**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ

PRÁCA

BRANISLAV TOMKA

**Implementácia konvolučnej hlbokej neurónovej siete pre diagnostiku rakoviny kože**

Vedúci práce: Ing. Lukáš Falát, PhD.

Registračné číslo: 988/2019

Žilina, 2020

**ŽILINSKÁ UNIVERZITA V ŽILINE**

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA

PRÁCA

ŠTUDIJNÝ ODBOR:

Inteligentné Informačné Systémy

BRANISLAV TOMKA

**Implementácia konvolučnej hlbokej neurónovej siete pre diagnostiku rakoviny kože**

Žilinská univerzita v Žiline

Fakulta Riadenia a Informatiky

Školiace pracovisko KMME

Žilina, 2020

#### [Čestné vyhlásenie](Pokyny_pre_vypracovanie_ZP.docx#Čestné_vyhlásenie)

Čestne prehlasujem, že som prácu vypracoval samostatne s využitím dostupnej literatúry a vlastných vedomostí. Všetky zdroje použité v diplomovej práci som uviedol v súlade s predpismi.

Súhlasím so zverejnením práce a jej výsledkov.

...........................................

V Žiline, dňa ............ Branislav Tomka

#### [Poďakovanie](Pokyny_pre_vypracovanie_ZP.docx#Poďakovanie)

„*Moje poďakovanie patrí školiteľovi diplomovej práce Ing. Lukášovi Falátovi, PhD. za odbornú pomoc, pripomienky a usmerňovanie pri tvorbe práce.*“

[ABSTRAKT V ŠTÁTNOM JAZYKU](Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt)

TOMKA, Branislav: ***Implementácia konvolučnej hlbokej neurónovej siete pre diagnostiku rakoviny kože***. [Diplomová práca]. – Žilinská univerzita. Fakulta Riadenia a Informatiky; Katedra makro a mikroekonomiky. – Školiteľ/Vedúci: Ing. Lukáš Falát, PhD. – Stupeň odbornej kvalifikácie: inžinier. – Žilina: FRI UNIZA, 2020. Počet strán .

[**Kľúčové slová**](Pokyny_pre_vypracovanie_ZP.docx#Kľúčové_slová)**:**

[ABSTRAKT V CUDZOM JAZYKU](Pokyny_pre_vypracovanie_ZP.docx" \l "Abstrakt_CJ)

**Key words**

[Obsah](Pokyny_pre_vypracovanie_ZP.docx#Obsah)

Zoznam obrázkov 10

Zoznam tabuliek 11

Zoznam skratiek 12

Úvod 13

Štatistiky 13

Prevencia 14

Metóda ABCDE 14

1 Metodológia 16

1.1 Supervizované učenie 16

1.2 Nesupervizované učenie 17

1.3 Posilňujúce učenie 17

1.4 Modely 17

1.4.1 Regresná analýza 17

1.4.2 Rozhodovacie stromy 17

1.4.3 Genetický algoritmus 18

1.4.4 Umelé neurónové siete 18

2 Konvolučné neurónové siete 19

2.1 CNN a NS 19

2.2 Štruktúra CNN 20

2.2.1 Konvolúcia a filter 20

2.2.2 Stride 23

2.2.3 Padding 23

2.2.4 Pooling 23

2.2.5 Aktivačné vrstvy 24

2.2.6 Plne prepojená vrstva 24

2.2.7 Softmax 25

2.3 Architektúry CNN 25

2.3.1 LeNet-5 26

2.3.2 AlexNet 27

2.3.3 VGG-16 27

3 Návrh a implementácia CNN 29

3.1 Jadro 30

3.2 Vrstvy 31

3.2.1 Rozhranie ILayer 31

3.2.2 Konvolučná vrstva 31

3.2.3 Plne prepojená vrstva 35

3.2.4 Softmax 37

3.2.5 Pooling vrstva 39

3.3 Adam 41

3.4 Batch 42

3.5 Algoritmické zrýchlenie 44

3.6 Návrh vlastnej architektúry CNN 44

4 Návrh nových vylepšení CNN nezávislých na architektúre 47

4.1 Vlastné pooling metódy 47

4.1.1 Weighted average pooling 47

4.1.2 Double max pooling 48

4.1.3 Median pooling 48

4.2 Vlastné aktivačné funkcie 48

4.2.1 CBRT 49

4.2.2 LeakyTReLU 49

4.2.3 SQCBRT 50

4.2.4 Log 50

4.3 Vlastné inicializácie váh 51

4.3.1 Student 51

4.3.2 GGD 52

4.3.3 *χ*2 52

4.3.4 Log-normal 53

4.4 Vlastné loss funkcie 53

4.4.1 Divided Binnary Cross-Entropy 55

4.4.2 Sensitivity Cross-Entropy 56

4.4.3 Shifted Binnary Cross-Entropy 57

4.5 Experimenty s navrhnutými vylepšeniami 58

4.5.1 Experiment 1 – pooling 58

4.5.2 Experiment 2 – aktivačná funkcia 59

4.5.3 Experiment 3 – inicializácia váh 59

4.5.4 Experiment 4 – loss funkcia 60

4.6 Najlepšie siete 61

4.6.1 Upravená LeNet-5 61

4.6.2 BraňoNet 62

5 Diskusia 63

5.1 Keras experimenty 63

5.1.1 Experiment 1 – pooling 64

5.1.2 Experiment 2 – Aktivačná funkcia 64

5.1.3 Experiment 3 – RGB 64

5.1.4 Experiment 4 – Inicializácia váh 65

5.1.5 Experiment 5 – Loss funkcia 66

5.2 Najlepšie keras siete 67

5.2.1 Upravená LeNet-5 67

5.2.2 Upravená AlexNet 67

6 Záver 69

Zoznam použitej literatúry 70

Zoznam príloh 71

Prílohy 72

# [Zoznam obrázkov](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

[Obrázok 3‑1 Znázornenie operácie konvolúcie 22](#_Toc40090289)

[Obrázok 3‑2 Vizualizácia filtrov od Krizhevskeho 23](#_Toc40090290)

[Obrázok 3‑4 AlexNet architektúra 27](#_Toc40090291)

[Obrázok 3‑5 VGG-16 architektúra 27](#_Toc40090292)

# [Zoznam tabuliek](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_obrázkov)

[Tabuľka 1‑1-1 Počet prípadov a úmrtí jednotlivých typov rakoviny 14](#_Toc40090227)

[Tabuľka 4‑1 Výsledky experimentov 1 40](#_Toc40090228)

[Tabuľka 4‑2 Výsledky experimentov 2 44](#_Toc40090229)

[Tabuľka 5‑1 Výsledky experimentu s BraňoNet 45](#_Toc40090230)

[Tabuľka 6‑1 Výsledky experimentov fázy 1 57](#_Toc40090231)

[Tabuľka 6‑2Výsledky experimentov fázy 2 57](#_Toc40090232)

[Tabuľka 6‑3 Výsledky experimentov fázy 3 58](#_Toc40090233)

[Tabuľka 6‑4 Výsledky experimentov fázy 4 59](#_Toc40090234)

# [Zoznam skratiek](Pokyny_pre_vypracovanie_ZP.docx" \l "Zoznam_skratiek)

# Úvod

Rakovina kože sa môže vyskytnúť na hociktorom mieste na tele ale najčastejšie na koži, ktorá je často vystavená slnečnému svetlu ako napríklad tvár, krk a ruky. Hlavné typy rakoviny kože sú: skvamocelulárny karcinóm, bazocelulárny karcinóm a melanóm. Melanóm je menej častý typ rakoviny ako zvyšné dva (non-melanómové), ale má väčšiu pravdepodobnosť, že napadne okolité tkanivo a že sa rozšíri do ostatných častí tela. Najviac smrtí je spôsobených melanómom. Melanóm je choroba, pri ktorej sa v kožných bunkách vytvoria zhubné bunky.

## Štatistiky

Rakovina kože je jedna z najbežnejších typov rakoviny. Melanóm tvorí len 4% všetkých nádorov kože, avšak spôsobuje až 73% všetkých úmrtí na rakovinu kože. Približne 2,3 % mužov a žien majú diagnostikovaný melanóm kože počas života. Za rok 2019 v Spojených Štátoch Amerických pribudlo 96 480 nových prípadov s melanómom kože a 7 230 ľudí zomrelo.

Tabuľka ‑ Počet prípadov a úmrtí jednotlivých typov rakoviny [1]

|  | **Typy rakoviny** | **Odhad nových prípadov 2019** | **Odhad úmrtí 2019** |
| --- | --- | --- | --- |
| 1. | Rakovina prsníka (Ženy) | 268,600 | 41,760 |
| 2. | Rakovina pľúc a priedušiek | 228,150 | 142,670 |
| 3. | Rakovina prostaty | 174,650 | 31,620 |
| 4. | Rakovina hrubého čreva a konečníka | 145,600 | 51,020 |
| **5.** | **Melanóm kože** | **96,480** | **7,230** |
| 6. | Rakovina močového mechúra | 80,470 | 17,670 |
| 7. | Rakovina lymfatických uzlín | 74,200 | 19,970 |
| 8. | Rakovina obličiek | 73,820 | 14,770 |
| 9. | Rakovina krčka maternice | 61,880 | 12,160 |
| 10. | Leukémia | 61,780 | 22,840 |

Ročne pribudne 22,2 nových prípadov melanómu na 100 000 ľudí podľa štatistík z rokov 2012-2016. Melanóm kože sa viac vyskytuje u mužov ako u žien. U mužov je to 28.8 a u žien 17.5 nových prípadov ročne na 100 000 ľudí. Prípadov melanómu kože sa vyskytuje pri bielej rase viac ako pri ktorejkoľvek inej rase. Najčastejšie sa melanóm kože diagnostikuje osobám v rokoch 65-74 (24,1%). Percento úmrtnosti je najväčšie pri ľuďoch v rokoch 75-84 (23,9% z všetkých úmrtí).

## Prevencia

Pravdepodobnosť výskytu rakoviny kože je možné znížiť používaním ochrany proti slnečnému žiareniu a vyhýbaniu sa ultrafialovému žiareniu. Ultrafialové žiarenie je na základe aktuálnych vedeckých poznatkov klasifikované Medzinárodnou agentúrou pre výskum rakoviny (IARC) ako dokázaný karcinogén. Závery epidemiologických štúdií IARC potvrdili, že vystavenie sa UV žiareniu v mladosti najviac prispieva k poškodeniu kože a riziku rozvoja melanómu v ďalšej časti života. V súčasnosti je vo svete zaznamenávaný významný nárast melanómu kože najmä medzi mladými ženami do 30 rokov. Z tohto dôvodu bola v roku 2019 schválená novela zákona o ochrane, podpore a rozvoji verejného zdravia, ktorá sa zaoberala obmedzením osobám mladším ako 18 rokov navštevovať solária s umelým zdrojom UV žiarenia.

Ďalší spôsob ako predísť zlým následkom rakoviny kože je pravidelná kontrola buď u dermatológa, alebo samokontrolou využitím metódy ABCDE.

### Metóda ABCDE

* **A** – Asymetry (Asymetria) - Dermatológa by sme mali navštíviť v prípade, že je naše znamienko asymetrické. To spoznáme tak, že si ho pomyselne rozdelíme na dve polovice a zameriame sa na jeho tvar. Pokiaľ sú obe polovice súmerné, tak nie je treba nič riešiť. Pokiaľ je ale každá polovica iná, mali by sme zbystriť pozornosť.
* **B** – Borders (Okraje) - Okrem plasticity by sme sa mali u znamienok sústrediť aj na ich okraje. Ohraničenie melanómu totiž býva často nepravidelné či rozpité, pričom zdravé znamienko má okraje pravidelné a ostré.
* **C** – Colour (Farba) - O mnohom môže vypovedať aj farba znamienka. Pokiaľ pozorujeme, že niektorá z jeho častí zrazu zmenila farbu, mali by sme si ho nechať skontrolovať, nepravidelne škvrnité alebo čierne znamienka môžu byť nebezpečné.
* **D** – Diameter (Priemer) - Všímajte si aj veľké pehy a znamienka. Melanómy bývajú zvyčajne väčšie než zdravé znamienka a väčšinou sa ich veľkosť pohybuje nad pol centimetra v priemere.
* **E** – Evolution (Vývoj) - Môžeme povedať, že je potrebné venovať väčšiu pozornosť znamienku vždy, keď  náhle zmení tvar, veľkosť, farbu či štruktúru. V takom prípade je na mieste vyhľadať pomoc kožného lekára, môže sa totiž jednať o nebezpečný melanóm. Dôležitá je pri samovyšetrení aj pravidelnosť. Keď budeme kontrolovať znamienka pravidelne, všimneme si aj drobné odchýlky, ktoré sa na ňom mohli vytvoriť priebehom času.

aasdasd

# Metodológia

Strojové učenie je vedecký odbor algoritmov a štatistických modelov, ktorý využívajú počítačové systémy na vykonanie špecifických úloh bez použitia explicitne definovaného postupu a namiesto neho sa spolieha na vzorce a vyhodnotenia. Považuje sa za podmnožinu umelej inteligencie. Algoritmy strojového učenia tvoria matematický model na základe nejakej vzory dát, ktorú označujeme ako „trénovacie dáta“, za účelom robiť predikcie alebo rozhodnutia bez explicitného naprogramovania vykonať danú úlohu. Mitchell (1997) definuje algoritmy strojového učenia “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” Teda strojové učenie nám umožňuje vyriešiť úlohy, ktoré sú príliš náročné na vyriešenie fixnými programami napísanými a navrhnutými ľuďmi. Z vedeckého a filozofického pohľadu je strojové učenie zaujímavé pretože rozvoj nášho chápania strojového učenia si vyžaduje rozvíjanie nášho chápania zásad, ktoré sú základom inteligencie.

Typy algoritmov strojového učenia sa líšia v ich prístupoch, typoch dát na vstupe a výstupe a type úloh alebo problému, ktoré majú riešiť.

## Supervizované učenie

Supervizované učenie začína s cieľom predpovedať známy výstup alebo cieľ. V tomto type algoritmov sa často porovnávajú výkonnosti rôznych modelov na základe výsledkov na rovnakých datasetoch. Medzi typické problémy supervizovaného učenia patrí rozpoznávanie ručne písaného textu, klasifikovanie obrázkov rôznych objektov (napr. je to mačka alebo pes?, je to lietadlo alebo auto?) alebo klasifikovanie dokumentov (napr. je to klinické skúšanie srdcového zlyhania alebo finančná správa?). Ide predovšetkým o úlohy, ktoré môže školený človek urobiť dobre, a preto sa počítač často snaží priblížiť ľudskému výkonu. Supervizované učenie sa zameriava na klasifikáciu, ktorá zahŕňa výber z podskupín, ktorá najlepšie opisuje novú skupinu údajov, alebo predikciu, teda odhad neznámeho parametra (napr. teplota v Žiline zajtra poobede).

## Nesupervizované učenie

Na rozdiel, pri nesupervizovanom učení neexistujú žiadne výstupy, ktoré by sme mali predpovedať. Namiesto toho sa snaží nájsť prirodzene sa vyskytujúce vzorce alebo zoskupenia v rámci daného datasetu. Toto hodnotiť je už zo svojej podstaty náročnejšia úloha a často sa takéto vzorce alebo zoskupenia, ktoré sa získali prostredníctvom nesupervizovaného učenia hodnotia podľa výkonnosti v následných úlohách supervizovaného učenia (napr. sú tieto nové vzorce nejakým spôsobom užitočné?).

## Posilňujúce učenie

Posilňujúce učenie je oblasť strojového učenia zaoberajúca sa tým, ako by softvéroví agenti mali konať v prostredí s cieľom maximalizovať nejaký prínos alebo odmenu. Tento typ učenia sa odlišuje od supervizovaného učenia tým, že nepotrebuje vopred označené vstupy prislúchajúcim výstupom. Namiesto toho sa zameriava na nájdenie rovnováhy medzi skúmaním (nepreskúmaného územia) a využívaním (súčasných poznatkov).

## Modely

Vykonanie strojového učenia zahŕňa vytvorenie modelu, ktorý je natrénovaný na nejakých trénovacích dátach a potom spracúva ďalšie dáta aby robil predikcie. Existuje niekoľko rôznych typov modelov, ktoré sa používajú v strojovom učení.

### Regresná analýza

Regresná analýza zahŕňa veľké množstvo štatistických metód na odhad vzťahu medzi vstupnými premennými a ich vlastnosťami. Najbežnejšia forma regresnej analýzy je lineárna regresia, ktorá vytvára funkciu, ktorá najlepšie vyhovuje daným dátam na základe matematických kritérií, ako napríklad suma najmenších štvorcových chýb (SSE). Toto kritérium sa často rozširuje pomocou regularizačných (matematických) metód na zmiernenie pretrénovania. Pri riešení nelineárnych problémoch sa používa napríklad logistická regresia alebo polynomiálna regresia (používa sa v MS Excel na určenie trendu).

### Rozhodovacie stromy

Učenie rozhodovacieho stromu používa rozhodovací strom ako prediktívny model na prechod z pozorovaní o nejakej položke (v strome predstavuje vetvu) k záverom o cieľovej hodnote položky (v strome predstavuje list). Stromy, v ktorých cieľová premenná môže mať diskrétnu množinu hodnôt, sa nazývajú klasifikačné stromy. V týchto stromových štruktúrach listy reprezentujú označenia triedy a vetvy predstavujú spojenia prvkov, ktoré vedú k týmto označeniam triedy. Stromy, v ktorých cieľová premenná môže mať spojité hodnoty (zvyčajne reálne čísla), sa nazývajú regresné stromy. Pri rozhodovaní sa môže použiť rozhodovací strom na vizualizáciu postupnosti rozhodnutí vedúcich k danému výsledku.

### Genetický algoritmus

Genetický algoritmus je heuristika, ktorá napodobňuje proces prirodzenej selekcie pomocou metód ako mutácia a kríženie, aby sa vytvorili nové genotypy v nádeji, že sa nájdu dobré riešenia daného problému. V 80. a 90. rokoch 20. storočia sa používali genetické algoritmy v strojovom učení. Dnes sa však často kombinujú genetické algoritmy s inými modelmi strojového učenia na dosiahnutie najlepšieho výsledku.

### Umelé neurónové siete

Umelé neurónové siete boli pôvodne vyvinuté na modelovanie činnosti nervového systému mozgu. Dnes už sa však môžu použiť na riešenie veľkého množstva úloh. Umelá neurónová sieť sa rovnako ako mozgová neurónová sieť skladá z neurónov a axónov, teda prepojení týchto neurónov. Každý neurón môže prijať signál a ten následne spracovať a preposlať do ďalších neurónov. Neurónovú sieť teda tvorí veľké množstvo navzájom prepojených neurónov, ktoré si medzi sebou posielajú signál od vstupu až po výstup. Spracovanie signálu v neuróne je reprezentované nelineárnou funkciou súčtu všetkých vstupov neurónu a výsledok je jeden výstup. Učenie umelej neurónovej siete prebieha v nastavovaní váh každého neurónu pre konkrétne vstupy. V dnešnej dobe sa umelé neurónové siete používajú na veľké množstvo úloh ako napríklad rozpoznanie jazyka, prekladanie, filtrovanie na sociálnych sieťach, pri videohrách a taktiež ich veľké využitie je v medicínskej diagnostike.

# Konvolučné neurónové siete

Konvolučná neurónová sieť (CNN) je špeciálny typ neurónovej siete (NN), ktorej hlavné využitie je klasifikovanie obrázkov. Napríklad rozhodovanie či je na obrázku mačka alebo pes, alebo rozpoznanie ručne písaného textu.



## CNN a NN

Konvolučná neurónová sieť je veľmi podobná klasickej neurónovej sieti. Pozostávajú z neurónov, ktoré majú váhy, ktoré sa sieť učí. Každý neurón dostane vstup, vykoná súčet vynásobení váh so vstupmi a prípadne sa aplikuje aktivačná funkcia. Celá sieť stále vyjadruje jednu diferencovateľnú funkciu. Z obrázku na vstupe vyprodukuje skóre jednotlivých tried na výstupe sieti. Avšak konvolučná neurónová sieť predpokladá že má na vstupe obrázok.



Hlavným dôvodom prečo sa na obrázky používajú konvolučné neurónové siete je, že obrázky sú veľké. Väčšina obrázkov, ktoré sa v dnešnej dobe používajú pri rozpoznávaní majú veľkosť 224x224 alebo väčšie. Farebné obrázky navyše bývajú reprezentované 3 farebnými kanálmi (RGB) a teda takýto obrázok má 224 \* 224 \* 3 = 150 528 vstupných bodov/pixelov. Typická skrytá vrstva v takejto sieti má 1024 uzlov, takže by sa neurónová sieť musela učiť 150 528 \* 1024 = 154 140 672 váh. Viac ako 150 miliónov váh len na jednu vrstvu. Takáto sieť by bola obrovská a bolo by takmer nemožné učiť takúto sieť. Navyše pri obrázkoch ani toľko váh nepotrebujeme, pretože ako vieme tak pixely jedného obrázka sú najviac popisujúce v rámci kontextu ich susedov. Všetky objekty obrázkov sú tvorené malými, presne umiestnenými vlastnosťami, ako napríklad kruhová dúhovka oka, alebo štvorcový hranatý roh papiera. Potom učiť sa váhy pre každý jeden pixel môže byť veľmi zbytočné.

Ďalším veľmi podstatným dôvodom nepoužívania klasickej neurónovej siete je fakt, že pozícia objektu sa môže meniť. Ak by bola sieť naučená rozpoznávať psa na obrázku, tak v prípade, že sa pes na obrázku posunie, sieť ho vôbec nemusí nájsť, pretože by sa neaktivovali potrebné neuróny a takáto sieť by bola zbytočná.

Konvolučná neurónová sieť rieši oba tieto problémy .

## Štruktúra CNN

## Vrstvy CNN

Konvolučná neurónová sieť sa rovnako ako klasická neurónová sieť skladá z rôznych vrstiev, ktoré sú medzi sebou prepojené. Konvolučná neurónová sieť sa skladá najmä z 4 hlavných vrstiev:

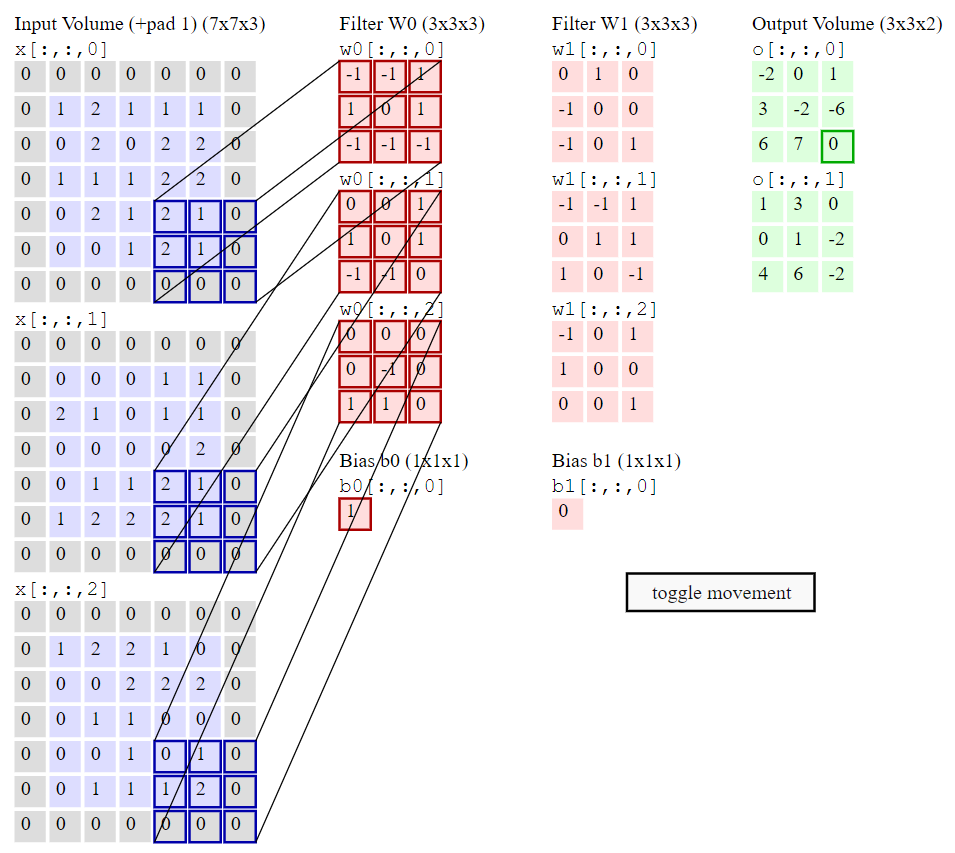
* Konvolučná vrstva
* Pool vrstva
* Plne prepojená vrstva
* Softmax vrstva (špeciálny prípad plne prepojenej vrstvy)

### Konvolučná vrstva

Hlavným rozdielom konvolučnej vrstvy oproti normálnej plne prepojenej vrstve je zdieľanie váh. Každý neurón konvolučnej vrstvy sa pozerá na štvorcový výrez obrázka. Neuróny zoberú na vstup každý pixel z tohto výrezu a zdieľa váhy bez ohľadu na to, kde sa tento výrez nachádza na obrázku, takže sú rovnocenné. Ďalšia interpretácia konvolučnej vrstvy namiesto množiny rovnocenných neurónov, je jeden neurón, ktorý nazývame filter. Tento filter je štvorcová matica, ktorá obsahuje váhy a konvolúciou prechádza celý obrázok výrez po výreze cez celú hĺbku vstupu. Napríklad typický filter vstupnej vrstvy môže mať rozmer 5x5x3 (5 pixelov široký a vysoký filter, ktorý má hĺbku 3, pretože farebné obrázky majú 3 RGB vrstvy). Konvolúcia pozostáva zo 4 hlavných bodov:

1. Prekrytie filtra cez jeden výrez obrázka.
2. Vykonanie násobenia po prvkoch medzi hodnotami vo filtri a ich zodpovedajúcimi hodnotami na obrázku.
3. Sčítanie všetkých výsledkov násobení po prvkoch a prípadné prevedenie cez aktivačnú funkciu ako pri klasickej vrstve. Výsledné číslo bude jeden pixel vo výstupnom obrázku.
4. Kroky sa opakujú cez všetky výrezy celého obrázka.

Sieť sa postupne bude učiť filtre, ktoré sa aktivujú keď uvidia nejakú špecifickú vizuálnu vlastnosť obrázka ako napríklad rôzne orientované okraje, nejakú škvrnu na vstupnej vrstve, prípadne celý vzorec podobný kolesu alebo tvári na hlbších vrstvách.



Obrázok ‑ Znázornenie operácie konvolúcie

Na obrázku 3-1 je vidieť ako postupuje konvolúcia. Na vstupe je obrázok 7x7x3 s hodnotou padding 1 a je prechádzaný dvoma filtrami o veľkosti 3x3x3 a stride je 2, teda výstup má rozmer 3x3x2.

Filter teda môžeme interpretovať aj ako nejaký detektor vlastností, ktorý vo výsledku vráti veľké číslo, ak detekuje danú vlastnosť.



Obrázok ‑ Vizualizácia filtrov od Krizhevskeho [3]

Na obrázku je 96 filtrov o veľkosti [11x11x3]. Na niektorých filtroch je jasne vidieť, ktoré vlastnosti obrázka aktivujú (rôzne okraje, tvary, ...).

#### Stride

Stride je parameter konvolučnej vrstvy, ktorý určuje veľkosť kroku medzi jednotlivými výrezmi, cez ktoré prechádza filter pri konvolúcii. Takmer vždy sa tento parameter nastavuje na 1, tak aby ďalší výrez bol hneď nasledujúci. Výhoda vyššej hodnoty tohto parametra je, že zníži výšku a šírku výstupu. Avšak v praxi stride 2 alebo viac môže byť považované za príliš agresívny prístup a teda zníženie veľkosti výstupu je za cenu straty niektorých informácií.

#### Padding

Pri konvolúcii pixely, ktoré sú viac pri strede, sú prejdené filtrom viac krát, ako tie pri kraji obrázka, takže tam môže dôjsť k strate nejakej informácie. Aby sa predišlo takejto strate informácií, je možné nastaviť parameter padding, ktorý pridá 0 na okraje obrázka, aby nebola pridaná žiadna informácia navyše, keďže násobením 0 sa výsledná hodnota nezvýši. Padding sa taktiež používa na kontrolu rozmeru výstupu. Konvolúcia zmenšuje veľkosť obrázku, avšak pridaním paddingu môžeme zachovať rozmer obrázku na výstupe rovnaký ako na vstupe.

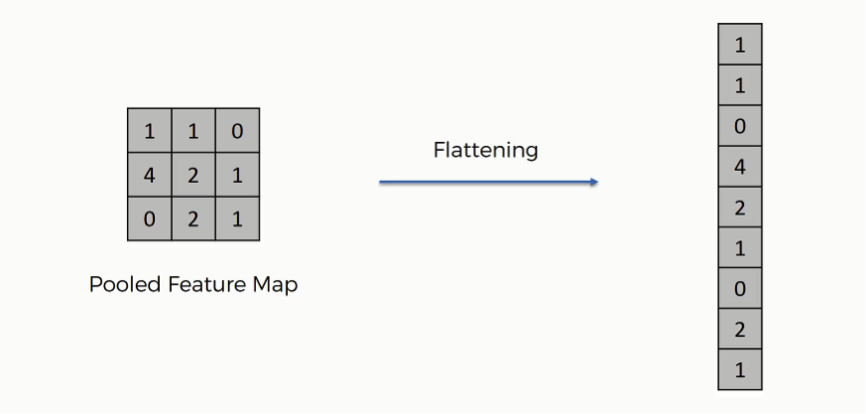
### Pool vrstva

Susedné pixely v obrázkoch zvyčajne majú podobné hodnoty, takže konvolučné vrstvy takisto dajú rovnaké hodnoty pre susedné pixely vo výstupe. Vo výsledku väčšina informácií, ktoré sa nachádzajú vo výstupe konvolučnej vrstve sú redundantné. Napríklad ak použijeme filtre na detekciu hrán a na niektorom mieste nájdeme hranu, je veľmi pravdepodobné, že hranu nájdeme aj o 1 pixel vedľa a teda je to stále tá istá hrana, takže sme nenašli nič nové.

Pooling vrstva rieši tento problém. Táto vrstva redukuje veľkosť vstupu združovaním hodnôt. Obvykle sa na pooling využívajú jednoduché matematické operácie ako min, max alebo priemer. Zvyčajne sa na združujú pixely po výrezoch veľkých 2x2 pixely. Teda šírka a výška výstupného obrázka bude o polovicu menšia ako vstupného obrázka. Napríklad pri max-pooling vyberieme z výrezu 2x2 len jedno najväčšie číslo.

### Plne prepojená vrstva

Plne prepojená vrstva sa pridáva za konvolučné vrstvy, kde sú všetky neuróny prepojené so všetkými výstupmi predošlej vrstvy. Výstup z poslednej konvolučnej vrstvy sa roztiahne do jedného vektora, ktorý sa použije ako vstup pre túto vrstvu.



Týchto plne prepojených vrstiev sa môže v CNN nachádzať aj viacej. Tieto vrstvy pracujú ako klasická neurónová sieť. Prenásobia vstupy s váhami, sčítajú ich a aplikujú na tento súčet aktivačnú funkciu.

### Softmax vrstva

Ako posledná, výstupná, vrstva CNN sa takmer vždy používa Softmax vrstva. Táto vrstva je plne prepojená vrstva s softmax aktivačnou funkciou. Táto vrstva obsahuje toľko neurónov, koľko rôznych tried je potrebné klasifikovať.

Softmax aktivačná funkcia je zovšeobecnenie sigmoidnej aktivačnej funkcie pretože tiež prideľuje pravdepodobnosť každému z neurónov (jeden neurón pre každú triedu). Pre neurón i a počet tried j, sa softmax funkcia vypočíta nasledovne:

Najskôr sa vypočíta exponent váhami prenásobené vstupy neurónu i a následne sa vydelí súčtom exponentov všetkých váhami prenásobených vstupov všetkých j neurónov, aby sa normalizoval výstup a teda aby pravdepodobnosti pre všetky triedy po sčítaní dali 1.

## Architektúry CNN

Ako aj normálne neurónové siete, aj pri konvolučné sa skladajú z rôznych vrstiev. Avšak pri CNN existuje viacero rôznych typov vrstiev. Vo všeobecnosti sa konvolučné neurónové siete skladajú z troch hlavných typov vrstiev: konvolučné (vrátane aktivačnej vrstvy), pool vrstvy (najčastejšie max) a plne prepojených vrstiev (často zakončených softmax vrstvou). Najbežnejší spôsob tvorby konvolučných neurónových sietí je nakopiť niekoľko konvolučných vrstiev nasledovaných pool vrstvami a toto vrstvenie opakovať až dokým sa obrázok postupne nezmenší na veľmi malý (niekedy až na veľkosť 1 pixela). Následne sa aplikuje niekoľko plne prepojených vrstiev zakončených softmax vrstvou. Inak povedané, najbežnejší vzorec pre tvorbu architektúry CNN je nasledovný:

Zvyčajne 0 < N <= 3, M >= 0, 0 < K < 3. Samozrejme počet jednotlivých vrstiev a ich usporiadanie nie je jediná vec, ktorá ovplyvňuje výkonnosť konvolučnej neurónovej siete. Záleží aj na nastavení rôznych hyperparametrov, ako je veľkosť a počet filtrov v jednotlivých konvolučných vrstvách, stride, padding. Veľkosť výrezu v pool vrstvách, počet neurónov v plne prepojených vrstvách a mnoho ďalšieho.

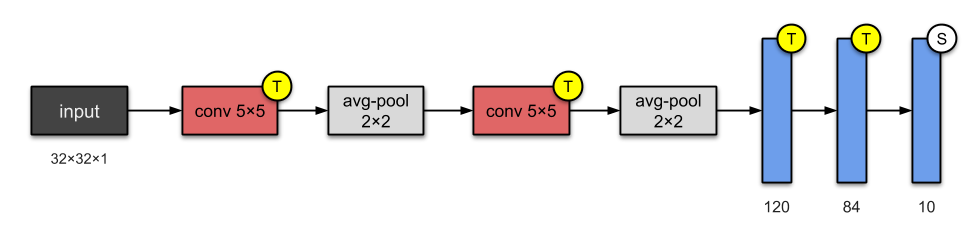
V posledných rokoch sa ukazuje vo svete veľké množstvo rôznych architektúr CNN. Mnohé z nich sú tak veľké a komplikované, že je veľmi náročné si predstaviť a zobraziť ako vyzerajú. V oblasti rozpoznávania obrazu bol zaznamenaný obrovský pokrok, avšak ako poznamenal výskumník z Googlu Christian Szegedy „väčšina tohto pokroku nie je iba výsledkom silnejšieho hardvéru, väčších datasetov a väčších modelov, ale hlavne dôsledkom nových nápadov, algoritmov a vylepšených architektúr sietí“. Avšak z toho veľkého množstva rôznych architektúr, existuje niekoľko veľmi často používaných sa dobrými výsledkami a je vidno, že aj keď nie sú veľmi veľké, tak ich výsledky sú skvelé.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Veľkosť | Top-1 presnosť | Top-5 presnosť | Parametre | Hĺbka |
| VGG-16 | 528 MB | 0,713 | 0,901 | 138 357 544 | 23 |
| InceptionV3 | 92 MB | 0,779 | 0,937 | 23 851 784 | 159 |
| ResNet50 | 98 MB | 0,749 | 0,921 | 25 636 712 | - |
| Xception | 88 MB | 0,790 | 0,945 | 22 910 480 | 126 |
| InceptionResNetV2 | 215 MB | 0,803 | 0,953 | 55 873 736 | 572 |
| ResNeXt50 | 96 MB | 0,777 | 0,938 | 25 097 128 | - |

V tabuľke je porovnaných 6 architektúr. Ich veľkosti, hĺbka a počet parametrov sa môžu veľmi líšiť a pritom výsledky sú veľmi podobné. Testované boli na ImageNet datasete a top-1 a top-5 sú ich výsledky na tomto datasete.

### LeNet-5

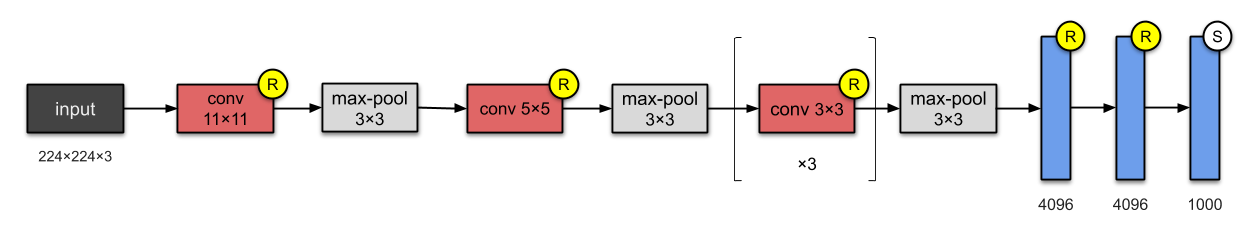
Jedna z najjednoduchších architektúr, ktorá má len 2 konvolučné a 3 plne prepojené vrstvy a len okolo 60 000 parametrov je LeNet-5 architektúra z roku 1998.



Obrázok ‑ LeNet-5 architektúra

### AlexNet

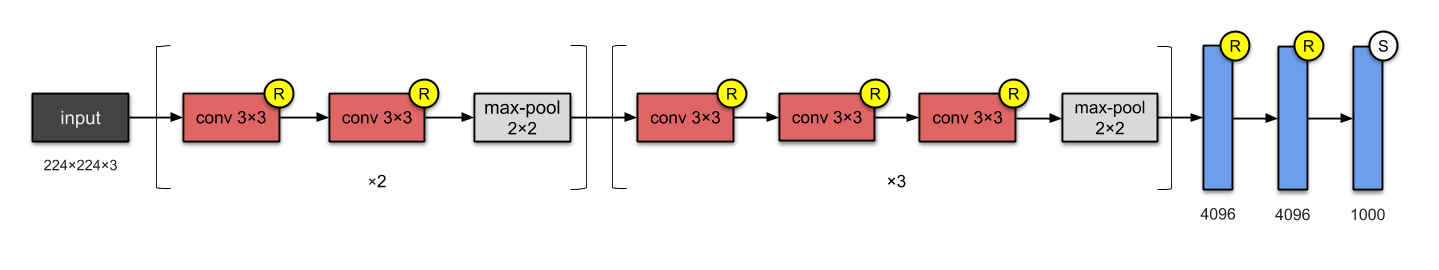
Novšia architektúra z roku 2012 AlexNet ako prvá prišla z myšlienkou vymeniť Tanh aktivačnú funkciu za ReLU. Má 8 vrstiev a okolo 60 miliónov parametrov. V tej dobe to bola najväčšia konvolučná neurónová sieť použitá na ImageNet dataset.



Obrázok ‑ AlexNet architektúra

### VGG-16

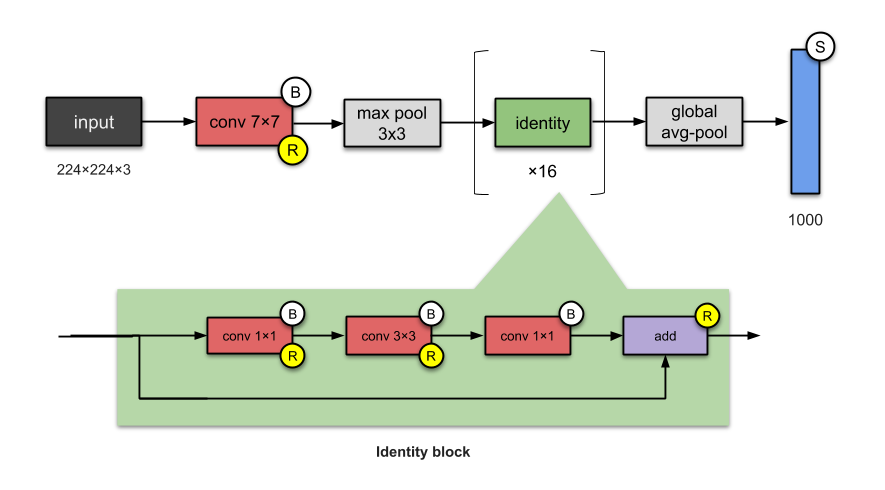
Je vidieť, že v poslednej dobe sa začali konvolučné neurónové siete zväčšovať, pretože najpriamejší a najjednoduchší spôsob ako zlepšiť CNN je zväčšiť jej veľkosť. Vo Visual Geometry Group (VGG) v roku 2014 vytvorili sieť zvanú VGG-16, ktorá má 13 konvolučných a 3 plne prepojené vrstvy. Táto architektúra pridala niekoľko vrstiev na AlexNet architektúru a použila menšie filtre (2x2 a 3x3). Aktivačnú funkciu ReLU aj max pooling ponechala z AexNet. Pozostáva z približne 138 miliónov parametrov a zaberá až 528MB miesta. Okrem VGG-16 vytvorili aj väčšiu VGG-19.



Obrázok ‑ VGG-16 architektúra

### ResNet-50

Za touto sieťou stoja výskumníci z Microsoft Research. Pri tvorbe tejto siete využili takzvané „skip“ prepojenia (skratky) ale zároveň zostrojili hlbšiu a väčšiu architektúru. ResNet bol medzi prvými, ktorý použili batch normalizáciu. Základným stavebným prvkom ResNet-50 sú Conv a Identity bloky. A keďže sú si veľmi podobné a komplexná ResNet-50 je veľmi náročná na zobrazenie, môžeme zjednodušenie zobraziť ResNet-50 nasledovne:



V pôvodnej ResNet-50 architektúre sa strieda Conv blok s niekoľkými Identity blokmi. Avšak jediný rozdiel medzi Conv a Identity blokmi je ten, že pri skip prepojení sa pri Conv bloku používa 1x1 konvolúcia s batch normalizáciou. ResNet-50 má 26 miliónov parametrov. ResNet-50 síce nebol prvý kto použil skip prepojenia, ale rozhodne ich veľmi spopularizoval a začali sa od vtedy viac používať. Ale bol medzi prvými kto začali používať batch normalizáciu.

## Hyperparametre

Medzi hyperparametre konvolučnej neurónovej siete patria všetky parametre, ktoré sa sieť neučí, resp. také parametre, ktoré je potrebné pred učením zadefinovať. Medzi tieto hyperparametre patrí aj počet jednotlivých vrstiev, typ poolingu, veľkosti filtra, stride, padding, typ aktivačnej funkcie, loss funkcia, inicializácia váh, počet epoch, veľkosť várky, optimalizačný algoritmus a mnoho ďalšieho. Niektoré hyperparametre sú popísané vo predchádzajúcich kapitolách.

### Aktivačné funkcie

Jeden zo základných veľmi dôležitých hyperparametrov je typ aktivačnej funkcie. Pri doprednom šírení obrázka cez sieť sa vždy upraví podľa stanovených váh, ktoré sa učia v procese trénovania. Tento upravený vstup sa následne pozmení použitím aktivačnej funkcie. Jednotlivé aktivačné funkcie majú rôzne vplyvy na výstup. Úplne najzákladnejšou aktivačnou funkciou môže byť binárna aktivačná funkcia, ktorá buď vstup aktivuje (1) alebo nie (0). Ďalším veľmi základným príkladom aktivačnej funkcie je lineárna aktivačná funkcia, pri ktorej sa na upravený vstup neuplatňuje žiadna transformácia. Súčasne sa však používajú už len pri regresných problémoch na výstupnej vrstve. V súčasnosti sa teda najviac používajú nelineárne aktivačné funkcie, ktoré pomáhajú neurónom učiť sa viac komplexné štruktúry v dátach.

#### Sigmoid

Hlavným dôvodom, prečo sa používa sigmoidná aktivačná funkcia je jej rozsah (existuje medzi 0 a 1). Preto sa používa obzvlášť pri modeloch, pri ktorých je potrebné predikovať pravdepodobnosť ako výstup.



Funkcia je diferencovateľná a monotónna. Logistická sigmoidná aktivačná funkcia môže spôsobiť, že sa neurónová sieť zasekne počas trénovania. Softmax aktivačná funkcia je všeobecnejšou logistickou aktivačnou funkciou, ktorá sa používa na klasifikáciu viacerých tried.

#### TanH

TanH (Hyperbolický Tangens) aktivačná funkcia je podobná logistickej sigmoidnej ale je lepšia. Rozdah TanH aktivačnej funkcie je od -1 po 1.



Výhodou oproti sigmoidnej je, že záporné hodnoty sa mapujú na silno záporné blízke -1 a nulové hodnoty blízko nuly. Funkcia je takisto diferencovateľná a monotónna. Jej časté využitie je pri klasifikácií medzi dvoma triedami.

#### ReLU

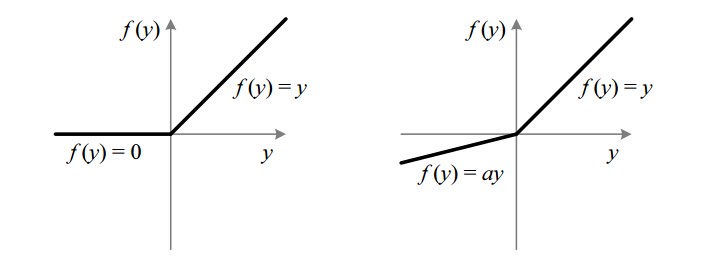
ReLU (Recified Linear Unit) je v súčasnosti najpoužívanejšia aktivačná funkcia vo svete. Jej využitie je najmä pri takmer všetkých konvolučných neurónových sieťach a pri hlbokom učení.



Ako vidno z obrázku, ReLU je napoly usmernená (zdola). f (z) je nula, keď z je menšie ako nula a f (z) je rovná z, keď z je nad alebo rovná nule, kde *z* predstavuje váhami upravený vstup. Na rozdiel od sigmoidnej a tanh funkcie je aj jej derivácia monotónna. Nevýhodou ReLU je, že všetky záporné hodnoty sa vynulujú a to znižuje schopnosť modelu natrénovať sa na správne na dátach.

#### Leaky ReLU

Ako názov napovedá, táto aktivačná funkcia je odvodená od ReLU. Snaží sa riešiť problém s vynulovaním záporných hodnôt pri ReLU.



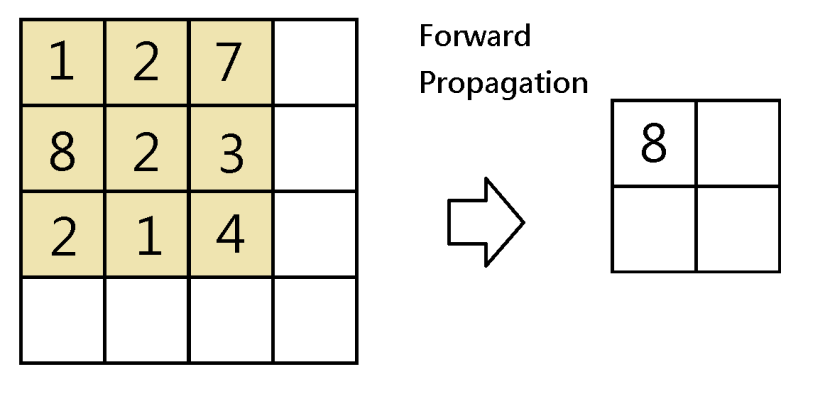
Pri Leaky ReLU je parameter *a=0,01*. Toto pomáha rozšíriť rozsah ReLU funkcie. V prípade že parameter *a* má inú hodnotu ako 0,01, nazýva sa takáto aktivačná funkcia Randomized ReLU.

### Pooling metódy

Ďalším hyperparametrom, ktorý je potrebné nastaviť pred trénovaním modelu je metóda použitá v pool vrstve (ak model obsahuje pool vrstvu). Keďže hlavnou úlohou pool vrstvy je zmenšiť rozmer obrázka. Pooling metóda definuje spôsob akým sa tento obrázok zmenší. V súčasnosti sa najčastejšie používa max a avg pooling. Avšak vo výnimočných prípadoch sa používa aj min pooling.

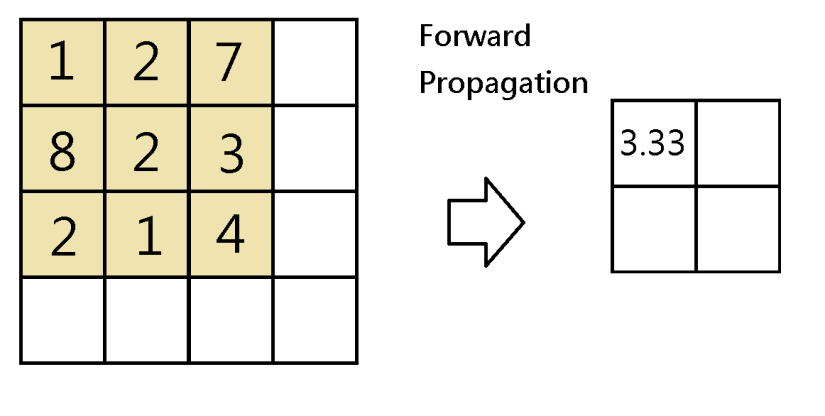
#### Max pooling

V súčasnosti najpoužívanejšou pooling metódou pri konvolučných neurónových sieťach je max pooling. Princípom tejto metódy je, že vždy z výrezu vstupu *x* o veľkosti *fxf* sa vyberie len pixel s najväčšou hodnotou a ten sa použije ako výstup *y* pre rad výstupu *r* a stĺpec *c*.



#### Avg pooling

Druhou najpoužívanejšou pooling metódou je avg pooling. Princípom tejto metódy je, že z výrezu vstupu *x* o veľkosti *fxf* sa vypočíta priemerná hodnota pixelov a táto sa použije ako výstup *y* pre rad výstupu *r* a stĺpec *c*.



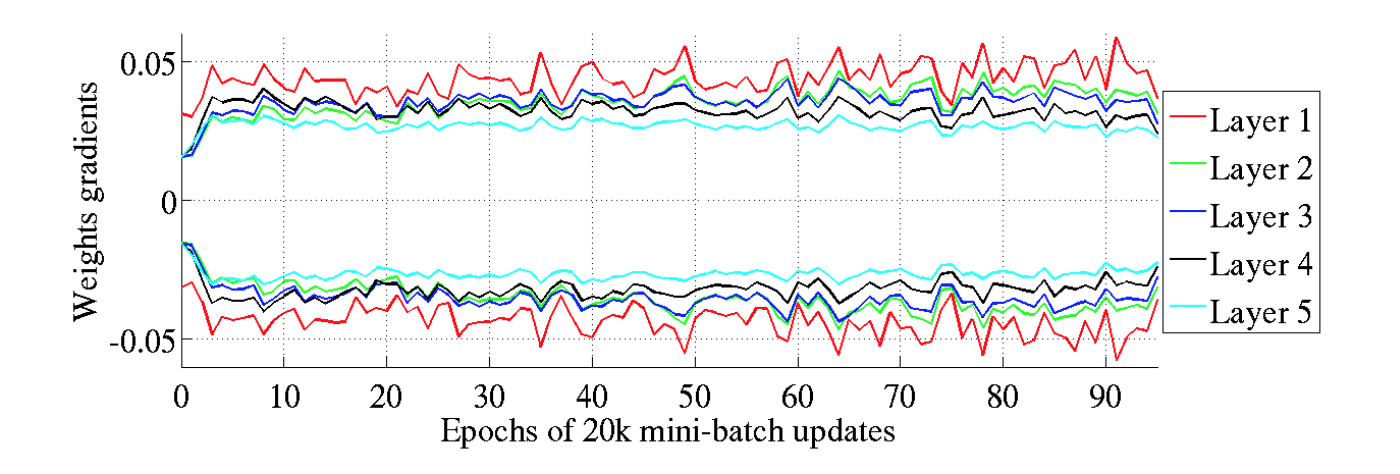
### Inicializácia váh

Pri trénovaní konvolučných neurónových sietí je veľmi dôležité správne inicializovať váhy vo všetkých vrstvách. Ak by tieto váhy boli príliš vysoké alebo príliš malé, prípadne boli zle rozdelené, učenie by bolo veľmi neefektívne a dokonca by to mohlo viesť k zastaveniu učenia. V súčasnosti sa najviac používajú dve hlavné inicializačné metódy Xavier a He.

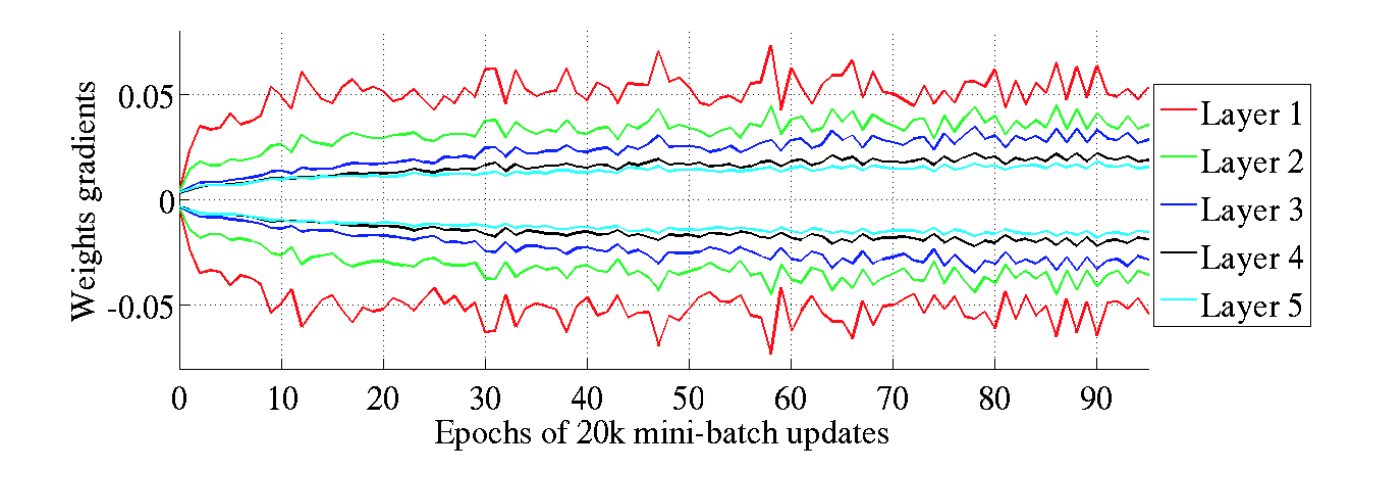
#### Xavierova inicializácia

Xavierova inicializácia nastaví váhy z náhodného rovnomerného rozdelenia, ktoré je ohraničené medzi:

Kde *ni* je počet prepojení predchádzajúcej vrstvy a *ni+1* je počet výstupov súčasnej vrstvy. Glorot a Bengio verili, že Xavierova inicializácia váh by zachovala rozptyl aktivácií a spätne šírených gradientov pri doprednom aj spätnom šírení po všetkých vrstvách siete. Vo svojich experimentoch zistili, že Xavierova inicializácia umožnila 5-vrstvovej sieti udržiavať takmer rovnaké odchýlky jej gradientov váh naprieč vrstvami.



Naopak, ukázalo sa, že pomocou „štandardnej“ inicializácie došlo k oveľa väčšej medzere v rozptyle medzi gradientmi váh v spodných vrstvách siete, ktoré boli vyššie, a tie v jej najvyšších vrstvách sa blížili k nule.



Glorot a Bengio ukázali, že siete, ktorých váhy boli inicializované Xavierovou inicializáciou dosiahli v úlohe klasifikácie obrázkov CIFAR-10 podstatne rýchlejšiu konvergenciu a vyššiu presnosť.

#### He inicializácia

Podobne ako Xavierova inicializácia váh, aj He inicializácia nastaví váhy z náhodného rovnomerného alebo normálneho rozdelenia, ktoré je ohraničené medzi:

Kde *ni* je počet prepojení predchádzajúcej vrstvy. He inicializácia dosahovala veľmi podobné výsledky ako Xavierova.

### Loss funkcie

Pri neurónových sieťach nedokážeme vypočítať presné váhy pretože obsahujú príliš veľa neznámych premenných. Preto namiesto exaktného výpočtu je problém učenia stanovený ako optimalizačný a daný algoritmus prechádza priestor možných kombinácií váh, ktoré by model mohol použiť aby dosiahol dobré, resp. dostatočne dobré predikcie. Zvyčajne sa model trénuje pomocou Stochastic Gradient Descent (xxx) optimalizačným algoritmom a váhy sa nastavujú pomocou backpropagation (xxx) algoritmom. Model určí predikcie a z nich následne vypočíta chybu. Pomocou SGD model hľadá takú kombináciu váh, aby bola chyba čo najmenšia. Na vyhodnotenie vhodnosti daných váh sa používa funkcia, ktorá sa nazýva cenová alebo stratová (loss) funkcia. Významnosť tejto funkcie spočíva v predpoklade, že čím je hodnota loss funkcie menšia, tým je model lepší. Preto je veľmi dôležité použiť takú loss funkciu, ktorá najlepšie vystihne dôležité vlastnosti daného problému. Existuje niekoľko preddefinovaných funkcií z ktorých najpoužívanejšie sú asi Mean Squared Error (MSE) a Cross-Entropy. Avšak často sa používajú aj iné odvodené od týchto ako napríklad Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), Root Mean Absolute Error (RMSE) či Binnary Cross-Entropy.

#### MSE

Hodnota tejto loss funkcie sa vypočíta ako priemerný štvorcový rozdiel medzi predikovanými a skutočnými hodnotami. Výsledná hodnota je vždy kladná bez ohľadu na znamienko odhadovaných alebo skutočných hodnôt. Najlepšia hodnota tejto loss funkcie je 0.

Kde *n* je počet pozorovaní, *Yi*je skutočná hodnota a *Y^i* je predikovaná hodnota. Jej najčastejšie využitie je v regresných problémoch, kde predikujeme reálnu hodnotu namiesto triedy a teda výstupom je len jeden neurón s lineárnou aktivačnou funkciou.

#### Cross-Entropy

Každá predikovaná pravdepodobnosť triedy je porovnaná so skutočnou triedou a skóre je vypočítané tak, že pokutuje pravdepodobnosti podľa vzdialenosti od očakávanej hodnoty. Pokuta je logaritmická, vďaka čomu pri malých rozdieloch (0,1 alebo 0,2) dáva malé skóre a pri veľkých rozdieloch (0,9 alebo 1) dáva podstatne väčšie skóre. Hodnota loss funkcie sa minimalizuje, takže model s menšou hodnotou loss funkcie považujeme za lepší. Model ktorý dokonalo predikuje pravdepodobnosti má hodnotu loss funkcie 0.

Pri tejto loss funkcií sa sčíta súčin skutočnej pravdepodobnosti triedy *i* (*pyi*) a prirodzený logaritmus predikovanej pravdepodobnosti triedy *i* (*py^i*). Za predpokladu, že každý vstup patrí práve do jednej triedy, platí že ak triedu, do ktorej patrí vstup je označená *c*, potom:

Teda pri určovaní práve jednej triedy je tvar loss funkcie zjednodušený:

Pri výpočte gradientu sa používa derivácia loss funkcie pre danú triedu:

Využitie tejto loss funkcie je hlavne pri problémoch predikcie jednej triedy danému vstupu. Základný tvar Cross-Entropy sa môže použiť aj na problémy predikcie viacerých tried pre jeden vstup.

## Učenie siete

# Návrh a implementácia CNN

Vlastná konvolučná neurónová sieť bola implementovaná v jazyku Python s využitím vývojového prostredia PyCharm IDE. V práci bol použitý Python 3.6 interpreter.

Implementácia bola rozdelená do troch častí. V prvej časti implementácie bo implementovaný len dopredný spôsob šírenia obrázku sieťou. V rámci toho bolo naimplementované jadro siete (trieda cnn), softmax vrstva (trieda softmax), konvolučná vrstva (trieda convolutional), pool vrstva (trieda pool) a taktiež rozhranie pre všetky vrstvy (interface ILayer). V druhej časti implementácie bolo postupne pridávané spätné šírenie obrázka sieťou do jednotlivých vrstiev softmax, pool a konvolučnej. Taktiež bola pridaná plne prepojená vrstva (trieda dense) s dopredným aj spätným šírením obrázka sieťou. Tretia časť implementácie zahŕňala rôzne algoritmické vylepšenia pre väčšiu rýchlosť a efektívnosť učenia.

Sieť bola navrhnutá tak, aby bolo možné si jednoducho vytvoriť vlastnú architektúru s použitím štyroch hlavných typov vrstiev konvolučných neurónových sietí:

* Konvolučná vrstva
* Pool vrstva
* Dense vrstva
* Softmax vrstva

Všetky vrstvy implementujú spoločný interface ILayer, ktorý má tri hlavné metódy forward, backward a updateWeights. Vďaka tomu je možné jednoducho pridávať rôzne vrstvy do jedného modelu, ktorý reprezentuje trieda CNN. Aplikácia taktiež poskytuje možnosť zadefinovať hodnoty rôznych hyperparametrov siete:

* Počet a veľkosť filtrov konvolučnej vrstvy
* Stride konvolučnej a pool vrstvy
* Padding konvolučnej vrstvy
* Aktivačnú funkciu
* Typ inicializácie
* Optimalizér
* Loss funkciu
* Počet predikovaných tried

## Jadro

Jadro celej aplikácie predstavuje trieda CNN, ktorá vytvára model konvolučnej neurónovej siete. Teda do tohto modelu je možné pridávať vrstvy, trénovať model a získať predikcie. Trieda taktiež vypočíta hodnotu danej loss funkcie a presnosť modelu.

Trieda má 2 hlavné atribúty:

* **layers** – je to atribút typu *List*, ktorý v sebe uchováva všetky vrstvy modelu, ktoré implementujú interface *ILayer*
* **num\_outputs** – tento atribút typu *Integer* určuje počet výstupných neurónov, tento atribút je potrebný pre niektoré loss funkcie.

Ďalej trieda obsahuje 4 metódy:

* **\_\_init\_\_** - inicializačná void metóda triedy
* **add** – void metóda, ktorá má ako parameter vrstvu a pridá túto vrstvu do zoznamu vrstiev *layers*
* **forwardBatch** – metóda, ktorá má ako parameter obrázky a triedy obrázkov. Táto metóda slúži na predikciu triedy obrázkov, všetky obrázky preženie postupne po vrstvách až k poslednej vrstve, ktorá určí pravdepodobnosti jednotlivých tried pre všetky obrázky. Následne vypočíta a vráti gradienty, priemernú presnosť a hodnotu danej loss funkcie pre dané obrázky a vopred špecifikovaný model.
* **train** – metóda slúži na trénovanie modelu. Ako parameter dostane obrázky, triedy obrázkov a *learn rate* (veľkosť kroku učenia). Najskôr metóda zavolá metódu *forwardBatch* na určenie tried a výpočet gradientu pre všetky obrázky. Výsledný gradient sa použije ako vstup do *backward* metódy poslednej vrstvy modelu, ktorá vypočíta gradient pre ďalšiu vrstvu. Následne sa zavolá metóda *updateWeights*, s danou hodnotou *learn rate* a nastavia sa váhy vo vrstve. Postupne sa takto upravia váhy pre všetky vrstvy modelu. Návratová hodnota metódy je priemerná loss funkcia a presnosť pre všetky dané obrázky.

## Vrstvy

Druhou hlavnou časťou aplikácie sú jednotlivé vrstvy CNN. Všetky vrstvy implementujú rozhranie **ILayer**, aby bolo možné ich spolu nakopiť v jednom modely.

### Rozhranie ILayer

Toto rozhranie obsahuje 3 hlavné metódy, potrebné na predikciu tried obrázkov a učenie modelu:

* **forward** – táto metóda slúži na posunutie obrázku cez danú vrstvu. Má jeden parameter *prev\_layer*, ktorý predstavuje vstup vrstvy, resp. výstup predchádzajúcej vrstvy. Metóda vracia už upravený obrázok alebo vstup danou vrstvou.
* **backward** – táto metóda slúži na výpočet gradientu, podľa ktorého sa následne upravujú váhy a ktorý sa ďalej pošle predchádzajúcej vrstve. Parameter metódy *prev\_layer* predstavuje výstup *backward* metódy ďalšej vrstvy, resp. podľa gradientu upravený výstup danej vrstvy.
* **updateWeights** – metóda upravuje váhy danej vrstvy podľa gradientu vypočítaného v metóde *backward* a parametru *learn rate*, ktorý je parametrom metódy.

### Konvolučná vrstva

Táto vrstva má na rozdiel od klasickej NN vrstvy váhy určené filtrami, ktoré sa učia hľadať určité vlastnosti obrázku, ktoré by mohli byť významné pre rozlíšenie jednotlivých tried. Avšak rovnako ako aj klasické NN vrstvy, často aj konvolučná vrstva používa aktivačné funkcie na výstup konvolúcie.

Základné atribúty vrstvy zahŕňali len počet filtrov, ich veľkosť, stride a padding. V inicializačnej metóde sa vytvorilo viacrozmerné pole filtrov. Všetky filtre sa inicializovali podľa *Xavierovej inicializácie*.

Pred samotnou konvolúciou je potrebné vypočítať veľkosť výstupného obrázka. Veľkosť výstupu sa inicializuje podľa veľkosti obrázka a hodnôt hyperparametrov stride, padding a veľkosti filtra. Šírku a výšku výsledného obrázku vypočítame podľa vzorca:

Keďže predpokladáme, že obrázok je vždy štvorcový, tak šírka aj výška bude rovnaká, preto sa počíta rovnakým vzorcom, kde *x* predstavuje šírku/výšku výstupu, *w* je šírka/výška vstupného obrázka, *pad* je hodnota paddingu, *f* je veľkosť filtra a *stride* je hodnota stride (veľkosť kroku). Pred samotnou operáciou konvolúcie ak bola hodnota hyperparametra padding väčšia ako 0 upraviť vstupný obrázok pridaním príslušného paddingu. Operácia konvolúcie bola implementovaná pomocou viacerých cyklov, kde sa vždy vyberie zo vstupného obrázka výrez o veľkosti filtra a tento výrez sa vynásobí s filtrom po prvkoch a výsledné pole sa sčíta a výsledok sa uloží na príslušné miesto výstupného poľa.

*Outputn,f,r,c* označuje výstupný pixel pre obrázok *n*, filter *f*, riadok *r* a stĺpec *c*. *Depth* predstavuje hĺbku vstupného obrázka, *f* označuje veľkosť filtra. *Input* je vstupný obrázok, *filter* sú filtre a *b*predstavujú biasy.

V druhej fáze implementácie bolo implementované **spätné šírenie** obrázka vrstvou spolu s**Stochastic Gradient Descent** (SGD) metódou učenia filtrov. Pre spätné šírenie a učenie flitrov si potrebujeme pri doprednom šírení ukladať šírku, výšku a hĺbku výstupu a taktiež samotný vstupný obrázok, ktorý budeme potrebovať pri učení.

**Spätné šírenie** tiež využíva operáciu konvolúcie. Ako vstup tejto operácie je gradient z predchádzajúcej vrstvy. Tento gradient má rovnaký rozmer ako výstup konvolúcie dopredného šírenia. Preto je potrebné ho zväčšiť na veľkosť vstupu. To sa vykoná pridaním príslušne veľkého paddingu. Následne sa vykoná operácia konvolúcie podobná doprednému šíreniu. Avšak ak má vstup pri doprednom šírení hĺbku viac ako 1, tak každý filter má tiež rovnakú hĺbku a pri konvolúcií sa výrez pre danú hĺbku vynásobí s filtrom z rovnakej hĺbky a výsledné súčiny sa sčítajú cez celú hĺbku. Pri spätnom šírení však gradient má hĺbku rovnakú ako je počet filtrov danej vrstvy. Preto pri konvolúcií spätného šírenia sa vynásobí príslúšná hĺbka gradientu s príslušným filtrom a výsledné súčiny sa sčítajú cez všetky filtre pre jednu hĺbku pre jednu vrstvu gradientu. Výsledok tejto konvolúcie spätného šírenia je gradient, pre ďalšie vrstvy.

*G\_outn,d,r,c* označuje výstupný gradient pre pixel obrázku *n*, hĺbky *d*, riadka *r* a stĺpca *c*. *Filters* predstavuje počet filtrov danej vrstvy, *f\_s* označuje veľkosť filtra. *G\_inp* je vstupný gradient, *filter* sú filtre a *b*predstavujú biasy.

Po vypočítaní gradientu nasleduje proces učenia. V konvolučnej vrstve proces učenia váh filtrov sa vykoná vynásobením gradientu, ktorý bol vstupom do spätného šírenia danej vrstvy a vstupom dopredného šírenia. Každý filter vypočítame súčtom vynásobeného gradientu pre jeden pixel vstupného gradientu a výrezom o veľkosti filtra pre vstup.

*Filterf,d,a,b* predstavuje jednu novú váhu filtra *f* pre hĺbku filtra *d*, v rade *a*a stĺpci *b*. *N* predstavuje počet obrázkov, resp. veľkosť batchu. *H* a *w* predstavujú šírku a výšku vstupného gradientu. Podľa SGD sa tieto nové filtre sa následne vynásobia parametrom *learn\_rate* (veľkosť kroku učenia) a výsledok sa odpočíta od súčasných váh filtrov a tým dostaneme nové váhy pre všetky filtre upravené podľa gradientu.

V aplikácií je konvolučná vrstva definovaná triedou **Convolutional**, ktorá má nasledujúce metódy:

* **\_\_init\_\_** - inicializačná metóda vrstvy. Inicializuje hlavné parametre triedy, ktoré sú aj parametrami tejto metódy:
  + *prev\_layer\_size* – veľkosť predchádzajúcej vrstvy, resp. hĺbka vstupného obrázka
  + *num\_filters* – počet filtrov
  + *filter\_size* – veľkosť filtrov. Číslo je typu *Integer*, pretože všetky filtre sú štvorcové, teda pri *filter\_size = 3*, budú mať všetky filtre rozmer 3x3
  + *stride* – určuje veľkosť kroku pre operáciu konvolúcie, po koľkých pixeloch sa má posúvať filter cez obrázok
  + *padding* – určuje koľko nulových pixelov okolo obrázka je treba pridať. Používa sa, ak sa významné časti obrázka nachádzajú pri kraji, alebo ak chceme zachovať veľkosť obrázka aj po vykonaní operácie konvolúcie
  + *initializer* – *String* typ inicializácie váh filtrov
  + *activation* – *String* typ aktivačnej funkcie, ktorá sa aplikuje na výstup operácie konvolúcie

V metóde sa taktiež inicializujú váhy a bias

* **initializeWeights** – metóda inicializuje váhy filtrov konvolučnej vrstve podľa zadaného typu inicializácie, ktorý je ako parameter metódy.
* **convolve** – metóda vykonáva operáciu konvolúcie nad obrázkami, ktoré sú zadané ako parameter metódy. Vstupný parameter metódy sú obrázky, následne sa vykoná konvolúcia so všetkými filtrami pre všetky obrázky a výsledný obrázok sa vracia ako návratová hodnota.
* **backprop** – metóda má ako parameter gradientom upravený výstup, ktorý sa upraví na požadovanú veľkosť, aby po vykonaní konvolúcie nad týmto výstupom mal obrázok veľkosť vstupu danej vrstvy. Takýto upravený vstup vráti metóda ako návratovú hodnotu pre ďalší krok backpropagation.
* **filterSet** – podľa zadaného upraveného výstupu metóda vypočíta priemerný gradient pre všetky obrázky, ktorý sa následne vynásobí veľkosťou kroku učenia alebo sa upraví nejakým optimalizačným algoritmom (napr. Adam).
* **activate** – táto metóda sa zavolá po vykonaní operácie konvolúcie nad obrázkom a „aktivuje“ tento výstup zavolaním konkrétnej aktivačnej funkcie. Názov aktivačnej funkcie je parameter tejto metódy a jej výstup je výstupom danej aktivačnej funkcie.
* **derivate** – ak bola pri procese backpropagation na výstup aplikovaná aktivačná funkcia je potrebné výstup vynásobiť deriváciou tejto aktivačnej funkcie. Na to slúži táto metóda, ktorá má ako parameter názov aktivačnej funkcie a výstupom je výstup derivácie aktivačnej funkcie.

### Plne prepojená vrstva

V konvolučnej neurónovej sieti predstavuje plne prepojená vrstva klasickú vrstvu neurónovej siete, kde všetky neuróny sú prepojené so všetkými vstupmi a každému prepojeniu je priradená váha. Každý pixel obrázka vstupujúceho do plne prepojenej vrstvy predstavuje jeden výstupný neurón vrstvy. Keďže výstup má na rozdiel od konvolučných vrstiev len jeden rozmer (počet neurónov siete) po plne prepojenej vrstve už nenasleduje konvolučná vrstva ale len ďalšie plne prepojené vrstvy.

Hlavnou úlohou filtrov konvolučnej vrstvy je naučiť sa a nájsť rôzne vlastnosti obrázkov (*feature extraction*) a hlavnou úlohou plne prepojenej vrstvy je určiť významnosť jednotlivým vlastnostiam a naučiť sa kedy sú, ktoré vlastnosti významné (*feature selection*).

V aplikácií je plne prepojená vrstva definovaná triedou **dense** a špeciálny prípad plne prepojenej vrstvy používajúci sa ako výstupná kategorizačná vrstva je trieda **softmax**. Medzi hlavné metódy triedy dense patria:

* **\_\_init\_\_** - inicializačná metóda triedy. Inicializuje hlavné parametre triedy, ktoré sú aj parametrami tejto metódy:
  + *layer\_size* – počet neurónov vrstvy
  + *prev\_layer\_size* – počet neurónov predchádzajúcej vrstvy. V prípade že je predchádzajúca vrstva konvolučná, tento parameter udáva veľkosť hĺbky výstupného obrázka.
  + *initializer* – typ inicializácie váh filtrov určený v podobe reťazca. Základná hodnota tohto parametra je *xavier\_uniform,* čo predstavuje inicializáciu váh rovnomerným rozdelením upraveným podľa Xavierovej inicializácie.
  + *activation* – typ aktivačnej funkcie určený v podobe reťazca. Základná hodnota tohto parametra je *relu,* ktorá je najpoužívanejšia aktivačná funkcia.

Metóda taktiež inicializuje váhy prepojení a ich biasy.

* **initializeWeights** – metóda inicializuje váhy všetkých prepojení podľa zadaného typu inicializácie, ktorý je ako parameter metódy.
* **activate** – táto metóda sa zavolá nad vstupom pri predikovaní triedy obrázka. Metóda najskôr zavolá metódu *\_\_calculate\_potential\_\_*, ktorá vynásobí vstup váhami a pripočíta biasy. Následne sa tento výstup „aktivuje“ zavolaním konkrétnej aktivačnej funkcie. Názov aktivačnej funkcie je parameter tejto metódy a jej výstup je výstupom danej aktivačnej funkcie.
* **derivate** – pri procese backpropagation sa na výstup aktivačnej funkcie aplikuje derivácia danej aktivačnej funkcie zavolaním príslušnej metódy. Názov aktivačnej funkcie je parametrom metódy a výstupom je výstup derivácie aktivačnej funkcie.
* **\_\_calculate\_potential\_\_** - metóda je volaná len z metódy *activate* a ako parameter je vstup danej vrstvy, ktorý sa v tejto metóde vynásobí s príslušnými váhami jednotlivých spojení a ku každému neurónu sa ešte pripočíta hodnota biasov. Výsledok metóda vráti ako návratovú hodnotu.
* **forward** – metóda prejde postupne cez všetky vstupy, vyrovná ich do jednorozmerného poľa a na každý vstup zavolá metódu *activate*. Návratová hodnota je pole „aktivovaných“ všetkých vstupov.
* **backprop** – parametrom metódy je upravený výstup z predchádzajúcej vrstvy pri procese backpropagation. Metóda postupne prechádza všetky takto upravené výstupy, vynásobí ich deriváciou aktivačnej funkcie a váhami a takto upravené vstupy následne vráti ako pole v návratovej hodnote.

### Softmax

Špeciálnym prípadom plne prepojenej vrstvy je softmax vrstva, ktorá sa v sieti nachádza vždy najviac raz a používa sa ako výstupná vrstva, pretože jej úlohou je určiť podľa daných vstupov pravdepodobnosť každej triede. Na základe týchto pravdepodobností sa následne robí predikcia (trieda s najväčšou pravdepodobnosťou = predikovaná trieda).

Softmax vrstva je vlastne plne prepojená (dense) vrstva, takže sa skladá z neurónov, ktoré sú všetky prepojené so všetkými vstupmi. Rovnako tak aj každé prepojenie má svoju váhu, preto je potrebné na začiatok inicializovať váhy aj pre túto vrstvu. Váhy sa ako pri ostatných vrstvách inicializujú *Xavierovou inicializáciou*.

Keďže už nie je potrebné mať obrázok vo viacrozmernom poli, obrázok sa vyrovná do vektora pomocou *flatten* metódy z knižnice *numpy.* Následne sa tento vstup vynásobí s váhami a pripočíta sa k ním bias.

*Tc* predstavuje výsledok skalárneho súčinu triedy *c* vrátane biasu *b*. *Inputi*je výstup *i* predchádzajúcej vrstvy a *wc,i* je váha prislúchajúca danému vstupu. Ďalej tento výstup využijeme pri výpočte pravdepodobností jednotlivých tried. Aby sme predišli pretekaniu pri počítaní exponentov v softmax aktivačnej funckií, vynulujeme maximálnu hodnotu tak, že od všetkých hodnôt *tn* pre daný vstup odpočítame maximálnu hodnotu a pripočítame malú hodnotu ε aby nevzniklo delenie nulou.

Potom vypočítame jednotlivé pravdepodobnosti *pc* pre každú triedu *c* zo všetkých *n* tried nasledovne:

Následne sa vypočíta gradient podľa danej loss funkcie v jadre aplikácie a tento gradient sa použije ako vstup do spätného šírenia softmax vrstvy. Z tohto gradientu následne vypočítame gradient pre ďalšie vrstvy. Najskôr si definujeme *S* ako súčet exponentov *t* pre všetky triedy:

Potom pre všetky triedy *k*, ktoré nie sú predikovaná trieda *c* deriváciu softmax aktivačnej funkcie vypočítame nasledovne:

A pre triedu *c*:

Následne túto deriváciu vynásobíme vstupným gradientom a dostaneme gradient softmax vrstvy. Keď tento gradient vynásobíme váhami dostaneme gradient pre ďalšie vrstvy. Keď však tento gradient vynásobíme vstupom, dostaneme nové váhy. Nové váhy sa podľa SGD vynásobia learn\_rate parametrom a odčítajú sa od súčasných váh. Tým dostaneme gradientom upravené váhy.

V aplikácií je táto vrstva definovaná triedou **softmax** a medzi jej hlavné metódy patria:

* **\_\_init\_\_** - inicializačná metóda triedy. Inicializuje hlavné parametre triedy, ktoré sú aj parametrami tejto metódy:
  + *layer\_size* – počet neurónov vrstvy
  + *prev\_layer\_size* – počet neurónov predchádzajúcej vrstvy. V prípade že je predchádzajúca vrstva konvolučná, tento parameter udáva veľkosť hĺbky výstupného obrázka.
  + *initializer* – typ inicializácie váh filtrov určený v podobe reťazca. Základná hodnota tohto parametra je *xavier\_uniform,* čo predstavuje inicializáciu váh rovnomerným rozdelením upraveným podľa Xavierovej inicializácie.

Metóda taktiež inicializuje váhy prepojení a ich biasy.

* **initializeWeights** – metóda inicializuje váhy všetkých prepojení podľa zadaného typu inicializácie, ktorý je ako parameter metódy.
* **forward** – metóda vynásobí vstupy príslušnými váhami, pripočíta biasy a následne určí podľa výsledku pravdepodobnosť všetkým triedam. Tieto pravdepodobnosti vracia ako návratovú hodnotu.
* **backprop** – parametrom metódy sú upravené pravdepodobnosti podľa loss funkcie. Metóda z týchto upravených pravdepodobností vypočíta upravený vstup, ktorý vráti ako návratovú hodnotu pre ďalšie vrstvy.

### Pooling vrstva

Posledná vrstva, ktorá bola implementovaná je pooling vrstva. Jej hlavnou úlohou je zmenšiť obrázok, tak aby všetky dôležité informácie v obrázku boli zachované. Používa sa vždy za konvolučnou vrstvou.

Naimplementované boli dva základné a najpoužívanejšie pooling metódy max a average. Rovnako ako pri konvolučnej vrstve, aj tu najskôr stanovíme veľkosť výstupu, keďže pooling má za úlohu zmenšiť obrázok, tak aby ponechal dôležité vlastnosti obrázka. Na výpočet šírky a výšky sa používa veľmi podobný vzorec:

Jediná zmena oproti konvolučnej vrstve je, že pri pooling sa nepoužíva padding. Väčšinou sa používa rovnaká veľkosť stride ako filtra (*f = stride*). Teda stačí šírku alebo výšku obrázka podeliť veľkosťou filtra a máme šírku alebo výšku výsledného obrázka. Podobne ako pri konvolučnej vrstve je obrázok prechádzaný viacerými cyklami, tak že vždy vyberie výrez obrázka o veľkosti filtra a následne sa z neho vyberie maximum alebo sa vypočíta priemer z celého výrezu a výsledok sa priradí na príslušný pixel výstupného obrázka.

*Outputn,d,r,c*predstavuje pixel výstupného obrázka *n* pre hĺbku obrázka *d*, v rade *r*a stĺpci *c*. *Input* predstavuje vstup a *f* je veľkosť poolingu.

Keďže pooling vrstva nemá žiadne váhy v druhej fáze implementácie sme implementovali len spätné šírenie obrázka sieťou. Keďže výstup pooling vrstvy je menší ako vstup, je potrebné gradienty, ktoré prídu ako vstup pri spätnom šírení upraviť správne tak, aby mali rovnaký rozmer ako vstup do dopredného šírenia. Max pooling a average pooling používajú rôzne spôsoby spätného šírenia. Pri metóde max pooling si musíme uložiť indexy všetkých prvkov, ktoré boli vybrané. Následne pri spätnom šírení nám stačí gradienty preusporiadať na tieto pozície. Pri metóde average pooling stačí keď gradienty rozšírime tak, že každý gradient rozšírime na rozmer filtra a následne výsledné gradienty vydelíme druhou mocninou veľkosti filtra.

V aplikácií boli tieto pooling metódy implementované v triede **pool** s metódami:

* **\_\_init\_\_** - inicializačná metóda triedy. Inicializuje hlavné parametre triedy, ktoré sú aj parametrami tejto metódy:
  + *stride* – veľkosť kroku, ktorý určuje o koľko sa má posúvať filter, ktorý zmenšuje obrázok
  + *filter\_size* – veľkosť filtra, ktorý prechádza obrázkom a zmenšuje jeho veľkosť. Rovnako ako pri konvolučnej vrstve je tento parameter hodnoty *Integer*, pretože sú filtre vždy štvrocové a teda rovnako pri hodnote *filter\_size = 2* bude mať filter rozmer 2x2.
  + *pool\_type –* typ poolingu typu *String* (max/avg)
* **maxPooling** – metóda filtrom postupne prejde cez celý vstupný obrázok a vždy vyberie z filtra jedno najväčšie číslo a takto sa zmenší veľkosť a zachovajú sa najvýznamnejšie pixely
* **avgPooling** *–* metóda podobne ako *maxPooling* prejde filtrom cez celý obrázok ale namiesto vybratia najväčšieho čísla vypočíta priemer zo všetkých čísel.

Síce trieda nemá žiadne váhy, ktoré sa učí, pri procese backpropagation musí upraviť výstup tak aby mohol pokračovať v procese. Na úpravu (zväčšenie) výstupu používa trieda dve metódy:

* **maxBackprop** – v metóde *maxPooling* si metóda ukladala indexy vybratých prvkov. Táto metóda postupne prechádza výstupom a vkladá prvky na zväčšený obrázok podľa indexov uložených v metóde *maxPooling* a teda všetky pixely, mimo tých, ktoré obsahovali vybraný pixel majú hodnotu 0.
* **avgBackprop** *–* každý pixel výstupu vynásobí 0.25 a rozšíri ho na rozmer filtra a tým sa obrázok rozšíri do požadovanej veľkosti.

## Adam

Po naimplementovaní celej konvolučnej neurónovej siete sme ju otestovali, či sa bude naša sieť učiť. Pomocou našej aplikácie sme zostavili model CNN podľa architektúry LeNet-5. Následne sme túto sieť otestovali na MNIST datasete. Tento dataset obsahuje 60000 obrázkov rukou písaných čísiel od 0 po 9 o veľkosti 28x28 pixelov. Teda úloha siete je naučiť sa predikovať číslo, ktoré je napísané na obrázku. Test sme vykonali na podmnožine datasetu, konkrétne 1000 obrázkov na trénovanie a 250 na testovanie. Ako loss funkciu sme použili klasickú kategorickú cross-entropy. Beh trval 1841447 ms (viac ako 30min) a výsledná presnosť na testovacom datasete bola 88,8% a hodnota loss funkcie 0,4098. Z výsledkov je jasné, že sieť sa učí, avšak doba učenia bola príliš vysoká a síce sme trénovali sieť len na menšom datasete na 10 epochách, výsledky by mohli byť aj lepšie. Preto sme sa rozhodli implementovať namiesto SGD lepší optimalizátor Adam. Ktorý sme popísali v kapitole xxx. Po implementovaní Adama do našej siete sme ju otestovali na rovnakej sieti s rovnakými dátami. Beh trval približne rovnako, avšak presnosť na testovacom datasete bola dokonca nižšia 87,2% a hodnota loss funkcie 0,4177. To môže byť zapríčinené tým, že sa nastavovali nové váhy sieti po každom jednom obrázku, Preto je aj vývoj loss funkcie a presnosti osciluje. Adam lepšie pracuje pri učení po várkach (kapitola 3.4).

Na výsledných grafoch je vidno, že s využitím Adam optimalizátora nám hodnota loss funkcie na začiatku veľmi rýchlo klesla a rovnako aj presnosť veľmi rýchlo stúpla v porovnaní s SGD, avšak neskôr sa už moc od seba nelíšili. Navyše pri použití Adama hodnoty loss funkcie a presnosť viac oscilovala.

## Batch

Doba trvania experimentu bola príliš vysoká a ani s Adam optimalizátorom sme presnosť moc nezlepšili. Tým že sieť sa učí podľa gradientov stanovených z každého jedného obrázku, môže sa stať že hodnota loss funkcie viac osciluje, obzvlášť pri použití Adam optimalizátora namiesto SGD. Preto sme sa rozhodli implementovať do našej aplikácie možnosť dopredného a spätného šírenia obrázkov po várkach (batch). Namiesto posielania obrázkov datasetu po jednom cez sieť a po každom upravovať váhy podľa výsledného gradientu, táto metóda posiela naraz cez sieť celú várku obrázkov (16,32,64,128,....) a až po tejto celej várke sa upravia váhy vrstiev podľa priemerného gradientu vypočítaného zo všetkých obrázkov celej várky. Po implementovaní učení po várkach do našej siete sme ju otestovali na rovnakej sieti s rovnakými dátami ale už namiesto SGD optimalizátora sme použili Adama. Beh bol len o trochu rýchlejší, pretože pri aktuálnej implementácií sa celá várka šíri postupne, akurát proces učenia sa spustí až po celej várke. Avšak presnosť aj hodnota loss funkcie bola už výraznejšie lepšia ako pri učení po 1. Presnosť na testovacom datasete po 10 epochách bola 92,4% a hodnota loss funkcie 0,3196.

Na grafoch je vidno, že síce pri učení po várkach nemalo taký prudký začiatok ako pri učení po 1 ale jej ustaľovanie je viac plynulé ako pri učení po 1.

## Algoritmické zrýchlenie

Pridaním Adama a učenia po várkach sme síce dokázali dosiahnuť lepšie výsledky, však rýchlosť sa moc nezlepšila. Preto sme sa rozhodli upraviť proces dopredného a spätného šírenia tak, aby bolo schopné spracovať celú várku naraz a nemusela sa várka šíriť po jednom. Pre toto vylepšenie sme využili knižnicu *numpy*, ktorá dokáže pracovať s viacrozmernými poľami. Výsledný algoritmus sme naprogramovali tak, že dokáže spracovať celú várku cez celú hĺbku várky naraz. Čím bola várka väčšia tým lepšie rýchlosti sme dosiahli. Vylepšenie sme implementovali najmä do konvolučnej a pool vrstvy, pretože tieto vrstvy stále zaberali prevažnú väčšinu výpočtového času. Znova po naimplementovaní sme otestovali našu sieť na rovnakej architektúre s rovnakými parametrami, avšak už s využitím nových algoritmov, optimalizátorom Adam a učenia cez várky. Výsledky boli rovnaké avšak beh trval už len 196586ms, čo je len niečo málo cez 3min. Teda nový algoritmus pre dopredné a spätné šírenie nám zrýchlil sieť takmer 10-násobne.

Výslednú aplikáciu sme následne otestovali na našom datasete znamienok. Na testovanie sme použili zase sieť LeNet-5. Ako dáta sme použili znova 1000 obrázkov na trénovanie a 250 na testovanie. Rovnako ako pri predošlých testoch sme trénovali sieť po dobu 10 epoch. Výsledná presnosť modelu bola 60,2% a hodnota loss funkcie bola 0,6991. Výsledná presnosť je nižšia ako na MNIST datasete. To môže byť spôsobené aj tým, že máme predikovať len 2 triedy, ktoré vyzerajú celkom podobne. Niektoré znamienka ťažko voľným okom rozoznáme či sú zhubné alebo nie, kdežto pri rukou písaných číslach to rozoznáme hneď. Hlavným dôvodom, že LeNet-5 sa tak dobre učí na MNIST datasete je preto, že bola špeciálne navrhnutá na tento dataset.

## Návrh vlastnej architektúry CNN

Keďže chceme aby výsledky boli čo najpresnejšie, tak musíme vykonať dostatočne veľký počet epoch vo viacerých behoch. Problém je, že pri väčších sieťach ako AlexNet, ResNet či VGG-16 by doba učenia bola príliš časovo náročná. Taktiež presnosť LeNet-5 nedosahuje zrovna najlepšie výsledky, pretože táto architektúra nebola navrhnutá na náš dataset. Preto sme sa rozhodli navrhnúť vlastnú architektúru CNN konkrétne na náš dataset.

Rozhodli sme sa pre jednoduchú architektúru len jednej konvolučnej a jednej plne prepojenej vrstvy. Keďže LeNet-5 požaduje na vstup obrázky o veľkosti 28x28, môže to znamenať, že niektoré vlastnosti obrázka sa stratia pri zmenšovaní. Preto sme sa rozhodli ako vstup do našej architektúry brať obrázky o veľkosti **64x64** pixelov. Ako aktivačnú funkciu sme použili **ReLU** a medzi konvolučnú a plne prepojenú vrstvu sme dali **max** **pooling**. Na konvolučnú vrstvu sme použili filtre o veľkosti **3x3** a veľkosťou kroku 2. Max pooling má nastavenú veľkosť filtra **3x3** s veľkosťou kroku 3. Počet filtrov konvolučnej vrstvy a počet neurónov v plne prepojenej vrstve sme sa rozhodli stanoviť algoritmom, ktorý vyskúšal všetky kombinácie mocniny 2. Pre počet filtrov to bolo od 4 po 64. Pre počet neurónov od 8 do 512. Pre rýchlejšie stanovenie výsledkov sme využili knižnicu keras. Pomocou tejto knižnice sme zostavili našu sieť a skúšali všetky kombinácie na našom datasete znamienok. Každá kombinácia bola trénovaná po dobu 30 epoch na 1000 dátach. Pre presnejšie výsledky sme vykonali 10 replikácií pre každú kombináciu a ako výsledok sme zaznačili priemernú hodnotu loss funkcie a priemernú presnosť na testovacích dátach, ktorých bolo 250 a boli to dáta, ktoré model predtým nevidel.

Tabuľka ‑ Výsledky experimentu s BraňoNet

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Filtre** | **4** | | **8** | | **16** | | **32** | | **64** | |
| **Neuróny** | **loss** | **acc** | **loss** | **acc** | **loss** | **acc** | **loss** | **acc** | **loss** | **acc** |
| **8** | 0.6789 | 54.80% | 0.6769 | 56.25% | 0.6679 | 54.30% | 0.6931 | 50.00% | 0.6030 | 62.55% |
| **16** | 0.6770 | 53.40% | 0.6466 | 58.40% | 0.6555 | 58.40% | 0.6178 | 62.45% | 0.6593 | 54.70% |
| **32** | 0.6585 | 61.10% | 0.6367 | 60.85% | 0.6515 | 56.70% | 0.5565 | 69.90% | 0.6199 | 61.95% |
| **64** | 0.6571 | 62.70% | 0.5919 | 68.20% | 0.5683 | 71.10% | 0.5802 | 66.90% | 0.6014 | 70.65% |
| **128** | 0.6409 | 64.60% | 0.5911 | 66.90% | 0.5709 | 69.15% | 0.5646 | 72.10% | 0.5411 | 74.40% |
| **256** | 0.6137 | 67.40% | 0.5800 | 69.35% | 0.5623 | 71.85% | 0.5569 | 74.00% | 0.5378 | 75.75% |
| **512** | 0.6360 | 69.60% | 0.5933 | 70.30% | 0.6013 | 71.75% | **0.5177** | **76.45%** | 0.5504 | 75.20% |

Najlepšie výsledky sme dosiahli pri kombinácií 32 filtrov konvolučnej vrstvy a 512 neurónov plne prepojenej vrstvy s hodnotou loss funkcie **0,5177** a presnosťou **76,45%**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vrstva | | Počet filtrov | Veľkosť výstupu | Veľkosť filtrov | Stride | Aktivačná funkcia |
| Vstupná | Obrázok | 1 | 64x64 | - | - | - |
| 1 | Konvolučná | 16 | 31x31 | 3x3 | 2 | ReLU |
| 2 | Max pooling | 16 | 10x10 | 3x3 | 3 | - |
| 3 | Dense | - | 64 | - | - | ReLU |
| Výstupná | Dense | - | 2 | - | - | Softmax |

# Návrh nových vylepšení CNN nezávislých na architektúre

Experimenty, boli vykonané na preddefinovanej sieti LeNet-5 a vlastnej sieti BraňoNet. Avšak pre rôzne problémy nám možno nestačia základné hyperparametre ako napríklad relu či tanh aktivačná funkcia, xavier či he inicializácia, max/average pooling, či klasická kategorická cross-entropy loss funkcia. Preto sme sa rozhodli vymyslieť niekoľko vlastných hyperparametrov, ktoré predpokladáme, že ak ich aplikujeme na našu BraňoNet alebo LeNet-5, tak pre náš špecifický problém predikcie rakoviny kože by tieto siete mohli dosahovať lepšie výsledky. Pri navrhovaní nových vylepšení sme sa zamerali na 4 rôzne hyperparametre:

* Pooling
* Aktivačné funkcie
* Inicializácie váh
* Loss funkcie

## Vlastné pooling metódy

Pri navrhovaní vlastných pooling metód sme vychádzali z existujúcich average a max, ktoré sme následne prepojili alebo upravili. Výsledkom sú tri rôzne nové metódy na pooling.

### Weighted average pooling

Táto metóda pri doprednom šírení používa rovnaký princíp ako average pooling. Avšak pri spätnom šírení sa gradienty nevynásobia ¼ ale hodnotou vypočítanou ako pomer hodnoty pixelu k maximálnej hodnote pixelu z výrezu pri doprednom šírení. Teda pri doprednom šírení sa nájde maximálna hodnota každého výrezu o veľkosti filtra a následne sú všetky hodnoty výrezu vydelené touto hodnotou. Výsledné pomery sa uložia do masky, ktorou sa prenásobia gradienty pri spätnom šírení. Tým sa zabezpečí, že na pixel s najväčšou hodnotou sa pri spätnom šírení nastaví plná hodnota gradientu a na ostatné časti výrezu sa nastaví proporčne menšia hodnota gradientu.

### Double max pooling

Metóda je kombináciou max a average poolingu. Predpokladom tejto metódy je, že pri aplikovaní klasického max poolingu sa môže stať že jeden výrazný pixel vo výreze odstráni menej výrazný ale možno rovnako významný pixel. Preto pri doprednom šírení namiesto klasického max poolingu vyberieme až dve najväčšie hodnoty z každého výrezu. Tak sa zachová požadované zmenšenie obrázka, ale zároveň umiernime najvýznamnejší pixel tak, aby strata informácie bola čo najmenšia. Po výbere dvoch najväčších hodnôt pixelov sa z nich vypočíta priemer. Tento priemer bude predstavovať príslušný pixel výstupného obrázka. Pri doprednom šírení sa do masky uloží umiestnenie dvoch najväčších pixelov z každého výrezu. Potom pri spätnom šírení sa hodnota gradientu uloží na ich miesta a na zvyšné ostane hodnota pixelu 0.

### Median pooling

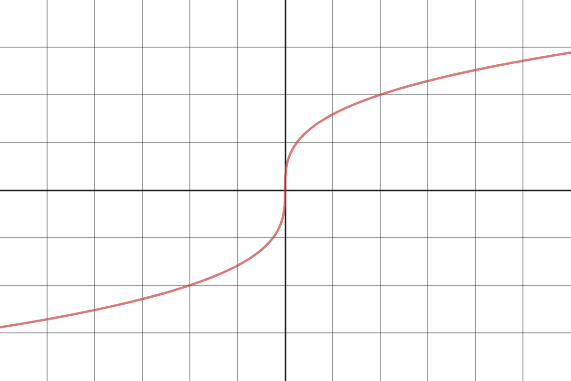
Pri average pooling metóde môže dôjsť k strate celkovej informácie z výrezu, pokiaľ niektorý pixel nadobúda príliš malé hodnoty, teda aj priemer výrezu následne bude nadobúdať primalé hodnoty a informácia sa stratí. Preto namiesto priemeru vypočítame medián z každého výrezu, tým sa zanedbajú pixely s príliš malou hodnotou a informácia sa nestratí. Táto metóda je takmer totožná s average pooling metódou. Jediný rozdiel je pri doprednom šírení, kedy namiesto priemeru vypočítame medián. Pri spätnom šírení sa gradient rovnako ako pri average pooling vynásobí ¼.

## Vlastné aktivačné funkcie

Niektoré vlastné návrhy sú kombináciou existujúcich aktivačných funkcií, alebo kombináciou existujúcej aktivačnej funkcie s vlastnou. Výsledkom sú 4 návrhy vlastných aktivačných funkcií:

* CBRT (Cube root)
* LeakyTReLU (TanH / LeakyReLU)
* SQCBRT (Square root / Cube root)
* Log (Logistic)

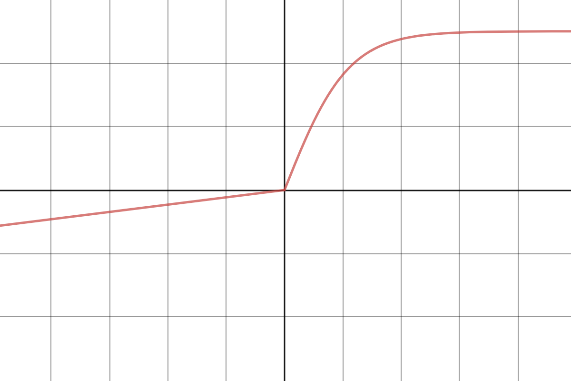
### CBRT

CBRT (Cube root) aktivačná funkcia je jednoduchá nelineárna funkcia, ktorá vážený vstup transformuje tak, že naň aplikuje tretiu odmocninu:

Derivácia funkcie:

### LeakyTReLU

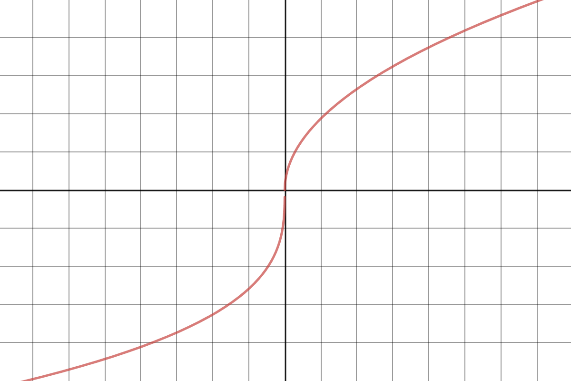
Aktivačná funkcia LeakyTReLU je kombináciou dvoch aktivačných funkcií LeakyReLU a TanH. Funkcia má základ v LeakyReLU, avšak namiesto lineárnej funkcie keď *x > 0* sa na vážený vstup upraví hyperbolickým tangensom aby sa narušila linearita funkcie.



Derivácia funkcie:

### SQCBRT

Táto aktivačná funkcia je kombináciou CBRT a druhej odmocniny. Vznikla upravením CBRT, tak aby kladné hodnoty boli trochu umiernenejšie. V prípade, že je *x > 0*, sa vážený vstup upraví namiesto tretej mocniny len druhou mocninou.

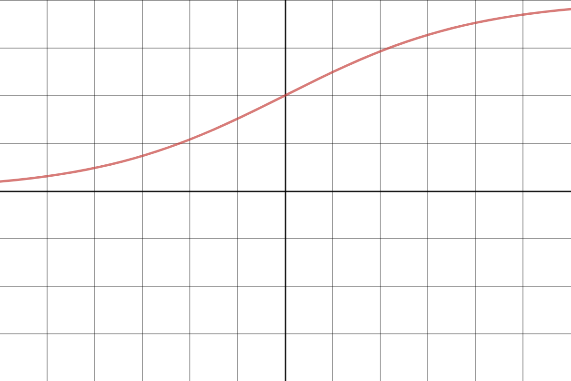


obr

Derivácia funkcie:

### Log

Aktivačná funkcia Log upraví vážený vstup logistickou funkciou.



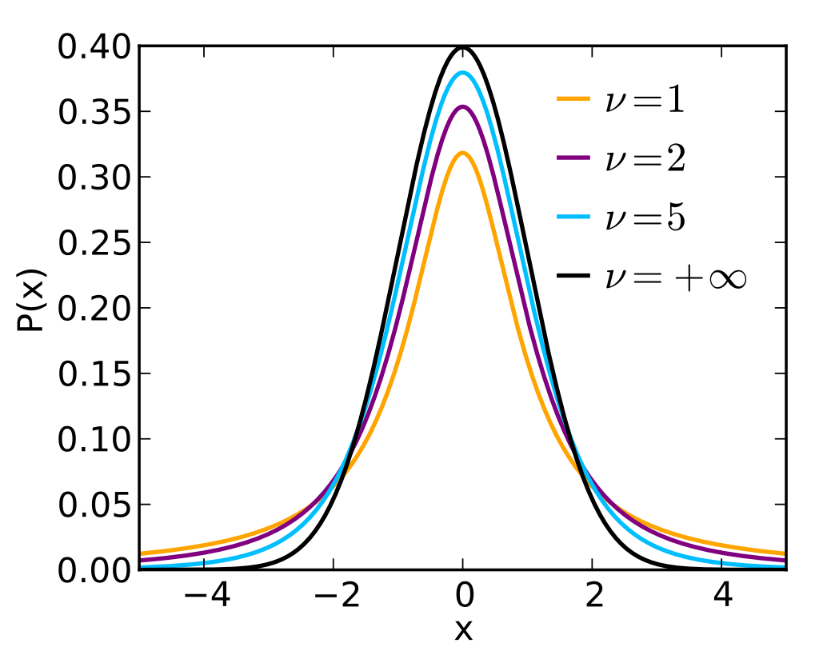
Derivácia tejto aktivačnej funkcie:

## Vlastné inicializácie váh

Pri návrhu nových metód inicializácie váh sme použili ako základ existujúce inicializácie ako **Xavier** alebo **He**, kde sme nahradili normálové alebo rovnomerné rozdelenie novým rozdelením. Pre inicializáciu sme navrhli 4 nové rozdelenia:

* Student
* GGD (Generalized Gaussian Distribution)
* *χ*2 (Chí-kvadrát)
* Log-normal (Logaritmicko-normálové)

### Student

Studentovo rozdelenie je podobné normálnemu rozdeleniu, avšak je náchylnejšie k vytváraniu hodnôt, ktoré sa nachádzajú ďalej od strednej hodnoty. To záleží aj od stanoveného stupňa voľnosti. Čím má Studentovo rozdelenie viac stupňov voľnosti tým viac sa podobá normálnemu rozdeleniu. Použitím tohto rozdelenia môžeme dostať viac hodnôt z koncov rozdelenia ako pri normálnom rozdelení.

### GGD

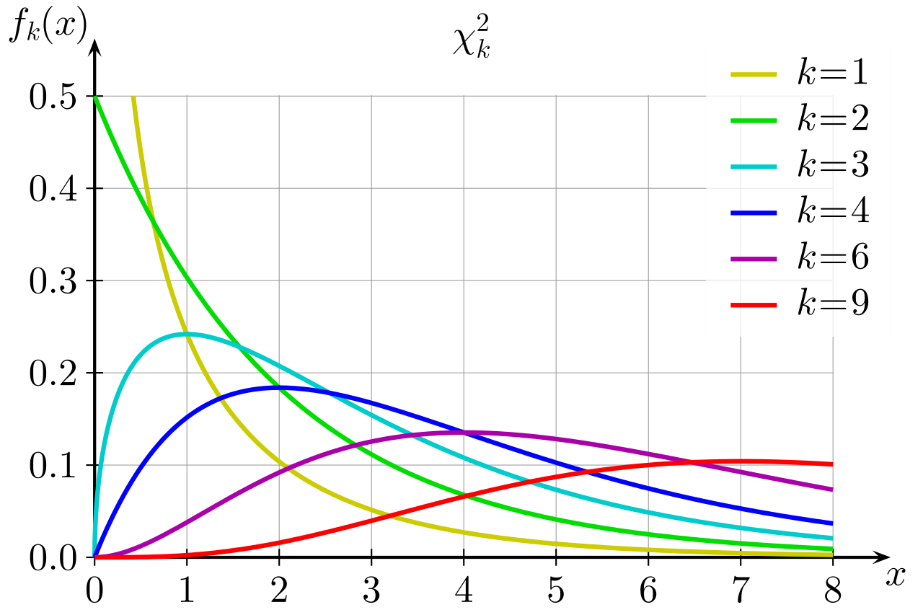
GGD (Generalized Gaussian Distribution) sa taktiež nazýva aj generalizované normálne alebo všeobecné rozdelenie. Je teda zovšeobecnením normálneho rozdelenia. Ak parameter Beta nastavíme na 2, dostaneme normálne rozdelenie. Preto zmenou tohto parametra môžeme meniť hodnoty na ktoré sa nastavia váhy jednotlivých vrstiev. Pri nízkych hodnotách parametra Beta môžeme dostávať čísla ďalej od strednej hodnoty omnoho častejšie ako pri normálnom rozdelení.

Obrázok, na ktorom je text, mapa

Automaticky generovaný popis

### *χ*2

Pokiaľ máme *k*nezávislých náhodných premenných, z ktorých každá má normované normálne rozdelenie, tak potom premenná predstavujúca sumu štvorcov týchto náhodných premenných má Chí-kvadrát rozdelenie s *k* stupňami voľnosti.



### Log-normal

Logaritmicko-normálne rozdelenie je také rozdelenie, že logaritmus náhodnej premennej z tohto rozdelenia má normálne rozdelenie pravdepodobnosti s rovnakou strednou hodnotou a odchýlkou.

Obrázok, na ktorom je svetlo, hodiny

Automaticky generovaný popis

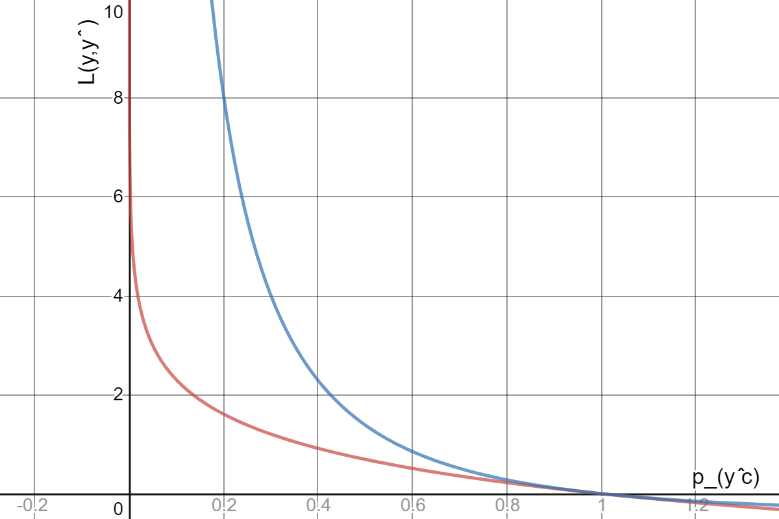
## Vlastné loss funkcie

Ako základ našich vlastných loss funkcií sme použili klasickú kategorickú cross-entropy loss funkciu (xxx). Pri navrhovaní vlastných loss funkcií sme sa zamerali na náš konkrétny problém predikcie rakoviny kože. Teda vieme, že máme len dve triedy, kde trieda 0 predstavuje, že pacient nemá rakovinu kože a trieda 1 predstavuje, že pacient má rakovinu kože. Keďže naša predikcia má slúžiť ako prevencia upravili sme loss funkciu tak, aby jej výsledná senzitivita bola čo najvyššia aj za cenu nižšej špecificity. Ak pacient nemá rakovinu a náš model by mu povedal, že ju na nejakú pravdepodobnosť má, môže ho to len naviesť ku kontrole u doktora, a to nepovažujeme za veľký problém. Avšak, ak by pacient mal rakovinu kože a náš model by mu predikoval, že nemá rakovinu kože, to už by problém mohol byť, pretože to môže pacienta odradiť od kontroly. Preto sme upravili loss funkciu tak, aby pri predikovaní falošných negatív bola pridaná pokuta k výslednej hodnote loss funkcie. Výsledkom sú tri nové loss funkcie:

* Divided Binnary Cross-Entropy
* Sensitivity Cross-Entropy
* Shifted Binnary Cross-Entropy

### Divided Binnary Cross-Entropy

Táto loss funkcia je závislá od konkrétnej triedy obrázka. Ak je trieda obrázka 0, je hodnota loss funkcie rovnaká ako pri kategorickej cross-entropy. Ak je trieda obrázka 1 je jej hodnota tiež ako pri kategorickej cross-entropy, ale výsledok je ešte podelený pravdepodobnosťou danej triedy. Keďže pravdepodobnosť bude vždy nadobúdať hodnoty v intervale <0,1>, tak ak má pacient rakovinu kože, tak čím menšiu pravdepodobnosť mu model predikuje, tým viac sa zväčší hodnota loss funkcie.

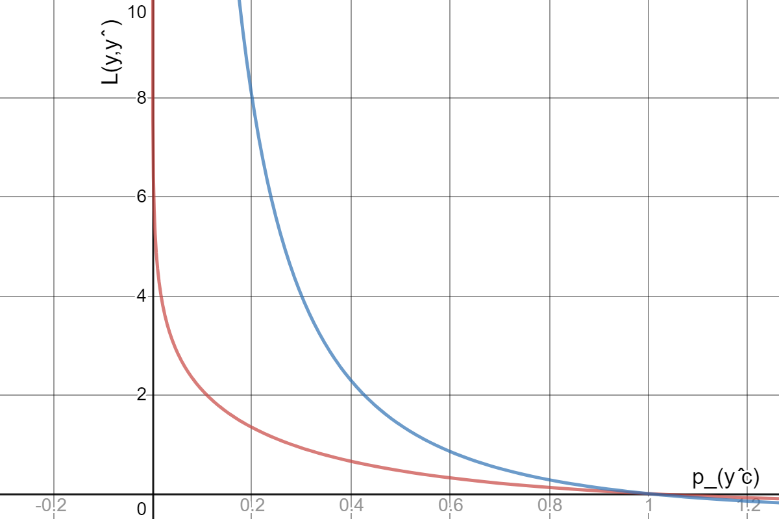


Červená krivka ukazuje priebeh hodnoty loss ak trieda *c = 0* a modrá ukazuje priebeh hodnoty loss funkcie ak trieda *c = 1*. Je vidno že ak c = 1, ale model predikoval 0 (hodnota py^c < 0.5), tak hodnota loss funkcie prudko stúpa. V porovnaní s klasickou loss funkciou jej hodnota pri falošných negatívach dosahuje ďaleko vyššie hodnoty.

Pre výpočet gradientu sa používa rovnako derivácia tejto loss funkcie.

### Sensitivity Cross-Entropy

Podobne ako Divided Binnary Cross-Entropy, aj táto loss funkcia je závislá na konkrétnej triede. Ak je trieda 1 hodnota loss funkcie je rovnaká ako pri Divided Binnary Cross-Entropy, avšak ak je trieda 0 je hodnota loss funkcie iná.

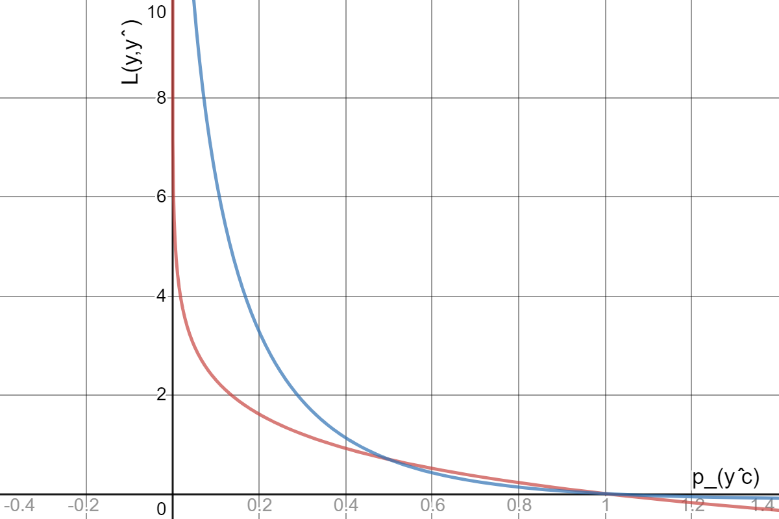


Červená krivka predstavuje vývoj hodnoty loss funkcie ak je trieda obrázka 0, modrá zase ak je trieda obrázka 1. Priebeh je veľmi podobný predchádzajúcej loss funkcii, avšak pri triede 0 hodnota loss funkcie stúpa s klesajúcou pravdepodobnosťou pomalšie, čím spôsobuje, že pokuta za falošné negatíva je väčšia.

Derivácia Sensitivity Cross-Entropy:

### Shifted Binnary Cross-Entropy

Táto loss funkcia podobne ako Divided Binnary Cross-Entropy sa záporný prirodzený logaritmus pravdepodobnosti triedy 1 delí. Avšak nedelí sa pravdepodobnosťou triedy 1, ale táto pravdepodobnosť je posunutá o 0,5 a umocnená na druhú. Touto úpravou dosiahneme, že ak model predikuje triedu 0 pri triede obrázka 1, tak sa výsledná loss zväčší, avšak ak predikuje správne tak hodnotu loss to zmenší.



Červená krivka ukazuje priebeh hodnoty loss ak trieda *c = 0* a modrá ukazuje priebeh hodnoty loss funkcie ak trieda *c = 1*.

Derivácia tejto loss funkcie:

## Experimenty s navrhnutými vylepšeniami

Po naimplementovaní všetkých vlastných vylepšení sme vykonali experimenty na BraňoNet a LeNet-5. Avšak pri najlepšej kombinácií poštu filtrov a neurónov pre BraňoNet (32 filtrov, 512 neurónov) doba jedného experimentu by bola príliš časovo náročná. Preto sme sa rozhodli vybrať pre vykonanie experimentov menšiu kombináciu, ktorá však stále dosahovala veľmi dobrých výsledkov. Konkrétne išlo o kombináciu 16 filtrov a 64 neurónov. Experimenty boli vykonané na vzorke 1250 dát, z ktorých je 625 fotiek nezhubných znamienok a 625 zhubných. Na tejto vzorke bola vykonaná 5-násobná krížová validácia. V každom cykle krížovej validácie bola sieť trénovaná na 1000 dátach a testované na 250. Každý model sa trénoval v batchoch po 128 dát v 30 epochách. Experimenty boli vykonané v 4 fázach, kde v každej fáze sa testovali všetky vylepšenia pre jeden typ hyperparametra (pooling, aktivačná funkcia, inicializácia, loss funkcia). V každej fáze sme vybrali najlepšiu hodnotu daného hyperparametra a v ďalšej fáze sa zafixovala táto najlepšia hodnota a experimenty pokračovali s najlepšou hodnotou hyperparametra. Všetky ostatné hodnoty hyperparametrov boli počas všetkých experimentov zafixované na rovnaké hodnoty.

### Experiment 1 – pooling

V prvej fáze sme sa zamerali na pooling metódy. Experimenty boli vykonané na všetkých našich pooling metódach ale aj na max a average pre lepšie porovnanie.

Tabuľka ‑ Výsledky experimentov fázy 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | LeNet-5 | | BranoNet | | |
| Pooling | Loss | presnosť | | Loss | Presnosť |
| AVG | 0.7038 | 53.04% | | 0.6363 | 65.84% |
| MAX | 0.6649 | 66.00% | | **0.6106** | **67.60%** |
| Median | 0.7064 | 53.36% | | 0.6620 | 62.56% |
| Vážený AVG | 0.6946 | 46.64% | | 0.6943 | 46.64% |
| MAX2 | **0.6598** | **64.00%** | | 0.7484 | 69.20% |

Z tabuľky vidno, že najlepšie výsledky sme dostali s použitím max2 metódy pri architektúre LeNet-5. Síce lepšia presnosť bola pri max pooling metóde, rozhodli sme sa vybrať metódu max2 kvôli nižšej hodnote loss funkcie. Z rovnakého dôvodu sme sa pri architektúre BraňoNet rozhodli vybrať metódu max pooling aj napriek tomu, že max2 mala lepšiu presnosť, no jej hodnota loss funkcie bola výrazne vyššia. Síce sme pri BraňoNet vybrali klasický max pooling, ostatné nami navrhnuté riešenia mali celkom dobré výsledky a pri LeNet-5 dokonca Max2 pooling mal lepšie výsledky ako klasický max pooling.

### Experiment 2 – aktivačná funkcia

V druhej fáze sme sa zamerali na aktivačné funkcie. Pre časovú efektivitu sme sa rozhodli použiť rovnakú aktivačnú funkciu na všetky vrstvy modelu. Rovnako ako pri prvej fáze, aj tu sme vykonali experimenty s klasickými aktivačnými funkciami tanh a relu pre lepšie porovnanie.

Tabuľka ‑Výsledky experimentov fázy 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LeNet-5 | | BranoNet | |
| Aktivačná funkcia | Loss | Presnosť | Loss | Presnosť |
| ReLU | 0.6893 | 64.32% | 0.6106 | 67.60% |
| TanH | 0.6598 | 64.00% | 0.6946 | 54.40% |
| CBRT | 0.6744 | 66.88% | 0.6828 | 68.32% |
| LeakyTReLU | 0.6412 | 63.84% | **0.6053** | **68.56%** |
| SQCBRT | **0.6768** | **67.20%** | 0.6581 | 68.24% |
| Log | 0.6947 | 50.00% | 0.6945 | 47.12% |

Z tabuľky vidno že pre LeNet-5 najlepšiu presnosť dosiahla aktivačná funkcia SQCBRT a najlepšiu hodnotu loss funkcie dosiahla aktivačná funkcia LeakyTReLU. Avšak hodnota loss funkcie pri SQCBRT nebola omnoho horšia ako presnosť pri LeakyTReLU, preto sme sa rozhodli pre ďalšie experimenty na LeNet-5 zafixovať aktivačnú funkciu SQCBRT. Pre BraňoNet najlepšiu presnosť aj najlepšiu hodnotu loss funkcie dosiahla aktivačná metóda LeakyTReLU.

### Experiment 3 – inicializácia váh

V tretej fáze sme sa zamerali na rôzne inicializácie váh, ktoré sme vždy stanovili rovnaké pre všetky vrstvy z rovnakého dôvodu ako aktivačné funkcie vo fáze 2. Okrem rôznych rozdelení sme taktiež skúšali aj rôzne parametre týchto rozdelení a taktiež sme skúšali aplikovať tieto rozdelenia na Xavier inicializáciu a vlastnú inicializáciu.

Tabuľka ‑ Výsledky experimentov fázy 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | LeNet-5 | | BranoNet | |
| Inicializácia | Loss | Presnosť | Loss | Presnosť |
| Xavier Normal | 0.6770 | 67.84% | 0.6100 | 67.84% |
| Xavier Uniform | 0.6768 | 67.20% | 0.6053 | 68.56% |
| He Normal | 0.6667 | 67.84% | 0.5965 | 70.08% |
| He Uniform | 0.6661 | 64.72% | **0.5762** | **71.68%** |
| Xavier Student | 3,8853 | 55.12% |  |  |
| Student | 0.6502 | 65.04% | 0.6004 | 70.08% |
| Student (degrees of freedom = 2) | 0.6662 | 65.20% | 0.6068 | 67.92% |
| student (degrees of freedom = 5) | 0.6841 | 64.40% | 0.6001 | 69.12% |
| Xavier GGD | **0.6808** | **68.40%** | 0.5907 | 69.20% |
| GGD | 0.6474 | 66.48% | 0.5985 | 68.72% |
| Xavier GGD (Beta = 0,5) | 3,8525 | 60.96% | 0.8370 | 62.00% |
| Xavier GGD (Beta = 2) | 0.6571 | 66.56% | 0.5966 | 69.12% |
| Xavier GGD (Beta = 3) |  |  | 0.6036 | 68.16% |
| Xavier χ2 | 0.7089 | 51.36% | 0.6553 | 60.40% |
| χ2 | 0.6865 | 56.96% | 0.6777 | 57.28% |
| Xavier Log-normal | 0.7417 | 48.32% |  |  |
| Log-normal | 0.7254 | 50.40% |  |  |

Niektoré experimenty neboli vykonané, pretože pri stanovených parametroch dochádzalo k pretekaniu alebo podtekaniu niektorých hodnôt pri výpočtoch. Pre LeNet-5 nám vyšlo najlepšie stanoviť inicializáciu váh pre všetky vrstvy pomocou Xavierovej inicializácie aplikovanej na generalizované normálne rozdelenie. Avšak pre BraňoNet najlepšie výsledky dosahovala inicializácia váh pomocou He inicializácie aplikovanej na rovnomerné rozdelenie.

### Experiment 4 – loss funkcia

V poslednej fáze sme sa zamerali na loss funkciu. Avšak nesledovali sme len hodnotu loss funkcie a presnosť. Hodnota loss funkcie sa ťažko porovnáva, keďže v zmysle inej loss funkcie má jej hodnota iný význam. Taktiež, keďže sme naše loss funkcie navrhli so zámerom zvýšiť senzitivitu, pozorovali sme taktiež hodnoty senzitivity a špecificity.

Tabuľka ‑ Výsledky experimentov fázy 4

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | LeNet-5 | | | BranoNet | | |
| Loss funkcia | Total acc | Spec | Sens | Total acc | Spec | Sens |
| Cross-Entropy | **68.40%** | **63.52%** | **73.28%** | **71.68%** | **78.08%** | **65.28%** |
| DBCE | 60.80% | 32.48% | 89.12% | 57.12% | 19.20% | 95.04% |
| SensCE | 60.32% | 31.20% | 89.44% | 54.40% | 11.52% | 97.28% |
| SBCE | 66.32% | 58.25% | 74.40% | 70.64% | 66.88% | 74.40% |

Keby sa zameriavame len na celkovú presnosť, tak tú ako sme čakali dosahujeme najlepšie pri klasickej kategorickej cross-entropy loss funkcie. Avšak, ak by sme sa chceli viac zamerať na senzitivitu, tak Divided Binnary Cross-Entropy a Sensitivity Cross-Entropy dosahujú veľmi vysoké hodnoty senzitivity. Pri LeNet-5 je to takmer 90% a pri BranoNet až 95% a 97%. Avšak taktiež vidno, že je to za cenu nižšej špecificity a teda čiastočne aj na celkovej presnosti.

## Najlepšie siete

Výsledkom našich experimentov sú 2 siete s rôznymi hyperparametrami. Jedným je upravená LeNet-5 a druhým BraňoNet.

### Upravená LeNet-5

Ako inicializáciu váh pre architektúru LeNet-5 sme vybrali Xavierovu inicializáciu so všeobecným normálnym rozdelením. Ďalej ako loss funkcia nám najlepšie vyšla kategorická cross-entropy, avšak ak sa chceme zamerať na senzitivitu, veľmi dobré výsledky dosahovali Divided Binnary Cross-Entropy a Sensitivity Cross-Entropy. Aktivačná funkcia pre všetky modely bola zvolená SQCBRT pre jej najlepšiu presnosť a ako pooling vrstvy sme použili Max2 pooling metódu.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vrstva | | Počet filtrov | Veľkosť výstupu | Veľkosť filtrov | Stride | Aktivačná funkcia |
| Vstupná | Obrázok | 1 | 28x28 | - | - | - |
| 1 | Konvolučná | 6 | 24x24 | 5x5 | 1 | SQCBRT |
| 2 | Max2 pooling | 6 | 14x14 | 3x3 | 2 | - |
| 3 | Konvolučná | 16 | 10x10 | 5x5 | 1 | SQCBRT |
| 4 | Max2 pooling | 16 | 5x5 | 2x2 | 2 | - |
| 5 | Konvolučná | 120 | 1x1 | 4x4 | 1 | SQCBRT |
| 5 | Dense | - | 84 | - | - | SQCBRT |
| Výstupná | Dense | - | 2 | - | - | Softmax |

### BraňoNet

Pre našu BraňoNet sieť sme pre inicializáciu váh vybrali He inicializáciu s rovnomerným rozdelením. Rovnako ako pri upravenej LeNet-5 aj tu vyšla celkovo najlepšie kategorická Cross-Entropy, avšak aj Shifted Binnary Cross-Entropy dosahovala veľmi podobných výsledkov a zároveň hodnota senzitivity bola o dosť vyššia ako pri kategorickej Cross-Entropy. Síce pri DBCE a SensCE bola senzitivita podstatne vyššia (95-97%) celková presnosť bola výrazne slabšia (54-57%). Preto sme sa rozhodli vybrať ako loss funkciu SBCE. Aktivačnú funkciu sme vybrali LeakyTReLU a pooling vrstvu sme vybrali max pooling, pretože mala lepšie výsledky ako max2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vrstva | | Počet filtrov | Veľkosť výstupu | Veľkosť filtrov | Stride | Aktivačná funkcia |
| Vstupná | Obrázok | 1 | 64x64 | - | - | - |
| 1 | Konvolučná | 16 | 31x31 | 3x3 | 2 | LeakyTReLU |
| 2 | Max pooling | 16 | 10x10 | 3x3 | 3 | - |
| 3 | Dense | - | 64 | - | - | LeakyTReLU |
| Výstupná | Dense | - | 2 | - | - | Softmax |

# Diskusia

Ako ďalší krok sme chceli porovnať naše vytvorené siete s niektorými existujúcimi architektúrami, pre ktoré by sme našli najlepšie hyperparamtre z niektorých existujúcich. Na porovnanie sme sa rozhodli vybrať LeNet-5 a AlexNet. Na nájdenie najlepších parametrov sme použili rovnaký postup ako pri hľadaní najlepších parametrov pre naše siete. Vyskúšali sme viacero hodnôt jedného typu hyperparametra, z nich sme vybrali hodnotu najlepšiu pre daný hyperparameter a túto hodnotu sme zafixovali pre ďalšie experimenty. Keďže AlexNet je príliš veľká, doba učenia by trvala príliš dlho. Preto sme sa rozhodli na vykonanie experimentov využiť knižnicu keras, ktorej proces učenia trvá kratšie. Avšak aj s využitím kerasu by učenie AlexNet bolo príliš časovo náročné, pretože je to veľká sieť, ktorá bola navrhnutá na ImageNet (dataset s 1,3miliónom obrázkov rozdelených do 1000 tried). Preto sme sa rozhodli upraviť architektúru AlexNet tak, že sme znížili počet filtrov v jednotlivých konvolučných vrstvách. Predpokladáme, že pri obrázkoch znamienok dvoch tried nebudeme potrebovať nájsť toľko veľa vlastností ako pri obrázkoch 1000 rôznych tried.

## Keras experimenty

Pri každom experimente sa sieť učila na 1000 obrázkoch nášho datasetu po dobu 30 epoch a následne bola otestovaná na 250 nových obrázkoch. Pre každý experiment sme vykonali 10 replikácií, z ktorých sme si vždy zapísali výsledné presnosti a hodnoty loss funkcie z testu. Z výsledkov jednotlivých replikácií sme potom urobili priemer a ten sme zapísali. Pri vykonávaní experimentov sme sa zamerali na 4 rôzne typy hyperparametrov:

* Pooling
* Aktivačná funkcia
* Inicializácia váh
* Loss funkcia

Taktiež sme testovali rozdiel medzi farebnými obrázkami a čiernobielymi obrázkami.

### Experiment 1 – pooling

Pri tomto experimente sme skúšali len LeNet-5 sieť. Porovnávali sme preddefinovaný avg pooling a max pooling.

|  |  |  |
| --- | --- | --- |
|  | LeNet-5 | |
| Pooling | Loss | Presnosť |
| AVG | 0.6830 | 57.50% |
| MAX | **0.7469** | **68.40%** |

Z tabuľky vidno, že max pooling metóda dosiahla výrazne väčšiu presnosť. Síce hodnota loss funkcie je väčšia ale presnosť je o viac ako 10% lepšia a preto sme sa rozhodli vybrať tento pooling pre ďalšie experimenty.

### Experiment 2 – Aktivačná funkcia

Pre testovanie aktivačných funkcií sme vybrali 3 základné TanH, Sigmoid a ReLU. A okrem týchto základných aktivačných funkcií, sme taktiež otestovali PReLU.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | LeNet-5 | | AlexNet | | |
| Aktivačná funkcia | Loss | Presnosť | | Loss | Presnosť |
| TanH | 0.7469 | 68.40% | | **0.9443** | **66.00%** |
| Sigmoid | 0.6941 | 50.00% | | 0.6940 | 50.00% |
| ReLU | 0.6160 | 69.70% | | **0.6673** | **64.20%** |
| PReLU | **0.6100** | **71.45%** | | 0.6846 | 62.75% |

Pre LeNet-5 nám vyšla najlepšia presnosť aj najlepšia hodnota loss funkcie s aktivačnou funkciou PReLU. Oproti preddefinovanej TanH aktivačnej funkcií jej presnosť stúpla o 3% ale hodnota loss výraznejšie klesla a preto sme ju zafixovali pre ďalšie experimenty na LeNet-5. Pri AlexNet mala najvyššiu presnosť TanH funkcia, avšak hodnota loss funkcie je výrazne väčšia ako pri preddefinovanej ReLU aktivačnej funkcií, ktorej presnosť je len o necelé 2% horšia. Keďže však výsledky boli veľmi podobné rozhodli sme sa ďalšie experimenty vykonať s oboma aktivačnými funkciami.

### Experiment 3 – RGB

Pri tomto experimente sme porovnávali výsledky pri farebných obrázkoch a pri čiernobielych obrázkoch. Keďže pri AlexNet porovnávame výsledky s dvoma rôznymi aktivačnými funkciami rozdelili sme výsledky do dvoch rôznych tabuliek.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  | | AlexNet | |
| Farebnosť | **Aktivačná** **funkcia** | | Loss | | Presnosť |
| Čiernobiele | TanH | | 0.9443 | | 66.00% |
| **ReLU** | | **0.6673** | | **64.20%** |
| Farebné | **TanH** | | **0.7507** | | **67.90%** |
| ReLU | | 0.6770 | | 64.20% |

|  |  |  |  |
| --- | --- | --- | --- |
|  | | LeNet-5 | |
| Farebnosť | Loss | | Presnosť |
| Čiernobiele | 0.6100 | | 71.45% |
| Farebné | **0.5679** | | **71.85%** |

Pri AlexNet najlepšiu hodnotu loss funkcie sme dosiahli pri aktivačnej funkcií ReLU a pri čiernobielych obrázkoch, zatiaľ čo najvyššiu presnosť sme dosiahli s aktivačnou funkciou TanH a farebnými obrázkami. Preto sme sa rozhodli pri ďalších experimentoch zafixovať tieto dve kombinácie. Pri LeNet-5 sme však dosiahli najlepšie výsledky pri farebných obrázkoch. Preto ďalšie experimenty s LeNet-5 budeme vykonávať na farebných obrázkoch.

### Experiment 4 – Inicializácia váh

Pri inicializácií váh sme porovnávali Xavier, He, Random a Lecun inicializáciu. Všetky spôsoby inicializácie sme skúšali s normálnym aj rovnomerným rozdelením.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | AlexNet | |
| Inicializácia | **Rozdelenie** | **Aktivačná** **funkcia** | Loss | Presnosť |
| Random | Rovnomerné | ReLU | 0.6697 | 59.60% |
| TanH | 0.6792 | 63.55% |
| Normálne | ReLU | 0.6594 | 65.65% |
| TanH | 0.7297 | 69.50% |
| Xavier | Rovnomerné | ReLU | 0.6770 | 64.20% |
| TanH | 0.7507 | 67.90% |
| Normálne | ReLU | 0.6489 | 66.25% |
| TanH | **0.6870** | **71.15%** |
| He | Rovnomerné | ReLU | 0.7514 | 67.15% |
| TanH | 0.6538 | 66.70% |
| Normálne | ReLU | 0.7932 | 67.10% |
| TanH | **0.6279** | **67.55%** |
| Lecun | Rovnomerné | ReLU | 0.7106 | 61.00% |
| TanH | 0.7269 | 68.55% |
| Normálne | ReLU | 0.7060 | 69.75% |
| TanH | 0.6739 | 66.60% |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | LeNet-5 | |
| Inicializácia | **Rozdelenie** | Loss | Presnosť |
| Random | Rovnomerné | 0.6549 | 62.80% |
| Normálne | 0.6290 | 66.65% |
| Xavier | Rovnomerné | 0.5914 | 71.50% |
| Normálne | 0.5679 | 71.85% |
| He | Rovnomerné | 0.6208 | 72.30% |
| Normálne | 0.6466 | 72.25% |
| Lecun | Rovnomerné | 0.6055 | 71.65% |
| Normálne | **0.5601** | **73.45%** |

Inicializácia váh s najlepšou hodnotou loss funkcie nám vyšla He inicializácia s normálnym rozdelením a TanH aktivačnou funkciou. Najlepšiu presnosť dosiahla Xavierova inicializácia rovnakým rozdelením aj rovnakou aktivačnou funkciou. Preto pre posledný experiment sme zafixovali už len farebné obrázky a TanH aktivačnú funkciu pre AlexNet. Avšak obe inicializácie mali veľmi podobné výsledky a preoto sme posledný experiment robili s oboma inicializáciami. Pri LeNet-5 nám najlepšie výsledky dala Lecun inicializácia s normálnym rozdelením.

### Experiment 5 – Loss funkcia

Na porovnanie s kategorickou Cross-Entropy sme vybrali Squared Hinge a Binnary Cross-Entropy.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | AlexNet | |
| Loss funkcia | **Inicializácia** | Loss | Presnosť |
| Categorical Cross-Entropy | **Xavier** | **0.6870** | **71.15%** |
| He | **0.6279** | **67.55%** |
| Binnary Cross-Entropy | Xavier | 0.7022 | 70.70% |
| He | 0.6311 | 69.10% |
| Squared Hinge | Xavier | 1.0992 | 68.35% |
| He | 1.0796 | 68.55% |

|  |  |  |
| --- | --- | --- |
|  | LeNet-5 | |
| Loss funkcia | Loss | Presnosť |
| Categorical Cross-Entropy | 0.5601 | 73.45% |
| Binnary Cross-Entropy | **0.5775** | **73.50%** |
| Squared Hinge | 1.0107 | 73.05% |

Aj pri AlexNet aj pri LeNet-5 nám celkovo vyšla najlepšie kategorická Cross-Entropy. Avšak pri LeNet-5 aj zvyšné dve loss funkcie dávali takmer rovnaké výsledky. Pri AlexNet tiež boli veľmi podobné ale rozdiel bol trochu väčší ako pri LeNet-5.

## Najlepšie keras siete

Podľa výsledkov našich experimentov, sme vybrali najlepšie hodnoty rôznych hyperparametrov.

### Upravená LeNet-5

Pre našu LeNet-5 sme vybrali Lecun inicializáciu váh s normálnym rozdelením. Ako aktivačnú funkciu sme vybrali PReLU namiesto preddefinovanej TanH. Namiesto avg pooling metódy sme vybrali max pooling, pretože dosahovala výrazne lepšie výsledky. Loss funkciu sme vybrali Binnary Cross-Entropy.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vrstva | | Počet filtrov | Veľkosť výstupu | Veľkosť filtrov | Stride | Aktivačná funkcia |
| Vstupná | Obrázok | 3 | 28x28 | - | - | - |
| 1 | Konvolučná | 6 | 24x24 | 5x5 | 1 | PReLU |
| 2 | Max pooling | 6 | 14x14 | 3x3 | 2 | - |
| 3 | Konvolučná | 16 | 10x10 | 5x5 | 1 | PReLU |
| 4 | Max pooling | 16 | 5x5 | 2x2 | 2 | - |
| 5 | Konvolučná | 120 | 1x1 | 4x4 | 1 | PReLU |
| 5 | Dense | - | 84 | - | - | PReLU |
| Výstupná | Dense | - | 2 | - | - | Softmax |

### Upravená AlexNet

Váhy sme sa rozhodli inicializovať podľa Xavierovej inicializácie s normálnym rozdelením. Výslednú aktivačnú funkciu sme vybrali TanH miesto preddefinovanej ReLU. Loss funkciu sme ponechali kategorickú Cross-Entropy.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Vrstva | | Počet filtrov | Veľkosť výstupu | Veľkosť filtrov | Stride | Aktivačná funkcia |
| Vstupná | Obrázok | 3 | 224x224 | - | - | - |
| 1 | Konvolučná | 6 | 54x54 | 11x11 | 4 | TanH |
| 2 | Max pooling | 6 | 27x27 | 3x3 | 2 | - |
| 3 | Konvolučná | 16 | 23x23 | 5x5 | 1 | TanH |
| 4 | Max pooling | 16 | 11x11 | 3x3 | 2 | - |
| 5 | Konvolučná | 32 | 9x9 | 3x3 | 1 | TanH |
| 6 | Konvolučná | 32 | 7x7 | 3x3 | 1 | TanH |
| 7 | Konvolučná | 16 | 5x5 | 3x3 | 1 | TanH |
| 8 | Dense | - | 128 | - | - | TanH |
| 9 | Dense | - | 256 | - | - | TanH |
| Výstupná | Dense | - | 2 | - | - | Softmax |

# Záver

# [Zoznam použitej literatúry](Pokyny_pre_vypracovanie_ZP.docx#Zoznam_použitej_literatúry)

[1] HOWLADER N., NOONE AM., KRAPCHO M., MILLER D., BREST A., YU M., RUHL J., TATALOVICH Z., MARIOTTO A., LEWIS DR., CHEN HS., FEUER EJ., CRONIN KA. 2020. *SEER Cancer Stat Facts: Melanoma of the Skin*. [onine]. Bethesda, MD:  National Cancer Institute, 2020.[cit. Apríl 2020]. Dostupné na internetovej stránke: <<https://seer.cancer.gov/statfacts/html/melan.html>>.

[3] KRIZHEVSKY A., SUTSKEVER I., HINTON G.E. 2012. *ImageNet Classification with Deep Convolutional Neural Network*. [online]. Dostupné na internetovej stránke:

<<https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>>

# [Zoznam príloh](Pokyny_pre_vypracovanie_ZP.docx" \l "Prílohy)

# 

# Prílohy