanie je veľmi rôznorodý a sieť nedokáže alebo nemá kapacity naučiť sa dostatočne rozpoznať vstupné dáta. Problém Under-fitting sa dá riešiť tým že sa pri učení použije viac vstupných dát, dáta sa rozdelia do menej skupín, ktoré má daná sieť hľadať, pridá sa na komplexnosti danej siete alebo sa pridá počet iterácií.

Pridanie šumu

Šum možno pridať k vstupným dátam, váham alebo aj výstupom zo siete. Šum je ďalšou alternatívou, ktorá prispieva k stabilite siete.

Bagging

Vstupné dáta spracujú nezávisle naučené siete a výstup zo siete je vyberaný pomocou hlasovania týchto modelov. Zakladá si na tom že viacero nezávisle učených modelov sa nedopustí rovnakej chyby. Siete bývajú trénované na rozdielnych tréningových vstupoch a tým pádom by vďaka nedeterminizmu a náhodnosti mali podávať pri rovnakých testovacích dátach iný výstup.

Chybové funkcie

Pri trénovaní s učiteľom sa pri učení využívajú chybové funkcie, ktorých úlohou je merať odchýlku výstupu a očakávaného výstupu zo siete na základe ktorých sa určuje ako sa učenie daného modelu zlepšuje na dátach použitých na trénovanie a validáciu. Je primárnym ukazovateľom učenia sa siete.

Každá funkcia má svoje výhody aj nevýhody. Niektoré môžu byť použité len pre normalizované hodnoty, niektoré sú vhodné pre riešenie regresívnych, iné pri klasifikačných problémoch. Zmena chybovej funkcie môže úplne zmeniť správanie sa neurónovej siete ak je zvolená nesprávne.

Pri problémoch regresie sa najčastejšie používa chybová funkcia Mean Square Error(MSE) alebo Mean Absolute Error(MAE)

Mean Square Error(MSE) je chybová funkcia, ktorá meria ako priemer štvorcového rozdielu medzi vypočítanou hodnotou a očakávanou. Nerieši smer ale len veľkosť odchýlky. MSE má vlastnosť že čím ďalej je výstupná hodnota od očakávanej tým viac bude penalizovaná a vďaka matematickým vlastnostiam sa pri tejto chybovej funkcií počítajú gradieny ľahšie. MSE je počítané vzťahom

$$MSE(n) = \frac{1}{n}\sum_{i=1}^{n}(y_i - y'i)^2, \tag{2.9}$$

kde n predstavuje počet prvkov, z ktorého je chybová funkcia počítaná, y je predpoveď v čase i a y' je očakávaný výstup v čase i.

Mean Absolute Error(MAE) sa oproti MSE počíta ako priemernú absolútnu odchýlku. Rovnako ako MSE neuvažuje smer ale len veľkosť ale narozdiel od MSE má komplikovanejšie počítanie gradienov ale zase má výhodu že je odolnejšie proti veľkým odchýlkam keďže nepoužíva štvorcové výpočty a nepenalizuje teda tak prísne veľké odchýlky. MAE sa vypočíta vzťahom

$$MAE(n) = \frac{1}{n}\sum_{i=1}^{n}|y_i - y'i|, \tag{2.10}$$

kde n predstavuje počet prvkov, z ktorého je chybová funkcia počítaná, y je predpoveď v čase i a y' je očakávaný výstup v čase i.

Pri problémoch Klasifikácie sa používa najčastejšie Cross-Entropy(CE) funkcia a jej variácie.

$$H(p, q) = -\sum_{x} p(x) \log q(x), \tag{2.11}$$

kde $p(x)$ je chcená pravdepodobnosť a $q(x)$ je vypočítaná pravdepodobnosť a x označuje triedu pre ktorú výpočet prebieha.

Optimalizácia siete

Keďže hlboké neurónové siete sú stále z veľkej časti nepochopené tak nie je isté ako sa bude správať sieť ktorá je učená. Preto existujú len doporučenia pre trénovanie siete ale nie priamo pravidlá pre to aby sieť dosahovala lepších výsledkov. Veľmi dobré je vstupné dáta sieti normalizovať a tým predísť nejakému neočakávanému správaniu siete. Rovnako sa odporúča pre dva a viac neurónov ktoré prijímajú rovnaké vstupné dáta a funkciu určiť iné hodnoty váh inak by vznikala symetria.

Kapitola 3

Konvolučné neuronové siete

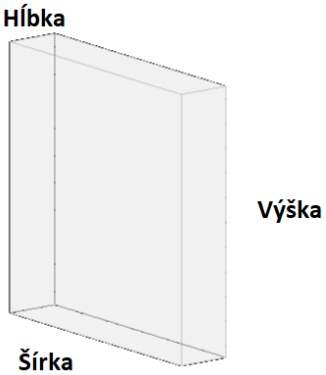
Keďže hlavnou témou tejto práce je práca s konvolučnými neurónovými sieťami, anglicky Convolutional Neural Network[3] skrátene CNN tak si ich rozoberieme trochu podrobnejšie.

Konvolučné neurónové siete sú jeden typ veľmi často používaných neurónových sietí. Používa niekoľko druhov vrstiev ktoré sa kombinujú a tým vzniká model konvolučnej neurónovej siete. Pomocou takzvaných konvolúcií spracováva vstupný vektor, ktorý pomocou jednoduchých funkcií spracuje a upraví váhy a prahy v neurónoch na základe spätnej väzby.

Konvolučná vrstva očakáva na vstupe obrázok, najčastejšie sa jedná o 4D vektor(tensor) ktorý je predaný sieti. Tým že sieť počíta so vstupným obrázkom dokáže efektívne a jednoducho odhadnúť jeho vlastnosti a tým aj zlepšiť jeho spracovanie.

Hlavnou výhodou je že CNN si dokáže veľmi efektívne v obraze identifikovať objekty, ich tvary, natočenie pomocou jednotlivých vrstiev neurónov. Čím viacej je sieť komplexnejšia tým detailnejšie sa vie naučiť rozoznávať informácie o objektoch v obrázku. CNN sa skladá zo základných vrstiev ktoré môžeme použiť a kombinovať medzi sebou.

Vstupný obrázok je prezentovaný ako pole ktoré má rozmery $[X], [Y], [Z]$ kde X je šírka obrázku v pixeloch, Y je výška v pixeloch a Z je hĺbka ktorá reprezentuje koľkým hodnotami je reprezentovaný každý pixel v obrázku. Pri RGB obrázku má hodnotu 3, pri čiernobielom obrázku stačí 1 hodnota.



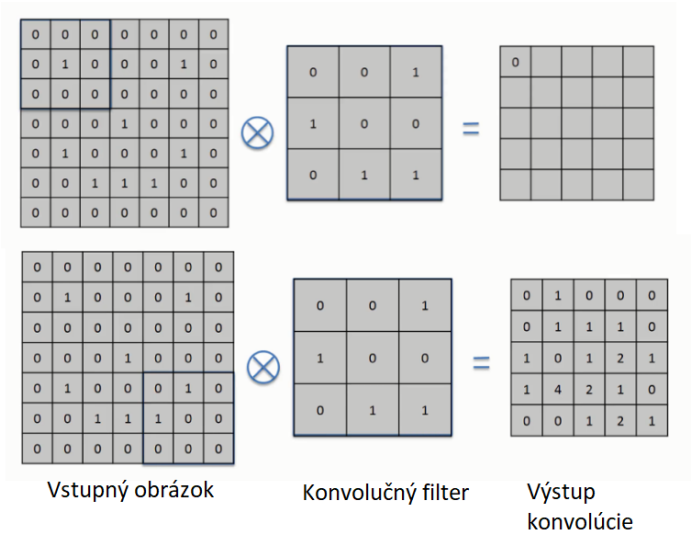
Obr. 3.1: Príklad toho ako je prezentovaný vstupný obrázok.

3.1 Konvolučná vrstva

Konvolučná vrstva je hlavnou vrstvou tvoriacou konvolučnú neurónovú sieť. Typicky ňou sieť začína, inak je počet použitých konvolučných vrstiev ľubovolný. Táto vrstva aplikuje na vstup matematickú operáciu dvojrozmernej diskrétnej konvolúcie.

Na obrázok je aplikovaný filter s takzvaným konvolučným jadrom(konvolučnou maskou). Tento filter je zložený z neurónov, ktoré majú rovnakú zdielanú váhu čo uľahčuje učenie siete. Filter je priložený na obrázok, tam sa vynásobí hodnota pixelu s hodnotou vo filtre a výsledky sa sčítajú. Výsledná hodnota je zapísaná do pixelu, ktorý leží pod stredom konvolučného filtru na výstup tejto vrstvy.

Úlohou tejto vrstvy je v obrázku detegovať rôznymi filtrami hrany,uhly a rôzne poznávacie značky podľa ktorých by bol obsah obrázku lepšie identifikovateľný. Výstup konvolúcie sa nazýva príznaková mapa a pre každú aplikáciu filtru dostaneme jednu príznakovú mapu, rovnako ako aj pre každú farebnú zložku. Takže pri aplikácií na jedného filtru na RGB obrázok dostaneme hneď 3 príznakové mapy. Zvyčajne je jeden filter nedostatočný a preto používame niekoľko filtrov. Výstupom je teda súbor niekoľkých príznakových máp. Počet filtrov je voliteľný, vyberá sa podľa komplexnosti zadania a nárokov na výpočtovú náročnosť.



Obr. 3.2: Príklad počítania konvolúcie nad obrázkom.

Veľkosť samotného filtru je voliteľná, v praxi sa typicky používajú nepárne hodnoty ako 3x3,5x5,7x7.Okrem rozmeru filtru môžme konvolucií zadať aj takzvaný padding a stride. Padding sa používa v prípade že chceme rozmer obrázku po konvolúcii zachovať výstupu po konvolúcií, prípadne ho zvýšiť. Ide vlastne o pridanie 0 na strany obrázku aby sa predišlo strate počas konvolúcie. Veľkosť padding aby ostali rozmery obrázku rovnaké možno vypočítať vzorcom

$$P = \frac{(K-1)}{2}, \tag{3.1}$$

kde P je hľadaný padding a K je veľkosť konvolučného filtra.

Stride je hodnota o ktorú sa konvolučný filter pri počítaní posúva. V základe sa používa stride = 1, v prípade že je nastavený na vyššiu hodnotu tak sa filter posúva o viac pixelov

naraz a tým redukuje rozmery výstupu konvolučnej vrstvy.

$$O = \frac{(W - K + 2P)}{S} + 1, \tag{3.2}$$

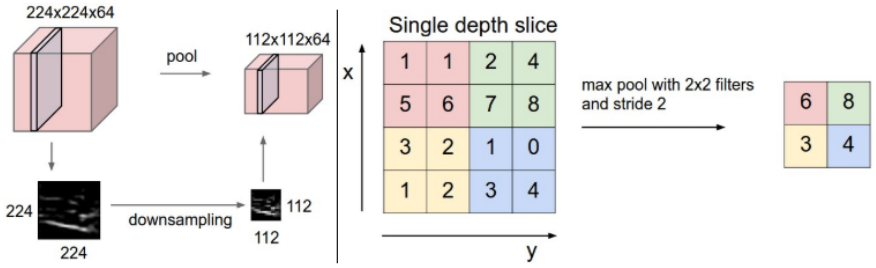
kde O označuje výstupnú šírku/výšku, K je vstupná výška/šírka, P je padding a S je stride.

3.2 ReLU (Rectified Linear Units) vrstva

ReLu vrstva[1] reprezentuje aktivačnú vrstvu, ktorá môže byť použitá kdekoľvek v sieti ale je zaužívaná konvencia použiť nelineárnu(aktivačnú) vrstvu po konvolúcií a tak sa spravidla používa po aplikovaní konvolučnej vrstvy, ale môže sa použiť aj za plne prepojenou vrstvou. Význam tejto vrstvy spočíva v tom že sa pomocou ReLu algoritmu sa neurónová sieť učí omnoho rýchlejšie ako v minulosti používané sigmoid alebo tanh vrstvy, ktoré dokázali sieť spomaliť. Rovnako sa veľmi dobre vysporiadava s miznúcim gradienom s čím mali aktivačné vrstvy v minulosti problém a to bez veľkého dopadu na presnosť siete. Vzorec pre výpočet ReLu je 2.7 je vlastne veľmi jednoduchý. Ak je číslo menšie ako 0 nahradí sa 0 inak ostane dané číslo nezmenené. Inými slovami zbavuje nás záporných hodnôt. Samozrejme záleží od typu neurónovej siete a je vhodné si pri trénovaní siete skúsiť viacero aktivačných funkcií, prípadne ich kombinácie.

3.3 Pooling vrstva

Pooling vrstva[1] je veľmi dôležitá, tiež býva označovaná aj ako zmenšovacia vrstva. Jej najčastejšie využitie je po aplikácií ReLu vrstvy. Jej účelom je znížiť rozlíšenie obrázku. Táto vrstva ma niekoľko možností, rovnako ako u konvolučnej vrstvy si môžeme zvoliť rozmery filtra, ktorý budeme aplikovať(najčastejšie 2x2 alebo 3x3) a aj stride, teda hodnotu o ktorú sa má filter posúvať, z pravidla to býva veľkosť samotného filtru. Najčastejšie používanou metódou pri pooling vrstve je takzvaná MaxPooling metóda ktorá zoberie zo všetkých hodnôt nachádzajúcich sa pod filtrom len tu najvyššiu a tú zapíše do výstupu vrstvy. Ďalšou je AvgPooling, ktorá tieto hodnoty priemeruje. Táto vrstva sama o sebe neukladá žiadne váhy podobne ako aktivačná vrstva ale len tranformuje vstup. Keďže hĺbka siete z pravidla rastie tak pooling vrstva aspoň uberá jej šírke a výške a tým dokáže veľmi znížiť potrebnú výpočtovú náročnosť na trénovanie siete. Len jedným použitím pooling vrstvy s filtrom 2x2 a stride 2 dokážeme znížiť výšku a šírku na polovicu a celkovú veľkosť o 75%. Ďalej pomáha predísť tomu aby sa sieť prestala generalizovať a nastal overfitting.



Obr. 3.3: Príklad aplikácie MaxPooling s filtrom 2x2 a stride 2.[1]

[1] http://cs231n.github.io/convolutional-networks

3.4 Plne prepojená vrstva

Označovaná aj ako FC(Fully-Connected) vrstva.Táto vrstva sa používa z pravidla na konci siete. Jej úloha je zobrať výstup z po predchádzajúcej vrstva, z pravidla ReLu alebo pooling vrstvy a ten transformovať na vektor o dĺžke N, kde N predstavuje počet tried, ktoré má sieť hľadať a ich hodnota predstavuje pravdepodobnosť výskytu danej triedy. Táto vrstva si namapuje výstupy z predchádzajúcej vrstvy tak aby čo najviac zodpovedali očakávaným výstupom a tým rozdelí neuróny pre detekciu jednotlivých tried.

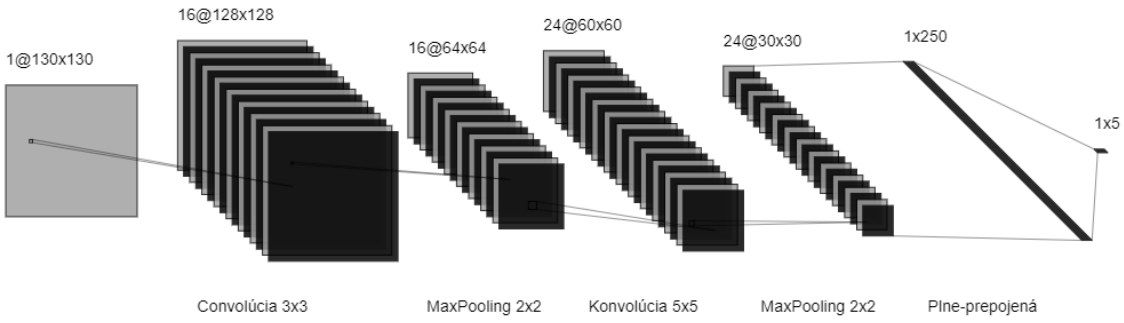
3.5 Chybová(Loss) vrstva

Poslednou vrstvou siete je vždy takzvaná chybová vrstva[7]. Ide o vrstvu, ktorá obsahuje aktivačnú funkciu, ktorá nám charakterizuje výstup zo siete a na jej základe sa počíta chybová funkcia. Ak je zvolená zle, môže skresliť alebo úplne znehodnotiť trénovanie siete.

3.6 Dropout vrstva

Ide o špeciálnu vrstvu, ktorá sa môže použiť, spravidla pred poslednou vrstvou siete na zahodenie určitého počtu neurónov. Toto sa často využíva v prípadoch kedy sieť trpí overfittingom a chceme ju žjednodušiťalebo v prípade že sieť je príliš výpočtovo náročná tak sa jej dá týmto spôsobom uľaviť.

A je použitá aspoň jedna konvolučná vrstva môžeme o danej sieti hovoriť ako o konvolučnej neurónovej sieti. Z pravidla nasleduje po každej konvolučnej vrstve ReLu vrstva nasledovaná pooling vrstvou a na konci aspoň jedna plne prepojená vrstva s loss vrstvou, ktorá bude reprezentovať výstup zo siete. Príklad jednoduchej konvolučnej sieti je na obrázku 3.4



Obr. 3.4: Príklad konvolučnej neurónovej siete, ktorá obsahuje 2 konvolučné vrstvy a hľadá 5 tried.

Kapitola 4

Praktická časť

Po zoznámení sa s neurónovými sieťami môžme začať aplikovať tieto znalosti v praxi a pokúsiť sa vyriešiť náš cieľ pomocou konvolučnej neurónovej siete. V prvej časti sa oboznámime s prístupom, ktorý bol zvolený, výbere jednotlivých knižníc, programovacích jazykov, zvolenie typov setov dát a cieľa, ktorý sa pokúsime dosiahnuť. V druhej časti sa budeme venovať vylepšeniu našej siete rôznymi zmenami aplikovanými na sieť a jej trénovanie, ktoré by nás mali priviesť ku žiadanému výsledku. V poslednej časti sa sústredíme na testovanie finálnej verzie/verzií siete na vzorové sety dát a zhodnotíme dosiahnuté výsledky. V tejto časti sa budem snažiť využiť nadobudnuté informácie k čo najlepšiemu riešeniu daného problému. Informácie k riešeniu tohto problému som čerpal prevažne z kníh *Deep learning with Python*[2], *Deep Learning*[5] a *Hands-On Machine Learning with Scikit-Learn and TensorFlow*[4]. Zároveň som čerpal aj z internetových zdrojov kde je veľmi veľa zaujímavých článkov o hlbokých neurónových sietiach, ktoré vysvetľujú mnohé otázky ohľadne neurónových sietí.

4.1 Analýza problému

Téma tejto práce je určenie natočenia hlavy v obrázku pomocou neurónovej siete, konkrétne špecifickým druhom neurónovej siete, konvolučnou neurónovou sieťou, ktorá je veľmi vhodná na spracovanie obrázkov. V prípade natočenia hlavy budeme brať do úvahy jej 2 možné natočenia a to sklonu hlavy, čo by sa dalo reprezentovať ako pohyb hlavy hore dolu a natočenie hlavy, čo by sa dalo reprezentovať ako pohyb hlavy doľava doprava. Jedna z možností by bolo zisťovať toto natočenie osobitne pomocou 2 sietí. Problém tohto riešenia je že jeden uhol je závislí na druhom a tak bude lepšie keď budeme obe natočenie odhadovať naraz.

Keďže naša sieť nebude klasifikovať ale bude sa snažiť odhadnúť uhol pod ktorým je hlava natočená musíme použiť regresiu ktorá bude odhadovať 2 hodnoty.

Pre to aby sme boli schopný tieto dve hodnoty schopný odhadovať musíme si pripraviť potrebné sady dát, v našom prípade obrázkov ktoré použijeme pre trénovanie a následne testovanie siete. Pre učenie a tvorbu dát budeme potrebovať niekoľko nástrojov a aplikácií ktoré si rozpíšeme v nasledujúcej časti.

Obr. 4.1: Ukážka hlavy s popisom vlastností hlavy ktoré sa sieť bude pokúšať určiť.

4.2 Zvolené nástroje

4.2.1 Python

Tento vyšší interpretovaný programovací jazyk je v súčasnej dobe veľmi populárny. Jeho výhody sú v jeho dostupnosti, ide o open-source softvére, v jeho rýchlosti v porovnaní s inými interpretovanými jazykmi a veľkej úrovne abstrakcie ktorú ponúka. Samotný jazyk ma zabudovaný garbage-collector a veľké množstvo užitočných knižníc, ktoré sú dostupné priamo po inštalácií alebo dodatočne dostupné. Pre naše účeli budeme používať verziu Python(3.6.7) .

TensorFlow je jedna z najznámejších knižníc pre tvorbu neurónových sietí v súčasnej dobe. je dostupná pre nami vybraný jazyk python. My budeme používať framework, ktorý je súčasťou TensorFlow a to **Keras**. Keras bola knižnica ,ktorá bežala nad Tensorflow a ďalšími knižnicami a poskytovala príjemné užívateľsky prívetivé užívateľské prostredie pre prácu s neurónovými sieťami. Neskôr sa stala súčasťou samotného TensorFlow a na naše účeli bude dostačujúca. Samotný tensorflow obsahuje 2 verzie, jednu pre výpočty vykonávané na procesory(CPU) a druhú pre výpočty na grafickej karte(GPU) pomocou CUDA jadier. Keďže je trénovanie na grafickej karte mnohonásobne rýchlejšie a mám prístup k výkonnej karte tak som zvolil práve túto možnosť ale vo výsledku by mali pracovať obe rovnako. Pre účel práce som zvolil verziu 1.13.1 .

OpenCV je veľmi populárna knižnica pre načítanie a spracovanie obrazových vstupov.

Numpy je knižnica ktorá sa používa prevažne na vedecké účeli. V našom prípade využijeme možnosť tvorby n-dimenzionálnych polí ktoré budú reprezentovať naše obrázky a budú slúžiť ako vstup prácu s neurónovou sieťou.

Matplotlib budeme používať na zobrazenie grafov pre lepšie znázornenie niektorých výsledkov získaných pri trénovaní siete.

Scikit-learn využívam pri Kfold algoritme, ktorý je v tejto knižnici implementovaný.

Mimo tieto knižnice bude využitých niekoľko knižníc dostupných v nainštalovanom balíku samotného jazyka.

4.2.2 SYDAGenerator

SYDAGenerátor[1] je program vyvíjaný skupinou STRaDE z Ústavu inteligentních systémů FIT VUT v Brně.Tento generátor nám poslúži pre generovanie testovacích dát ktorými budeme simulovať reálne obrázky. Pomocou tohto a ďalších aplikácií ktorými pred spracujeme obrázky osôb získame 3D model, ktorý môžme natočiť nad ľubovolným pozadím a získame tým obrázok, ktorý nám poslúži pre simuláciu. Pre vytvorenie 3D modelu pre využijeme program zephyr [2] a následne využijeme fbx-conv [3], ktorý poslúži konvertovanie .obj súboru z programu Zephyr na súbor, ktorý je vhodný pre použitie v aplikácií SYDAgenerator.

4.3 Dátové sady

Keďže potrebujeme našu sieť trénovať s čo najkvalitnejšou dátovou sadou, tak som rozhodoval medzi niekoľkými variantami. Pri dátovej sade je veľmi dôležité mať nielen dostatočnú variáciu obrázkov, v našom prípade s čo najviac kombináciami uhlov a hláv. Pri malom počte hláv môže začať nastupovať problém s overfittingom, ktorý môže viesť k neúspechu siete na obrázkoch, ktoré nikdy nevidela pretože nemala pri učení dostatočnú variáciu. Rovnako je veľmi dôležité aby boli dáta kvalitne a presne anotované aby sa s nimi dalo pracovať. Z tohoto dôvodu som sa rozhodol pre 2 zdroje dát s ktorými budem pracovať.

Prvý plán bol použiť len mnou vygenerované sady dát v programe SYDAgenerátor ale zistil som že tieto dáta nebudú najvhodnejšie pre učenie našej siete. Na generáciu obrázku s hlavou potrebujeme obrázok pozadia a 3D model hlavy. K tomuto účelu som mal prístup k 6 modelom hláv pomocou ktorých je možné vygenerovať prakticky ľubovolný počet obrázkov. Problém je že pri 6 hlavách je veľmi ťažké rozdeliť dáta tak aby nedochádzalo k overfittingu. Zároveň sa neutrálna poloha hlavy, to je poloha kedy sa sklon aj natočenie rovná 0, určuje manuálne a tým pádom môžu vznikať nepresnosti. Ďalej pri generovaní model neotáča hlavou ale otáča sa celý model čo môže niekedy vytvoriť nechcené efekty. Preto som sa rozhodol že tento spôsob budem používať až ako druhý, prípadne na samotné testovanie vytvorenej siete a budem nimi simulovať reálne obrázky keďže sú anotované budú na tento účel vhodné. Tvorbu tohto setu je popísaný v samostatnej časti 4.9. Túto sadu budeme ďalej označovať ako **SYDA** sada.

Druhý zdroj dát je Head Pose Image Database[9], ktorý je voľne dostupný a veľmi kvalitne spracovaný. Táto sada obsahuje fotky 15 osôb z ktorej každá osoba má 2 série fotiek všetkých kombinácií uhlov natočenia a sklonu hlavy na intervale <-90,90> s určitým krokom. Ďalej je táto sada veľmi kvalitne anotovaná a okrem údajov o uhloch obsahuje aj údaje o presnej pozícií hlavy na obrázku, ktoré sa dajú využiť pri ich spracovaní alebo pre siete, ktoré majú iné určenie ako naša. Popis sady a toho ako bol vytvorený je dostupný na webovej stránke[4]. kde je ho možné stiahnuť. Počet dostupných obrázkov je 2745, 186 pre každú osobu. Túto sadu budeme ďalej nazývať **HPID** sada.

4.3.1 Tvorba datasetu

Samotnú dátovú sadu musíme rozdeliť na 3 časti: tréningovú časť, validačnú časť a testovaciu časť. Ak je sieť pomocou sady len trénovaná je dobrým pomerom rozdeliť sadu 80 ku 20, kde 80 percent dát slúži na trénovanie a 20 na validáciu. Pri rozdeľovaní kde sa počíta

[1] https://www.fit.vutbr.cz/units/UITS/prod/index.php?id=564
[2] https://www.3dflow.net/3df-zephyr-free/
[3] https://github.com/libgdx/fbx-conv
[4] http://www-prima.inrialpes.fr/perso/Gourier/Faces/HPDatabase.html

Obr. 4.2: Príklad obrázku z Head Pose Image Database.

Obr. 4.3: Príklad obrázku vygenerovaného pomocou SYDAgenerator.

s testovacou časťou sa tieto dáta doporučuje rozdeliť v pomere 70 ku 15 ku 15. Samozrejme toto sú len doporučené hodnoty a sú môžu byť zvolené ľubovolne. V mojom prípade som sa rozhodol si pre začiatok vytvoriť dve dátové sady z HPID. Jednu ktorá bude pozostávať z náhodne premiešaných obrázkov, ktoré rozdelím v pomere 70:15:15(1954:418:418 obrázkov) a druhú ktorú rozdelím na základe osôb. Tady bude pomer 11 osôb pre trénovanie, 2 osoby na validáciu a a 2 osoby na testovanie výsledkov, to je 2046 obrázkov pre tréning, 371 pre validáciu a 371 pre testovanie. Počet obrázkov v oboch tréningových sadách je veľmi podobný ale ich rozloženie je o dosť iné. Zatiaľ čo prvý set predstavuje úplne náhodne rozpoloženie, kde sa budú vyskytovať všetky tváre v každej časti tak druhý neobsahuje všetky a bude viac simulovať reálny problém pri testovaní kde bude sieti predstavená neznáma tvár. Prvý dátový set som pomenoval **DATASET_1**, druhý s oddelenými hlavami **DATASET_2**

4.4 Tvorba siete

Keďže u neurónových sietí sa často nevie aké budú výsledky alebo aké sú najlepšie konfiguračné nastavenia musí sa začať s nejakými základnými hodnotami a tie na základe výsledkov experimentov so sieťou upravovať a pokúšať sa dosiahnuť lepší a lepší výsledok. V samotná sieť má obrovský počet parametrov, ktoré sa môžu pred jej tréningom nastaviť a tak ovplyvniť výsledok. Postupne som sa zameral na tie ktoré som považoval za najdôležitejšie a tie som upravoval.

Model siete

Na začiatok som si zvolil model z ktorého budem vychádzať a pomocou ktorého budem zisťovať iné parametre nastavenia siete. Na hľadanie samotného modelu som sa zameral až neskôr.

Ako základný model siete som zvolil sieť, ktorá obsahuje dve konvolučné vrstvy s filtrom 3x3 kde je každá nasledovaná pooling vrstvou s filtrom 2x2. Keďže naše vstupné dáta sú obrázky tak hodnoty s ktorými bude sieť na vstupe pracovať budú v obore hodnôt <0;255>, tak mi prišla aktivačná funkcia ReLu 2.7 ako veľmi vhodná. Čo sa týka normalizácie dát, tak som sa rozhodol normalizovať dáta na vstupe do rozsahu 0 až 1. Pri testoch na prvý pohľad nebol medzi správaním sa siete rozdiel ale je dobrou praxou dáta normalizovať aby sa lepšie zobrazovali, upravovali a tiež sa tak vyhli potenciálnym výkyvom počas trénovania siete.

Pre vstup som zvolil delenie číslom 255, jedná sa o veľmi jednoduchú normalizáciu ktorá upraví hodnoty do intervalu <0;1>. Anotácie o uhle som sa rozhodol tiež normalizovať na interval 0 až 1 aj keď to nie je tak potrebné ako normalizácia vstupných dát.

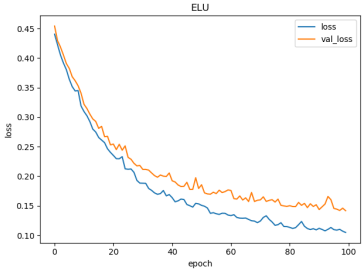
Samotné obrázky som sa rozhodol načítať namiesto formátu RGB, bezfarebne len v odtieňoch šedej a v základom rozlíšení(384x288) aj keď je časté ich spracúvať v štvorcovom formáte. Keďže samotné obrázky obsahujú len hlavu človeka bez nejakého väčšieho rušenia a hlavy zaberajú väčšinu plochy tak mi prišlo toto riešenie ako vhodné. Takže jeden náš obrázok bude reprezentovať vektor 384x288x1.

```
def create_model_0():
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), input_shape=input_shape))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, kernel_size=(3, 3)))
    model.add(Activation("relu"))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(2))
    model.summary()
    return model
```

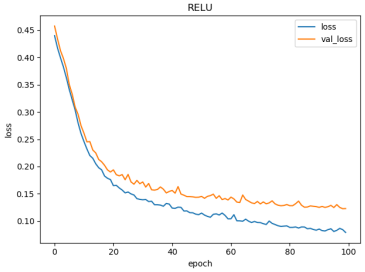
Aktivačná funkcia

Čo sa týka aktivačnej funkcie je v súčasnej dobe pri konvolučných neurónových sietiach veľmi populárna ReLu funkcia ale nemusí byť vždy ta najlepšia čo sa týka podávaných výsledkov. Pre vstupnú vrstvu som zvolil ReLu lebo máme na vstupe dáta, ktoré nebudú nikdy záporné a tým pádom je určite táto funkcia vhodná pre toto použitie a nezhodnotí nám vstup, čo by mohla ak by boli na vstupe záporné hodnoty. Čo sa týka vnútorných vrstiev tam to tak jednoznačné nie je a zvoliť aktivačnú funkciu pre tieto vrstvy je komplikovanejšie. Môžu sa používať ich kombinácie alebo ich používať podla vrstvy, ktorá bola použitá pred nimi. Preto som sa rozhodol urobiť niekoľko experimentov a na základe ich výsledkov vybrať aktivačnú funkciu, ktorá bude používaná v modely.

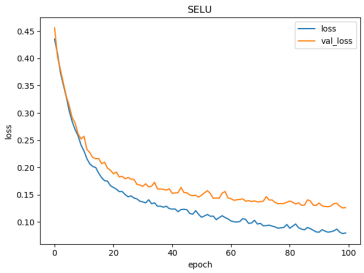
Samotný experiment pozostával z už spomenutého modelu siete v ktorom bola vždy vnútorná funkcia obmenená za testovaciu. Ostatné nastavenia siete, vrátane všetkých vstupných dát a ich premiešania, boli rovnaké pre každú jednu funkciu. Experiment bol prevedený viacero-krát na oboch datasetoch.



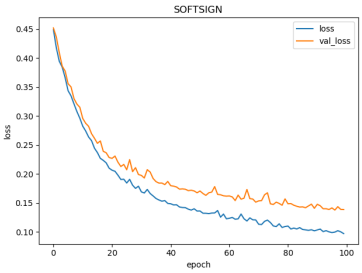
(a) ELU funkcia, chyba = 0,1048.



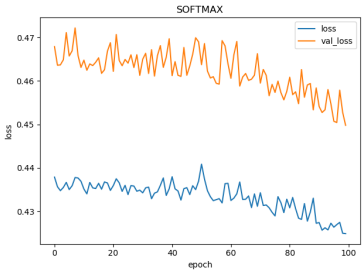
(b) ReLU funkcia, chyba = 0,0791.



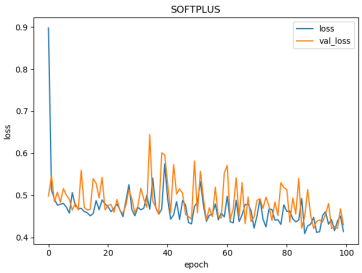
(c) SELU funkcia, chyba = 0,0795.



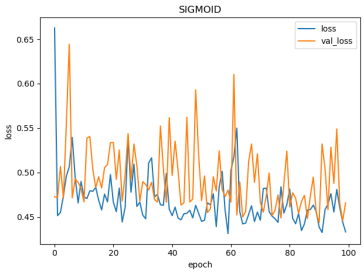
(d) SoftSigm funkcia, chyba = 0,0970.



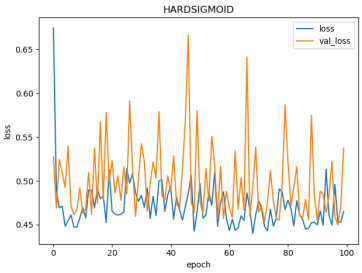
(e) SoftMax funkcia, chyba = 0,4248.



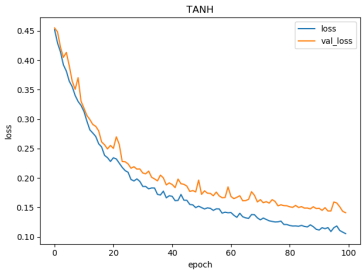
(f) SoftPlus funkcia, chyba = 0,4138.



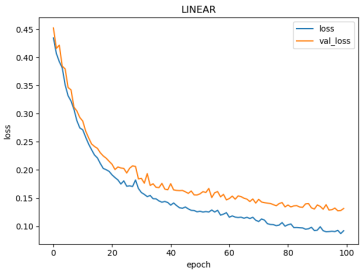
(g) Sigmoid funkcia, chyba = 0,4330.



(h) HardSigmoid funkcia, chyba = 0,4649.



(i) Tanh funkcia, chyba = 0,1054.



(j) Linear funkcia, chyba = 0,0920.

Obr. 4.4: Grafy aktivačných funkcií.

Ako je vidieť na jednotlivých grafoch, tak niektoré funkcie mali veľmi podobný priebeh zatiaľ čo iné zaznamenali veľké kmitanie a nestabilitu počas trénovania. Ak zanedbáme hodnotu validačnej chyby a budeme brať v ohľad len hodnoty chyby na konci trénovania tak najlepšia je práve funkcia ReLU, preto som sa rozhodol pokračovať s ňou.

Chybová funkcia

Tiež známa aj ako loss funkcia nám určuje ako dobre tréning napreduje na tréningovej sade, rovnako sa využíva pri validačnej sade ako ukazovateľ smeru či sa naša sieť zlepšuje aj na dátach z ktorých sa priamo neučí. Podľa správania sa chybovej funkcie na validačných dátach vieme detegovať problémy ako sú napríklad overfitting, ktorý nám spôsobí horšie chovania sa siete na testovacích dátach. Keďže objekt tejto práce spadá do kategórie regresívnych problémov musel som vybrať spomedzi chybových funkcií vhodných k riešeniu tohto problému. Samotná chybová funkcia je veľmi dôležitá keďže meria chybu predikcia a jej postupné znižovanie je hlavným ukazovateľom vývoja neurónovej siete. Rozhodoval som sa medzi funkciami **MSE**(2.9) a **MAE**(2.10), keďže MAE je *miernejšia* pri penalizovaní rozhodol som sa preň stým že neskôr pri finalizovaní siete porovnám rozdiel medzi nimi a vyberiem tú, ktorá bude podávať lepšie výsledky aj keď ich správanie by malo byť veľmi podobné.

Learning rate

Learning rate je hyperparameter, ktorý nám určuje do akej miery sa získané informácie prenesú do váh našej siete. Na začiatku sú z pravidla nastavené hodnoty váh modelu náhodne a postupne sa počas trénovania upravujú a snažíme sa nájsť ich ideálne hodnoty. V prípade že sa váhy prestanú meniť náš model uviazol a jeho výsledky sa ďalej už nebudú zlepšovať.

Počas každej epochy trénovania sa počíta hodnota chybovej funkcie a na základe jej hodnoty je vypočítaný gradient ku každej váhe. Tieto gradienty sú následne násobené hodnotou learning rate. Získaná hodnota sa použije pri tvorbe novej váhy tým že sa od jej súčasnej hodnoty odpočíta. Týmto spôsobom sa neurónová sieť snaží získať najlepšie hodnoty váh, ktoré nám dajú čo najnižší výsledok chybovej funkcie. Správnu hodnotu learning rate sa nedá určite presne, je potrebné si nejakú hodnotu zvoliť a experimentami nad dátovou vybrať hodnotu, ktorá sa nám bude zdať optimálna. Hodnota learning rate musí byť vždy menšia ako 1, zväčša sa hodnota learning rate pohybuje od 0,0001 do 0,4 ale je to veľmi závislé na danej sieti a jej architektúre. Ak zvolíme hodnotu learning rate príliš veľkú, tak sa môžu hodnoty chybovej funkcie meniť prirýchlo a po veľkých skokoch, čo môže spôsobiť nežiadúci výsledok vo forme veľkej výslednej chybovej funkcie. Naopak aj je hodnota veľmi nízka sieť sa zlepšuje príliš pomaly a predlžuje zbytočne proces trénovania.

Pri hľadaní optimálnej hodnoty som použil už predtým vytvorenú sieť stým že som menil hodnoty learning rate a počet epoch(čím menšia hodnota learning rate, tým viac epochách potrebujeme na to aby sme získali výsledok, zatiaľ čo pri vyšších stačí menší počet pretože sieť sa trénuje rýchlejšie). Výsledky som zhrnul do tabuľky 4.1.

Learning rate	Počet epoch	Výsledná hodnota chybovej funkcie
0,1	10	0,435
0,01	20	0,433
0,001	40	0,064
0,0001	80	0,030
0,00001	160	0,069
0,000001	320	0,123
0,0000001	720	0,322

Tabuľka 4.1: Tabuľka zobrazujúca hľadanie hodnoty learning rate.

Ako je vidieť v tabuľke tak najlepšie výsledky dosiahla learning rate s hodnotou 0,0001, pričom hodnoty menšie väčšie ako 0,001 postupovali príliš rýchlo a mali veľkú chybu. Pri nižších hodnotách bol zase nedostatočný počet epoch pre to aby dosiahli lepších výsledkov. Takže v ideálnom prípade by sme zvolili túto ale podľa grafu chybových funkcií pri týchto hodnotách nastala príliš rýchlo konvengercia a tým pádom overfitting. Pri hodnote 0,0001 sa zlepšovanie chyby na validačných dátach zastavilo už pri 10 epoche pričom sa chyba na tréningových dátach stále zlepšovala. Z tohto dôvodu som zvolil radšej hodnotu 0,00001 s tým že bude trénovanie pomalšie a náročnejšie ale bude sa lepšie optimalizovať stým že hodnotu 0,0001 môžem využiť neskôr.

Epochy

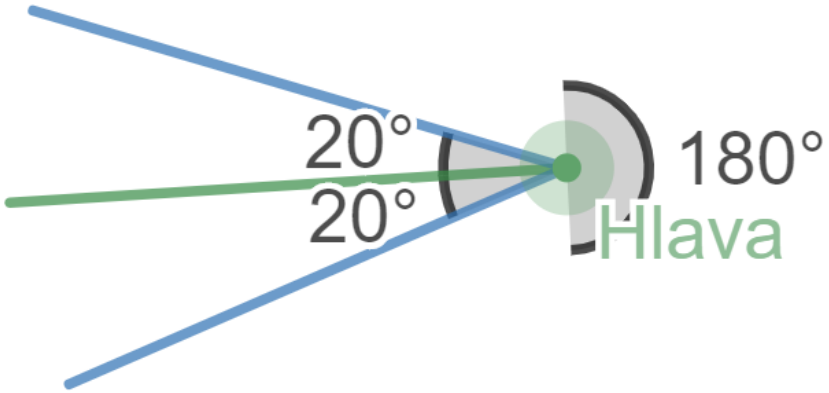
Počet epoch ako takých nie je nejako určený a môže sa nastaviť na ľubovolnú hodnotu väčšiu ako 1. Počet epoch je priamo úmerný dĺžke trénovania siete preto je vhodné toto číslo nedávať veľmi vysoké aby trénovanie netrvalo príliš dlho. Zároveň čím dlhšie sa sieť trénuje tak tým sa zvyšuje pravdepodobnosť toho že nastane overfitting. Urobil som pár experimentov s tým že som porovnával výsledné hodnoty chybovej funkcie pre trénovacie a validačné dáta. Následne som sa rozhodol že budem začínať s hodnotou 150.

Tréningová dávka

Tréningová dávka, tiež označovaná aj ako batch, je vlastne počet vektorov, ktoré sieť spracuje predtým ako na základe chybovej funkcie upraví váhy. Každá tréningová sada je teda rozdelená na n dávok. Jeden cyklus vyhodnotenia všetkých dávok nazývame **epochou**. Takže pre nás určuje veľkosť tréningovej dávky to koľko obrázkov sa naraz spracuje. Týmto sa zrýchľuje trénovanie siete keďže ide robiť viac vecí paralelne. Pri veľmi malých dávkach môže byť učenie siete veľmi pomalé a zároveň sa môže správať chaoticky keďže nemusí mať dostatočnú variáciu a jej hodnoty váh sa preto môžu veľmi meniť medzi jednotlivými dávkami, preto je aj veľmi dôležité mať nejakým spôsobom tréningovú sadu premiešanú aby nenastávali prípady že v jednej dávke budú až príliš podobné vektory. Naopak pri veľkej dávke sa môžu váhy meniť málo a zároveň je tento proces náročný na hardvér. Pri hľadaní optimálnej hodnoty som použil rovnaký model ako predtým stým že som menil len veľkosti tréningovej dávky. Podľa výsledkov dosahovala najnižšie hodnoty chybovej funkcie hodnota 20, čo naznačovalo prítomnosť overfittng ale aj napriek tomu som sa rozhodol túto hodnotu použiť stým že sa budem snažiť dávky dobre premiešavať.

Testovanie siete

Keďže základný model bol nakonfigurovaný bolo treba overiť jeho kvalitu, či dokáže detegovať to čo chceme a celkovo jeho správanie na obrázkoch, ktoré nikdy nevidel. Keďže naučiť sieť detegovať presný uhol 2 dvoch rôznych osiach je pri tejto vzorke dát, ktoré som sa rozhodol použiť veľmi nepravdepodobné navrhol som jednoduché riešenie, ktoré porovná výsledné hodnoty sklonu a natočenia hlavy predikované sieťou s reálnymi hodnotami ale stým, že bude nastaviteľná odchýlka o ktorú sa môže odhadovaný uhol líšiť od tej reálnej hodnoty. Ako základnú hodnotu, ktorú som sa rozhodol dosiahnuť bolo ±20° v oboch určovaných vlastnostiach, ktoré sa sieť snaží predikovať.



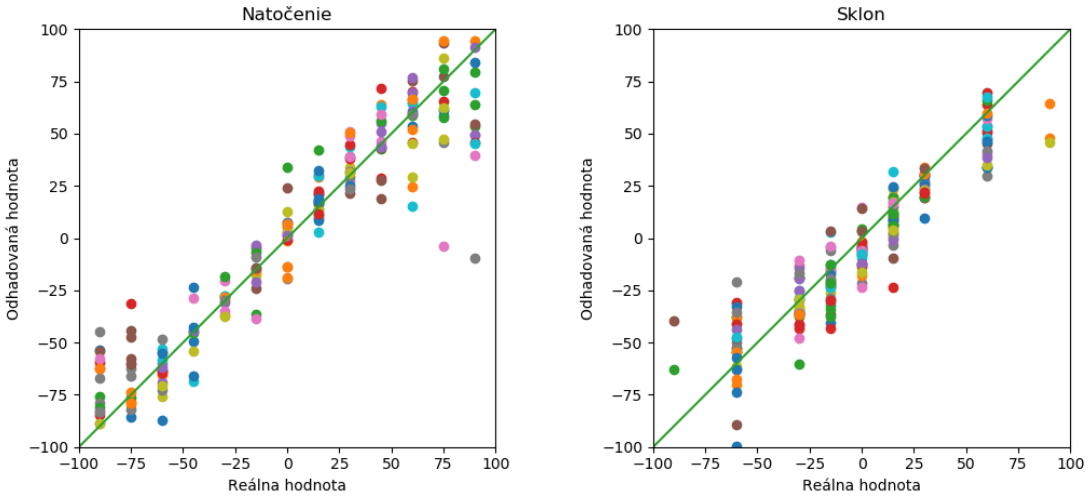
Obr. 4.5: Ukážka hlavy a odchýlky, ktorú sa budeme snažiť získať.

A ako nejaký ukazovateľ chovania sa siete pri ďalších úpravách som sa rozhodol sieť overovať na tréningovej, validačnej aj testovacej sade. Sieť bola trénovaná na DATASET_1 aj na DATASET_2 aby boli vidieť rozdieli pri použití dvoch sád, ktoré využívajú rovnaké zdrojové dáta ale sú inak rozdelené. Samozrejme kvôli uviaznutiu v lokálnych minimách bolo prevedených niekoľko behov a na základe výsledkov bol jeden vybraný ako vzorový.

Sada	Trénovacie dáta	Validačné dáta dáta	Testovacie dáta
DATASET_1	87,71 %	77,99 %	73,68 %
DATASET_2	90,09 %	55,37 %	15,05 %

Tabuľka 4.2: Tabuľka zobrazujúca prvé výsledky siete.

Ako je možné vidieť v tabuľke 4.2 tak výsledky potvrdili očakávanie a DATASET_1 si viedol o niečo lepšie a to hlavne na obrázkoch, ktoré nevidel. Aj napriek tomu že sieť nedosiahla vysokých čísiel bolo už z chovania siete zrejmé že dochádza k overfitting a je treba upraviť tréningoví proces. Preto som sa rozhodol zamerať v prvom rade na vylepšenie modelu siete, zistení aký počet vrstiev bude najvhodnejší a následne použiť niektoré metódy využívané na zmiernenie problému overfitting. Na grafoch 4.6a a 4.6b je možné vidieť hodnoty, ktoré boli očakávané a hodnoty, ktoré boli predikované sieťou.



(a) Graf zobrazujúci očakávané verzus odhadované hodnoty pre natočenie.

(b) Graf zobrazujúci očakávané verzus odhadované hodnoty pre sklon.

Obr. 4.6: Grafické znázornenie výsledkov pre testovacie dáta.

4.5 Model siete

Keďže nie je spôsob ako zistiť aký model je pre daný problém najlepší, preto je treba nutné experimentovať. Zvolil som niekoľko modelov, tie som podrobil niekoľkým behom a ten ktorý sa osvedčil najviac som zobral a ďalej upravoval a porovnával s modelmi ktoré som si myslel že by mohli fungovať lepšie. Po dlhšom trénované som usúdil že najvhodnejší bude model ktorý nebude obsahovať priveľa skrytých vrstiev pretože sa tieto modely osvedčili na tréningových dátach ale na testovacích nepodávali najlepšie výsledky.

Po dlhšej dobe kedy sa mi nedarilo získať lepšie výsledky alebo sa získaný výsledok líšil len o minimálne rozdiely ale bol omnoho náročnejší na tréningový proces. V praxi sa najviac zo začiatku osvedčili modely, ktoré mali 2 konvolučné vrstvy, jednu na vstupe a jednu skrytú, ktoré boli nasledované ReLU vrstvami a veľmi dôležitá bola plne prepojená vrstva pred výstupnou vrstvou, ktorá sa ukázala že má veľký dopad na učenie siete.

Experimentoval som aj s dropout vrstvami ale keďže tieto vrstvi zahadzujú náhodné neuróny bolo lepšie model zjednodušiť ako ho navrhovať komplexnejší a následne zahadzovať neuróny.

4.6 Kfold

Kfold je jeden z najpopulárnejších spôsobov ako riešiť oferfitting a preto som sa ho rozhodol aplikovať aj tu. Už s predtým vybraným modelom som urobil niekoľko experimentov s rôznymi hodnotami k, počtom epoch a hodnotou learning rate. Výsledky experimentov, ktoré som navrhol neboli veľmi úspešné a preto som sa rozhodol ďalej túto metódu nepoužívať ako nutnú súčasť siete.

4.7 Early stopping

Ako ďalšiu metódu proti problému overfitting som vybral Early stopping. Tento spôsob jednoducho zastaví učenie siete v prípade že sa prestane zlepšovať chovanie siete na validačných dátach a tak sieť ostane v stave kedy nemá váhy príliš prispôsobené tréningovým dátam. Navrhol som niekoľko experimentov, ktorými sa potvrdilo že early stopping zlepšuje chovanie dát nad validačnými a testovacími dátami.

4.8 Predspracovanie obrázkov

Veľmi dôležitou časťou pri trénovaní neurónových sietí je ich prespracovanie. Obrázky často obsahujú šum alebo iné nedokonalosti, ktoré nám môžu ovplyvňovať trénovanie siete a je dobré sa ich aspoň čiastočne zbaviť.

Ako prvé som sa rozhodol obrázky vždy používať vo odtieňoch šedej. Keďže naša sieť sieť nepotrebuje k určeniu uhlu informácie o farbe tak je lepšie využiť možnosti mať namiesto troch kanálov pre farbu len jeden kanál a tým pádom trikrát menší vstupný vektor čo značne urýchli a stabilizuje učenie siete.

Tohoto výsledku ide dosiahnuť niekoľkými spôsobmi. Najjednoduchší spôsob je takzvané priemerovanie. Ten ide popísať vzorcom ako

$$X = (R + G + B)/3, \tag{4.1}$$

kde X je a R,G,B sú jednotlivé hodnoty RGB. Keďže je ale podstata jednotlivých zložiek iná je vhodnejšie použiť vzorec

$$X = (Rr + Gg + Bb), \tag{4.2}$$

kde r,g,b sú koeficienty priradené jednotlivým farebným prvkom podľa ich významnosti. Tieto koeficienty sa môžu líšiť podľa použitého štandardu a žiadaného cieľa ale ich súčet je vždy 1 aby sme neprekročili maximálnu hodnotu pre jednotlivý pixel. Ja som sa rozhodol používať základnú konverziu implementovanú priamo v knižnici OpenCV2.

(a) Pôvodný obrázok.

(b) Obrázok v odtieňoch šedej.

Obr. 4.7: Ukážka rozdielu medzi originálnym obrázkom a obrázkom v odtieňoch šedej.

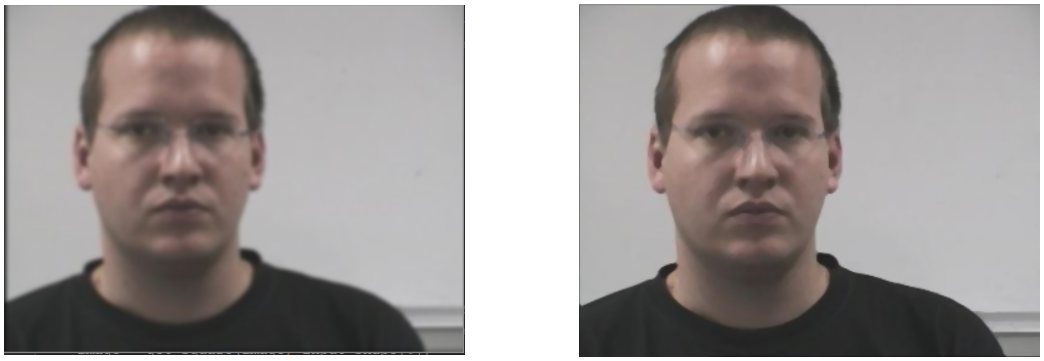
Ako ďalšie som sa rozhodol obrázky normalizovať na jednotné rozlíšenie, keďže ani obrázky nemusia mať hodnoty rozlíšenia rovnaké je dobré ich zmeniť na jednotnú hodnotu. Preto som sa rozhodol že obrázky pretvorím bez zmeny pomeru strán na rozlíšenie 250x250 s tým že doplním čiernu farbu aby nebol pomer narušený.

Na to aby som zredukoval šum som testoval 2 funkcie na rozmazanie obrazu. Prvé bolo Gaussovo filtrovanie, ktoré využívam rovnakomennú funkciu.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}, \tag{4.3}$$

kde x a y sú polomere rozmazania pre jednotlivé osy a σ je štandardná odchýlka Gaussového rozloženia.

Ďalšie bolo medián filtorvanie, ktoré za pomoci mediánu počíta jednoduchú hodnotu pre každý pixel. Zoberú sa všetky hodnoty okolitých pixelov a medián nahradí hodnotu pixelu. Ide o nelineárne filtrovanie, ktoré má oproti Gaussovému filtrovaniu výhodu že čiastočne uschováva hrany.

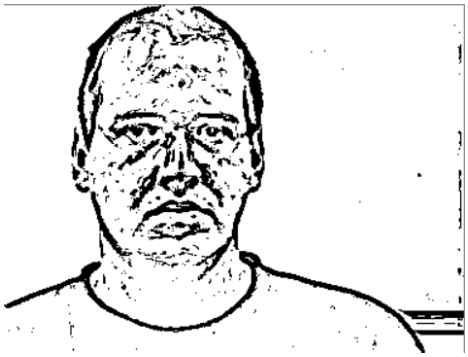


(a) Aplikácia Gaussovho filtru s veľkosťou 9. (b) Aplikácia medián filtru o velkosti 5.

Obr. 4.8: Ukážka rozdielu medzi rozmazaním obrazu.

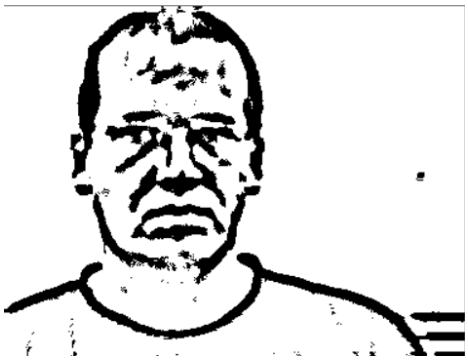
Z šedého obrázku som sa následne rozhodol pomocou prahovania vytvoriť obrázok, ktorý bude len v bielej a čiernej a zvýrazní tak plochy tváre. Tu som sa rozhodoval či použijem adaptívne prahovanie s priemerom alebo Gausovým rozložením.

Priemerné rozloženie sa vypočíta ako priemer okolia pixelu o veľkosti filtra a odpočíta sa konštanta C. Zatiaľ čo pri Gaussovom rozložení sa jedná o vážený súčet okolitých bodov o veľkosti filtru na ktoré je aplikovaná Gaussova matica.

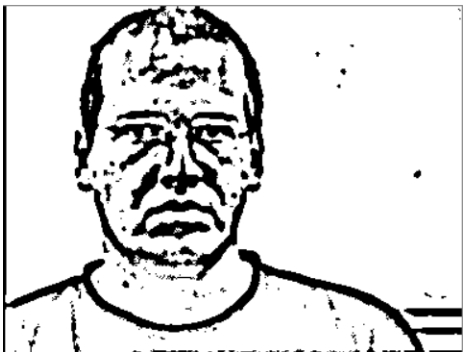


(a) Kombinácia medián rozmazania a Gaussového adaptívneho prahovania.

(b) Kombinácia medián rozmazania a priemerného adaptívneho prahovania.



(c) Kombinácia Gaussového rozmazania a Gaussového adaptívneho prahovania.



(d) Kombinácia Gaussového rozmazania a priemerného adaptívneho prahovania.

Obr. 4.9: Ukážka kombinácií rozmazania a prahovania.

4.9 Tvorba SYDAGenerator sady

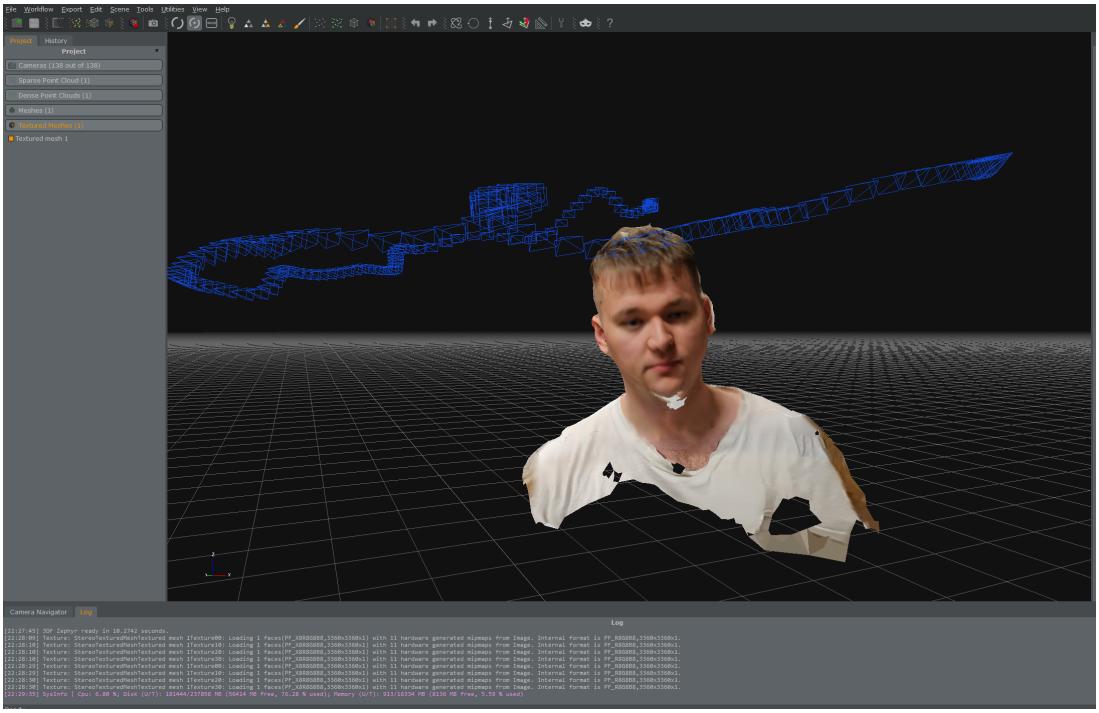
Môj pôvodný plán bolo si vytvoriť celú testovaciu sadu pomocou aplikácie SYDAGenerator a na tejto sade učiť sieť. Ale po niekoľkých problémoch, ktoré nastali som sa rozhodol túto sadu nepoužívať ako primárnu ale nechať si ju na záverečné experimenty a na to aby som overil funkčnosť siete na úplne neznámych tvárach v neznámom prostredí a celkovo nie ideálnou pozíciou tváre na obrázku. Týmto budem simulovať reálnu fotku.

Tvorba 3D modelu

Na to aby som mohol vytvoriť obrázky, ktoré budú obsahovať tváre v rôznych pozíciach je potrebný 3Dmodel tváre. Ten som sa snažil vytvoriť v aplikácií 3DF Zephyr, ktorá je na tento účel vhodná.

Aplikácia berie na vstupe snímky alebo video z ktorého extrahuje snímky automaticky. Tieto snímky sa pokúsi spárovať a v prípade úspechu je možné niekoľkými skoro plne automatickými krokmi vytvoriť 3D model hlavy. Vzniknutý model je vrátane okolia a nežiadúcich efektov, ktoré je treba manuálne odstrániť. Toto je možné priamo v programe alebo v inom externom programe, ktorý pracuje s .obj súbormi. Po tom ako je model vyčistený je potreba ešte zredukovať počet polygónov aby nebol model príliš komplexný a mohli sme ho ďalej spracovať. Tohto sa dá dosiahnuť v samotnom Zephyr zadaním hodnoty a on počet zredukuje automaticky. Ideálna hodnota je niekde okolo 20 000.

Po tom ako je model hotový je treba ho exportovať ako .obj súbor spolu s textúrami vo formáte jpg. Následne už stačí len tento súbor vložiť do fbx-conv, ktorý nám skonvertuje tento model do formátu g3db, ktorý je ľahšie spracovateľný pre rôzne aplikácie ako je práve SYDAGenerator. Týmto je pripravený model, ktorý môže byť vložený do akéhokoľvek obrázku.



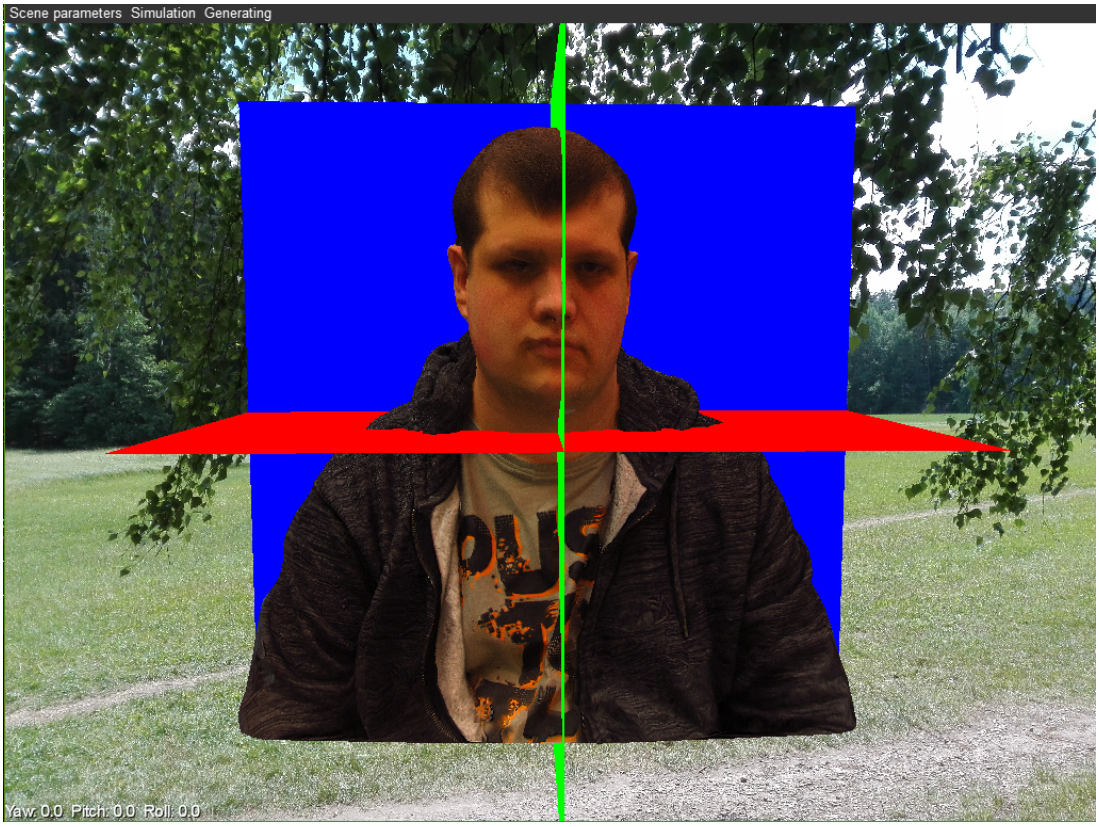
Obr. 4.10: Príklad toho ako vyzerá model vytvorený v programe Zephyr z videa.

Problém tohto riešenia je že Zephyr má problémy s tým že ak je video alebo snímky inak nasvietené tak nedokáže tieto fotky spárovať. Tento problém sa vyskytol pri mojich pokusoch veľmi často a nepodarilo sa mi získať kvôli tomuto problému dostatočný počet modelov. Ďalším problémom boli anomálie spôsobené pohybom pri natáčaní videa alebo fotografovania, ktoré spôsobili buď diery v modely alebo sa niektoré časti vygenerovali zle(najčastejšie uši alebo vlasy). Nakoniec sa podarilo v školskom prostredí získať niekoľko použiteľných modelov.

Tvorba snímkov

Pred samotným generovaním sa musí otvoriť každý model a pomocou zobrazených rovín sa model manuálne vycentruje aby bol s týmito rovinami zarovno a mali sme tak neutrálnu nulovú pozíciu okolo ktorej sa bude model otáčať. Po tom stačí model uložiť a aplikácia si uloží toto nastavenie pre konkrétny model. Následne stačí nastaviť v konfiguračnom súbore uhly od ktorého po ktorý chceme generovať s vybraným krokom.

Program po spustený generovania vygeneruje všetky obrázky a pomenuje ich tak aby názov obsahoval anotácie o uhloch pod ktorými sa model nachádza. Samotná aplikácia vie generovať 3 rôzne otáčania modelu a ich kombinácie. Podarilo sa mi generovať obrázky kde bolo len 1 otočenie alebo kombinácie všetkých troch. Keďže som sa rozhodol určovať len 2 pozície hlavy tak budem využívať len snímky kde sa mení natočenie hlavy a sklon ostáva rovnaký.



Obr. 4.11: Príklad SYDAgenerator s rovinami, ktoré slúžia na centrovanie modelu.

Keďže tento proces ide zautomatizovať tak sa dali generovať sady o veľkom počte snímkov. Toto riešenie je vhodné ak chcete generovať veľké sady, ktoré som aj plánoval ale môj problém bol že som mal k dispozícií len 6 modelov hláv a pri pokusoch o učenie sietí dochádzalo k veľkému overfitting.

4.10 Výsledky experimentov

Po ukončený experimentov kde som upravoval všetky parametre siete pomocou ktorých sa sieť učila a rôznym pomerom rozdeľoval dáta som dospel k finálnym verziám sietí, ktoré považujem za najúspešnejšie. Tieto siete som podrobil testom na sade vytvorenej pomocou SYDAGenerator, ktorá reprezentuje náhodné obrázky ľudí.

Na to aby bolo lepšie vidieť úspešnosť siete som sa rozhodol okrem merania, ktoré nám porovnáva výstupný uhol zo siete a očakávaný uhol s odchýlkou ešte ďalšie dve metriky.

Prvá metrika je počítanie priemernej odchýlky kde sa vyhodnotí odchýlka pre natočenie a sklon. Výhodou je že túto metriku ovplyvňujú úplne všetky výstupu zo siete a tak udáva lepšie celkové chovanie sa siete nad vstupnými dátami. Tento vzťah ide popísať aj matematicky ako:

$$f(real, pred) = \frac{\sum_{x=1}^{n} ||real_x| - |pred_x||}{n}, \tag{4.4}$$

kde n je počet vstupov, $real_x$ je očakávaná hodnota a $pred_x$ je hodnota, ktorú odhaduje sieť.

Ďalšia metrika čo som pridal je metrika ktorá nám zhrnie výsledky do kategórií podľa toho ako veľmi sa líšili od očakávaného výstupu. Pomocou tejto metriky ide jednoducho zobraziť do akej odchýlky je naša sieť nad danými dátami vhodnejšia a naopak kedy od akej odchýlky má už slabšie výsledky.

Keďže som vo výsledku použil niekoľko trénovaných sietí tak zobrazím získané výsledky a porovnám ich chovanie nad rôznymi dátami. Pre testovanie použijem 2 sady vygenerované pomocou SYDAGenerator. SYDASET_1 bude tvorený obrázkami kde bude na pozadí fotka zatiaľ čo SYDASET_2 bude mať jednofarebné pozadie. Týmto môžem porovnať aký veľký vplyv má pozadie na vytvorenú sieť. Nad každým z týchto dvoch setov sú prevedené 2 merania. Jedno je nad obrázkom ako celkom, v druhom je vystrihnutý stred obrázku, takže sa zredukuje obsah nepotrebnej plochy obrázku a maximalizuje pomer hlavy.

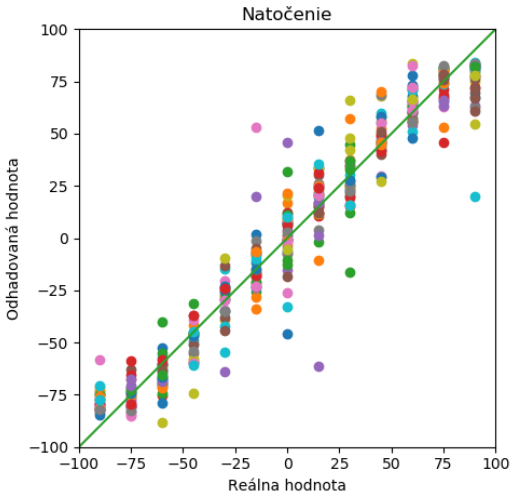
Na konci bolo na výber niekoľko podobných sietí, ktoré boli učené inými spôsobmi a prejavovali iné vlastností pri finálnom testovaní. Tie, ktoré boli podla mňa najúspešnejšie som vybral a zhodnotil ich výsledky.

Sieť č.1

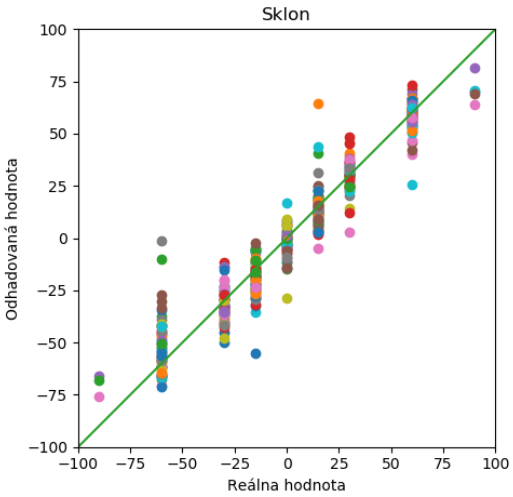
Sieť č.1 je sieť, ktorá bola trénovaná nad DATASET_1 mala najlepšie výsledky pri tréningu. Pri jej trénovaní boli obrázky načítané v odtieňoch šedej a upravené do štvorca. Ale keďže jej sada obsahovala známe tváre aj pri testovaní tak nedokázala dobre rozoznávať neznáme tváre a na obrázkoch zo SYDAGenerator sady. Výsledky sú zhrnuté v nasledujúcej tabuľke.

Sada	Presnosť	Priemerná odchýlka-natočenie	Priemerná odchýlka-sklon
DATASET_1 test	89,47 %	9,00	7,36
SYDASET_1 originál	14,19 %	20,69	26,86
SYDASET_1 orezané	17,09 %	17,56	24,43
SYDASET_2 originál	18,68 %	17,10	30,77
SYDASET_2 orezané	9,21 %	18,82	38,25

Tabuľka 4.3: Tabuľka zobrazujúca výsledky pre sieť č.1.



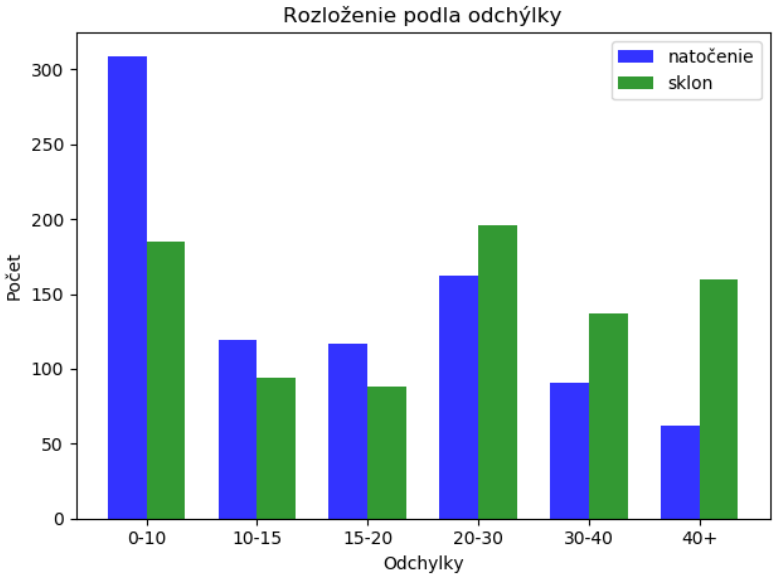
(a) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu natočenia.



(b) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu sklonu.

Obr. 4.12: grafické znázornenie výsledkov pre sieť č.1.

Ako je vidieť v tabuľke 4.3 tak na úplne neznámych dátach sa tejto sieti príliš nedarilo narozdiel od tréningovej sady. Ale ak sa pozrieme na graf rozdelenia podľa odchýlky tak vidíme že sieť bola pri určovaní sklonu hlavy omnoho horšia ako pri určovaní natočenia, pritom vo všetkých vstupoch bol sklon hlavy 0. Toto bolo pravdepodobne spôsobené uhlom pod ktorým bol model vytvorený a nedokonalosťou implementácie, ktorú som zvolil. Ale čo sa týka natočenia hlavy tak viac ako polovica prípadov mala odchýlku menšiu ako ±20°.



Obr. 4.13: Rozloženie odchýlky na SYDASET_1 s orezanými obrázkami pri použití siete č.1.

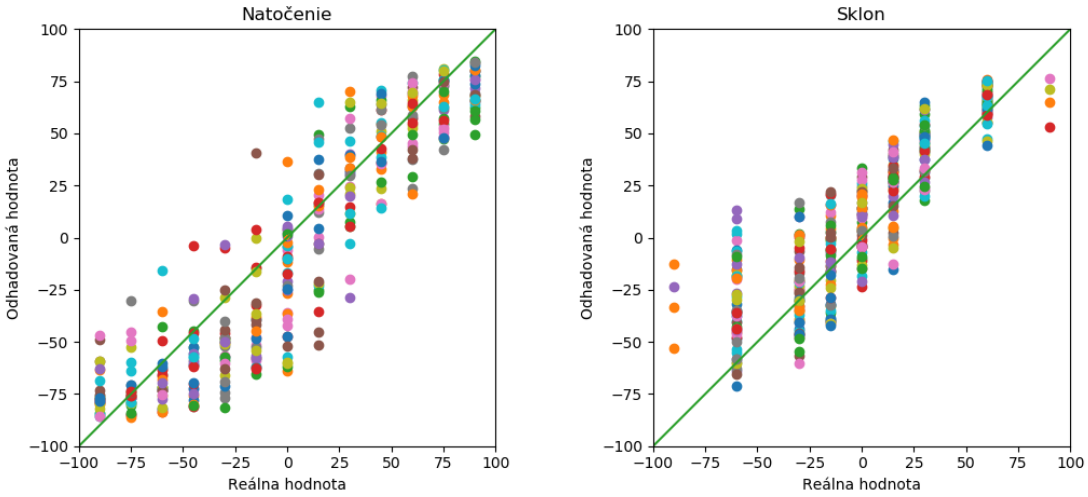
Zaujímavým výsledkom tejto siete je že pri teste nad SYDASET_2 kde je vyrezaný z obrázku len stred s hlavou dosahuje sieť horších výsledkov ako nad obrázkami kde tvorí hlava menšiu časť.

Sieť č.2

Sieť č.2 je sieť, ktorá bola trénovaná nad DATASET_2 s úpravou rozdelenia a pomeru tvárí. Rovnako bez úprav vstupov ako v prípade siete č.1. Mala síce horšie výsledky pri tréningu ako sieť č.1 ale používala na validáciu a testovanie neznáme tváre, preto sa očakávalo že bude na SYDAGenerator setoch podávať stabilnejšie výsledky.

Sada	Prenosť	Priemerná odchýlka-natočenie	Priemerná odchýlka-sklon
DATASET_2 test	47,04 %	17,53	15,58
SYDASET_1 originál	18,60 %	18,82	25,27
SYDASET_1 orezané	23,26 %	16,93	20,05
SYDASET_2 originál	20,26 %	20,08	25,75
SYDASET_2 orezané	13,95 %	18,71	24,63

Tabuľka 4.4: Tabuľka zobrazujúca výsledky pre sieť č.2.

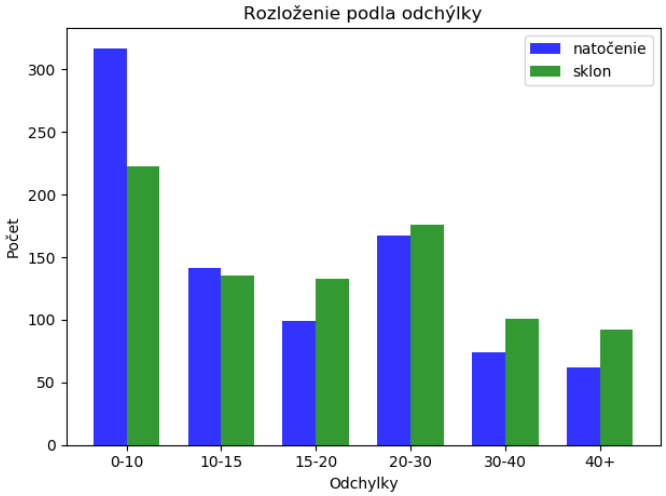


(a) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu natočenia.

(b) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu sklonu.

Obr. 4.14: Grafické znázornenie výsledkov pre sieť č.2.

.



Obr. 4.15: Rozloženie odchýlky na SYDASET_1 s orezanými obrázkami pri použití siete č.2.

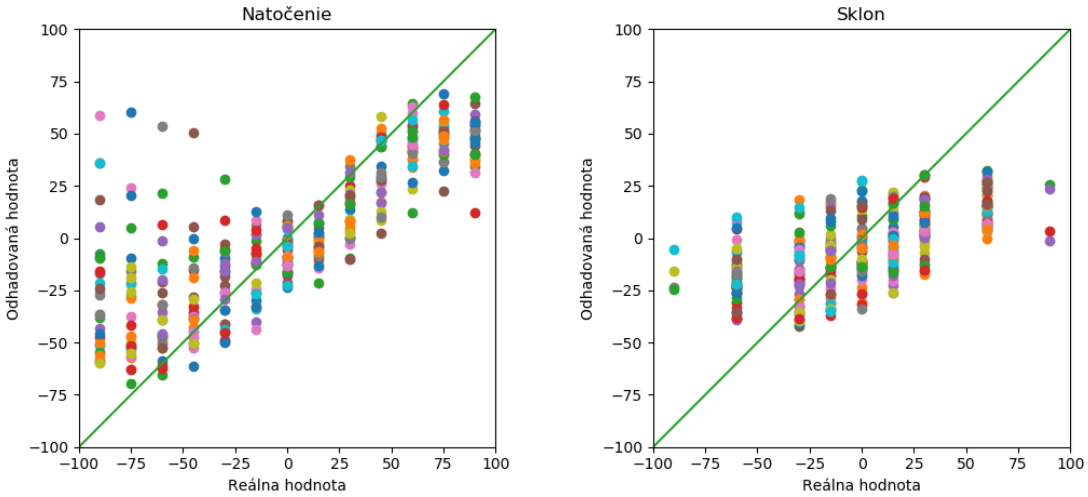
Ako je vidieť tak sieť č.2 bola naozaj úspešnejšia v záverečných testoch pretože dokázala lepšie generalizovať. Narozdiel od siete č.1 nie sú už priemerné odchýlky také veľké. Ako je vidieť na obrázku 4.15 tak viac ako dve tretiny odhadov mali odchýlku nižšiu ako ±30°.

Sieť č.3

Sieť č.3 je sieť, ktorá bola trénovaná nad DATASET_2, ktorý mal upravený pomer tréningových a validačných dát . Pri tvorbe tejto siete boli použité rôzne úpravy vstupov za cieľom čo najlepšie pred spracovať obrázky aby ich sieť ľahšie dokázala rozpoznávať a učiť sa z nich. Výsledky tejto siete sú zhrnuté v nasledovnej tabuľke.

Sada	Presnosť	Priemerná odchýlka-natočenie	Priemerná odchýlka-sklon
DATASET_2 test	23,12 %	21,71	22,22
SYDASET_1 originál	33,14 %	28,11	13,45
SYDASET_1 orezané	31,51 %	23,46	11,19
SYDASET_2 originál	31,58 %	25,14	12,25
SYDASET_2 orezané	32,37 %	22,86	11,84

Tabuľka 4.5: Tabuľka zobrazujúca výsledky pre sieť č.3.

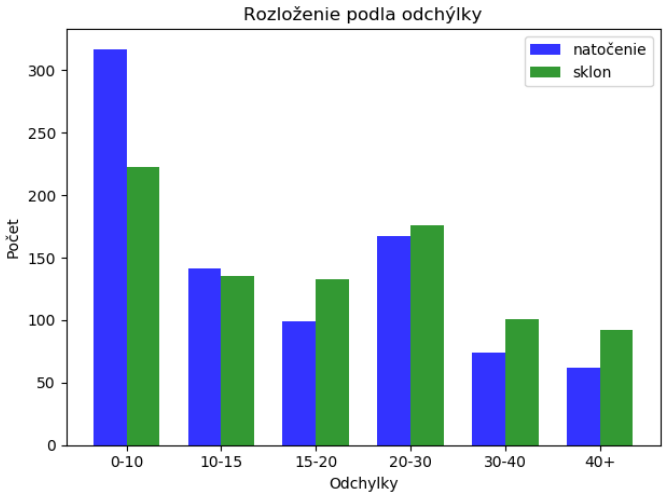


(a) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu natočenia.

(b) Graf zobrazujúci rozdiel odhadovaného a očakávaného uhlu sklonu.

Obr. 4.16: Grafické znázornenie výsledkov pre sieť č.3.

.



Obr. 4.17: Rozloženie odchýlky na SYDASET_1 s orezanými obrázkami pri použití siete č.3.

Ako je vidieť tak sieť č.3 bola omnoho úspešnejšia čo sa týka presnosti triafania sa do $\pm 30°$ od presnej hodnoty ale čo ta týka priemernej odchýlky tak dosahovala dobré výsledky iba pre sklon, pri natočení podávala horšie výsledky ako predchádzajúca sieť.

Zhrnutie výsledkov

Tak ako je vidieť v tabuľkách 4.3, 4.4 a 4.5 tak každá sieť bola úspešnejšia v niečom inom. Mimo týchto sietí som vytvoril veľa iných, ktoré podávali dobré výsledky na určitom type vstupov ale na inom boli zase omnoho horšie. Toto mohlo byť spôsobené mojím návrhom

riešenia, ktorý som pre tento problém zvolil. Správanie umelej neurónovej siete sa nedá predpovedať a jej chovanie sa overí až pri experimentoch. Preto je možné že moje riešenie nebolo vhodné pre riešenie tohto problému a keby som ho navrhol inak bolo by možné dosiahnuť lepších výsledkov.

Čo sa týka výsledkov tak neboli úplne najpresnejšie vzhľadom k vzorke dát s ktorou som pracoval. Prvý krokom pre získanie lepších výsledkov je určite rozšírenie dátových sád o veľký počet nových tvári, ideálne s rôznym pozadím. Tohto by šlo dosiahnuť pomocou SYDAgenerator ak by boli použité značky ako pri HeadPose Image Database[9] a stabilnej kamery v nie príliš osvetlenej miestnosti. Žiaľ toto riešenie je veľmi náročné na realizáciu. Ďalej by šlo so sieťou ďalej experimentovať pomocou iných nastavení hyperparametrov, iným spôsobom spracúvať vstup, či už vo forme RGB, normalizácie alebo iného navrhnutia spracovania celého problému. Samotný tréningový proces bol veľmi náročný ako časovo tak aj na výkon počítača potrebný pri tréningu a preto je pri jeho prípadnej znovu realizácií lepšie navrhnúť model pretože ja som dlho pracoval s príliš komplikovaným modelom, ktorý obsahoval príliš veľa naučiteľných parametrov a v konečnom výsledku sa lepšie overili siete s menším počtom.

Kapitola 5

Záver

Cieľom tejto práce bolo určiť uhol natočenie hlavy pomocou konvolučnej neurónovej siete. Na základe znalostí, ktoré som dosiahol, som sa pokúsil takúto neurónovú sieť skonštruovať. Postupným vykonávaním experimentov nad mnou navrhnutými dátovými sadami, som chovanie siete upravoval a snažil sa získať čo najlepší výsledok.

Na začiatok bolo potrebné si naštudovať teóriu neurónových sietí, následne zanalyzovať riešený problém, vytvoriť postup riešenia, získať a rozdeliť potrebné dáta, nad ktorými mohla byť následne sieť trénovaná, a nakoniec zhodnotiť výsledky, ktoré boli dosiahnuté.

Aj keď sieť neurčovala hodnoty natočenia a sklonu hlavy veľmi presne v rozmedzí $\pm 20^{\circ}$ nad dátami, ktoré som vytvoril pomocou modelov hláv vytvorených z fotiek, tak nad tréningovou sadou boli omnoho úspešnejšie. Z výsledkov je viditeľné, že naučené siete mali pri trénovaní porovnateľné výsledky pre natočenie aj sklon, zatiaľ čo pri finálnych testoch sa tieto hodnoty viacej líšili. Pri známych tvárach bola úspešnosť cez 80 %, zatiaľ čo pri nevidených tvárach v rovnakom prostredí bola nižšia, niekde okolo 50 % v najlepších prípadoch a približne 30 % v prípadoch kedy boli neznáme hlavy v neznámom prostredí.

Počas riešenia problému tejto práce som nadobudol vedomosti v obore strojového učenia, konkrétne v smere hlbokých neurónových sietí, ich tvorení a experimentovania. Dozvedel som sa o problémoch, ktoré pri práci s nimi môžu nastať, ako ich riešiť, že príprava dát je veľmi dôležitá, a že najlepšie výsledky dosiahnuté pri trénovaní nemusia byť aj najlepšie pri nasadení siete.

Pri trénovaní bol najväčší problém to, že sieť nedokázala veľmi dobre generalizovať vstupné dáta a dochádzalo často k overfitting problému. Z tohto dôvodu som sa rozhodol pre použitie viacerých zdrojov pre potreby tejto práce, keďže pôvodne zamýšľané riešenie neposkytovalo dostatočný počet a kvalitu dát. Ďalším problémom bola náročnosť na trénovanie a experimentovanie so sieťou, ktoré som si sám komplikoval príliš veľkými modelmi, ktoré sa učili pomaly.

Ako zlepšenie a rozšírenie tejto práce by som navrhoval učenie siete za pomoci sady, ktorá obsahuje veľkú škálu tvárí a kombinácií uhlov spolu z rôznymi druhmi pozadia, aby sa sieť naučila čo najlepšie generalizovať a tak mať lepšie správanie pri použití neznámych dát.

Literatúra

[1] Can, A.: Layers of a Convolutional Neural Network. [Online; navštívené 17.02.2019].
URL https://wiki.tum.de/display/lfdv/Layers+of+a+Convolutional+Neural+Network/#LayersofaConvolutionalNeuralNetwork-PoolingLayer

[2] Chollet, F.: *Deep learning with Python*. Manning Publications Company, 2017, ISBN 9781617294433.
URL https://books.google.cz/books?id=Yo3CAQAACAAJ

[3] CS231N: Convolutional Neural Networks. [Online; navštívené 15.02.2019].
URL http://cs231n.github.io/convolutional-networks/

[4] Géron, A.: *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2017, ISBN 9781491962244.
URL https://books.google.cz/books?id=bRpYDgAAQBAJ

[5] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. Adaptive Computation and Machine Learning series, MIT Press, 2016, ISBN 9780262035613.
URL https://books.google.cz/books?id=Np9SDQAAQBAJ

[6] Hagan, M.; Demuth, H.; Beale, M.: *Neural network design*. Martin Hagan, 2014, ISBN 9780971732117.
URL https://books.google.sk/books?id=4EW9oQEACAAJ

[7] Hao, Z.: Loss Functions in Neural Networks. [Online; navštívené 21.02.2019].
URL https://isaacchanghau.github.io/post/loss_functions/

[8] Khan Academy: Overview of neuron structure and function. [Online; navštívené 20.01.2019].
URL https://www.khanacademy.org/science/biology/human-biology/neuron-nervous-system/a/overview-of-neuron-structure-and-function

[9] N. Gourier, J. L. C., D. Hall: *Estimating Face Orientation from Robust Detection of Salient Facial Features*. Proceedings of Pointing 2004, ICPR, International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK.
URL http://www-prima.inrialpes.fr/perso/Gourier/Faces/HPDatabase.html

[10] Ruizendaal, R.: Deep Learning 3: More on CNNs Handling Overfitting. [Online; navštívené 20.03.2019].
URL https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d

[11] Svozil, D.; Kvasnička, V.; Pospíchal, J.: Introduction to multi-layer feed-forward neural networks. [Online; navštívené 27.02.2019].
URL https://www.sciencedirect.com/science/article/pii/S0169743997000610

[12] Venkatachalam, M.: Recurrent Neural Networks. [Online; navštívené 30.03.2019].
URL
https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce