UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA
PRÍRODOVEDECKÁ FAKULTA

ADAPTÁCIA NEURÓNOVÝCH SIETÍ PRE PROBLÉM POČÍTAČOVÉHO VIDENIA V REÁLNOM ČASE

2021 **Bc. Tomáš KEKEŇÁK**

UNIVERZITA PAVLA JOZEFA ŠAFÁRIKA
PRÍRODOVEDECKÁ FAKULTA

ADAPTÁCIA NEURÓNOVÝCH SIETÍ PRE PROBLÉM POČÍTAČOVÉHO VIDENIA V REÁLNOM ČASE

CHYBA! NENAŠIEL SA ŽIADEN ZDROJ ODKAZOV.

DIPLOMOVÁ PRÁCA

Študijný program:	Informatika
Pracovisko (katedra/ústav):	Ústav informatiky
Vedúci diplomovej práce:	Mgr. Alexander Szabari, PhD.
Konzultant diplomovej práce:	

Košice 2021

Bc. Tomáš KEKEŇÁK

Zadanie záverečnej práce

Zadanie záverečnej práce (ďalej len „zadanie“) je dokument, ktorým vysoká škola stanoví študentovi študijné povinnosti v súvislosti s vypracovaním záverečnej práce. Zadanie spravidla obsahuje: typ záverečnej práce, názov záverečnej práce, meno, priezvisko a tituly študenta, meno, priezvisko a tituly školiteľa, v prípade externého školiteľa meno, priezvisko a tituly konzultanta, školiace pracovisko, meno, priezvisko a tituly vedúceho pracoviska, anotáciu záverečnej práce, jazyk, v ktorom sa práca vypracuje, dátum schválenia zadania.

Poďakovanie

Na tomto mieste môže byť vyjadrenie poďakovania napr. vedúcemu práce resp. konzultantom za pripomienky a odbornú pomoc pri vypracovaní práce. Nie je zvykom ďakovať za rutinnú kontrolu, menšiu spoluprácu alebo všeobecné rady. Vyjadrenie poďakovania v prípade využitia inej práce sa uskutočňuje formou citácie na konci hlavného textu práce a odkazy na citáciu sa musia uviesť aj na zodpovedajúcich miestach v texte.

Abstrakt v štátnom jazyku

V súčasnosti môžeme pozorovať veľký nárast popularity minimalizácie neurónových sietí. Súčasné modely sú výpočtovo veľmi zložité, a preto pomalé. V našej práci sa venujeme zmenšovaniu neurónových sietí pre problém detekcie tvárí. Cieľom práce je zrýchliť výpočet pri minimálnej strate presnosti. Pracovali sme s odoberaním celých konvolučných filtrov, keďže práve tento prístup garantuje reálne zníženie počtu operácií a tým aj zrýchlenia výpočtu. Implementovali sme viacero spôsobov minimalizácie modelov pomocou knižnice PyTorch. Dosiahli sme veľmi dobré výsledky, pri zmenšení počtu operácií o 73 % sme stratili na presnosti iba 1 %. Taktiež sme vytvorili metriky, ktoré ponúkajú jednoduchý návod na výber, koľko percent parametrov máme odobrať z pôvodnej siete, aby sme zabezpečili najlepší pomer zrýchlenia a straty presnosti.

Abstrakt v cudzom jazyku

Text abstraktu v svetovom jazyku je potrebný pre integráciu do medzinárodných informačných systémov (napr. The Network Digital Library of Theses and Dissertations). Ak nie je možné jazykovú verziu umiestniť na jednej strane so slovenským abstraktom, je potrebné umiestniť ju na samostatnú stranu (cudzojazyčný abstrakt nemožno deliť a uvádzať na dvoch stranách).

Obsah

Zoznam ilustrácií

Zoznam tabuliek

Úvod

V posledných rokoch sme svedkami veľkého rozmachu strojového učenia a umelej inteligencie. Najväčšie pokroky sa podarilo dosiahnuť v oblasti hĺbkového učenia (deep learning) pomocou hlbokých neurónových sietí. Tieto prediktívne algoritmy sa podarilo aplikovať na množstvo takých problémov, kde mali v minulosti algoritmy strojového učenia veľmi zlú úspešnosť.

Veľké úspechy sa dosiahli v oblastiach počítačového videnia, rozpoznávania hlasu a textových analýz. Tieto úspechy sú následkom najmä toho, že v súčasnosti máme k dispozícii oveľa väčšiu a lacnejšiu výpočtovú silu a pre tieto algoritmy je dostupné obrovské množstvo trénovacích dát.

V tejto práci sa budeme venovať počítačovému videniu, konkrétne detekcii tvárí. Detekcia tvárí má obrovské využitie v reálnom živote, či už ide o hľadanie zločincov, bezpečnostné systémy budov, odblokovania nášho smartphonu, alebo vyhľadávanie mien osôb na fotke na sociálnych sieťach.

Detekcia tvárí, ako aj iné podobné algoritmy sú však výpočtovo veľmi zložité. Neurónové siete, ktoré riešia tento problém majú obrovské množstvo skrytých neurónov a konvolučných filtrov, aby pri predikcii dosiahli dostatočnú presnosť.

V aplikáciách je žiaduce, aby detekčné systémy mohli fungovať aj na mobilných telefónoch alebo na rôznych embedovaných zariadeniach. Tieto zariadenia nie sú výpočtovo až také výkonné, čo má za následok, že mnoho výpočtov trvá veľmi dlho a aplikácie nestíhajú fungovať v reálnom čase. Ďalším dôležitým aspektom je, že tieto zariadenia často fungujú na batériu, ktorá sa pri zložitých výpočtoch rýchlo vybije.

Preto je potrebné zjednodušiť modely na detekciu tvárí, teda zmenšiť počet neurónov, konvolučných filtrov alebo počet skrytých vrstiev v danom modeli. Práve touto tematikou sa budeme zaoberať v našej práci.

1 Orezávanie konvolučných filtrov

Ako už bolo spomenuté v úvode článku, v aplikáciách je veľmi žiaduce, aby sme zmenšovali a zrýchľovali prediktívne modely, vďaka čomu sú využiteľnejšie v praxi. Jednou z možností zrýchlenia je orezávanie (pruning).

Základný a najpoužívanejší model orezávania orezáva váhy neurónov. Orezávanie váh sa robí na základe ich veľkostí (t. j. orezávajú sa malé váhy, keďže je predpoklad, že prispievajú veľmi malou hodnotou k výsledku celej siete). Kým pri jednoduchých dopredných sietiach tento typ orezávania čiastočne zrýchli výpočet siete, pri konvolučných modeloch to až tak neplatí.

Tento spôsob orezávania váh síce zmenší počet parametrov v modeli, avšak väčšina váh v konvolučných sieťach sa nachádza v posledných plne prepojených vrstvách, kde sú všetky neuróny prepojené so všetkými ostatnými v ďalšej vrstve. Tieto vrstvy obsahujú síce veľa parametrov, ale k celkovému času výpočtu prispievajú vo veľmi malej miere. Väčšina výpočtov sa totiž udeje v konvolučných sieťach, ktoré majú málo parametrov, keďže sa na konvolučné filtre môžeme pozerať aj ako na zdieľané váhy. Ako príklad môžeme uviesť známu sieť VGG-16, v ktorej je až 90 % parametrov obsiahnutých v posledných plne prepojených vrstvách, avšak tieto parametre tvoria menej ako 1 % celkových výpočtov.

Z toho vidno, že orezávanie váh nám vie prakticky len zmenšiť veľkosť modelu, čo je pri súčasných hardvérových možnostiach nezaujímavé, keďže pamäte sú veľmi lacné. Ako príklad stačí zobrať mobilné telefóny, kde už aj najlacnejšie modely majú operačnú pamäť 1-2 Gb. Teda z pohľadu real-time systémov je oveľa kritickejší čas výpočtu samotného modelu.

Ďalej treba poznamenať aj skutočnosť, že ak zmažeme váhu zo samotného konvolučného filtra, rýchlosť výsledného výpočtu to vo väčšine prípadov neovplyvní. Bežne používané frameworky na strojové učenie totiž nie sú nijako optimalizované na výpočty s riedkymi maticami a vo väčšine prípadov sa zmazané váhy vo filtroch len nahradia nulami.

To nás vedie k myšlienke, že by sme mali mazať celé konvolučné filtre. Ak vymažeme celý konvolučný filter, tak priamo urýchlime celý výpočet bez ohľadu na to, aký framework používame, keďže môžeme vynechať všetky výpočty daného filtra.

1.1 Orezanie konvolučného filtra

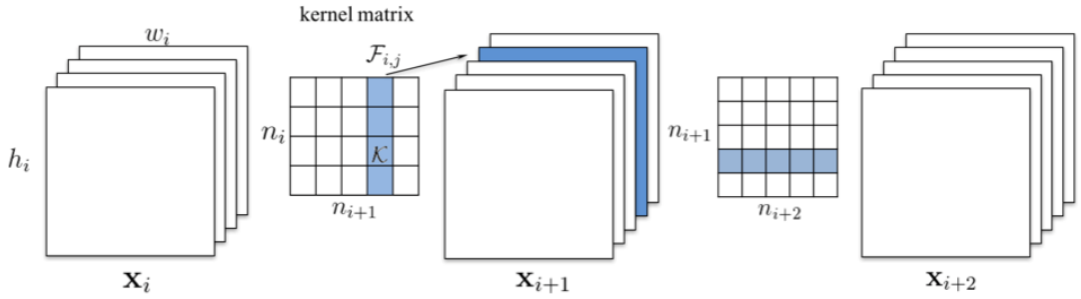
V tejto podkapitole uvedieme, ako vieme zmazať celý filter a aj to, v akej miere nám to presne zníži počet operácií vo výpočtoch [9].

Nech n_i je počet vstupných kanálov do i-tej konvolučnej vrstvy. Nech h_i je výška, w_i je šírka vstupných máp príznakov. Konvolučná vrstva transformuje tieto vstupné mapy príznakov $x_i \in \mathbb{R}^{n_i \times h_i \times w_i}$ na výstupné mapy príznakov $x_{i+1} \in \mathbb{R}^{n_{i+1} \times h_{i+1} \times w_{i+1}}$, ktoré sú aj vstupnými mapami príznakov v ďalšej, $(i+1)$-vej vrstve. Jednotlivé mapy príznakov označme postupne $x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(n_i)}$.

Transformácia x_i na x_{i+1} sa udeje aplikovaním n_{i+1} 3D konvolučných filtrov $\mathcal{F}_{i,j} \in \mathbb{R}^{n_i \times k \times k}$ na n_i vstupných máp príznakov, kde každý filter vytvorí jednu novú výstupnú mapu príznakov. Každý 3D konvolučný filter sa skladá z n_i 2D filtrov (kernelov), ktoré sú veľkosti $k \times k$ (napr. $3 \times 3, 5 \times 5$). Všetky filtre spolu tvoria maticu $\mathcal{F}_i \in \mathbb{R}^{n_i \times n_{i+1} \times k \times k}$.

Následne vieme spočítať celkový počet operácií, ktoré táto vrstva vykoná. Je to presne $n_i n_{i+1} k^2 h_{i+1} w_{i+1}$ (je to vlastne použitie $\mathcal{F}_i$ na každý pixel mapy príznakov).

Ak zmažeme konvolučný filter $\mathcal{F}_{i,j}$, tak jeho výstupná mapa $x_{i+1,j}$ sa tiež vymaže. To zníži celkový počet operácií na tejto vrstve o $n_i k^2 h_{i+1} w_{i+1}$. Kernely (teda 2D filtre), ktoré sa aplikovali v ďalšej, $(i+2)$-hej vrstve na zmazanú výstupnú mapu $x_{i+1,j}$, sa tiež odoberú zo siete. To nám ušetrí ďalších $n_{i+2} k^2 h_{i+2} w_{i+2}$ operácií.



Obr. 1 Orezávanie konvolučného filtra [9]

Zmazanie jedného filtra na i-tej vrstve zmenší počet výpočtov o $\frac{n_i k^2 h_{i+1} w_{i+1}}{n_i n_{i+1} k^2 h_{i+1} w_{i+1}} = \frac{1}{n_{i+1}}$ pôvodných výpočtov na i-tej vrstve. Na $(i+1)$-vej vrstve sa počet výpočtov zmení o $\frac{n_{i+2} k^2 h_{i+2} w_{i+2}}{n_{i+1} n_{i+2} k^2 h_{i+2} w_{i+2}} = \frac{1}{n_{i+1}}$.

Aplikovaním týchto výpočtov m-krát, zmazanie m filtrov na i-tej vrstve zmenší počet výpočtov o $\frac{m}{n_{i+1}}$ pôvodných výpočtov na i-tej a $(i+1)$-vej vrstve. To vedie k dôležitému výsledku, že zmazanie p percent filtrov v každej vrstve zrýchli výpočet o $\approx (2p - p^2)$ percent (okrem prvej a posledných plne prepojených vrstiev).

1.1.1 Príklad

Pre lepšiu pochopiteľnosť si ukážeme, ako funguje orezávanie jedného filtra v praxi a o koľko presne sa zmenší počet operácií vykonávaných daným modelom po odstránení filtra. Ako príklad ukážeme orezávanie druhej vrstvy siete O-net z modelu MTCNN.

Ako vidíme, vstup pre túto vrstvu má dimenzie $23 \times 23 \times 32$, výstup má veľkosť $10 \times 10 \times 64$. Teda druhá vrstva má 64 príznakových vstupných máp, ktorých výška a šírka je 23. Výstupných máp je 64, a preto táto vrstva obsahuje $n_3 = 64$ 3D filtrov, z ktorých každý má $n_2 = 32$ kernelov veľkosti 3×3 (teda $k = 3$, použili sme konvolúciu 3×3).

Teda vieme, že $n_2 = 32,\ h_2 = 32,\ w_2 = 32$. Podobne vidno, že $n_3 = 64,\ h_3 = 10,\ w_3 = 10$. Analogicky, $n_4 = 64,\ h_4 = 4,\ w_4 = 4$.

Celkový počet operácií vykonaných druhou vrstvou je preto $n_2 n_3 k^2 h_3 w_3 = 32 \times 64 \times 9 \times 10 \times 10 = 1\,843\,200$. Počet operácií vykonaných treťou vrstvou je $n_3 n_4 k^2 h_4 w_4 = 64 \times 64 \times 9 \times 4 \times 4 = 589\,824$.

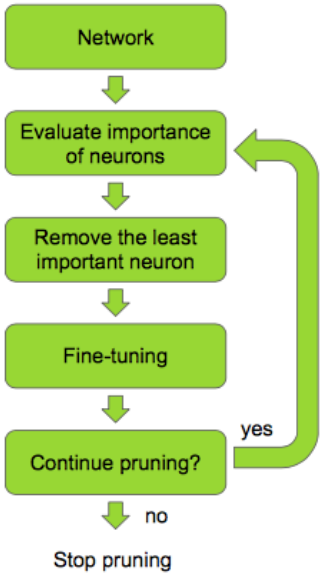
Po odobratí jedného filtra z druhej vrstvy počet operácií v druhej vrstve sa zníži o $n_2 k^2 h_3 w_3 = 32 \times 9 \times 10 \times 10 = 28\,800$. Podobne počet operácií v tretej vrstve sa zníži o $n_4 k^2 h_4 w_4 = 64 \times 9 \times 4 \times 4 = 9216$. V oboch prípadoch celkový počet operácií v danej vrstve klesne o $\frac{28\,800}{1\,843\,200} = \frac{9\,216}{589\,824} = \frac{1}{64} = 0{,}015625$ pôvodných operácií.

Ak odoberieme 25 % pôvodných filtrov v druhej vrstve, tak odoberieme $64 \times 0{,}25 = 16$ filtrov. Počet operácií vykonávaných druhou vrstvou sa zmenší o $25 \times 28\,800 = 720\,000$, počet operácií tretej vrstvy sa zmenší o $25 \times 9\,216 = 230\,400$. V oboch prípadoch je to 25 % pôvodného počtu operácií danej vrstvy, teda dosiahneme zrýchlenie 25 % výpočtov na týchto vrstvách.

1.1.2 Metóda orezávania filtrov

Takmer všetky najpoužívanejšie metódy na orezávanie filtrov využívajú nasledujúce kroky pre dosiahnutie najlepších výsledkov:

1. Vyrátame dôležitosť filtrov na základe zvolenej metriky.
2. Odoberieme najmenej dôležité filtre.
3. Dotrénujeme sieť, aby sme získali lepšie výsledky.
4. Ak ešte chceme orezávať, tak sa vrátime na krok 1, inak končíme.



Obr. 2 Orezávanie schematicky [10]

Orezávanie končíme, keď nájdeme dobrý kompromis medzi zrýchlením modelu a poklesom presnosti predikcie (chceme, aby model zostal stále presný a jeho výpočet sa čo najviac zrýchlil).

1.2 Výber filtrov na zmazanie

V tejto podkapitole popíšeme ako sa problém odoberania filtrov môžeme pozerať ako na optimalizačný problém. Označme našu dátovú sadu $\mathcal{D} = \{X, Y\}$, kde X sú vstupné dáta a Y je ich ohodnotenie. Parametre danej siete označme $\mathcal{M}$. Tieto parametre sú optimalizované, aby sa minimalizovala chyba predikcie, teda nejaká chybová funkcia $\mathrm{C}(\mathcal{D}|\mathcal{M})$ (ktorá závisí od dátovej sady a je nezávislá od typu orezávania).

Metóda orezávania filtrov maže filtre už z dobre natrénovanej siete a snaží sa minimalizovať pokles presnosti predikcie modelu. V článku [2] autori predstavili myšlienku, že na orezávanie filtrov sa dá pozerať ako na optimalizačný problém. To znamená, že k pôvodným parametrom $\mathcal{M}$ sa snažíme nájsť také $\mathcal{M}'$, pre ktoré $\mathrm{C}(\mathcal{D}|\mathcal{M}') \approx \mathrm{C}(\mathcal{D}|\mathcal{M})$. Presnejšie, snažíme sa nájsť také $\mathcal{M}'$, ktoré minimalizuje $|\triangle \mathrm{C}(h_i)| = |\mathrm{C}(\mathcal{D}|\mathcal{M}') - \mathrm{C}(\mathcal{D}|\mathcal{M})|$ a popritom minimalizovali aj $\|\mathcal{M}'\|_0$, teda l_0 - normu (počet) parametrov $\mathcal{M}'$.

Na to, aby sme našli najlepšiu kombináciu odobratých parametrov $\mathcal{M}'$, musíme preskúmať všetkých $2^{|\mathcal{M}|}$ možností podmnožín $\mathcal{M}$. Pri aktuálnych veľkostiach neurónových sietí, ktoré majú tisíce parametrov, to však nie je výpočtovo možné.

Preto treba použiť iné metódy výberu správnej podmnožiny parametrov. Väčšina súčasných metód používa nejaký greedy prístup k tomuto problému, keď v každom kroku odoberieme také parametre (v našom prípade také filtre, resp. im prislúchajúce príznakové mapy), ktoré sú pre nás lokálne „najlepšie“ z určitého hľadiska.

Je potrebné poznamenať, že v rôznych vedeckých publikáciách sa niekedy orezávajú filtre, inokedy príznakové mapy. V predošlej časti sme ukázali, že každý filter vytvára práve jednu príznakovú mapu a každá príznaková mapa je vytvorená jedným filtrom (okrem vstupných máp x_0), čiže ide o tú istú vec, len z dvoch rôznych pohľadov.

1.3 Metriky výberu filtrov na odstránenie

V ďalšej podkapitole uvedieme niekoľko možností, ako vybrať vhodné filtre (príznakové mapy) na orezanie. V ďalších úvahách Θ_i bude označovať „dôležitosť“ (rank) filtra i.

1.3.1 Oracle

Najlepší spôsob, ako vybrať najmenej dôležité parametre (filtre), je presne empiricky vypočítať rozdiel v zmene presnosti siete odobratím daných parametrov pre všetky kombinácie výberu parametrov na zmazanie. To znamená, že ak sieť obsahuje n filtrov, z ktorých chceme zmazať k kusov, tak vypočítame zmenu v presnosti pre všetkých $\binom{n}{k}$ možností, kde odoberieme vybraných k filtrov. Kvôli tejto vlastnosti sa zvykne aj nazývať *kombinatorický oracle*.

Je zrejmé, že táto metóda dáva najlepšie výsledky, ale za cenu obrovských výpočtov, ktoré sú v praktickom živote takmer nemožné. Preto existuje aj iné, „chamtivé“ riešenie, nazývané *greedy oracle*. Tu sa parametre siete odoberajú po jednom, teda k parametrov sa odoberie v k krokoch. V každom kroku sa vypočíta rozdiel v zmene presnosti siete odobratím daného parametra (len jedného, nie kombinácií). Týmto spôsobom stačí vypočítať zmenu v presnosti siete „len“ $n + (n-1) + \cdots + (n-k) \approx nk$-krát.

Tento spôsob pre malý počet odoberaných filtrov dáva veľmi podobné výsledky ako kombinatorický oracle pre malý počet odoberaných filtrov [10], ale je exponenciálne rýchlejší.

1.3.2 Náhodné odoberanie filtrov

Ako napovedá už názov samotnej metódy, filtre sú odoberané náhodne. Metóda funguje na veľmi jednoduchom princípe: každému filtru sa priradí náhodné číslo, t. j. dôležitosť filtra. V každom kroku sa odoberú najmenej dôležité filtre.

Táto metóda slúži na základné porovnávanie metód výberu najmenej dôležitých filtrov ako tzv. baseline. Od komplikovanejšej metódy na určenie dôležitosti filtrov prirodzene očakávame lepšie výsledky ako od náhodného odoberania, keďže v opačnom prípade naša metóda odoberá dôležité filtre namiesto menej dôležitých, čo nedáva veľký zmysel.

1.3.3 Minimálna l_1-norma

Pri tejto metóde na začiatku zvolíme jednu vrstvu siete, ktorú chceme orezať. Vo zvolenej i-tej vrstve sa vypočíta relatívna dôležitosť daného filtra $\mathcal{F}_{i,j}$ na základe svojej l_1-normy $\left\|\mathcal{F}_{i,j}\right\|_1$, t. j. na základe sumy absolútnych hodnôt jeho váh.

Keďže vo všetkých filtroch $\mathcal{F}_{i,j}$ je počet vstupných kanálov n_i, to znamená, že toto číslo reprezentuje aj priemernú veľkosť váh jeho kernelov. Toto nám umožní predpokladať dôležitosť jeho výstupnej mapy príznakov. Filtre s malými váhami zvyčajne vytvárajú málo dôležité mapy príznakov v porovnaní s ostatnými výstupnými mapami príznakov danej vrstvy.

Autori článku [9] zistili, že použitie najmenšej l_1-normy dáva lepšie výsledky ako zvolenie najväčšej l_1-normy, resp. náhodne zvolenie filtrov. Ďalej zistili, že zvolenie najmenšej l_1-normy je dobré kritérium na orezávanie filtrov, pričom dáva dobré výsledky bez ohľadu na dáta, s ktorými pracujeme.

Algoritmus na orezávanie m filtrov v i-tej vrstve funguje nasledovne:

1. Pre každý filter $\mathcal{F}_{i,j}$ vypočítam jeho $l_1 -$ normu, teda vypočítame
$$\Theta_{i,j} = \sum_{l=1}^{n_i} \sum_{w \in K_l} |w|\,, \text{ kde } K_l \text{ označuje l} - \text{ty kernel filtra } \mathcal{F}_{i,j.}$$
2. Zoradíme filtre podľa $\Theta_{i,j}$.
3. Zmažeme m filtrov a im patriace príznakové mapy. Kernely v nasledujúcej vrstve, ktoré patrili zmazaným príznakovým mapám tiež odoberieme.
4. Vytvoríme nové matice $\mathcal{F}_i, \mathcal{F}_{i+1}$, ostatné váhy prekopírujeme do nového modelu.

1.3.4 Minimálna l_2-norma

Táto norma funguje rovnako ako metóda s l_1-normou. Autori [9] taktiež zistili, že tieto dve metódy dávajú takmer rovnaké výsledky.

1.3.5 Taylorov rozvoj

V článku [10] autori predstavili novú sofistikovanú metódu na výber filtrov na odstránenie. Ako už bolo spomenuté, na orezávanie filtrov sa dá pozerať ako na optimalizačný problém, kde sa snažíme nájsť také $\mathcal{M}'$, ktoré minimalizuje

$$|\triangle \mathrm{C}(h_i)| = |\mathrm{C}(\mathcal{D}|\mathcal{M}') - \mathrm{C}(\mathcal{D}|\mathcal{M})|.$$

Teda pomocou Taylorovho rozvoja vieme priamo aproximovať zmenu chybovej funkcie, ktorá nastane pri odobratí daného parametra (odobratí daného filtra). Označme h_i výstup z i-teho parametra (teda výstupnú mapu príznakov i-teho filtra). Teda v našom prípade $h = \left\{x_0^{(1)}, x_0^{(2)}, \dots, x_L^{(n_l)}\right\}$, kde $x_i^{(j)}$ sú príznakové mapy (neurónová sieť obsahuje l vrstiev a dokopy L konvolučných filtrov). Ak predpokladáme nezávislosť parametrov, tak dostaneme:

$$|\triangle \mathrm{C}(h_i)| = |\mathrm{C}(\mathcal{D}, h_i = 0) - \mathrm{C}(\mathcal{D}, h_i)|,$$

kde $\mathrm{C}(\mathcal{D}, h_i = 0)$ označuje veľkosť chyby (hodnota chybovej funkcie), keď parameter h_i je orezaný, a $\mathrm{C}(\mathcal{D}, h_i)$ označuje veľkosť chyby, keď parameter h_i nie je orezaný. Napriek tomu, že parametre nie sú nezávislé, počítame s nimi, akoby boli nezávislé pri každom kroku gradientného zostupu.

Aby sme aproximovali $|\triangle \mathrm{C}(h_i)|$, použijeme Taylorov polynóm prvého rádu. Pre ľubovoľnú funkciu $f(t)$ Taylorov polynóm v bode a vyzerá nasledovne:

$$f(t) = \sum_{p=0}^{P} \frac{f^{(p)}(a)}{p!}(t-a)^p + \mathcal{R}_p(t),$$

kde $f^{(p)}(a)$ je p-ta derivácia funkcie f v bode a, $\mathcal{R}_p(t)$ je zvyšok p-teho stupňa.

Následne, ak aproximujeme $\mathrm{C}(\mathcal{D}, h_i = 0)$ Taylorovým polynómom prvého rádu v blízkosti bodu $h_i = 0$, tak dostávame:

$$\mathrm{C}(\mathcal{D}, h_i = 0) = \mathrm{C}(\mathcal{D}, h_i) - \frac{\partial \mathrm{C}}{\partial h_i} h_i + \mathcal{R}_1(h_i = 0)$$

Zvyšok $\mathcal{R}_1(h_i = 0)$ môžeme vypočítať pomocou Lagrange-ovho tvaru:

$$\mathcal{R}_1(h_i = 0) = \frac{\mathrm{C}^{(2)}(\xi)}{2!}(h_i - 0)^2 = \frac{\partial^2 \mathrm{C}}{\partial(h_i^2 = \xi)} * \frac{h_i^2}{2},$$

kde ξ je kladné číslo menšie ako h_i. Tento zvyšok ale v ďalších výpočtoch zanedbáme kvôli zrýchleniu výpočtov.

Dosadením vzťahu (5) do rovnice (3) a zanedbaním zvyšku dostávame:

$$\Theta_i = |\triangle \mathrm{C}(h_i)| = |\mathrm{C}(\mathcal{D}, h_i = 0) - \mathrm{C}(\mathcal{D}, h_i)| \approx \left|\mathrm{C}(\mathcal{D}, h_i) - \frac{\partial \mathrm{C}}{\partial h_i} h_i - \mathrm{C}(\mathcal{D}, h_i)\right|$$

$$= \left|\frac{\partial \mathrm{C}}{\partial h_i} h_i\right|$$

Ak označíme $g_i = \frac{\partial \mathrm{C}}{\partial h_i}$, tak výraz nadobudne nasledujúci tvar: $\Theta_i = (g_i h_i)^2$.

Teda toto kritérium orezáva tie parametre, ktoré majú veľmi plochý (takmer rovnaký) gradient chybovej funkcie podľa príznakovej mapy h_i.

V praxi výpočet hodnoty Θ_i sa dá vypočítať pomocou spätnej propagácie, stačí nám vždy len vypočítať gradient chybovej funkcie a aktivačnú hodnotu. Pre príznakové mapy (teda filtre) výpočet vyzerá nasledovne:

$$\Theta_{l,k} = \left|\frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathrm{C}}{\partial x_{l,i}^{(k)}} x_{l,i}^{(k)}\right|,$$

kde N označuje veľkosť mapy príznakov.

1.3.6 Taylorov rozvoj druhého rádu

V publikácii [17] autori pokračovali v myšlienke z podkapitoly 1.3.5 a nahradili Taylorov polynóm prvého rádu Taylorovým polynómom druhého rádu.

Označme $H_{i,j} = \frac{\partial^2 C}{\partial h_i h_j}$ prvky Hessiánu H a i-ty riadok Hessiánu označme H_i. Potom chyba je $\Theta_i = \left(g_i h_i - \frac{1}{2} h_i H_i \mathcal{M}\right)^2$ v blízkosti bodu $\mathcal{M}$.

Tento prístup dáva trochu lepšie výsledky ako použitie aproximácie prvého rádu, ale za cenu zvýšenej výpočtovej zložitosti. Pri tomto prístupe je nutné vypočítať Hessián, čo je výpočtovo veľmi náročné. Autori [17] preto použili jeho diagonálnu aproximáciu. Ani tá im však nepomohlo pri použití tohto prístupu pri probléme ImageNet, kde napriek ich takmer neobmedzeným hardvérovým možnostiam (ide o pracovníkov NVIDIA) narazili na problémy s nedostatočnou pamäťou.

1.3.7 Priemerný počet núl

V článku [11] navrhli, aby sa najmenej dôležité filtre vyberali pomocou hustoty aktivácií, teda výstupných príznakových máp. Takmer všetky neurónové siete využívajú aktivačnú funkciu ReLU, ktorá zvyšuje riedkosť výstupu oproti vstupu. Preto percento kladných aktivácií vie implikovať, ktoré neuróny sú dôležitejšie a ktoré, naopak, menej.

Táto myšlienka sa dá podobne preniesť aj na celé konvolučné filtre. Problémom tohoto prístupu je však, že filtre v prvých vrstvách majú veľmi podobný počet núl, preto túto metódu vo všeobecnosti nie je dobré použiť na výber filtrov „na začiatku“ siete.

1.3.8 Aktivačná hodnota

Popularita aktivačnej funkcie ReLU spočíva vo zvýšenej riedkosti výstupu. Táto vlastnosť umožňuje konvolučným filtrom, aby plnili úlohu detektorov čŕt. Preto môžeme predpokladať, že ak je aktivačná hodnota malá, tak daný detektor čŕt (teda konvolučný filter) nie je dôležitý. Preto aj priemerná aktivačná hodnota (výstup) filtra je dobrým kandidátom, ako určiť dôležitosť filtrov.

1.4 Orezávanie pred tréningom

Všetky doterajšie metódy orezávania boli použiteľné na už natrénovanú sieť, ktorú zmenšujeme po tréningu. V úplnom kontraste s týmito metódami, autori [18] sformulovali a testovaním overili hypotézu lístka z lotérie (Lottery ticket hypothesis).

Hypotéza znie nasledovne: V plne prepojenej doprednej neurónovej sieti (tvrdenie sa dá preniesť aj na konvolučné siete) pri náhodnej inicializácii existujú riedke podsiete, ktoré aj pri oddelenom trénovaní dosiahujú presnosť pôvodnej (celej) siete.

Toto zistenie otvorilo úplne novú oblasť výskumu a za necelé dva roky sa v tejto tematike publikovalo množstvo článkov.

1.4.1 Výhra v lotérii

V tejto časti popíšeme, čo presne znamená vyššie sformulovaná hypotéza a aké dôsledky prináša. Hypotéza hovorí, že pri náhodnej inicializácii veľkej neurónovej siete existujú riedke podsiete, ktoré vyhrali v lotérii pri inicializácii, t. j. váhy týchto sietí boli inicializované na také dobré hodnoty, že samotné podsiete dokážu dosiahnuť podobnú presnosť ako pôvodné siete.

Táto vlastnosť má za následok, že ak vieme určiť „výhercov v lotérii“, tak vieme natrénovať oveľa menšiu sieť, teda oveľa rýchlejšie ako pôvodnú sieť. To je hlavný rozdiel oproti metódam orezávania po tréningu, ktoré nezrýchlia samotný tréning siete, len ohodnotenie výsledkov.

1.4.2 Nájdenie výherných lístkov

V publikácii [18] je popísaný spôsob, ako identifikovať výherné lístky. Začíname trénovať pôvodnú neurónovú sieť a postupne ju orezávame, až kým nenájdeme výhercov. Algoritmus nazývaný Iterative Magnitude Pruning (IMP) vyzerá nasledovne:

1. Náhodne inicializujme neurónovú sieť $f(x, \Theta_0)$.
2. Trénujeme neurónovú sieť j iterácií, čím získame parametre Θ_j.
3. Orežeme $p^{\frac{1}{n}}$ % váh s najmenšou veľkosťou.
4. Parametre siete zmeníme späť na pôvodné parametre Θ_0.

Body 2 – 4 opakujeme n-krát, čím zmenšíme svoju sieť o p percent. Ak sa lepšie pozrieme na tento algoritmus, tak je veľmi podobný tým, ktoré orezávajú sieť po tréningu. Hlavný rozdiel spočíva v bode 4, teda v tom, že parametre po každom orezaní zmeníme na tie pôvodné.

Treba poznamenať, že pri tomto postupe je kľúčové nastavenie váh na pôvodné hodnoty, keďže pri náhodnej reinicializácii váh už nebolo možné natrénovať menšiu sieť, aby dosahovala dostatočnú presnosť. To poukazuje na kľúčový fakt výherného lístka, že nielen samotný dizajn podsietí je dôležitý, ale aj inicializácia ich váh.

1.4.3 Nezrovnalosti v doterajších výsledkoch

Za krátky čas, odkedy sa výskumníci aktívne zaoberajú tematikou orezávania sietí, vyšlo niekoľko publikácií, ktoré navrhujú úplne iný pohľad na danú problematiku. Hlavnou nezrovnalosťou vo výsledkoch oproti algoritmu IMP je, že autori [20], [21] tvrdia, že pôvodná inicializácia váh nie je dôležitá, t. j. dosahujú podobné výsledky aj pomocou náhodnej reinicializácie váh.

Liu a spol. [20] tvrdia, že riedke podsiete, ktoré nájdu pomocou svojou metódy, môžu byť náhodne reinicializované a aj takto dosahujú dobré výsledky. Ďalej, kým autori IMP (pôvodného algoritmu na nájdenie výherných lístkov) potrebujú sieť natrénovať mnohokrát, Liu a spol. stačí natrénovať sieť len raz. Toto robí ich algoritmus výrazne rýchlejším, a teda aj využiteľnejším v praktických aplikáciách.

Autorom [21] sa dokonca stačí pozrieť len na jeden mini-batch dát, na základe ktorého vedia identifikovať dobre trénovateľné podsiete. Ich metóda SNIP (Single-shot network pruning based on connection sensitivity) orezáva sieť hneď na začiatku tréningu (t. j. nie je ani raz nutné natrénovať celú sieť).

Táto metóda, ako napovedá jej názov, sleduje citlivosť chybovej funkcie v závislosti od jednotlivých parametrov siete. Následne orezáva tie parametre (váhy), pre ktoré je najmenej citlivá chyba siete. Citlivosť je určená pomocou virtuálneho parametra $c = 1$, na základe ktorého sa dá vypočítať $\frac{\partial L}{\partial c}$, kde L označuje chybu siete.

1.4.4 Určenie príčin nezrovnalostí

Frankle a spol. [19] identifikovali niekoľko faktorov, ktoré môžu mať za následok tieto rozdiely. Hlavnou príčinou bolo, že autori neorezávali pôvodnú sieť na rovnakú riedkosť. V publikáciách [20], [21] sa z neurónovej siete orezalo maximálne 80 % parametrov, v [18] sa orezalo až 99,5 % parametrov.

Autori [19] označili algoritmus SNIP ako za veľmi nádejný najmä vďaka jeho rýchlosti, avšak napr. pri sieti VGG19 nedosahoval také dobré výsledky ako algoritmy [18] a [20]. Podobne testovaním zistili, že do určitej riedkosti (ktorú odhadujú na 80 %) sa všetky 3 algoritmy chovajú rovnako, teda pôvodné váhy nie sú dôležité. Akonáhle však

prekročia túto hranicu, algoritmus IMP dáva výrazne lepšie výsledky ako algoritmy [20], [21].

Toto poukazuje na fakt, že žiaden z algoritmov nie je najlepší, a je potrebné si vybrať kompromis medzi cieľovou riedkosťou podsiete, presnosťou algoritmu a samotným časom výpočtu.

1.4.5 Orezávanie bez dát

Dajú sa určiť dôležité podsiete aj bez použitia dát a bez tréningu? Všetky doterajšie známe publikácie ich potrebovali. Tanaka a spol. [12] však tvrdia, že dôležitosť váh sa dá určiť aj bez použitia tréningových dát. Ich algoritmus *SynFlow* (*Iterative Synaptic Flow Pruning*) ich nepotrebuje.

Autori tvrdia, že jednou z hlavných príčin, prečo sa orezané podsiete nedokážu natrénovať na pôvodnú presnosť siete je zmazanie vrstvy (tzv. *layer collapse*), teda zmazanie všetkých parametrov jednej vrstvy. Ďalej tvrdia, že orezávacie algoritmy by mali spĺňať axiómu maximálnej kompresie, t. j. oddialiť zmazanie vrstvy, ako sa len dá.

Základom algoritmu je nová chybová funkcia nazvaná *synaptic flow*, ktorá má nasledujúci tvar:

$$C = \vec{1}^T \left(\prod_{l=1}^{L} \left| \Theta^{[l]} \right| \right) \vec{1}$$

$\vec{1}$ označuje jednotkový vektor, $\left|\Theta^{[l]}\right|$ je vektor absolútnych hodnôt parametrov l-tej vrstvy.

Parametre sa orezávajú vo viacerých iteráciách. Pomocou chybovej funkcie sa určí dôležitosť jednotlivých parametrov (teda synaptic saliency) ako $\frac{\partial C}{\partial \Theta} \odot \Theta$, kde $\odot$ označuje Hadamardov súčin matíc. Následne sa odoberú najmenej dôležité parametre.

Dá sa dokázať, že tento algoritmus spĺňa axiómu maximálnej kritickej kompresie pomocou zachovania kladnej *synaptic saliency*. Tento prístup preto prináša vylepšenie oproti iným gradientovým metódam, ako napr. algoritmom SNIP [21] alebo Taylorovmu rozvoju prvého rádu [10], ktoré môžu viesť k zmazaniu jednej celej vrstvy.

Najväčšou zaujímavosťou algoritmu je však to, že nepotrebuje žiadne dáta. Napriek tomu autori empiricky overili, že algoritmus dával najlepšie výsledky pre 4 populárne architektúry na troch rôznych dátových sadách.

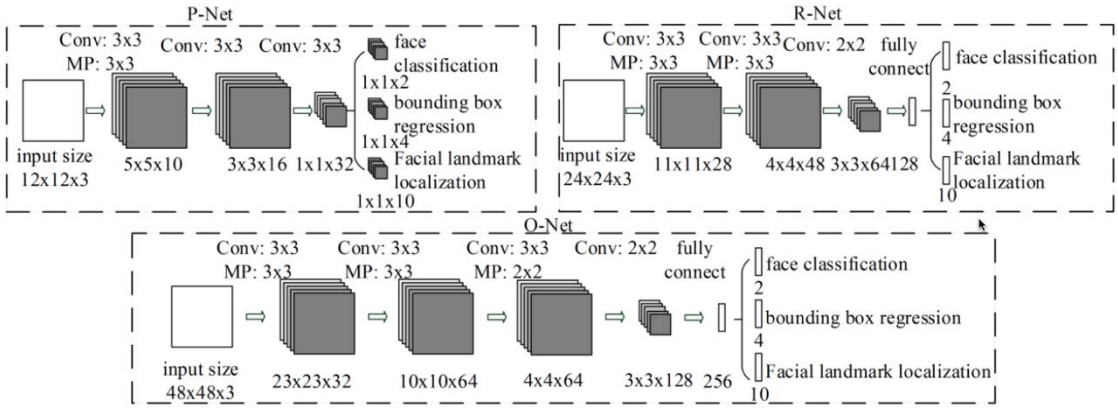
2 Implementácia

Na základe poznatkov z prvej kapitoly sme vytvorili program, ktorý natrénuje neurónovú sieť pre problém detekcie tvárí a následne ju minimalizuje pomocou metód popísaných v podkapitolách 1.2 a 1.3. V tejto kapitole popíšeme implementačné detaily tohto programu, najmä samotnej štruktúri a trénovania siete MTCNN a následného orezávania konvolučných filtrov.

2.1 MTCNN

Neurónová sieť MTCNN (Multitask Cascaded Convolutional Neural Network) [2] je známa konvolučná sieť, ktorá sa využíva na detekciu tvárí na obrázkoch. Multitask znamená, že táto sieť nám, okrem časti obrázku, na ktorej sa nachádza tvár, teda ohraničujúcich rámcov (bounding box), vráti aj orientačné body (landmark) danej tváre. Konkrétne sú to oči, nos, ľavý a pravý roh úst. Tento model sa skladá z troch neurónových sietí:

1. P-net (Proposal network) – Plne konvolučná sieť (neobsahuje žiadne iné vrstvy ako konvolučné), ktorá sa skladá z troch konvolučných vrstiev a slúži na vygenerovanie kandidátov pre rámčeky tvárí. Následne sa použije NMS algoritmus [3] na zjednotenie veľmi sa prekrývajúcich rámcov a výsledok (nájdené rámce) sa použije ako vstup ďalšej siete. Algoritmus NMS je popísaný v nasledujúcej podkapitole 2.1.1.
2. R-net (Refinement network) – Slúži na zdokonaľovanie rámcov, teda zamieta zlých kandidátov, teda false-positive rámce. Táto sieť už okrem konvolučných vrstiev obsahuje aj poslednú plne prepojenú vrstvu. Tiež sa následne použije NMS algoritmus a výsledok sa posunie nasledujúcej sieti.
3. O-net (Output network) – Funguje podobne ako R-net, ale nájde aj orientačné body nájdenej tváre. Kvôli zvýšenej komplexite úlohy je táto sieť najhlbšia zo všetkých troch.



Obr. 3 Architektúra siete MTCNN [2]

Architektúra siete sa líši od iných často využívaných modelov na rozpoznávanie objektov najmä tým, že kým iné siete riešia klasifikáciu do viacerých tried, detekcia tvárí je len binárny klasifikačný problém. Preto sa používajú menšie 3x3 filtre namiesto obvyklých 5x5 filtrov, a použil sa hlbší model, aby sa zlepšila presnosť detekcie.

Ďalšou inovatívnou myšlienkou, ktorá priniesla zvýšenú presnosť predikcie je spojenie dvoch podobných problémov nájdenia ohraničujúcich rámcov a orientačných bodov. Práve vysoká korelácia medzi týmito dvoma úlohami spôsobuje, že sieť sa vyznačuje vyššou presnosťou predikcie v oboch oblastiach ako iné siete, ktoré boli natrénované iba pre jeden z týchto dvoch problémov.

Napriek relatívne veľkej hĺbke sa sieť dá použiť na detekcie tvárí z videa v reálnom čase aj pomocou bežného počítača.

My sme v implementácii použili počiatočné nastavenie váh siete z projektu [4], odkiaľ sme využili aj samotnú štruktúru siete, ktorá vyzerá nasledovne:

```
PNet(
  (conv1): Conv2d(3, 10, kernel_size=(3, 3), stride=(1, 1))
  (prelu1): PReLU(num_parameters=10)
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv2): Conv2d(10, 16, kernel_size=(3, 3), stride=(1, 1))
  (prelu2): PReLU(num_parameters=16)
  (conv3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1))
  (prelu3): PReLU(num_parameters=32)
  (conv4_1): Conv2d(32, 2, kernel_size=(1, 1), stride=(1, 1))
```

```
  (softmax4_1): Softmax(dim=1)
  (conv4_2): Conv2d(32, 4, kernel_size=(1, 1), stride=(1, 1))
)

RNet(
  (conv1): Conv2d(3, 28, kernel_size=(3, 3), stride=(1, 1))
  (prelu1): PReLU(num_parameters=28)
  (pool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv2): Conv2d(28, 48, kernel_size=(3, 3), stride=(1, 1))
  (prelu2): PReLU(num_parameters=48)
  (pool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv3): Conv2d(48, 64, kernel_size=(2, 2), stride=(1, 1))
  (prelu3): PReLU(num_parameters=64)
  (dense4): Linear(in_features=576, out_features=128, bias=True)
  (prelu4): PReLU(num_parameters=128)
  (dense5_1): Linear(in_features=128, out_features=2, bias=True)
  (softmax5_1): Softmax(dim=1)
  (dense5_2): Linear(in_features=128, out_features=4, bias=True)
)

ONet(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1))
  (prelu1): PReLU(num_parameters=32)
  (pool1): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
  (prelu2): PReLU(num_parameters=64)
  (pool2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1))
  (prelu3): PReLU(num_parameters=64)
  (pool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv4): Conv2d(64, 128, kernel_size=(2, 2), stride=(1, 1))
  (prelu4): PReLU(num_parameters=128)
  (dense5): Linear(in_features=1152, out_features=256, bias=True)
  (prelu5): PReLU(num_parameters=256)
  (dense6_1): Linear(in_features=256, out_features=2, bias=True)
  (softmax6_1): Softmax(dim=1)
  (dense6_2): Linear(in_features=256, out_features=4, bias=True)
  (dense6_3): Linear(in_features=256, out_features=10, bias=True)
)
```

2.1.1 NMS

Metóda NMS (Non-Max Suppression) je algoritmus, ktorý sa často využíva v detekcii objektov (teda aj napr. tvárí). Algoritmy generujú množstvo návrhov (v našom prípade napr. sieť PNet), ktoré sú často duplicitné, teda popisujú tie isté objekty (v našom prípade tváre). Tieto duplicitné návrhy chceme vynechať a vybrať iba tie „najlepšie“. Na to slúži algoritmus NMS.

Vstup: list návrhov ohraničujúcich rámcov *B* a ich skóre dôveryhodnosti *S*, prah (threshold) prekrytia *N*

Výstup: list filtrovaných návrhov F

Algoritmus:

- Vytvor prázdny list *F*.
- Kým B nie je prázdne:
    - Nájdi v *B* návrh s najväčším skóre, odober ho z *B* a pridaj do *F*.
    - Porovnaj vybraný návrh so všetkými v *B*, vypočítaj IoU skóre týchto návrhov. Ak je IoU väčšie ako prah *N*, tak odober návrh z *B*.

Je potrebné poznamenať, že existujú aj modifikácie tohto algoritmu, kde napr. v IoU skóre je zjednotenie rámcov vymenené na minimum z veľkostí rámcov. Táto verzia sa využíva aj v MTCNN po sieti ONet.

Existujú aj komplikovanejšie modifikácie pôvodného algoritmu, ako napr. Soft-NMS [16], alebo aj „simulácia“ NMS pomocou konvolučnej siete [3].

2.2 Trénovanie neurónovej siete MTCNN

V tejto časti popíšeme spôsob, akým sme natrénovali neurónovú sieť MTCNN. Keďže pri našom spôsobe orezávania konvolučných filtrov musíme dotrénovať sieť po každej iterácii, museli sme naprogramovať aj samotné trénovanie siete. Inšpirovali sme sa projektom [13], ktorý však využíva mierne odlišnú verziu siete.

Ako tréningovú vzorku pre hľadanie ohraničujúcich rámcov sme použili dataset Wider Face [14], pre nájdenie orientačných bodov tvárí sme použili dataset CNN Face Point [15].

Tréning celej siete MTCNN prebieha v troch krokoch, samostatne pre každú sieť PNet, RNet, ONet. Treba poznamenať, že dátovú sadu CNN Face Point sme použili len pri tréningu siete ONet, keďže len táto sieť slúži aj na nájdenie orientačných bodov.

2.2.1 Wider Face

Wider Face je voľne prístupná dátová sada určená na rozpoznávanie tvárí, ktorá je odvodená z obrázkov v datasete WIDER (Web Image Dataset for Event Recognition). Pôvodný dataset WIDER, ako napovedá aj jeho názov, slúži na klasifikáciu rôznych tipov udalostí (na obrázkoch je zobrazených 61 rôznych tipov udalostí, ako napr. športové podujatia, pohreby, tlačovky...).

Wider Face pridáva informácie, kde sa na daných obrázkoch nachádzajú tváre. Každá tvár je popísaná obdĺžnikom, ktorý ju ohraničuje na obrázku. Dataset tvorí 32 203 obrázkov, ktoré obsahujú dokopy 393 703 tvárí. Tváre na obrázkoch sú zachytené v rôznych uhloch, polohách a veľkostiach, čo robí dataset ideálnym na určenie všeobecnej presnosti modelov.

Obrázky sú rozdelené na tréningové, validačné a testovacie v pomere 40 %, 10 % a 50 %, pričom v každej časti sa nachádzajú obrázky zo všetkých tipov udalostí. My sme využili tréningové a validačné dáta. K testovacím obrázkom nie sú zverejnené ohraničujúce rámce (bounding boxes), keďže testovacie dáta slúžia len na overenie presnosti modelov tvorcami datasetu.

2.2.2 CNN Face Point

Dátová sada CNN Face Point vznikla v rámci vedeckej publikácie [15] na The Chinese University of Hong Kong v spolupráci s Čínskou akadémiou vied. Obsahuje 13 466 obrázkov, z ktorých 5 590 pochádza zo známeho datasetu LFW. Na každom obrázku je vyznačených 5 orientačných bodov tvárí. 10 000 obrázkov je označených ako tréningové, ostatných 3 466 je ponechaných na validáciu.

2.2.3 Predspracovanie dát Wider Face

Obrázky sú uložené v adresároch podľa typu udalostí, ktoré zobrazujú. Popisné súbory ohraničujúcich rámcov sú v matlabovskom formáte. Tieto súbory sme najprv prerobili na textové súbory pomocou scriptu *transform.py*.

Týmto spôsobom sa vytvoria súbory, v ktorých každý riadok popisuje jeden obrázok v datasete. Prvé slovo je vždy názov obrázku s celou cestou. Za ním nasledujú čísla, ktorých počet sa rovná 4-krát počtu tvárí. Každá štvorica popisuje jeden ohraničujúci

rámec vo formáte *x1*, *y1*, *w*, *h*, kde *x1*, *y1* sú súradnice ľavého horného okraja rámca, *w* a *h* označujú šírku a výšku rámca.

2.2.4 Predspracovanie dát CNN Face Point

Dáta z tohto datasetu nebolo nutné nijako formátovať. Popisy jednotlivých obrázkov sú v podobnom formáte ako pri Wider Face. To znamená, že každý riadok popisuje jeden obrázok. Prvé slovo je cesta k obrázku, za ním nasleduje 14 čísel. Prvé štyri popisujú ohraničujúci rámec vo formáte *x1, y1, x2, y2,* kde *x1*, *y1* sú súradnice ľavého horného okraja rámca a *x2, y2* sú súradnice pravého dolného okraja rámca. Ďalších 10 čísel popisuje x-ovú a y-ovú súradnicu orientačných bodov tváre, t. j. ľavé oko, pravé oko, nos, ľavý okraj úst, pravý okraj úst.

2.2.5 Príprava dát pre sieť PNet

Sieť Pnet slúži na nájdenie kandidátov pre ohraničujúce rámce. Na základe odporúčania autorov modelu [2] sme vytvorili tréningové dáta pre sieť. Z obrázkov z WIDER Face sme vyrezali náhodné rámce rôznych veľkostí. Pre tieto rámce (obrázky) sme našli najbližšiu tvár na pôvodnom obrázku a vypočítali sme preň IoU.

IoU (intersection over union) je skóre, ktoré dostaneme ako podiel prieseku a zjednotenia ohraničujúceho rámca, ktorý sme vygenerovali a skutočného rámca najbližšej tváre. Na základe IoU sme vygenerované obrázky rozdelili do troch kategórii:

a) Negatívne príklady: príklady, ktorých IoU je menšie ako 0,3.
b) Čiastočne dobré príklady: príklady, ktorých IoU je medzi 0,4 a 0,65.
c) Pozitívne príklady: príklady, ktorých IoU je väčšie ako 0,65.

Vyrezané obrázky generujeme tak, aby pomer počtu negatívnych, čiastočne dobrých a pozitívnych bol 3:1:1, ako je to popísané v článku [2]. Orezané obrázky prerobíme na obrázky veľkosti 12x12 pomocou knižnice *cv2*.

2.2.6 Príprava dát pre sieť RNet

Sieť RNet slúži na vylúčenie väčšiny zlých kandidátov, ktorých sieť PNet označila ako pozitívnych. Keďže vstup pre túto sieť je výstup siete PNet, preto najprv musíme natrénovať sieť PNet, aby sme vedeli vytvoriť správne testovacie dáta pre sieť RNet.

Pre každý obrázok vygenerujeme kandidátov pomocou (už natrénovanej) siete PNet. Príliš malé obrázky (obrázky, ktorých šírka je užšia ako 20 pixelov) a obrázky, ktoré vyčnievajú z pôvodného obrázka, vylúčime. Ostatné podobným spôsobom ako pri sieti PNet rozdelíme na základe skóre IoU do troch skupín pozitívne, čiastočne dobré, negatívne a obrázky prerobíme na veľkosti 24x24 podobnou metódou ako pri sieti PNet.

2.2.7 Príprava dát pre sieť ONet

Sieť ONet slúži na nájdenie ohraničujúcich rámcov tvárí a na nájdenie orientačných bodov. Vstup pre sieť ONet je totožný s výstupom siete RNet, preto podobne ako v predchádzajúcej časti musíme najprv natrénovať sieť RNet (a teda aj sieť PNet), aby sme vedeli získať tréningové dáta pre sieť ONet.

Rovnakým spôsobom vylúčime príliš malé a vyčnievajúce obrázky. Výstupy zo siete RNet sa kategorizujú do troch skupín (pozitívne, čiastočne dobré, negatívne) pomocou skóre IoU a upravia sa na veľkosť 48x48.

Tréningové dáta na nájdenie orientačných bodov z datasetu CNN Face Point vygenerujeme podobným spôsobom ako dáta pre sieť PNet. To znamená, že náhodne vyrežeme rámce z obrázka. Následne vyrezaným častiam určíme IoU. Rozdiel je v tom, že v tomto prípade pridáme do tréningovej množiny len dáta s IoU väčším ako 0,65, teda iba pozitívne príklady, keďže na obrázkoch bez tváre nemá zmysel hľadať ani orientačné body.

2.2.8 Stratová funkcia

Našu neurónovú sieť sme trénovali pre tri rozličné problémy: detekcia tváre, nájdenie ohraničujúcich rámcov a nájdenie orientačných bodov tváre (len pri ONet). Pre každý problém sme použili inú stratovú funkciu:

1) Detekcia tváre: Problém je definovaný ako binárny klasifikačný problém, kde pre každú vzorku $x_i$nám model vráti pravdepodobnosť p_i, či daná vzorka je tvár alebo nie. Pre každú vzorku sme vypočítali krížovú entropiu:

$$L_i^{det} = -\left(y_i^{det}\log(p_i) + \left(1 - y_i^{det}\right)(1 - \log(p_i))\right),$$

kde $y_i^{det} \in \{0,1\}$ označuje skutočnosť, či daná vzorka je tvár alebo nie (1 – áno, 0 – nie).

2) Nájdenie ohraničujúcich rámcov: Nájdenie ohraničujúcich rámcov sme formulovali ako regresný problém. Pre každý nájdený rámec sme uvažovali vzdialenosť od najbližšieho skutočného rámca. Ako stratovú funkciu sme

definovali euklidovskú vzdialenosť týchto dvoch rámcov (presnejšie vzdialenosť ľavej strany, hornej strany, šírky a výšky dvoch rámcov). Stratovú funkciu teda vieme popísať ako:

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2$$

3) Nájdenie orientačných bodov: Tento regresný problém je definovaný podobne ako nájdenie ohraničujúcich rámcov, teda pre všetky nájdené orientačné body vypočítame ich euklidovskú vzdialenosť od skutočných orientačných bodov najbližšej tváre. Stratová funkcia vyzerá nasledovne:

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2$$

Výsledná stratová funkcia sa určí ako vážený súčet predošlých troch funkcií. V prípade sietí PNet, RNet môžeme uvažovať nulovú váhu 3). Presnosť detekcie sme vypočítali ako percento správne klasifikovaných vzoriek.

2.2.9 Tréning

Ako sme už spomenuli, samotný tréning prebiehal vo viacerých krokoch.

1) Najprv sme predspracovali dáta z datasetov WIDER Face, CNN Face Point.
2) Pripravili sme tréningové a validačné dáta pre sieť PNet pomocou scriptu *gen_Pnet_train_data.py*, nastavením parametra type na train/val.
3) Natrénovali sme sieť PNet.
4) Pripravili sme tréningové dáta pre sieť RNet pomocou scriptu *gen_Rnet_train_data.py*, nastavením parametra type na train/val.
5) Natrénovali sme sieť RNet.
6) Pripravili sme tréningové dáta pre sieť ONet pomocou scriptu *gen_Onet_train_data.py*, nastavením parametra type na train/val.
7) Natrénovali sme sieť ONet.

Pri tréningu sme používali batch size 32 (počet obrázkov v jednej dávke) pri všetkých sieťach, kým pri validácii bola hodnota nastavená na 64. Dáta sme načítavali pomocou *torch.utils.data.DataLoader*. Zoradili sme ich náhodne, pomocou parametra *shuffle,* aby sa zachovala variabilita obrázkov. Parameter *num_workers* sme nastavili na 8, aby sme pomocou paralelizmu zrýchlili načítavanie dát.

2.3 Orezávanie siete

V tejto časti uvedieme implementačné detaily samotného orezávania.

2.3.1 Orezanie vybraných filtrov

Orezávanie filtrov robí väčšina výskumníkov tromi spôsobmi:

1. Vymazané váhy nastavia na 0. Takto dané váhy nemajú žiadny vplyv na výsledok výpočtu. Pri zmazaní konvolučného filtra sa nastavia všetky parametre filtra na nulu. Aj keď tento spôsob je jednoduchý na implementáciu, praktický výpočet to nezrýchli.
2. Použijú pomocnú premennú, ktorá určí, či je daný parameter platný alebo nie. Tiež je to relatívne jednoduché na implementáciu, ale nastáva podobný problém ako pri prvom riešení.
3. Vytvorí sa nová sieť s orezanou štruktúrou, do ktorej sa prekopírujú váhy zo starej siete. Tento spôsob sme si zvolili aj my, keďže v tomto prípade sa počet výpočtov novej siete reálne zmenší, ako to bolo popísané v [9].

Zmazanie konvolučného filtra sa udeje vo viacerých krokoch. Najprv orežeme filtre z danej vrstvy, ktoré chceme odstrániť. Vytvoríme novú vrstvu, ktorá má rovnaký počet vstupných kanálov, ale počet výstupných kanálov je počet výstupných kanálov pôvodnej vrstvy mínus počet filtrov na zmazanie. Ostatné parametre zostanú nezmenené.

Teraz už stačí len správne prekopírovať váhy zo starej vrstvy do novej. Je to jednoduché, iba vynecháme tie, ktoré chceme zmazať pomocou príkazu *np.delete*.

Následne upravíme aj veľkosť PReLU vrstvy, ktorú upravíme podobným spôsobom ako konvolučnú vrstvu, z ktorej sme zmazali filter.

Následne určíme, či je nasledujúca vrstva konvolučná alebo plne prepojená. Na základe jej typu dopočítame, ktoré filtre/neuróny majú byť zmazané. Na určenie typu vrstvy sme použili príkaz *isinstance(layer, torch.nn.modules.conv.Conv2d)* pre nájdenie konvolučnej vrstvy a príkaz *isinstance(layer, torch.nn.Linear)* pre lineárnu vrstvu.

Pri konvolučnej vrstve musíme zmazať všetky konvolučné filtre, ktoré boli aplikované na zmazané výstupné mapy, presne podľa postupu popísaného v podkapitole

1.1. Pre lineárnu vrstvu sme zmazali rovnaké neuróny, aké boli aplikované na zmazané výstupy.

Treba poznamenať, že v našej implementácii je dôležité, aby sa jednotlivé parametre mazali v poradí podľa vrstiev (teda najprv zmažeme parametre z prvej, potom druhej vrstvy atď.).

2.3.2 Náhodný výber filtrov

Táto metrika bola najjednoduchšia na implementáciu. Každému filtru sme priradili náhodnú váhu pomocou funkcie *np.random.uniform*, ktorá vygeneruje náhodné číslo zo zadaného intervalu pomocou rovnomerného rozdelenia.

2.3.3 Minimálna l_2-norma

V tomto prípade sme každý konvolučný filter ohodnotili pomocou veľkosti jeho parametrov. Našťastie, knižnica Pytorch nám ponúka metódu *torch.norm*, ktorá vypočíta l_2-normu filtra.

2.3.4 Taylorov polynóm prvého rádu

Implementácia výpočtu ohodnotenia dôležitosti jednotlivých konvolučných filtrov pomocou Taylorovho polynómu prvého rádu využíva praktické výhody algoritmu spätnej propagácie, ktoré už boli spomenuté v podkapitole 1.3.5. Spomeňme si, že dôležitosť jednotlivých parametrov bola definovaná ako $\Theta_i = (g_i h_i)^2$, kde g_i je gradient a h_i je aktivačná hodnota i-teho parametra.

Využijeme funkciu *register_hook* knižnice *Pytorch*, ktorú použijeme na každý parameter, ktorý chceme potenciálne orezať. Táto funkcia sa zavolá pri každom výpočte gradientu, teda pri spätnej propagácii. Výhoda tohto prístupu je, že nemusíme zvlášť vypočítavať jednotlivé gradienty, čo zjednodušuje samotnú implementáciu a v neposlednom rade urýchľuje aj výpočet.

Pomocou tejto funkcie registrujeme metódu *compute_rank_taylor*, ktorá má na vstupe samotný gradient (grad). Aktivačnú hodnotu jednoducho získame z parametra *self.activation*. Z toho už jednoducho určíme hodnotu $(g_i h_i)^2$.

Toto sa udeje pre každý jeden tréningový vstup. Výslednú dôležitosť získame ako súčet jednotlivých dôležitostí. V tomto prípade parameter *batch_size* sme nastavili na hodnotu 32, podobne ako pri tréningu.

2.3.5 Normalizácia ohodnotení

Po vypočítaní ohodnotenia sme normalizovali ohodnotenie (*rank*) každého konvolučného filtra pomocou jeho l_2 normy filtrov, ktoré sú na danej vrstve. Tu sme využili štandardné príkazy *torch.abs* a *np.sqrt*.

2.3.6 Uloženie čiastkových modelov

Neurónové siete PNet, RNet, ONet sme orezávali v 9 iteráciách. V každej sme orezali 10 % parametrov, ktoré sa dali orezať bez modifikácie výstupov sietí. Po každej iterácii sme model po orezaní dotrénovali a váhy modelu sme uložili do súborov.

Súbory majú formát „*typ siete*“+*_pruned_* + „*percento parametrov*“+ *.pt*, kde typ siete môže byť *pnet, rnet, onet* a percento parametrov ukazuje, aké percento parametrov nám zostalo. Teda napr. *rnet_pruned_60.pt* obsahuje parametre orezanej siete *rnet*, ktorá obsahuje 60 % pôvodných parametrov.

Samotné súbory *.pt* obsahujú okrem parametrov aj štruktúru orezaných sietí. Treba dodať, že výstupy samotných sietí sme nezmenili. To je výhodné, lebo takto ich vieme hneď použiť namiesto pôvodných sietí *pnet, rnet, onet* bez toho, aby sme museli meniť ostatné časti zdrojových kódov.

Napríklad, ak chceme namiesto pôvodných sietí použiť naše orezané verzie, tak ich môžeme použiť aj v knižnici *facenet_pytorch*.

Vyzerá to nasledovne:

```
mtcnn = MTCNN()
pnet30 = torch.load(„pnet_pruned_30“)
rnet70 = torch.load(„rnet_pruned_70“)
mtcnn.pnet = pnet30
mtcnn.rnet = rnet70
```

Ako môžeme vidieť, zmenili sme sieť *pnet* na verziu len s 30 percentami svojich parametrov, sieť *rnet* na orezanú 70 percentnú verziu a pre sieť *onet* sme ponechali jej pôvodnú implementáciu.

Tento postup je výhodný, lebo vieme zvoliť najlepšiu kombináciu orezaní, ktorá má pre všetky tri podsiete najlepší pomer zrýchlenia a presnosti. Nemusíme zvoliť to isté percento pre všetky tri siete.

Keďže vieme tieto podsiete jednoducho vložiť namiesto pôvodných parametrov v danej knižnici, neprídeme o užitočné funkcionality, ktoré nám ponúka knižnica, napríklad vrátenie orezaného obrázka namiesto parametrov rámca tváre, možnosti vrátenia všetkých tvárí alebo len najlepšej tváre či dávkové spracovanie obrázkov. Ušetríme si veľa programovania, ktoré by sme museli spraviť, ak by sme zmenili formát výstupov sietí.

3 Výsledky orezávania

V tejto kapitole rozoberieme, aké výsledky sa nám podarilo dosiahnuť pomocou jednotlivých metód orezávania.

3.1 Všeobecná metodológia ohodnotenia výsledkov

Orezávanie siete MTCNN, ako už bolo spomenuté v predošlej kapitole, sme testovali pomocou datasetov WIDER Face a CNN Face Point. Každú podsieť sme orezávali v deviatich iteráciách. V každej iterácii sme odobrali 10 % z pôvodného počtu orezateľných parametrov. Takto sme získali 10 rôznych minimalizácií pôvodnej siete, z ktorých sme všetky otestovali na validačnej vzorke datasetu.

Zamerali sme sa na nasledujúce metriky, ktoré určujú presnosť siete:

1) Presnosť predikcie ohraničujúcich rámcov – najlepšie uchopiteľná chybová funkcia, ktorá popisuje, v koľkých percentách sa predikované ohraničujúce rámce zhodujú so skutočnými rámcami (presnejšie, v akom percente pokrývajú tú istú časť obrázka).
2) Celková strata – táto hodnota sa určila ako vážený priemer stratových funkcií 3, 4, 5.
3) Chyba detekcie tváre – veľkosť krížovej entropie medzi detegovanými a skutočnými tvárami.
4) Chyba ohraničujúcich rámcov – euklidovská vzdialenosť medzi predikovanými ohraničujúcimi rámcami tvárí a skutočnými rámcami.
5) Chyba charakteristických bodov tváre – euklidovská vzdialenosť medzi predikovanými charakteristickými bodmi tvárí a skutočnými bodmi

Presný popis výpočtu chybových funkcií uvádzame v podkapitole 2.2.8.

Pre každú metódu orezávania parametrov sme vypočítali presnosť predikcie hneď po orezaní a takisto sme overili presnosť modelu po dotrénovaní. Samotné dotrénovanie orezanej siete bolo vždy realizované v šiestich epochách. Následne pre lepšiu názornosť sme pre každú sieť vytvorili sumarizačný graf, ktorý porovnáva presnosť jednotlivých metód orezávania po dotrénovaní.

Dôvod, prečo sme sa zaoberali najmä prvou metrikou je jednoduchá interpretácia a fakt, že sieť MTCNN sa používa najmä na vytvorenie vstupu pre ďalšiu neurónovú sieť na klasifikáciu tvárí. Viac o tomto probléme a jeho praktických výsledkoch je

popísaných v kapitole 5. Výsledky ostatných metrík je možné nájsť v priložených súboroch.

3.2 Minimalizácia siete PNet

V tejto podkapitole uvedieme výsledky jednotlivých metód orezávania na sieti PNet.

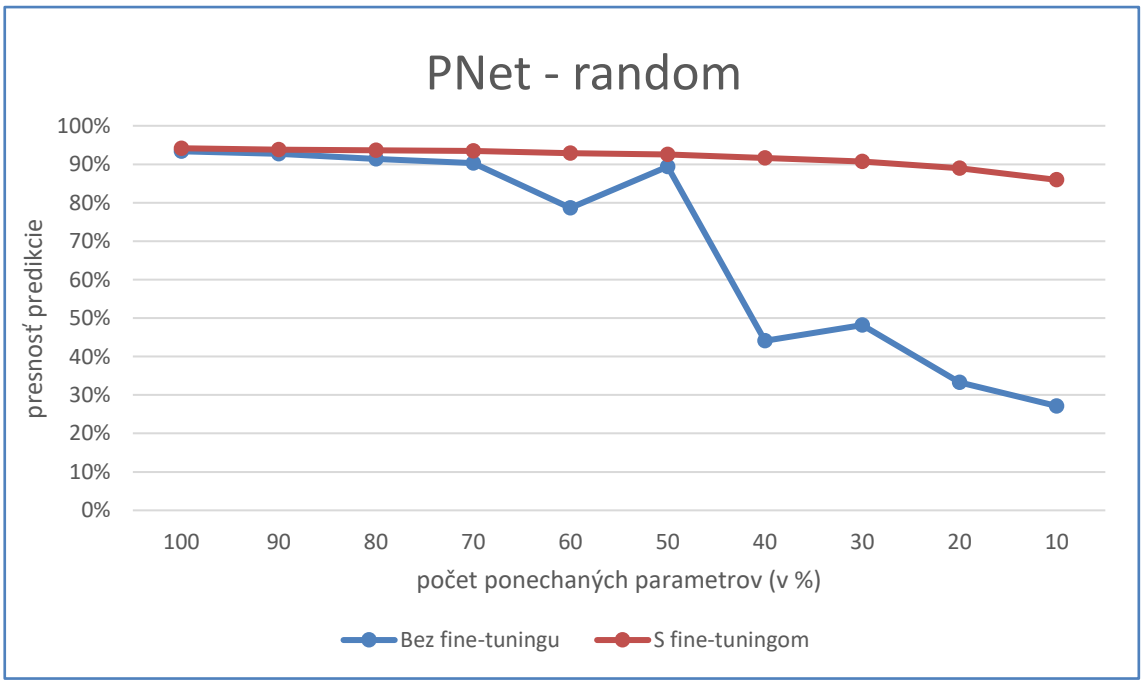
3.2.1 Náhodné orezávanie

Odstraňovanie konvolučných filtrov pomocou náhodného odoberania konvolučných filtrov prebiehalo, ako vyplýva z názvu podkapitoly, náhodne. V prvých iteráciách orezanie spôsobilo minimálne straty presnosti. Následne však v šiestej iterácii sa zrazu presnosť predikcie prepadla skoro o 50 %. Je zaujímavé, že napriek tomu, že orezané siete bez dotrénovania dávali slabé výsledky, vždy sa rýchlo dotrénovali na vysokú úspešnosť. Všetky výsledky sú zapísané v tabuľke nižšie:

Percento orezania	100	90	80	70	60	50	40	30	20	10
Bez fine-tuningu	0,9337	0,9273	0,9136	0,9031	0,7868	0,8937	0,4413	0,4817	0,3324	0,2714
S fine-tuningom	0,9416	0,9378	0,9364	0,9349	0,9288	0,9258	0,9166	0,9068	0,8895	0,8594

Tab. 1 Orezávanie PNet – algoritmus náhodného odoberania filtrov

Výsledky je možné vidieť aj v nasledujúcom grafe:



Obr. 4 Orezávanie PNet – algoritmus náhodného odoberania filtrov

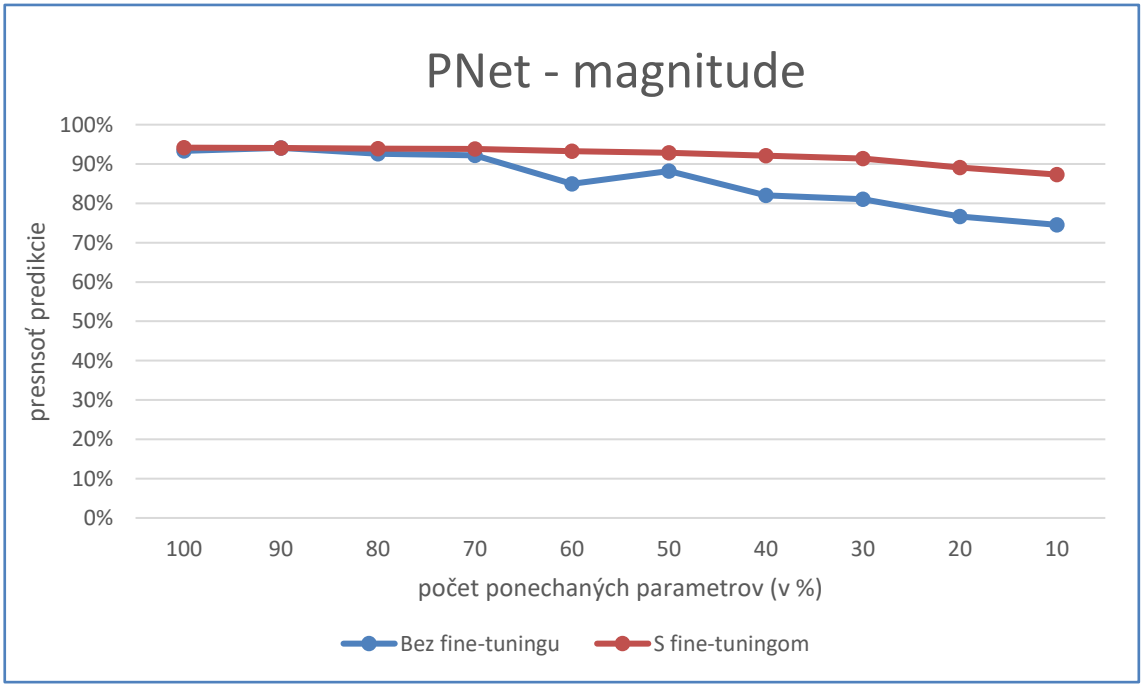
3.2.2 Minimálna l_2 nomra

Odstraňovanie konvolučných filtrov s minimálnou l_2 normou dávalo veľmi dobré výsledky. Dokonca aj neurónová sieť obsahujúca len 50 % orezateľných parametrov stratila na presnosti iba 1,3 % oproti pôvodnej sieti. Kompletné výsledky sú zhrnuté v tabuľke nižšie:

Percento orezania	100	90	80	70	60	50	40	30	20	10
Bez fine-tuningu	0,9337	0,941	0,926	0,9224	0,8492	0,8825	0,8205	0,8104	0,7666	0,7454
S fine-tuningom	0,9416	0,9411	0,9387	0,9382	0,9324	0,9288	0,9209	0,9136	0,8914	0,8732

Tab. 2 Orezávanie PNet – algoritmus minimálnych váh

Výsledky sme znázornili aj graficky:



Obr. 5 Orezávanie PNet – algoritmus minimálnych váh

3.2.3 Taylorov rozvoj

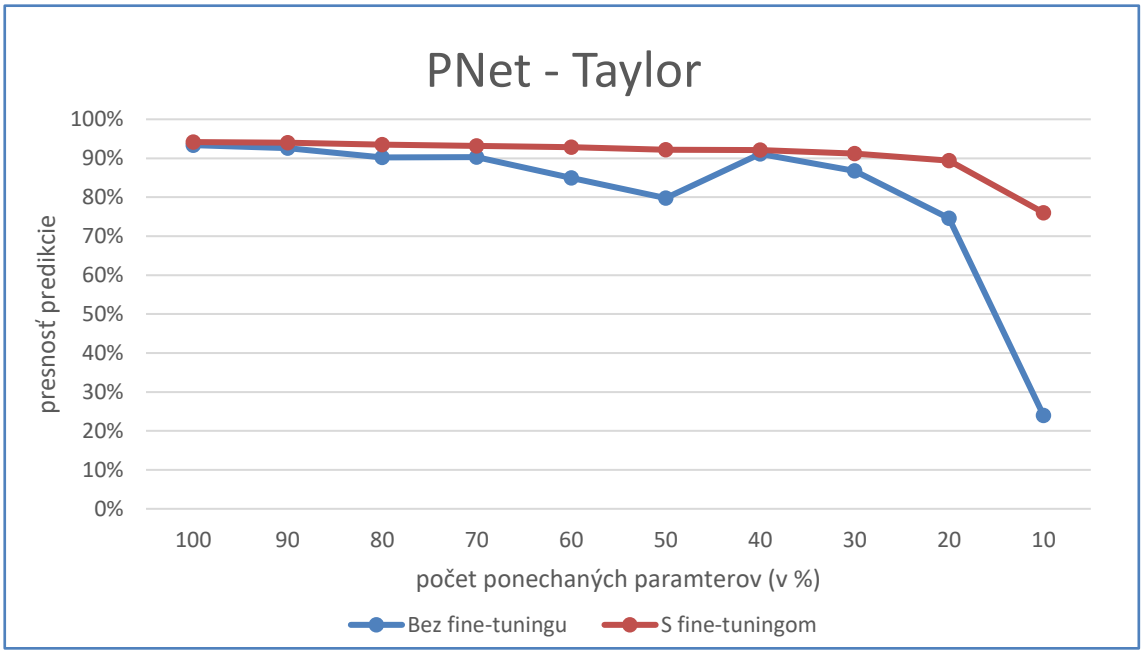
Odstraňovanie konvolučných filtrov pomocou aproximácie Taylorovho polynómu prvého rádu tiež dosahovalo dobré výsledky. Presnosť predikcie začala prudko klesať iba

pri 20 %, čo nám umožňuje výrazne zmenšiť pôvodnú neurónovú sieť. Kompletné výsledky sú zhrnuté v tabuľke:

Percento orezania	100	90	80	70	60	50	40	30	20	10
Bez fine-tuningu	0,9337	0,9258	0,902	0,9027	0,8498	0,798	0,9111	0,8673	0,7461	0,2397
S fine-tuningom	0,9416	0,9399	0,9354	0,9318	0,9284	0,9216	0,9207	0,9123	0,8941	0,7603

Tab. 3 Orezávanie PNet – algoritmus Taylorovho rozvoja

Výsledky sme znázornili aj pomocou grafu:



Obr. 6 Orezávanie PNet – algoritmus Taylorovho rozvoja

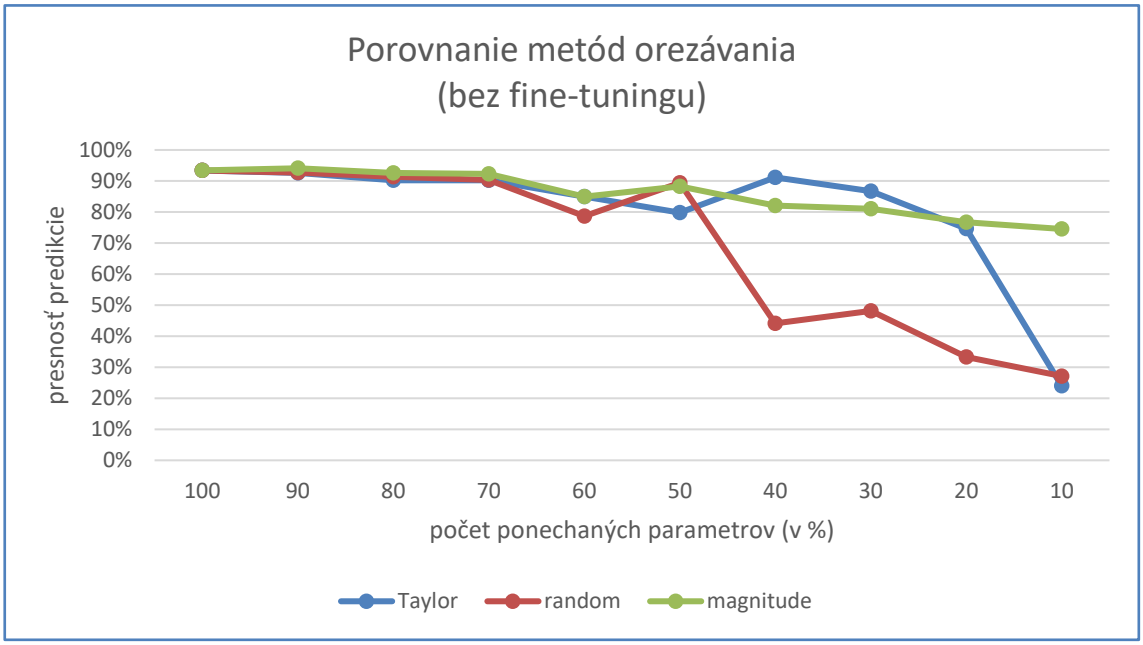
3.2.4 Sumarizácia výsledkov

Výsledky jednotlivých metód sme sumarizovali v dvoch tabuľkách. V prvej sú uvedené výsledky presnosti predikcie jednotlivých orezaných sietí bez dotrénovania (fine-tuningu), v druhej výsledky po dotrénovaní. Vidíme, že pre sieť PNet dáva najlepšie výsledky algoritmus minimálnych váh.

Nasledujúca tabuľka a graf znázorňuje sumarizáciu výsledkov bez dotrénovania (fine-tuningu). Ako vidíme, algoritmus minimálnych váh dáva najkonzistentnejšie výsledky, ale napr. algoritmus Taylorovho rozvoja je výrazne lepší okolo 30 – 40 % parametrov.

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	0,9337	0,9258	0,902	0,9027	0,8498	0,798	0,9111	0,8673	0,7461	0,2397
random	0,9337	0,9273	0,9136	0,9031	0,7868	0,8937	0,4413	0,4817	0,3324	0,2714
magnitude	0,9337	0,941	0,926	0,9224	0,8492	0,8825	0,8205	0,8104	0,7666	0,7454

Tab. 4 Orezávanie PNet – porovnanie metód orezávania (bez dotrénovania)



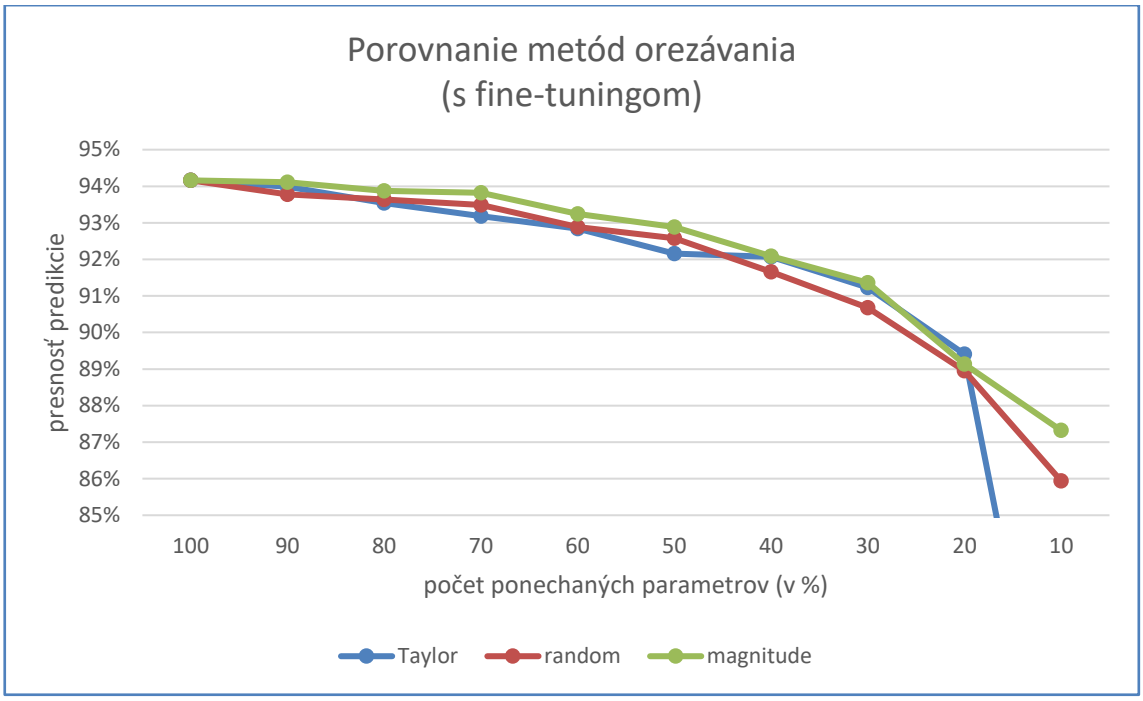
Obr. 7 Orezávanie PNet – porovnanie metód orezávania (bez dotrénovania)

Nasledujúca tabuľka a graf znázorňuje sumarizáciu výsledkov s dotrénovaním (s fine-tuningom):

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	0,9416	0,9399	0,9354	0,9318	0,9284	0,9216	0,9207	0,9123	0,8941	0,7603
random	0,9416	0,9378	0,9364	0,9349	0,9288	0,9258	0,9166	0,9068	0,8895	0,8594
magnitude	0,9416	0,9411	0,9387	0,9382	0,9324	0,9288	0,9209	0,9136	0,8914	0,8732

Tab. 5 Orezávanie PNet – porovnanie metód orezávania (s dotrénovaním)

V tomto prípade rozdiely vo výsledkoch už nie sú také veľké, ako pri porovnávaní bez dotrénovania. Môže za to najmä dotrénovanie, ktoré dokáže dotrénovať aj menej presné siete pomocou malého počtu iterácií na takmer pôvodnú presnosť.



Obr. 8 Orezávanie PNet – porovnanie metód orezávania (s dotrénovaním)

3.3 Minimalizácia siete RNet

V ďalšej podkapitole porovnávame presnosť predikcie orezaných sietí RNet, ktoré sme získali jednotlivými metódami orezávania popísanými v predošlej kapitole. Uvedieme už len sumarizáciu výsledkov, keďže výsledky sú veľmi podobné ako pri sieti PNet.

3.3.1 Sumarizácia výsledkov

Výsledky jednotlivých metód sme sumarizovali v dvoch tabuľkách. V prvej uvádzame výsledky presnosti predikcie jednotlivých orezaných sietí bez dotrénovania (fine-tuningu), v druhej výsledky po dotrénovaní.

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	0,958	0,9615	0,9645	0,9485	0,9157	0,9199	0,854	0,8896	0,7421	0,7623
random	0,958	0,9539	0,8529	0,9455	0,9099	0,9142	0,9179	0,7891	0,7985	0,7619
magnitude	0,958	0,9628	0,9459	0,945	0,9314	0,923	0,8759	0,8541	0,7863	0,7622

Tab. 6 Orezávanie RNet – porovnanie metód orezávania (bez dotrénovania)

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	0,9732	0,9716	0,9703	0,9694	0,9667	0,9656	0,9625	0,9566	0,9511	0,9266
random	0,9732	0,9718	0,9698	0,9685	0,966	0,9646	0,9617	0,9549	0,9492	0,9189
magnitude	0,9732	0,9721	0,9709	0,9704	0,968	0,9662	0,9635	0,9578	0,9472	0,8968

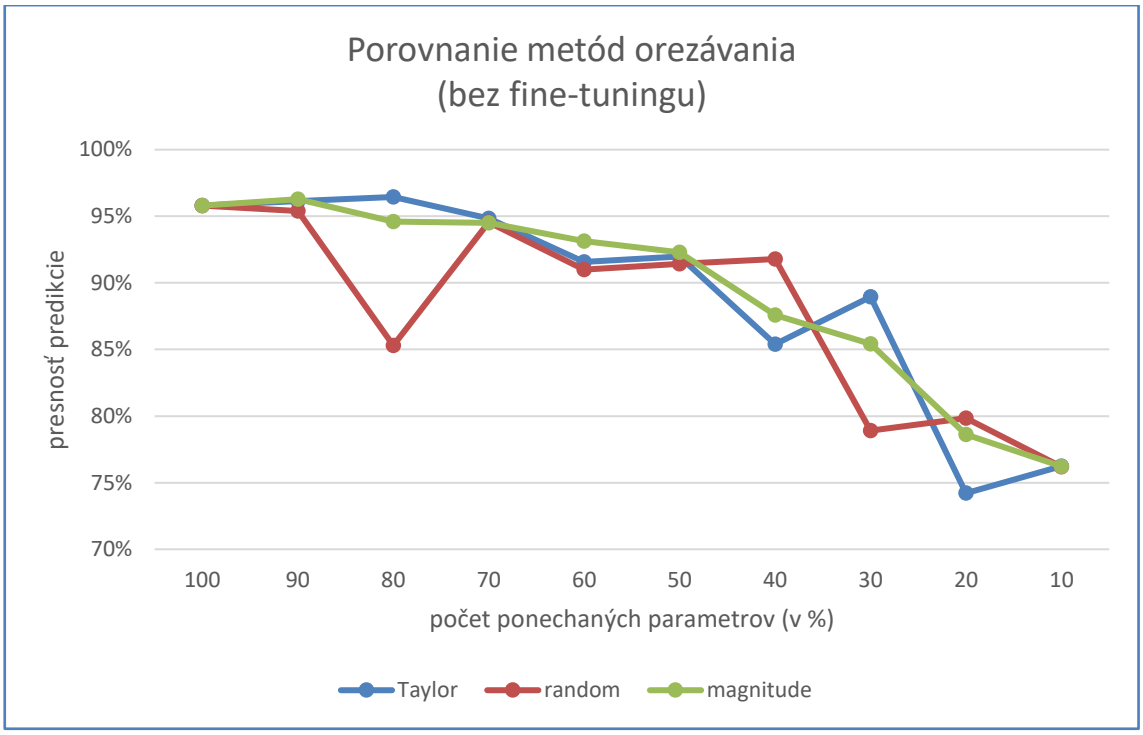
Tab. 7 Orezávanie RNet – porovnanie metód orezávania (s dotrénovaním)

Jednotlivé algoritmy dávali podobné výsledky ako pri sieti PNet. Najlepšie výsledky dávali striedavo algoritmus minimálnych váh a aproximácia pomocou Taylorovho rozvoja. Algoritmus náhodného odoberania sa správal najviac nepredvídateľne, ale v istých prípadoch dával najlepšie výsledky.

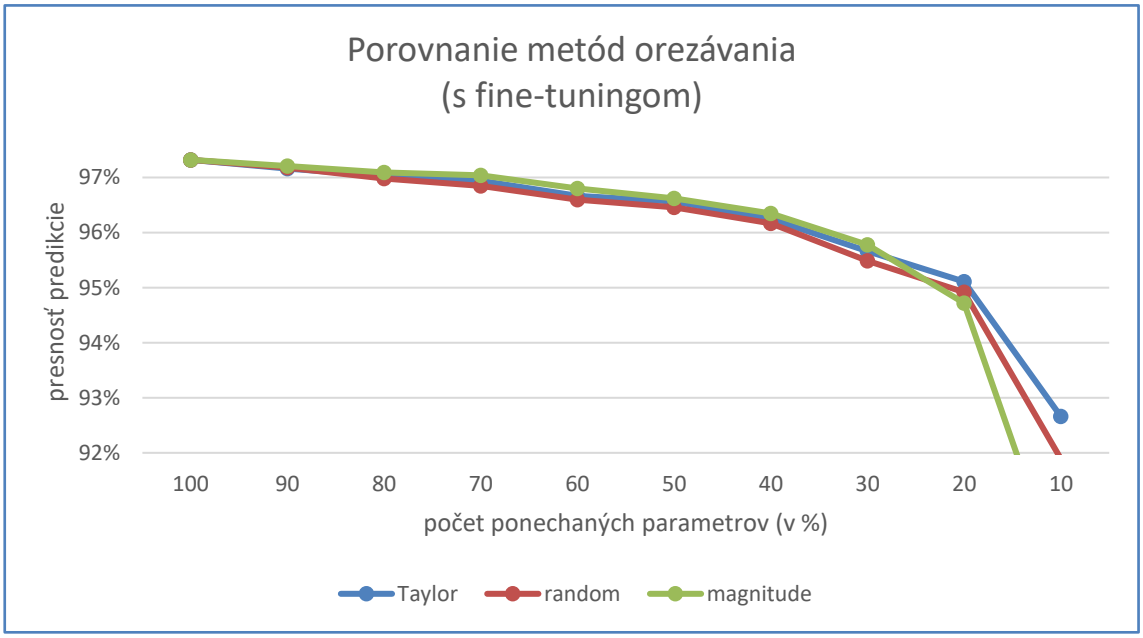
Je zaujímavé si uvedomiť, že poradie presnosti algoritmov pred a po dotrénovaní nie je rovnaké. Teda sa mohlo stať, že niektorý algoritmus mohol orezať sieť lepšie. Napriek tomu, optimalizátor nebol schopný dotrénovať sieť z toho stavu tak dobre, ako z menej presnej pozície.

Pre lepšiu pochopiteľnosť sa pozrime na výsledok orezávania pri 40 % parametrov. Pred dotrénovaním dával najlepší výsledok algoritmus *random*, ktorého presnosť predikcie bola 91,79 %. To je o viac ako 4 % viac ako presnosť predikcie algoritmu *magnitude*. Po dotrénovaní však algoritmus *magnitude* dosiahol lepšiu presnosť (96,35 %) ako algoritmus *random* (96,17 %).

Pre lepšiu názornosť sme dáta z tabuliek zobrazili aj graficky:



Obr. 9 Orezávanie RNet – porovnanie metód orezávania (bez dotrénovania)



Obr. 10 Orezávanie RNet – porovnanie metód orezávania (s dotrénovaním)

3.4 Minimalizácia siete ONet

V ďalšej podkapitole porovnávame presnosť predikcie orezaných sietí ONet, ktoré sme získali jednotlivými metódami orezávania popísanými v predošlej kapitole. Tiež uvedieme len sumarizáciu výsledkov.

3.4.1 Sumarizácia výsledkov

Výsledky jednotlivých metód sme sumarizovali v dvoch tabuľkách: v prvej sú výsledky presnosti predikcie jednotlivých orezaných sietí bez dotrénovania (fine-tuningu), v druhej výsledky po dotrénovaní.

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	0,938	0,9622	0,8153	0,9637	0,94	0,8948	0,9238	0,4759	0,4819	0,4488
random	0,938	0,9508	0,9229	0,8901	0,4487	0,9085	0,7926	0,8858	0,455	0,4486
magnitude	0,938	0,9699	0,9603	0,9504	0,9278	0,9468	0,9207	0,9021	0,7883	0,4486

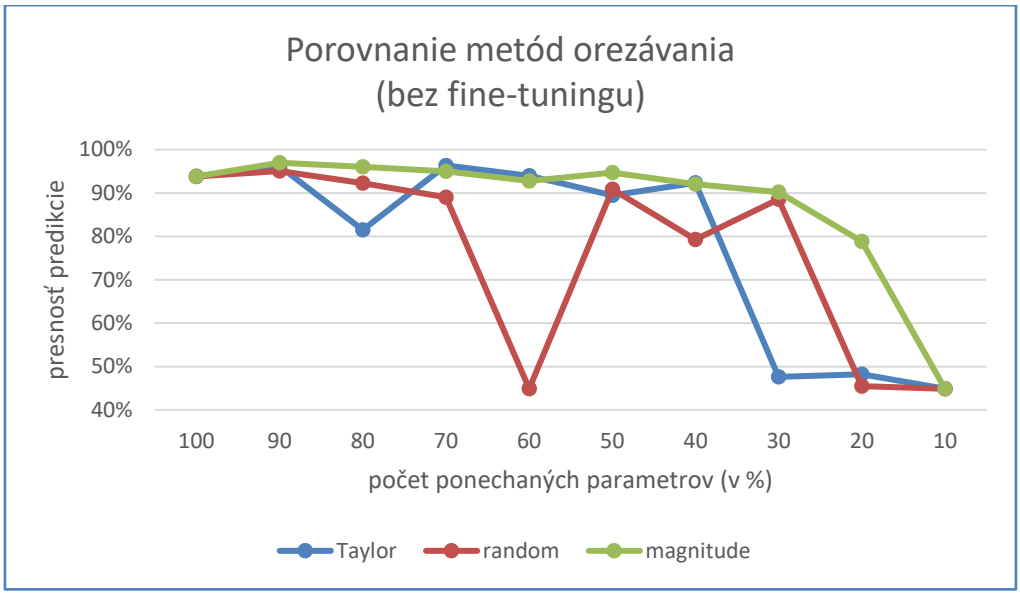
Tab. 8 Orezávanie ONet – porovnanie metód orezávania (bez dotrénovania)

Percento orezania	100	90	80	70	60	50	40	30	20	10
Taylor	97,34%	97,11%	97,08%	96,92%	96,88%	96,68%	96,23%	95,85%	94,87%	92,87%
random	97,34%	97,25%	96,99%	96,90%	96,71%	96,55%	96,17%	95,84%	94,84%	93,00%
magnitude	97,34%	97,25%	97,16%	96,82%	96,86%	96,65%	96,34%	95,86%	95,08%	90,85%

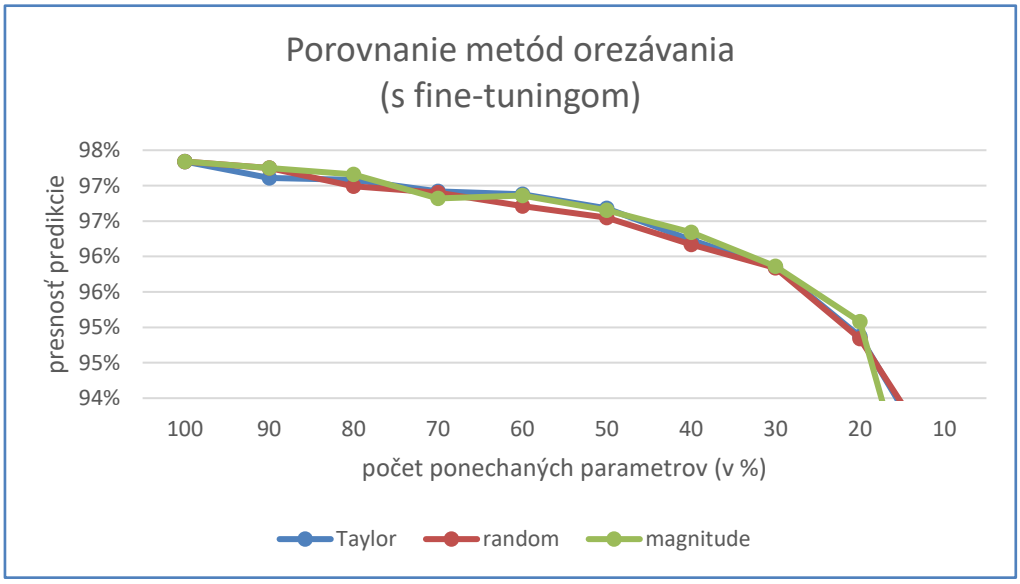
Tab. 9 Orezávanie ONet – porovnanie metód orezávania (s dotrénovaním)

Po dotrénovaní dávali všetky tri algoritmy veľmi podobné výsledky. Pred dotrénovaním dával najlepšie priemerné výsledky algoritmus minimálnych váh. Pri orezávaní siete na malý počet parametrov (10 – 20 %) všetky algoritmy rapídne strácajú na presnosti, čo je pravdepodobne spôsobené komplexnejšou úlohou siete ONet, ktorá okrem rámcov tvárí deteguje aj charakteristické body tváre.

Pre lepšiu názornosť sme dáta z tabuliek zobrazili aj graficky:



Obr. 11 Orezávanie ONet – porovnanie metód orezávania (bez dotrénovania)



Obr. 12 Orezávanie ONet – porovnanie metód orezávania (s dotrénovaním)

4 Návrh výberu orezaných sietí

V podkapitolách 3.2, 3.3 a 3.4 sme vyhodnotili presnosť jednotlivých algoritmov orezávania. Neodpovedali sme však na asi najdôležitejšiu otázku: ktorú z orezaných sietí máme použiť pri predikcii v reálnej aplikácii?

Na túto otázku neexistuje jednoznačná odpoveď. Skúsime však priblížiť niekoľko možných prístupov, ako vybrať možnosť, ktorá nám najviac vyhovuje. Táto úvaha je o to zaujímavejšia, že v iných publikáciách sa nikde nevyskytuje odporúčanie autorov napr. k tomu, do akého pomeru je vhodné orezávať danú neurónovú sieť nejakým algoritmom, popisujú iba výsledky presnosti predikcie. Nejednoznačnosť odpovede je daná najmä tým, že sa sami musíme rozhodnúť, či nám viac záleží na presnosti predikcie siete alebo na jej rýchlosti.

Označme S pôvodnú neurónovú sieť s plným počtom parametrov. (V našom konkrétnom príklade môže S označovať jednu z trojice PNet, RNet, ONet.) Nech $S_{i,a}$ označuje i-percentnú podsieť pôvodnej siete, teda orezanú sieť, ktorá obsahuje i % pôvodných parametrov a ktorá bola získaná pomocou orezávacieho algoritmu a. Presnosť predikcie modelu X označme $P(X)$.

Definovali sme viacero kritérií výberu vhodných podsietí:

4.1 Maximálny pokles predikcie

V tomto prípade treba určiť, aký najväčší pokles predikcie vieme ešte tolerovať. Potom sa už stačí iba pozrieť na presnosť predikcie jednotlivých podsietí a vybrať tú s najmenším počtom parametrov. Pri rovnosti počtu parametrov (pre rôzne algoritmy výberu filtrov na orezanie) vyberieme ten algoritmus, teda tú sieť, ktorá dáva presnejšiu predikciu.

Zvolili sme si 3 rôzne hodnoty maximálneho poklesu predikcie: 0,5 %, 1 %, 2 %. Pre každú z týchto hodnôt sme pre všetky algoritmy orezávania určili sieť s najmenším počtom parametrov, ktorá spĺňa podmienku, že jej presnosť predikcie je aspoň (pôvodná presnosť predikcie – maximálny pokles).

Pre každý typ siete sme výsledky znázornili v tabuľke. Červenou farbou je znázornený „výherca“, teda najlepší algoritmus pre daný pokles:

Maximálny pokles pedikcie	0,50%	1%	2%
Taylor	90	70	50
random	90	70	50
magnitude	70	60	50

Tab. 10 Maximálny pokles predikcie – PNet

Maximálny pokles pedikcie	0,50%	1%	2%
Taylor	70	50	30
random	70	50	30
magnitude	70	40	30

Tab. 11 Maximálny pokles predikcie – RNet

Maximálny pokles pedikcie	0,50%	1%	2%
Taylor	60	50	30
random	70	50	30
magnitude	60	40	30

Tab. 12 Maximálny pokles predikcie – ONet

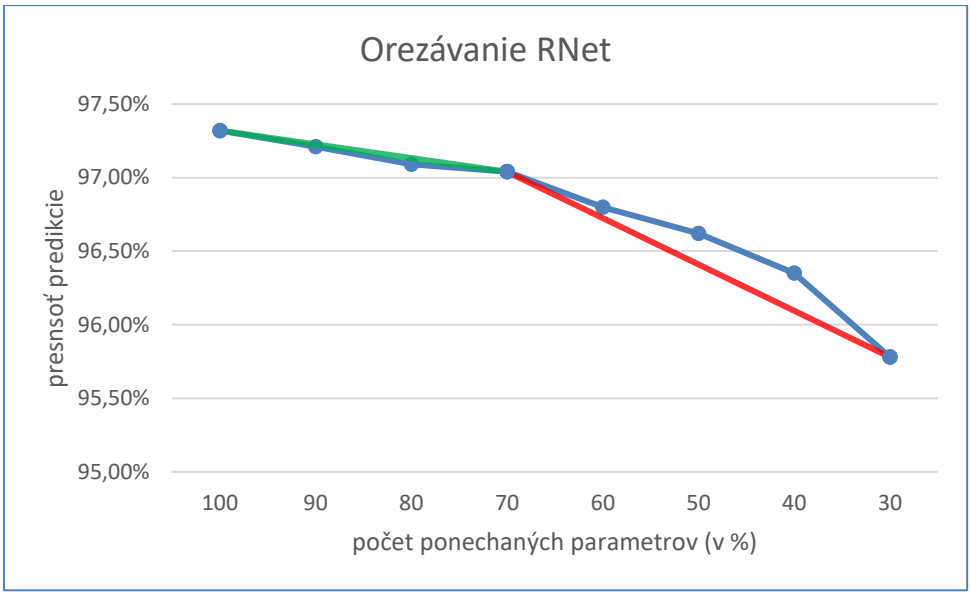
Vidíme, že v tomto prípade sa algoritmus minimálnych váh javí ako najlepší, keďže „vyhral“ 9-krát z 10.

4.2 Minimálne zrýchlenie

Tento algoritmus je veľmi jednoduchý, stačí vybrať podsieť s daným počtom parametrov, ktorá dosahuje najväčšiu presnosť. Napr. ak chceme minimálne 30 % zrýchlenie pre sieť RNet, tak len stačí vybrať najpresnejšiu podsieť $S_{70,a}$ z orezaných sietí pre rôzne a, ktorá spĺňa toto kritérium.

4.3 Relatívna strata presnosti

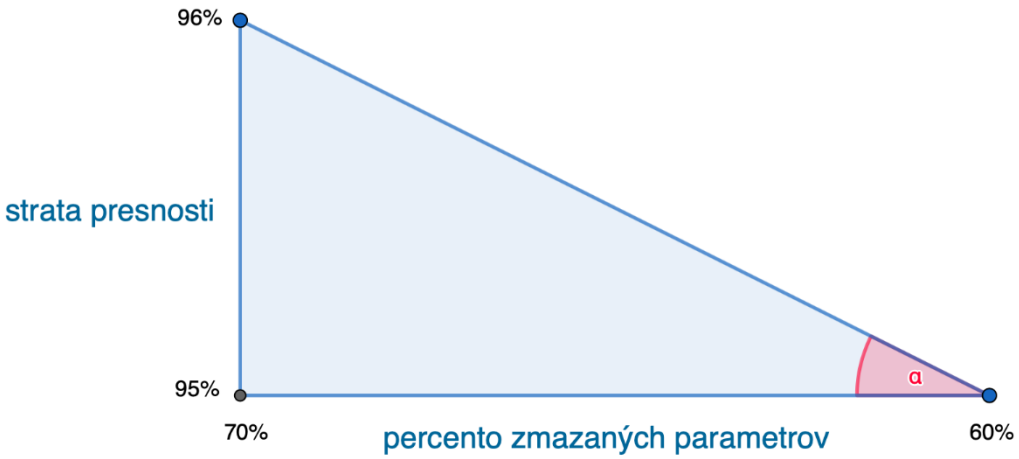
Ak sa pozrieme na graf, ktorý je podgraf grafu z podkapitoly 3.3.1, tak vidíme, že napr. pri použití algoritmu minimálnych váh je výber siete so 70-timi parametrami dobrou voľbou, keďže zmena v presnosti je minimálna a zrýchlime výpočet (vyznačené zelenou farbou).



Tab. 13 Orezávanie siete RNet – algoritmus minimálnych váh

Napríklad voľba siete 60, 50 už nie je taká dobrá (označené červenou farbou), pretože krivka presnosti začne klesať prudšie. To nám dáva nápad, aby sme definovali relatívnu stratu presnosti. Základná myšlienka spočíva v tom, že dovtedy budeme orezávať parametre, teda zrýchľovať sieť, kým presnosť predikcie príliš nepoklesne.

Tento problém je možné definovať aj matematicky. Chceme zistiť strmosť krivky poklesu presnosti predikcie, ktorá je definovaná uhlom α sklonu krivky. Veľkosť tohto uhla je definovaná pomerom $k = \frac{strata\ presnosti}{percento\ zmazaných\ parametrov}$, čiže tangensom uhla α.



Obr. 13 Relatívna strata presnosti

Stačí definovať maximálnu hodnotu k_{max}, pomocou ktorej vieme regulovať hodnotu dôležitosti zrýchlenia na úkor presnosti predikcie. Teda zvolením väčšieho k pripúšťame väčší pokles presnosti predikcie, záleží nám viac na zrýchlení siete.

V ďalšom postupe kvôli prehľadnosti zanedbáme parameter a algoritmu orezávania. Označme $S_{i_1}, S_{i_2}, \dots, S_{i_l}$, kde $i_1 \geq i_2 \geq \cdots \geq i_l$, postupnosť podsietí, ktoré sme získali jednotlivými iteráciami orezávania. V našom prípade je to S_{90}, S_{80}, ... S_{10}. S_{i_0} nech označuje S, teda pôvodnú sieť so všetkými parametrami.

Na určenie najvhodnejšej orezanej siete pomocou relatívnej straty presnosti existujú dve stratégie:

a) Lokálna strata

V tomto prípade sa sleduje lokálna relatívna strata presnosti v každej iterácii orezávania, teda porovnáva sa presnosť predikcie s presnosťou podsiete z predošlej iterácie. Matematicky sledujeme $k_j = \frac{P(S_{i_{j-1}}) - P(S_{i_j})}{i_{j-1} - i_j}$. Začíname s $j = 1$ a kým $k_j \leq k_{max}, j = j + 1$.

b) Globálna strata

Sledujeme globálnu relatívnu stratu presnosti v každej iterácii orezávania, teda relatívnu stratu oproti predikcii pôvodnej siete. Matematicky sledujeme $k_j = \frac{P(S) - P(S_{i_j})}{100 - i_j}$. Začíname s $j = 1$ a kým $k_j \leq k_{max}, j = j + 1$.

Je jednoduché uvedomiť si, že pomocou lokálnej straty vyberieme aspoň takú veľkú podsieť, ako pomocou globálnej straty. (Globálna strata toleruje lokálne aj väčší relatívny pokles, ak relatívny pokles bol predtým veľmi malý).

Globálnu stratu môžeme zmeniť aj spôsobom, že ju definujeme ako najväčšie j také, že platí $k_j \leq k_{max}$. Teda v tomto prípade môže existovať také $i < j$, pre ktoré $k_i > k_{max}$, ale potom nasleduje niekoľko iterácií s menšou relatívnou stratou ako k. Takto by sme dostali ešte menšie podsiete pre dané k ako v prípade globálnej straty.

4.4 Iné metriky na meranie relatívnej straty

V podkapitole 4.3 sme definovali relatívnu stratu presnosti ako podiel straty presnosti a percenta zmazaných parametrov. Pod parametrami rozumieme v tomto prípade jednotlivé konvolučné filtre, keďže práve tie orezávame.

Všeobecne môže byť táto metrika popísaná ako $\frac{strata\ presnosti}{zlepšenie\ modelu}$, kde pod zlepšením máme na mysli zmenšenie veľkosti, zrýchlenie výpočtu, atď.

Pri určovaní *zlepšenia modelu* percento orezaných filtrov (z orezateľných parametrov) nemusí byť najlepšia metrika, ktorú môžeme použiť. Preto navrhujeme aj ďalšie možnosti, ako merať *zlepšenie modelu*. Pri každej možnosti uvedieme pre a proti daného spôsobu.

4.4.1 Zrýchlenie výpočtu

Jednou z možností ako pozorovať *zlepšenie modelu* je zmeranie času výpočtu pomocou knižnice *time* v Pythone. Jednoducho vypočítame časový rozdiel medzi koncom a začiatkom výpočtu.

Tento prístup je veľmi jednoduchý a popisuje reálne časové zrýchlenie, ktoré sme získali. V skutočnosti je veľmi ťažké zabezpečiť, aby sme získali nezávislý výsledok, ktorý nie je ovplyvnený inými procesmi na počítači. Preto tento spôsob nemusí dávať presné výsledky, čo sa potvrdilo aj v našich testoch. Napriek stálemu zaťaženiu systému (bez iných spustených programov) bolo vyhodnotenie menších modelov niekedy pomalšie ako siete s väčším počtom parametrov.

4.4.2 Skutočný počet parametrov

V podkapitole 4.3 sme ako parametre počítali jednotlivé filtre. Vieme však, že tieto filtre nie sú rovnako veľké. Niektoré obsahujú väčší počet váh (teda parametrov), čím prirodzene rastie aj ich výpočtová zložitosť.

Preto je možné parameter *zlepšenie modelu* nahradiť percentom zmazaných váh. Tým získame presnejší pohľad na to, aké reálne zrýchlenie môžeme získať. Použili sme knižnicu *thop* a jej metódu *profile* na určenie parametrov jednotlivých orezaných sietí.

Problémom tohto prístupu je, že skutočné zrýchlenie závisí aj od iných faktorov, najmä od paralelizácie výpočtov a počtu výpočtov na jednotlivých vrstvách siete.

4.4.3 Počet vykonaných operácií

Poslednou, v publikáciách často využívanou metrikou je počítanie vykonaných operácií, ktoré vykoná daná neurónová sieť pri výpočte. Používajú sa na to dve metriky, počet operácií s desatinnou čiarkou (FLOP – floating point operation) alebo počet operácií vynásob-akumuluj (MAC – multiply-accumulate operation), medzi ktorými platí stav 1 MAC ≅ 2 FLOP.

Obe metriky je možné vypočítať buď teoreticky, pomocou vzorcov spomínaných v podkapitole 1.1, alebo pomocou knižníc. My sme zvolili druhú možnosť a využili sme metódu *profile* z knižnice *thop*.

Výhodou tejto metriky je, že vypočítame presný počet operácií, ktoré vykoná daný model. Vďaka tomu je táto metrika jednou z najvhodnejších na určenie *zlepšenia modelu*. Ani táto metrika však nedáva úplné riešenie, keďže niektoré operácie sa môžu vykonať paralelne, iné len sekvenčne, podľa charakteru siete, použitého hardvéru a ďalších faktorov.

4.5 Vyhodnotenie relatívnej straty

4.5.1 Výsledky na dátach

Všetky metódy uvedené v podkapitolách 4.4.1, 4.4.2 a 4.4.3 sme vyhodnotili pre náš dataset a neurónové siete PNet, RNet, ONet, pre jednotlivé metódy orezávania popísané v kapitole 2. Vytvorili sme skript PruningTimeTest.py, ktorý pre každú orezanú podsieť určí 4 metriky: percento orezaných filtrov, počet operácií, čas potrebný na výpočet a počet skutočne orezaných parametrov.

Kvôli krátkosti uvedieme len výsledky na sieti RNet metódou aproximácie pomocou Taylorovho polynómu prvého rádu. Sieť RNet má 99910 parametrov, z ktorých je cca. 75000 v posledných dvoch lineárnych vrstvách, ktoré nemajú skoro žiaden vplyv na rýchlosť výpočtu siete. Pridali sme ešte jednu metriku, počet skutočne orezaných parametrov okrem lineárnych vrstiev (v tabuľke označované ako Počet parametrov - NL), keďže zameranie výlučne na „konvolučné“ parametre dáva presnejší predpoklad o zrýchlení siete ako pohľad na všetky parametre.

Výsledky jednotlivých podsietí a metrík sme zhrnuli v nasledujúcej tabuľke:

Percento orezania	100	90	80	70	60	50	40	30	20	10
1. Percento orezaných filtrov	100	90	80	70	60	50	40	30	20	10

2. Počet operácií	2,23E+11	1,87E+11	1,55E+11	1,28E+11	1,04E+11	8,17E+10	6,08E+10	4,07E+10	3,13E+10	1,93E+10
3. Testovací čas (s)	84,83	80,7	69,82	66,75	57,97	45,42	44,44	34,72	30,6	21,63
4. Počet parametrov	99910	96120	92600	89778	86650	84278	81766	79690	77661	75914
5. Počet parametrov - NL	24910	21120	17600	14778	11650	9278	6766	4690	2661	914
Presnosť modelu	97,32%	97,16%	97,03%	96,94%	96,67%	96,56%	96,25%	95,66%	95,11%	92,66%

Tab. 14 Porovnanie metrík – zlepšenie siete

Aby sme vedeli porovnať jednotlivé metriky, normalizovali sme ich podľa „veľkosti“ pôvodného modelu a vypočítali sme zlepšenie oproti pôvodnému modelu v percentuálnom tvare. Teda ak označíme pre nejakú metriku skóre pôvodného modelu m_0, skóre orezaného modelu m_1, tak sme vypočítali normalizované *zlepšenie siete* $\frac{m_0 - m_1}{m_0}$. Výsledky môžeme vidieť v nasledujúcej tabuľke:

Percento orezania	100	90	80	70	60	50	40	30	20	10
1. Percento orezaných filtrov	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
2. Počet operácií	0%	16%	31%	43%	54%	63%	73%	82%	86%	91%
3. Testovací čas	0%	5%	18%	21%	32%	46%	48%	59%	64%	75%
4. Počet parametrov	0%	4%	7%	10%	13%	16%	18%	20%	22%	24%
5. Počet parametrov - NL	0%	15%	29%	41%	53%	63%	73%	81%	89%	96%

Tab. 15 Porovnanie metrík – normalizované zlepšenie siete

Pre každú metriku sme vypočítali lokálnu relatívnu stratu, teda koeficient *k*:

Percento orezania	100	90	80	70	60	50	40	30	20	10
1. Percento orezaných filtrov	-	0,016	0,013	0,009	0,027	0,011	0,031	0,059	0,055	0,245
2. Počet operácií	-	0,010	0,009	0,007	0,025	0,011	0,033	0,065	0,131	0,456
3. Testovací čas	-	0,033	0,010	0,025	0,026	0,007	0,268	0,051	0,113	0,232
4. Počet parametrov	-	0,042	0,037	0,032	0,086	0,046	0,123	0,284	0,271	1,401
5. Počet parametrov - NL	-	0,011	0,009	0,008	0,022	0,012	0,031	0,071	0,068	0,349

Tab. 16 Porovnanie metrík - lokálna relatívna strata

Taktiež sme pre každú metriku vypočítali aj globálnu relatívnu stratu, teda koeficient *k*:

Percento orezania	100	90	80	70	60	50	40	30	20	10

1. Percento orezaných filtrov	-	0,016	0,014	0,013	0,016	0,015	0,018	0,024	0,028	0,052
2. Počet operácií	-	0,010	0,009	0,009	0,012	0,012	0,015	0,020	0,026	0,051
3. Testovací čas	-	0,033	0,016	0,018	0,021	0,016	0,022	0,028	0,035	0,063
4. Počet parametrov	-	0,042	0,040	0,037	0,049	0,049	0,059	0,082	0,099	0,194
5. Počet parametrov - NL	-	0,011	0,010	0,009	0,012	0,012	0,015	0,020	0,025	0,048

Tab. 17 Porovnanie metrík - globálna relatívna strata

Vidno, že napr. použitie metriky 2, teda počtu operácií, ktoré sieť vykoná, je takmer totožná s metrikou 5 počet parametrov (bez lineárnych vrstiev). Avšak napríklad porovnať metriku 2 a 3 už je oveľa ťažšie, keďže koeficienty *k* v metrike 3 sú násobne vyššie ako pri metrike 2. Preto pre porovnanie jednotlivých metrík musíme použiť štatistickú metódu.

4.5.2 Porovnanie metrík *zlepšenia siete*

Na porovnanie jednotlivých metrík sme použili Pearsonov korelačný koeficient, ktorý sme určili pre každú dvojicu metrík. Výsledky sme zhrnuli do tabuliek.

Porovnanie metrík pre normalizované *zlepšenie siete*:

Korelácia zlepšenia siete	*1.*	*2.*	*3.*	*4.*	*5.*
1.	1,00000				
2.	0,98625	1,00000			
3.	0,99488	0,98324	1,00000		
4.	0,99328	0,99847	0,98940	1,00000	
5.	0,99328	0,99847	0,98940	1,00000	1,00000

Tab. 18 Korelácia zlepšenia siete pre rôzne metriky

Porovnanie metrík pre lokálnu relatívnu stratu *k*:

Korelácia lokálneho *k*	*1.*	*2.*	*3.*	*4.*	*5.*
1.	1,00000				
2.	0,99098	1,00000			
3.	0,61356	0,61065	1,00000		
4.	0,99912	0,99215	0,60400	1,00000	
5.	0,99912	0,99215	0,60400	1,00000	1,00000

Tab. 19 Korelácia lokálneho k pre rôzne metriky

Porovnanie metrík pre globálnu stratu *k*:

Korelácia globálneho *k*	*1.*	*2.*	*3.*	*4.*	*5.*
1.	1,00000				
2.	0,99744	1,00000			
3.	0,95045	0,93277	1,00000		
4.	0,99782	0,99985	0,93413	1,00000	
5.	0,99782	0,99985	0,93413	1,00000	1,00000

Tab. 20 Korelácia globálneho k pre rôzne metriky

Na základe výsledkov korelácie môžeme povedať, že pri normalizovanom *zlepšení siete* a pri globálnej strate je vysoká korelácia medzi ľubovoľnou dvojicou metrík. Metódy 4, 5 majú rovnakú koreláciu, keďže sú len posunuté o konštantu. Pri lokálnej strate môžeme vidieť, že metrika 3 slabšie koreluje s ostatnými metrikami, ale zvyšné metriky sú navzájom podobné.

Všeobecne sa dá konštatovať, že hodnoty koeficientu *k* veľmi silno korelujú pre skoro všetky dvojice rôznych metrík. Pre nás to znamená, že je takmer jedno, ktorú metriku použijeme, výsledky budú veľmi podobné.

Samozrejme, treba si uvedomiť, že pri zvolení rôznych metrík je potrebné zvoliť rôznu prahovú hodnotu k_{max}, aby sme dosiahli podobné výsledky, ale pri zvolení vhodného k_{max} budú výsledky prakticky totožné bez ohľadu na zvolenú metriku.

5 Rozpoznávanie tvárí v reálnom čase

Rozpoznávanie tvárí je veľmi populárny problém v oblasti strojového učenia. Má veľký potenciál na využitie v praktických aplikáciách, od hľadania zločincov cez bezpečnostné systémy budov a odblokovanie smartphonu až po vyhľadávanie osôb na fotke na sociálnych sieťach.

V tejto kapitole opíšeme návrh systému na rozpoznávanie tvárí konkrétnych ľudí na obrázkoch. Pomocou neho vyhodnotíme úspešnosť orezávaných sietí MTCNN v situácii z reálneho života.

5.1 Rozpoznávanie tvárí

Problém rozpoznávania tvárí znamená identifikáciu konkrétnej osoby na fotke a nie len nájdenie tváre na fotke, ktorá sa nazýva detekcia tvárí. Všeobecný pojem rozpoznávanie tvárí sa používa na viacero druhov problémov ako napr.:

1. Na vstupe dostaneme dva obrázky a máme rozhodnúť, či sa na nich nachádza tá istá osoba.
2. Na vstupe je fotka osoby a máme rozhodnúť, či je to daná osoba. Toto sa dá riešiť napr. porovnaním vstupného obrázka so všetkými obrázkami danej osoby.
3. Na vstupe je fotka osoby a máme rozhodnúť, ktorá osoba (alebo žiadna) sa na nej nachádza z danej databázy ľudí.

My sme sa zaoberali tretím problémom, teda sme sa snažili klasifikovať obrázky na základe osôb, ktoré sa na nich nachádzajú. Takto sme mohli otestovať presnosť predikcie aj zmenu v rýchlosti detekcie tvárí orezaných modelov MTCNN na úplne inom datasete aj pre iný, komplexnejší druh problému.

Tento spôsob testovania je prirodzený, keďže detekcia tvárí sa v praktických aplikáciách primárne využíva ako vstup pre klasifikáciu tvárí. Preto sme skúmali, ako orezávanie vplýva na presnosť predikcie pri rozpoznávaní.

5.1.1 Príklad

Pre lepšie pochopenie si uvedieme príklad, prečo nám nezáleží len na presnosti detekcie tváre, ale najmä na presnosti celého systému rozpoznávania tvárí.

Predpokladajme, že naša sieť MTCNN mala presnosť 95 % pri detekcii tvárí, a celý systém (teda presnosť klasifikácie, kto je na danom obrázku) mal presnosť 97 %.

Uvažujme orezanú sieť MTCNN, ktorá je dvakrát rýchlejšia a má predikciu 91 %. Ak presnosť klasifikácie celého systému klesne na 96,5 %, tak je zmysluplné zamyslieť sa na používaní orezanej siete. Ak ale presnosť klasifikácie celého systému klesne napr. na 86 %, tak je lepšie použiť pôvodnú verziu siete. Z toho vidno, že nás viac zaujíma presnosť konečnej klasifikácie ako presnosť čiastočných výsledkov.

5.2 Návrh systému

Na základe odporúčaní [6] a po porovnaní najlepších súčasných modelov (ktoré dosahujú tzv. „state of the art“ výsledky) na rozpoznávanie tvárí sme navrhli systém, ktorý pracuje v troch častiach:

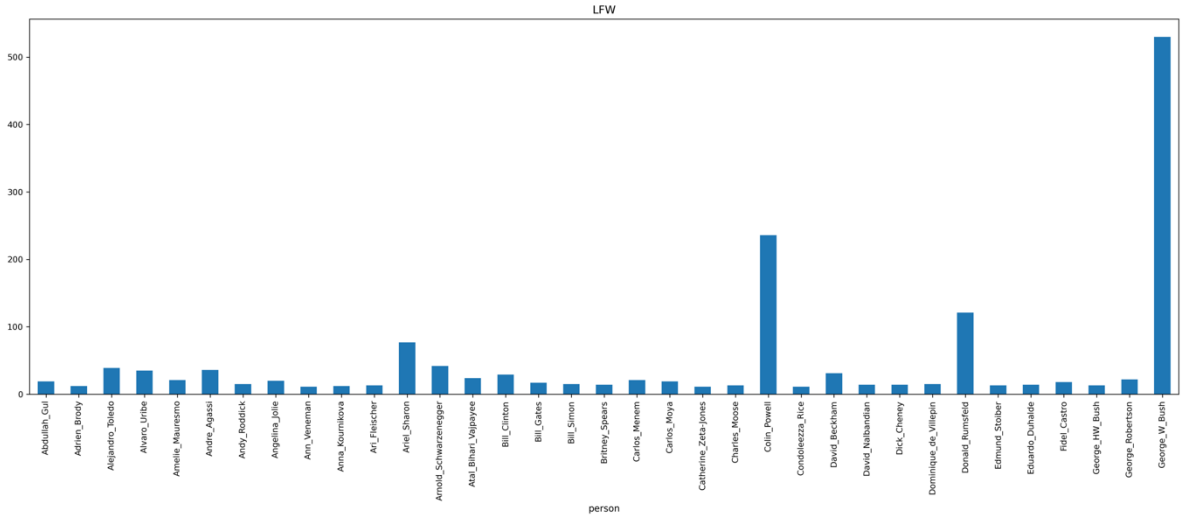
1. Najprv sa určí alebo určia tváre na danom obrázku alebo obrázku z videa pomocou MTCNN siete.
2. Obrázky sa normalizujú a dajú sa na vstup hlbokej neurónovej sieti, ktorá na výstupe vytvorí vektor (tzv. embedding) prislúchajúci danej tvári. Pre tento výstupný vektor by malo platiť, že jeho vzdialenosť od vektorov tvárí tej istej osoby (napr. z inej fotky) by mala byť čím menšia. Naopak, vzdialenosť vektora od vektorov tvárí iných osôb by mala byť čím väčšia.
3. Výstupný vektor sa porovná s tvárami (presnejšie vektormi tvárí) osôb v databáze, čím sa určí, o ktorú osobu ide.

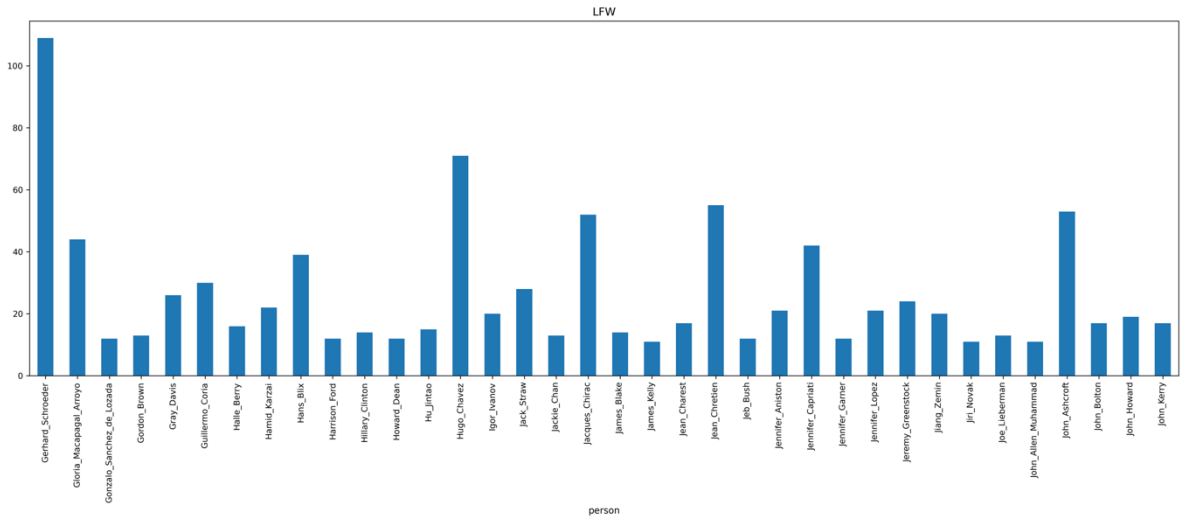
5.2.1 Dataset

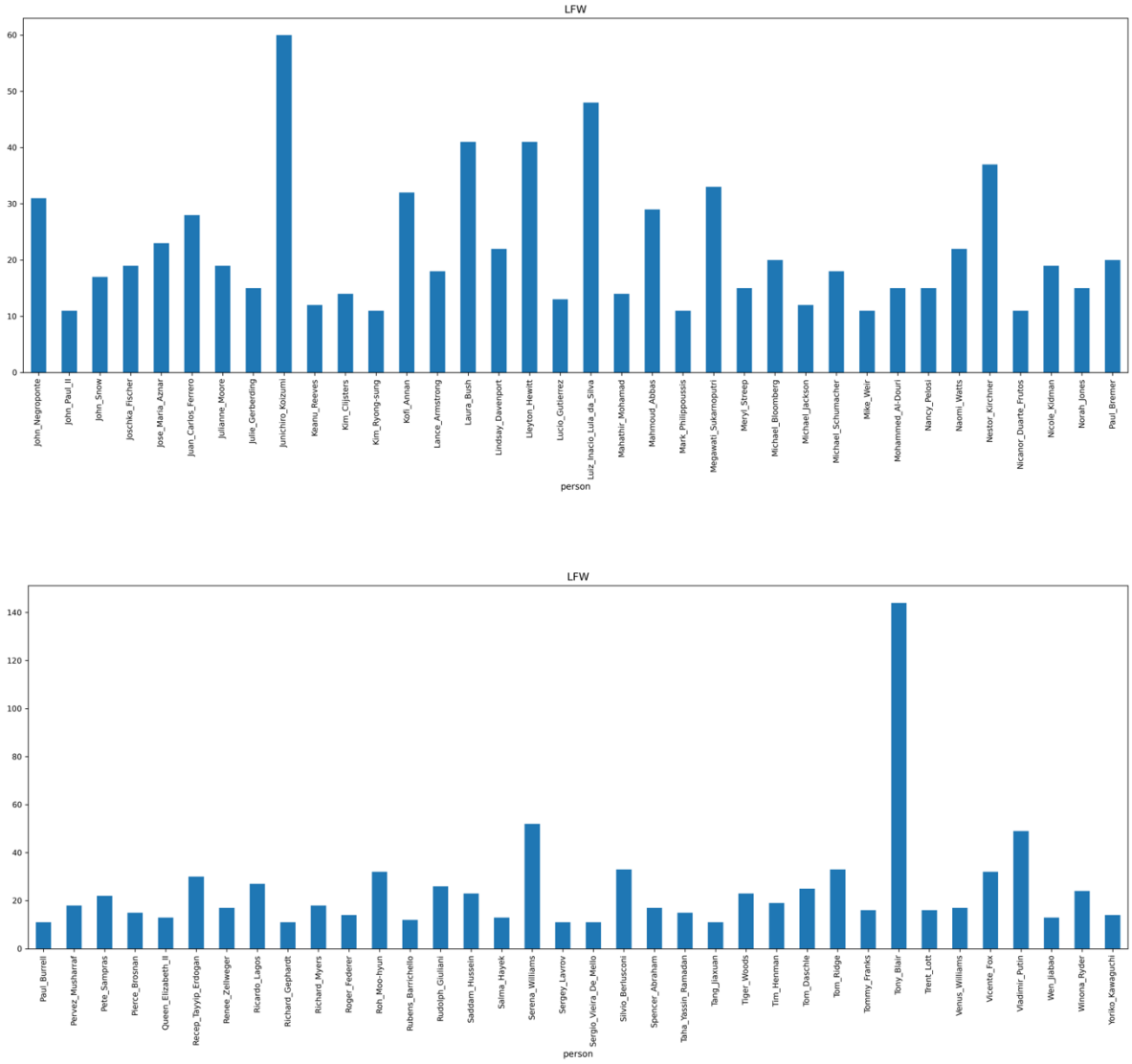
Použili sme dataset LFW (Labelled Faces in the Wild Home) [1]. Tento dataset vznikol na University of Massachusetts a postupne sa stal najznámejším benchmarkovým datasetom pre rozpoznávanie tvárí. Dataset obsahuje viac ako 13 000 obrázkov, ktoré sa získali z webu pomocou Viola-Jones detektora tvárí. Názov každého obrázka obsahuje meno osoby, ktorá sa na ňom nachádza. Viac ako 1680 ľudí je zobrazených na aspoň dvoch obrázkoch. Obrázky sú uložené vo formáte JPEG, majú veľkosť 250x250 a väčšina je farebná.

5.2.2 Predspracovanie údajov

Keďže sa v datasete nachádza veľa ľudí, ktorí majú len veľmi málo obrázkov, vyfiltrovali sme len tých, ktorí majú aspoň 10 obrázkov, aby sme neskôr vedeli správne urobiť krížovú validáciu. Týmto sme svoju dátovú sadu obmedzili na 152 ľudí, ktorí sa dokopy vyskytovali na 4174 obrázkoch. Nasledujúce grafy zobrazujú mená osôb na obrázkoch a počet obrázkov, na ktorých sa daná osoba nachádza.







Obr. 14 Znázornenie datasetu LFW

5.2.3 FaceNet

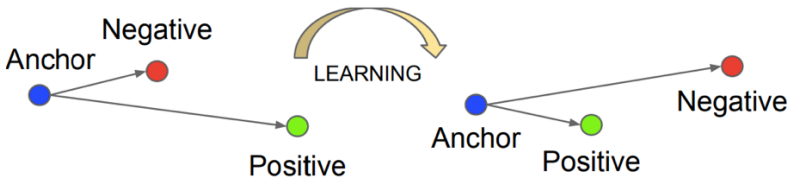
Na vytvorenie vektorov popisujúcich tváre (embedding) sme použili sieť FaceNet, ktorá je jedna z najlepších aktuálnych modelov na rozpoznávanie tvárí. Použili sme implementáciu [4], ktorá je totožná s implementáciou známej siete [5] Davida Sandberga. Líši sa len v tom, že je implementovaná v knižnici PyTorch a nie pomocou knižnice TensorFlow. Sieť bola predtrénovaná na datasete VGGFace2 [7], keďže tento model dáva najlepšie výsledky pre veľké množstvo datasetov pre rozpoznávanie tvárí.

Tento model používa trochu pozmenený model siete od tej, akú používali autori v pôvodnom článku [6], takzvaný Inseption-Resnet [8], ktorý spája incepčnú architektúru (Inception architecture) a reziduálne prepojenia (residual connections).

Model dostane na vstupe obrázok tváre (ktorý sme orezali z celého obrázka pomocou MTCNN) a na výstupe vráti vektor tváre dĺžky 512. Malo by platiť, že dva vektory tvárí toho istého človeka sú podobnejšie (majú menšiu vzájomnú euklidovskú vzdialenosť) ako dva vektory tvárí dvoch rôznych ľudí. Túto vlastnosť nám zabezpečí stratová funkcia triplet loss.

5.2.4 Triplet loss

Táto stratová funkcia je motivovaná hľadaním najbližšieho suseda (k-nearest neighbours) a slúži na vytvorenie vektora tvárí, ktorý spĺňa vyššie popísané vlastnosti. Na vstupe dostaneme počas tréningu tri obrázky: jeden základný (anchor), jeden obrázok tváre toho istého človeka a jeden obrázok tváre iného človeka. Triplet loss zabezpečí, aby sa pozitívny príklad priblížil základnému a negatívny príklad sa vzdialil od pôvodného. Pre lepšiu predstavu obrázok nižšie znázorňuje, čo sa stane s jednotlivými vektormi tvárí:



Obr. 15 Chybová funkcia triplet loss [6]

Označme obrázok tváre i-teho základného príkladu (anchor) x_i^a, obrázok tváre kladného príkladu x_i^p a záporného x_i^n. Nech $f(x)$ je výstup siete pre obrázok x. Nech α je minimálne kladné číslo také, aby kladné a záporné príklady pre jedného človeka mali vzdialenosť aspoň α (preto sa α zvykne označovať aj ako okraj, keďže je to veľkosť okraja medzi jednotlivými triedami). Stratová funkcia teda vyzerá nasledovne:

$$L = \sum_i \left(\left\| f(x_i^a) - f\left(x_i^p\right) \right\| - \left\| f(x_i^a) - f(x_i^n) \right\| + \alpha \right)$$

5.3 Klasifikácia vektorov tvárí pre jednotlivých ľudí

Neurónová sieť FaceNet má na výstupe iba vektor, ktorý dobre popisuje charakteristické črty tvárí na obrázkoch, ale nepovie nám, kto je na danom obrázku. Preto sme museli použiť klasifikačný algoritmus, ktorý na základe vektora určí triedu daného obrázka (identifikuje človeka na obrázku). Keďže optimalizačný algoritmus na trénovanie FaceNet siete je motivovaný práve hľadaním najbližších susedov, na

klasifikáciu sme využili aj tento model. Okrem neho sme použili aj iné modely, z ktorých niektoré dávali porovnateľné, niektoré dokonca lepšie výsledky. Pri implementácii sme využili knižnicu *scikit-learn* pre všetky nasledujúce metódy:

5.3.1 *k*-najbližších susedov

Algoritmus *k*-najbližších susedov (*k*-nearest neighbours, v skratke KNN) funguje na veľmi jednoduchom princípe. Pre dané číslo k sa nájde *k*-najbližších susedov pre daný objekt (v našom prípade k-najbližších vektorov tvárí). Formálne pre vstup x sa nájde množina $M = \{\langle x_i, c_i\rangle : 1 \leq i \leq k\}$, kde x_i je i-ty najbližší sused x a c_i je trieda, do ktorej patrí. Označme K_i počet prvkov triedy i z množiny M (teda K_i je počet tých najbližších susedov, ktorí sú klasifikovaní do triedy i). Výsledok algoritmu je teda $y = \arg\max\left\{\frac{K_i}{K}\right\}$.

5.3.2 Metóda podporných vektorov

Metóda podporných vektorov (support vector machines, v skratke SVM) patria medzi najpoužívanejšie klasifikačné metódy, pretože dávajú veľmi dobré výsledky pre rozličné datasety. Základný model SVM vie klasifikovať len do dvoch tried. V klasifikácii sa vyberie taká nadrovina v priestore, ktorá rozdeľuje naše dáta do dvoch tried, ktoré hľadáme (pri predpoklade, že naše dáta sú lineárne separovateľné). Z týchto nadrovín sa vyberie tá, ktorá má maximálny okraj.

Ak dáta nie sú lineárne separovateľné, použije sa tzv. kernel trik, ktorý sa využíva aj pri iných metódach strojového učenia. Dáta sa pomocou vhodne zvolenej nelineárnej funkcie (kernelu) pretransformujú do iného vysokodimenzionálneho priestoru, v ktorom sú dáta lineárne separovateľné s vyššou pravdepodobnosťou. Najpoužívanejšie kernely sú polynomiálne, exponenciálne (rbf) a sigmoidálne.

Na klasifikáciu do viacerých tried sa používa viacero metód. Použili sme tú, v ktorej sa pre každú dvojicu tried natrénuje jeden SVM klasifikátor. Tieto klasifikátory „hlasovaním“ rozhodnú, do ktorej triedy patrí daný vstup. Tento prístup sa zvykne označovať ako „jeden proti jednému“.

5.3.3 Náhodný les

Náhodný les (random forest) patrí medzi ensemble metódy. To znamená, že sa paralelne trénuje veľa rozhodovacích stromov (my sme použili základné nastavenie 100

z knižnice *scikit-learn*). Každý rozhodovací strom sa natrénuje na inej časti tréningovej sady. Táto časť sa vyberá náhodne pre každý rozhodovací strom. Pre ďalšiu regularizáciu modelu sa každý strom natrénuje na iba nejakej (tiež náhodne vybranej) podmnožine atribútov. Výsledná klasifikácia sa vykoná na základe „hlasovania“ všetkých stromov.

5.3.4 Gradient boosting

Podobne ako random forest, aj gradient boosting využíva rozhodovacie stromy. Ide o ensemble metódu, teda výsledná klasifikácia vznikne spojením viacerých slabších klasifikátorov. Hlavným rozdielom oproti random forestu je, že kým ten trénuje jednotlivé stromy paralelne a nezávisle od seba, gradient boosting postupne pridáva ďalšie stromy tak, aby zlepšili klasifikáciu na tých miestach, kde dávali predošlé rozhodovacie stromy chybnú predikciu. To sa udeje spôsobom, že algoritmus zvýši váhu (dôležitosť) nesprávne klasifikovaných príkladov z tréningovej vzorky a „novú“ tréningovú vzorku dostane na vstup nový rozhodovací strom. Je zrejmé, že kvôli vyššej váhe „zlých príkladov“ bude tento strom klasifikovať lepšie tie príklady, ktoré predošlé stromy klasifikovali nesprávne.

5.3.5 Zmenšenie počtu parametrov pre výslednú klasifikáciu

Skúsili sme zmenšiť počet parametrov využívaných na klasifikáciu obrázkov, aby sme dosiahli zrýchlenie výpočtu a zlepšenie presnosti predikcie. Na to sme použili dve rôzne metódy.

PCA

PCA (Principal component analysis – Analýza hlavných komponentov) je štatistická metóda, ktorá slúži na zmenšenie dimenzie vysokorozmerných dát. Ide o ortogonálnu transformáciu pôvodného vstupného priestoru (teda pôvodných vstupných dát, v našom prípade vektorov tvárí) do nového priestoru, v ktorom budú nové osy (stĺpce) principálne (hlavné) komponenty pôvodných dát. Týmto spôsobom zmeníme pôvodné korelované stĺpce na nové s menšou dimenziou, ktoré sú už lineárne nekorelované. Principálne komponenty sú také lineárne kombinácie pôvodných premenných, ktoré sú vyberané tak, aby sa zachovalo čo najviac z pôvodnej variancie dát.

Chi2 test

Chi2 test, presnejšie Pearsonov chi2 test, je štatistická metóda, ktorá sa používa na overenie, či má náhodná veličina vopred definované rozdelenie pravdepodobnosti. To nám umožňuje vypočítať skóre pre každý atribút, ktoré nám udáva závislosť výsledku klasifikácie od daného atribútu. Na základe výsledného skóre vieme vybrať tie atribúty, ktoré majú najväčší vplyv na výsledok (teda v našom prípade vyberieme tie atribúty vektorov tvárí, ktoré najviac ovplyvňujú, koho tvár je na danom obrázku). Tento typ redukcie dimenzionality sa pozerá na každý atribút zvlášť, ostatné atribúty sa „zahodia“, na rozdiel od PCA, kde sa do nových atribútov zakóduje informácia z viacerých starých atribútov.

5.3.6 Vyhodnotenie klasifikácie

Na vyhodnotenie správnosti modelu sme použili viacero evaluačných kritérií. Prvou z nich bola 10-násobná krížová validácia. Základnou ideou tejto evaluačnej metriky je náhodné rozdelenie dát do desiatich skupín. Postupne z nich vyberieme jednu, ktorá sa využije ako testovacia množina, zvyšných 9 sa využije na tréning. Výsledná presnosť modelu sa určí ako priemerná presnosť na desiatich tréningových/testovacích sadách. Táto evaluačná metóda je populárna najmä vďaka tomu, že veľmi dobre popisuje presnosť predikcie modelu na nových dátach.

Následne sme kvôli možnosti zistenia viacerých štatistických ukazovateľov modelu (ako napr. F-score) rozdelili dáta na trénovacie a testovacie v pomere 3:1. Takto sme zistili, ktorých ľudí klasifikoval model nesprávne.

Ako tretí ukazovateľ presnosti modelu sme zvolili AUC (area under curve), teda veľkosť plochy pod ROC krivkou (receiver operating characteristic curve). Táto grafická krivka znázorňuje, ako vie náš model rozlišovať medzi jednotlivými triedami klasifikácie. Ak sa hodnota AUC blíži k 1, tak náš model vie danú triedu odlíšiť od ostatných. Ak sa AUC rovná 1, tak náš model klasifikuje všetky prvky danej triedy správne.

5.4 Výsledky na dátach

Z pôvodných dát sme vytvorili 7 rôznych datasetov:

1. Pôvodný dataset – pôvodné vektory tvárí (**X**).
2. Dataset obsahujúci 100 najdôležitejších atribútov na základe chi2 testu (**X100**).
3. Dataset obsahujúci 80 najdôležitejších atribútov na základe chi2 testu (**X80**).
4. Dataset obsahujúci 50 najdôležitejších atribútov na základe chi2 testu (**X50**).

5. Dataset vytvorený pomocou PCA zachovávajúci 99 % variancie (**X_PCA99**). Použilo sa 47 principálnych komponentov.
6. Dataset vytvorený pomocou PCA zachovávajúci 95 % variancie (**X_PCA95**). Použilo sa 41 principálnych komponentov.
7. Dataset vytvorený pomocou PCA zachovávajúci 90 % variancie (**X_PCA90**). Použilo sa 36 principálnych komponentov.

Na výslednú klasifikáciu sme použili tieto modely:

1. **Nearest Neighbours 3**

V algoritme *k*-najbližších susedov sme ako *k* použili číslo 3, ktoré sme dostali ako najlepšie počas testovania.

2. **SVC**

Použili sme lineárne SVM, teda dáta sa netransformujú do iného priestoru pomocou kernel funkcie.

3. **Random forest 100**

Natrénovali sme 100 stromov v náhodnom lese.

4. **Random forest 300**

Natrénovali sme 300 stromov v náhodnom lese.

5. **Gradient boosting**

Natrénovali sme 150 stromov, pričom každý strom bol natrénovaný na polovici datasetu (subsample sme nastavili na 0,5). Maximálnu hĺbku stromu sme ohraničili číslom 3, učiaci pomer sme nastavili na 0,05.

5.4.1 Výsledky krížovej validácie

V nasledujúcej tabuľke môžeme vidieť presnosť predikcie jednotlivých modelov na základe 10-násobnej krížovej validácie na pôvodnej dátovej sade. Algoritmy sme zoradili podľa ich presnosti na testovacej dátovej sade.

Názov algoritmu	Tréningová presnosť	Testovacia presnosť	Čas
SVC	0,979212	0,971784	25,326
KNeighborsClassifier3	0,976051	0,968777	0,106891
KNeighborsClassifier5	0,969421	0,968315	0,110745

KNeighborsClassifier7	0,9714	0,968315	0,110314
RandomForrestClassifier300	1	0,964155	94,3843
RandomForrestClassifier100	1	0,96346	31,0042
GradientBoosting	1	0,73185	4193,694

Tab. 21 Výsledky krížovej validácie pre dátovú sadu X

Vo všetkých ostatných dátových sadách skončil najlepšie okrem PCA99 algoritmus najbližších susedov so zvolením k = 3. Najlepšie algoritmy na jednotlivých datasetoch sme zhrnuli do nasledujúcej tabuľky:

Dataset	Názov algoritmu	Tréningová presnosť	Testovacia presnosť	Čas
X	SVC	0,979212	0,971784	25,326
X50	KNeighborsClassifier3	0,97271	0,961607	0,0075242
X80	KNeighborsClassifier3	0,973867	0,966465	0,0097555
X100	KNeighborsClassifier3	0,969498	0,966698	0,0113944
X_PCA90	KNeighborsClassifier3	0,976539	0,966235	0,00529299
X_PCA95	KNeighborsClassifier3	0,976925	0,966003	0,0074175
X_PCA99	SVC	1	0,967161	3,46693

Tab. 22 Výsledky najlepších algoritmov pre jednotlivé dátové sady

Je zaujímavé, že pri PCA dátových sadách je presnosť všetkých algoritmov veľmi podobná, od 96,1 % po 96,7 %. Môže to súvisieť najmä so spôsobom, akým sa vytvárajú principálne komponenty.

Ďalším zaujímavým zistením je, že kým SVC klasifikátor dáva najlepšie výsledky pri pôvodnom datasete, pri menšom počte atribútov zvolených pomocou chi2 testu jeho presnosť rapídne klesá: pri 50 atribútoch je o už len 67 %.

Ak zoberieme do úvahy aj časovú zložitosť algoritmov, tak jednoznačným víťazom je algoritmus *k*-najbližších susedov, ktorý je mnohonásobne rýchlejší oproti ostatným algoritmom (najmä z dôvodu, že ho nie je potrebné trénovať).

Algoritmus gradient boosting nepriniesol očakávané výsledky, keďže napriek obrovskej časovej náročnosti generalizoval veľmi zle na našich dátach. Je možné, že pri lepšom nastavení parametrov sa s ním dajú získať lepšie výsledky, ale práve kvôli veľkej časovej zložitosti sme sa radšej viac zamerali na ostatné algoritmy.

Pri náhodných lesoch sa potvrdilo pravidlo, že väčší počet stromov je lepší, pokiaľ nám to počítačový výkon dovoľuje. 300 natrénovaných stromov prekonalo 100 stromov na všetkých datasetoch, rozdiely v presnosti však boli minimálne. Je teda len na nás, aby sme sa rozhodli, či nám trojnásobný výkon stojí za o 0,1 – 0,5 % lepšiu predikciu.

Rozdelenie dát v pomere 3:1

Pri rozdelení dát na tréningové a testovacie v pomere 3:1 sme dosiahli nasledujúce výsledky:

Názov algoritmu	Presnosť
SVC	0,972248
KNeighborsClassifier3	0,965772
KNeighborsClassifier5	0,967623
KNeighborsClassifier7	0,966698
RandomForrestClassifier300	0,968548
RandomForrestClassifier100	0,956522

Tab. 23 Presnosť predikcie pri rozdelení v pomere 3:1

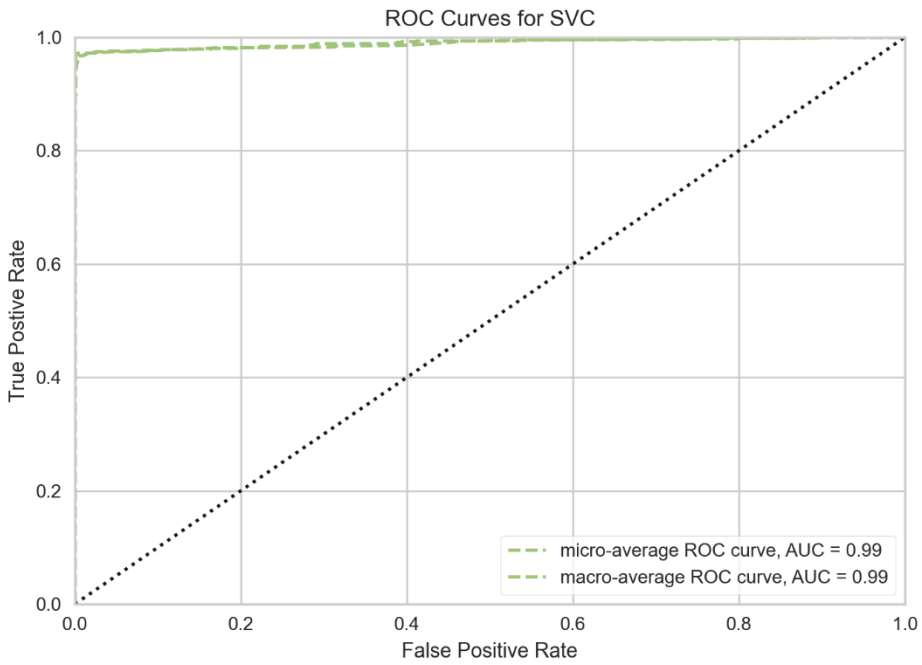
Najlepší model mal nasledujúce štatistické ukazovatele:

	SVC
Presnosť	0,98
Recall	0,97
F-score	0,97

Tab. 24 Štatistické ukazovatele modelu SVC

5.4.2 ROC krivka

Pre model SVC sme vykreslili aj ROC krivku, ktorá je zobrazená nižšie. AUC je rovné 0,99, čo znamená, že model veľmi dobre rozpoznáva jednotlivé triedy.

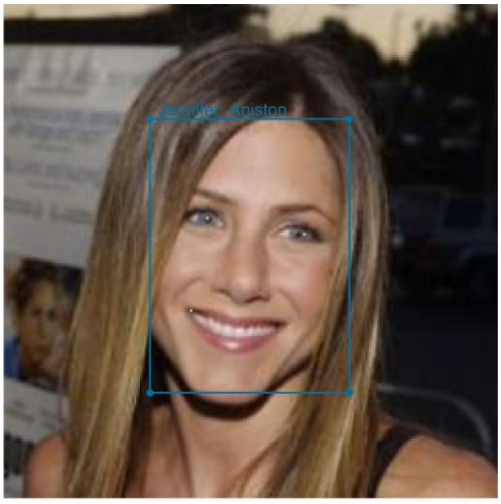


Obr. 16 ROC krivka klasifikácie pomocou SVC

5.4.3 Výsledky v praxi

Vytvorili sme program, pomocou ktorého vieme vizualizovať činnosť programu v praxi. Program dostane na vstupe fotku, na ktorej nájde tváre a vyznačí dôležité body, ktoré nájde sieť MTCNN. Potom sa pomocou siete FaceNet vytvorí embedding (vektor tváre), na základe ktorého už vie natrénovaný klasifikátor – v našom prípade *k*-najbližších susedov – zistiť, ktorá osoba sa nachádza na obrázku.

Napr. ako vidíme, na nasledujúcom obrázku je Jennifer Aniston, čo predikuje aj náš program:



Obr. 17 Detekcia tváre – Jennifer Aniston

Podobne na nasledujúcom obrázku je podľa programu Tom Cruise, čo je tiež pravda:

Obr. 18 Detekcia tváre – Tom Cruise

5.5 Zrýchlenie výpočtu

Na výpočet siete sme použili zložený model MTCNN, ktorý sa skladal zo siete PNet s 70 percentami, siete RNet so 40 percentami a siete ONet s 60 percentami pôvodných parametrov. Tieto siete sme vyberali pomocou algoritmu relatívnej straty popísaného v podkapitole 4.3.

Prvotne však bola táto sieť ešte pomalšia ako pôvodná sieť MTCNN, a to z dôvodu, že orezaná sieť PNet70 generovala oveľa viac návrhov tvárí ako pôvodná sieť PNet. To

spôsobilo zvýšenie výpočtov pre ďalšie siete RNet a ONet. Preto sme museli nastaviť nové hodnoty thresholdov pre jednotlivé siete PNet, RNet a ONet z pôvodných 0,6, 0,7, 0,7 na 0,96, 0,7, 0,7. Tieto nové hodnoty (presnejšie, zmenila sa len prahová hodnota pre sieť PNet) sme získali experimentálne, aby orezávaná sieť generovala podobne veľa kandidátov na tváre ako pôvodná sieť.

Takto sa nám podarilo získať pozoruhodné výsledky v zrýchlení aj v presnosti detekcie tváre. Ako klasifikačný dataset sme použili dataset LFW a ako metódu overovania presnosti modelu sme použili desaťnásobnú krížovú validáciu rovnako ako v podkapitole 5.4.1. Ako klasifikátor sme použili algoritmus najbližších susedov KNeighborsClassifier3.

Výsledky sme zhrnuli do nasledujúcej tabuľky:

Model	Počet operácií MTCNN	Počet operácií FaceNet	Presnosť klasifikácie
pôvodný	1 207 658 048 952	6 152 533 535 104	0,9688
orezávaný	462 840 088 560	6 152 533 535 104	0,9743

Tab. 25 Porovnanie modelov MTCNN pre problém rozpoznávania tvárí

Ako sa z tabuľky dá vyčítať, počet operácií vykonaných modelom MTCNN sme zmenšili o 62 %, pričom presnosť klasifikácie sa ešte zlepšila o viac ako pol percenta. Samotné časové zrýchlenie, ktoré sme namerali, bolo okolo 35 %. To znamená, že kým pôvodný model MTCNN vedel spracovať cca. 7,5 obrázkov za sekundu, orezaný model vie spracovať viac ako 10,5 obrázkov za sekundu. Toto je tiež významné zlepšenie, ak berieme do úvahy, že presnosť predikcie sa zlepšila.

Samozrejme, najlepšie zrýchlenie celého systému by sme vedeli získať, ak by sme orezali aj sieť FaceNet, keďže práve výpočet tejto siete zaberá viac ako 50% z celkového výpočtu. Preto orezávanie tejto siete môže byť ďalším zaujímavým bodom výskumu minimalizácie neurónových sietí.

Záver

Súčasná popularita minimalizácie neurónových sietí je následokom rapídneho zlepšenia predikcie neurónových sietí, najmä v oblasti počítačového videnia. Zlepšenie spôsobilo, že súčasné predikčné modely sú obrovské a vyžadujú si ohromnú výpočtovú silu, čo značne sťažuje ich využitie v real-time aplikáciách alebo lacných embedded zariadeniach.

Práve tento problém sa snaží vyriešiť minimalizácia neurónových sietí, ktorá zmenšuje počet parametrov sietí a tým zrýchľuje výpočet. V našej práci sme sa zaoberali touto tematikou, konkrétnejšie orezávaním konvolučných filtrov, ktoré je veľmi pravdepodobne najlepšie riešenie z pohľadu zrýchlenia výpočtov. Konkrétne sme sa zaoberali minimalizáciou siete na detekciu tvárí, keďže tento typ neurónovej siete má široké spektrum využitia v praktických aplikáciách.

Z viacerých možných prístupov a algoritmov sme získali najlepšie výsledky pomocou algoritmu minimálnych váh. Pri zmenšení počtu operácií vykonaných neurónovou sieťou o 73 % klesla presnosť predikcie o 1 %. Dokonca pri komplexnej klasifikácii, aká osoba sa nachádza na obrázku, zlepšil minimalizovaný model, ktorý vykonal o 62 % menej operácii, presnosť klasifikácie o viac ako 0,5 %.

V kapitole 4.3 a 4.4 sme popísali viacero vlastných metrík na výber najlepšej minimalizovanej siete, teda kedy ukončiť odoberanie parametrov z pôvodného modelu. Tieto metriky nám umožňujú vybrať si najlepšiu možnosť v pomere „cena–výkon“, teda najlepší pomer medzi zrýchlením modelu a poklesu presnosti predikcie. Z nášho výskumu vyplýva, že tieto metriky sú takmer rovnocenné. Výsledky vykazujú vysokú mieru korelácie, čo umožňuje vybrať si vždy implementačne najjednoduchšiu alebo výpočtovo najrýchlejšiu podľa konkrétnej situácie.

Táto práca poukazuje na to, že minimalizáciou neurónových sietí vieme masívne zrýchliť výpočty predikčných modelov pri veľmi malej až žiadnej strate. Navyše tieto algoritmy sú použiteľné na takmer ľubovoľnú konvolučnú neurónovú sieť, a preto môžu byť použiteľné na veľké množstvo rozličných problémov počítačového videnia. Táto vlastnosť v spojení s veľmi dobrými doterajšími výsledkami ich robí atraktívnou aj pre ďalší výskum.

Zoznam použitej literatúry

[1] Gary B. Huang,Manu Ramesh, Tamara Berg, and Erik Learned-Miller: Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments [prevzaté 12. marca 2020] [Online] http://vis-www.cs.umass.edu/lfw/lfw.pdf

[2] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10):1499–1503.

[3] Jan Hosang, Rodrigo Benenson, Bernt Schiele: Learning non-maximum suppression. arXiv 2017

[4] Face Recognition Using Pytorch [prevzaté 12. marca 2020] [Online] https://github.com/timesler/facenet-pytorch

[5] David Sandberg. Face recognition using Tensorflow. [prevzaté 12. marca 2020] [Online] https://github.com/davidsandberg/facenet

[6] Florian Schroff and Dmitry Kalenichenko and James Philbin.(2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. 1503.03832

[7] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, A. Zisserman VGGFace2: A dataset for recognising face across pose and age. International Conference on Automatic Face and Gesture Recognition, 2018. [prevzaté 13. marca 2020] [Online] http://www.robots.ox.ac.uk/~vgg/publications/2018/Cao18/cao18.pdf

[8] Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. 2016, Inception-v4, Inception-ResNet and the impact of residual connections on learning, arXiv:1602.07261v2 [cs.CV].

[9] Hao Li and Asim Kadav and Igor Durdanovic and Hanan Samet and Hans Peter Graf. 2016, Pruning Filters for Efficient ConvNets, arXiv:1608.08710.

[10] Pavlo Molchanov and Stephen Tyree and Tero Karras and Timo Aila and Jan Kautz. 2017, Pruning Convolutional Neural Networks for Resource Efficient Inference, arXiv: 1611.06440

[11] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, arXiv:1607.03250

[12] Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, Surya Ganguli. 2020, Pruning neural networks without any data by iteratively conserving synaptic flow, arXiv:2006.05467

[13] MTCNN_Tutorial. [prevzaté 21. septembra 2020] [Online] https://github.com/xuexingyu24/MTCNN_Tutorial

[14] Multimedia Laboratory, Department of Information Engineering, The Chinese University of Hong Kong, WIDER FACE: A Face Detection Benchmark, [prevzaté 12. júla 2020] [Online] http://shuoyang1213.me/WIDERFACE/

[15] Y. Sun, X. Wang, and X. Tang. Deep Convolutional Network Cascade for Facial Point Detection. 2013, In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[16] Bodla, N.; Singh, B.; Chellappa, R.; and Davis, L. S. 2017, Improving Object Detection With One Line of Code. In arXiv preprint arXiv:1704.04503.

[17] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, Jan Kautz. 2019, Importance Estimation for Neural Network Pruning (CVPR 2019)

[18] Frankle Jonathan, Carbin Michael. 2019, The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. arXiv:1803.03635

[19] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, Michael Carbin. 2020, Stabilizing the Lottery Ticket Hypothesis. arXiv:1903.01611

[20] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, Trevor Darrell. 2019, Rethinking the Value of Network Pruning. arXiv:1810.05270

[21] Namhoon Lee, Thalaiyasingam Ajanthan, Philip H. S. Torr. 2019, SNIP: Single-shot Network Pruning based on Connection Sensitivity. arXiv:1810.02340

Prílohy

Príloha A: CD médium – diplomová práca v elektronickej podobe, prílohy v elektronickej podobe.

Príloha B: Používateľská príručka

Príloha C: Systémová príručka

Táto časť diplomovej práce je povinná a obsahuje zoznam všetkých príloh vrátané elektronických nosičov. Názvy príloh v zozname musia byť zhodné s názvami uvedenými na príslušných prílohách. Tlačené prílohy majú na prvej strane identifikačné údaje – informácie zhodné s titulnou stranou diplomovej práce doplnené o názov príslušnej prílohy (Systémová príručka, Používateľská príručka). Identifikačné údaje sú aj na priložených diskoch alebo disketách. Ak je médií viac, sú označené aj číselne v tvare I/N, kde I je poradové číslo a N je celkový počet daných médií.

Každá príloha začína na novej strane a je označená samostatným písmenom alebo číslom (Príloha A, Príloha B, ... alebo Príloha 1, Príloha 2, ...). Číslovanie strán príloh nadväzuje na číslovanie strán v hlavnom texte.