

Praktikum z neurónových sietí pre počítačové videnie

Domáca úloha 1

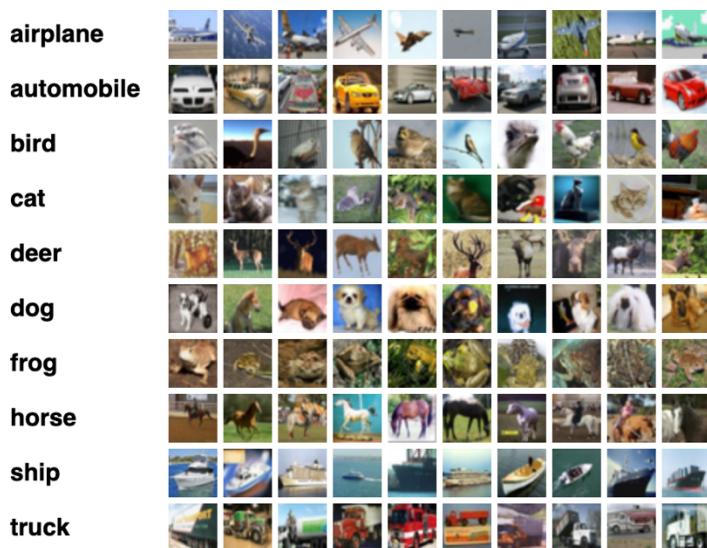
Richard Dominik

12.11.2021

Tento dokument popisuje rôzne konfigurácie, postupy a vyhodnotenia trénovania neurónových sietí na datasete CIFAR-10. Kód je implementovaný v jazyku Python za pomocí frameworku PyTorch. Trénovanie priebehalo v prostredí Google Colaboratory s GPU NVIDIA Tesla K80 s pamäťou 12GB. Kód úlohy sa nachádza v GitHub repozitári <https://github.com/RichardDominik/neural-networks-CV> v súbore PNSPV_hw1_final.ipynb

1. Dataset CIFAR-10

Dataset CIFAR-10 obsahuje 60 000 farebných obrázkov s rozmerom 32x32. Obrázky sú rozdelené do 10tich tried s 6000 obrázkami pre každú triedu. Trénovacia množina obsahuje 50 000 obrázkov a testovacia množina obsahuje 10 000 obrázkov [a]. Ukážku datasetu môžeme vidieť na obrázku 1.



Obrázok 1 Ukážka dát z datasetu CIFAR-10 [a]

2. Základný model

Ako základný model pre ďalšie modifikácie sme si zvolili neurónovú sieť, ktorá sa na začiatku skladá z troch konvolučných vrstiev s veľkosťou jadra 3x3 a paddingom 1. Za každou konvolučnou vrstvou následne aplikujeme operáciu max pooling s veľkosťou jadra 2x2 a stridom 2. Následne po operácii max pooling aplikujeme aktivačnú funkciu Relu. Po konvolučných vrstvách nasleduje operácia Flatten a 4 plne prepojené vrstvy s aktivačnou funkciou Relu. Grafickú reprezentáciu základného modelu môžeme vidieť na obrázku 2. Ako optimalizátor sme si zvolili Adam s learning rateom 1e-3, ďalej sme si zvolili veľkosť jedného batchu 32 a ako stratovú funkciu Cross Entropy. Základný model sme natrénovali na 10tich

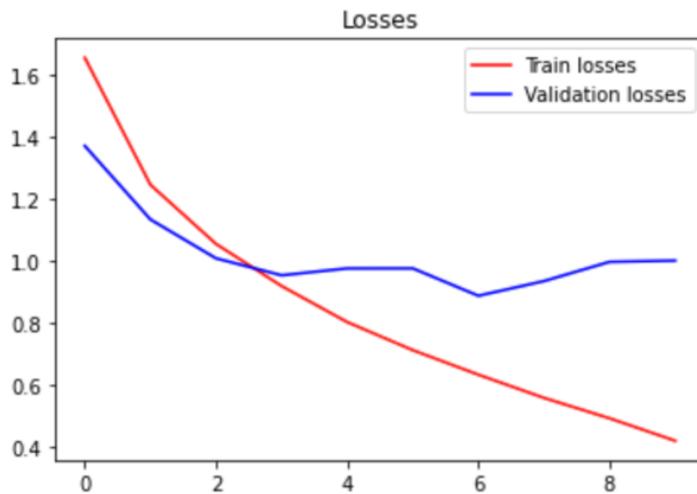
epochách a dosahuje presnosť 72.26% (Obrázok 4). Pri experimentovaní sme zistili, že po 10tej epoche dochádza k javu, ktorý nazývame overfitting (Obrázok 3).

```

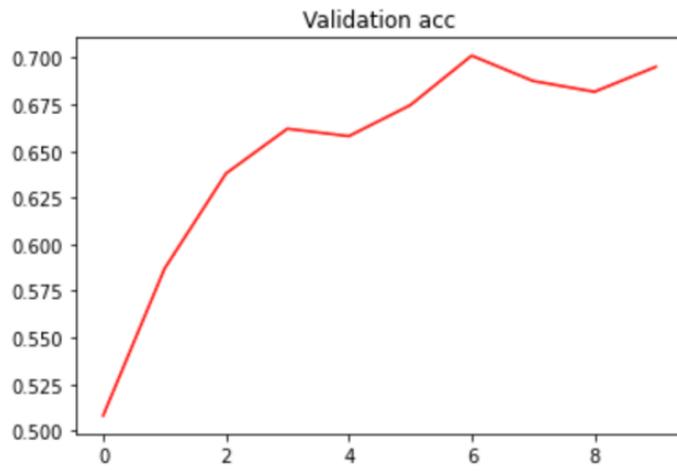
Sequential(
(0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(2): ReLU()
(3): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(5): ReLU()
(6): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
(8): ReLU()
(9): Flatten(start_dim=1, end_dim=-1)
(10): Linear(in_features=1024, out_features=512, bias=True)
(11): ReLU()
(12): Linear(in_features=512, out_features=256, bias=True)
(13): ReLU()
(14): Linear(in_features=256, out_features=128, bias=True)
(15): ReLU()
(16): Linear(in_features=128, out_features=10, bias=True)
)

```

Obrázok 2 Architektúra základného modelu



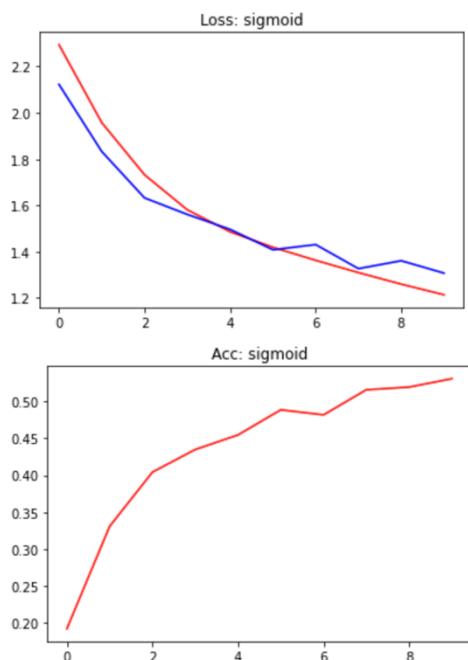
Obrázok 3 Trénovacia a validačná chyba základného modelu



Obrázok 4 Presnosť základného modelu

3. Aktivácie

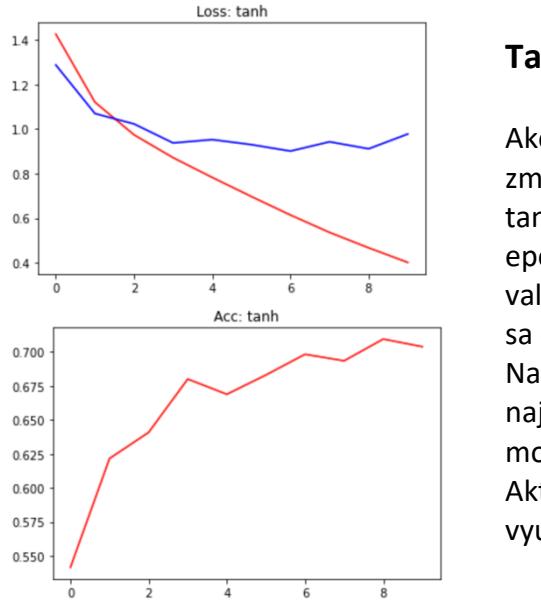
V tejto časti úlohy sme sa zaoberali zmenou aktivačnej funkcie Relu v základnom modely za aktivačné funkcie Sigmoid, Tanh, ELU, LeakyReLU a PReLU. Pomocou spomínaných aktivačných funkcií sme pomocou rovnakých hyperparametrov (okrem hyperparametrov špecifických pre aktivačné funkcie) dosiahli nasledujúce výsledky:



Obrázok 5 Graf 1: Trénovacia a validačná chyba pre aktivačnú funkciu sigmoid, Graf 2: Presnosť modelu pre aktivačnú funkciu sigmoid

Sigmoid

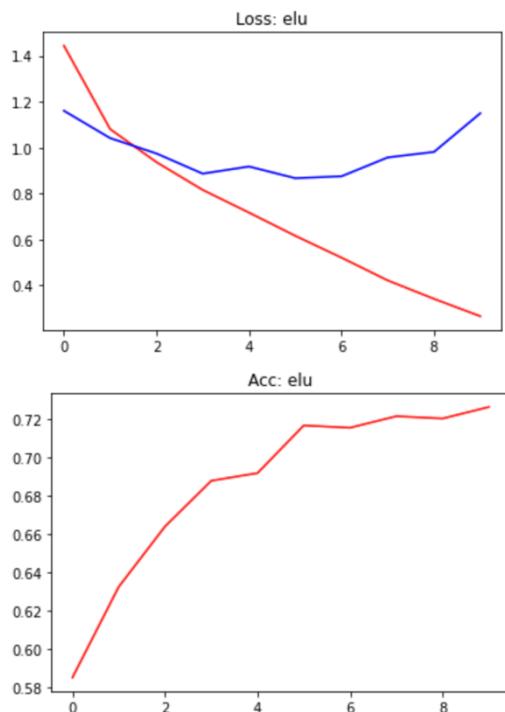
Ako môžeme vidieť na prvom grafe na obrázku 5, zmenou aktivačnej funkcie Relu za sigmoid sa počas trénovalia na rovnakom počte epoch (10) neprejavil overfitting (validačná chyba nezačala rásť) a zároveň trénovacia chyba stále klesala. Na druhom grafe môžeme vidieť, že sme dosiahli najlepšiu presnosť 53.08%. V porovnaní so základným modelom sme teda dosiahli o 19.18% menšiu presnosť. Aktivačná funkcia sigmoid sa ale neodporúča využívať v úlohách pre hlboké učenie.



Obrázok 6 Trénovacia a validačná chyba pre aktivačnú funkciu tanh, Graf 2: Presnosť modelu pre aktivačnú funkciu tanh

Tanh

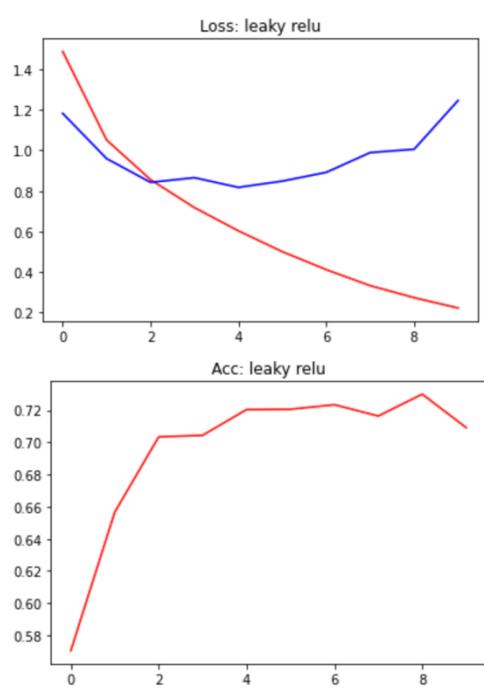
Ako môžeme vidieť na prvom grafe na obrázku 6, zmenou aktivačnej funkcie Relu za Tanh (hyperbolický tangens) sa počas trénovalia na rovnakom počte epoch (10) trénovacia chyba stále zničovala, ale validačná sa už po tretej epoche prestala zničovať až sa od 8nej epochy začala znova zvyšovať (overfitting). Na druhom grafe môžeme vidieť, že sme dosiahli najlepšiu presnosť 70.38%. V porovnaní so základným modelom sme teda dosiahli o 1.88% menšiu presnosť. Aktivačná funkcia Tanh sa rovnako neodporúča využívať v úlohách pre hlboké učenie.



Obrázok 7 Trénovacia a validačná chyba pre aktivačnú funkciu ELU, Graf 2: Presnosť modelu pre aktivačnú funkciu ELU

ELU

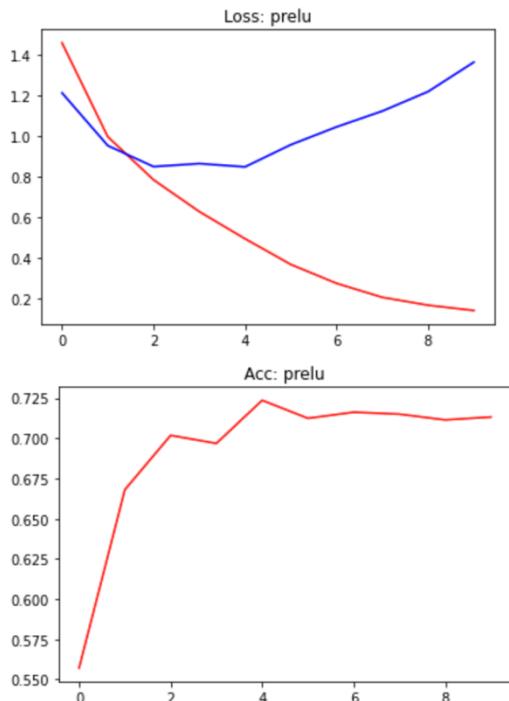
Ako môžeme vidieť na prvom grafe na obrázku 7 zmenou aktivačnej funkcie Relu za ELU sa počas trénovania na rovnakom počte epôch (10) trénovacia chyba stále znižovala, ale validačná sa už po 6tej epoche začala zvyšovať (overfitting). Na druhom grafe môžeme vidieť, že sme dosiahli najlepšiu presnosť 72.6%. V porovnaní so základným modelom sme teda dosiahli o 0.34% lepšiu presnosť ale oveľa skôr sme dosiahli overfitting na základnom počte dát z datasetu.



Obrázok 8 Trénovacia a validačná chyba pre aktivačnú funkciu LeakyReLU, Graf 2: Presnosť modelu pre aktivačnú funkciu LeakyReLU

LeakyReLU

Pri aktivačnej funkcií LeakyReLU sme si zvolili parameter alfa = 0.1. Ako môžeme vidieť na prvom grafe na obrázku 8 zmenou aktivačnej funkcie Relu za LeakyReLU sa počas trénovania na rovnakom počte epôch (10) trénovacia chyba stále znižovala, ale validačná chyba sa už po 4tej epoche začala zvyšovať (overfitting). Na druhom grafe môžeme vidieť, že sme dosiahli najlepšiu presnosť 71.32%. V porovnaní so základným modelom sme teda dosiahli o 0.94% horšiu presnosť modelu a oveľa skôr sme dosiahli overfitting na základnom počte dát z datasetu. Model by sme mohli vylepšiť vhodnejším nastavením parametru alfa, ale vzhľadom na počet otestovaných aktivačných funkcií to bolo mimo rozsahu zadania tejto úlohy.



PReLU

Ako môžeme vidieť na prvom grafe na obrázku 9 zmenou aktivačnej funkcie Relu za PReLU sa počas trénovania na rovnakom počte epôch (10) trénovacia chyba stále znižovala, ale validačná chyba sa už po 4tej epoche začala zvyšovať (overfitting). Na druhom grafe môžeme vidieť, že sme dosiahli najlepšiu presnosť 72.36%. V porovnaní so základným modelom sme teda dosiahli o 0.1% lepšiu presnosť modelu ale oveľa skôr sme dosiahli overfitting na základnom počte dát z datasetu.

Obrázok 9 Trénovacia a validačná chyba pre aktivačnú funkciu PReLU, Graf 2: Presnosť modelu pre aktivačnú funkciu PReLU

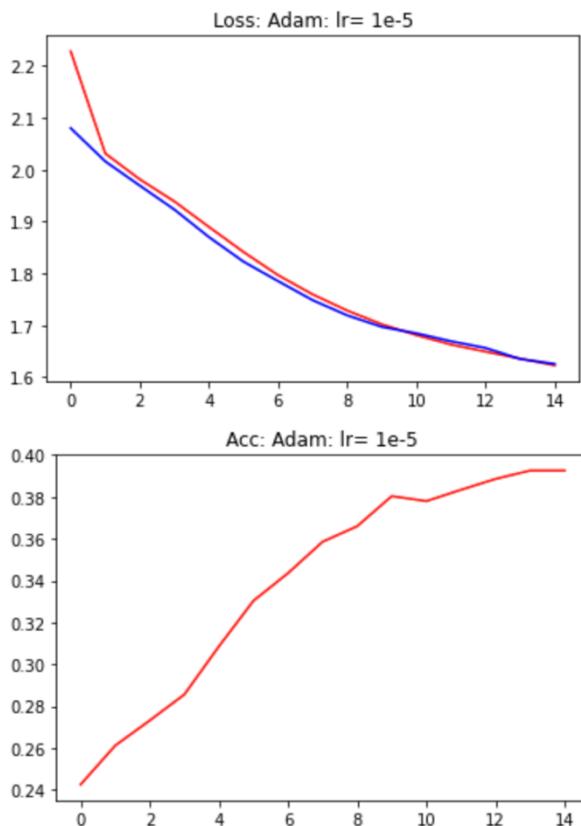
Z predchádzajúcich grafov môžeme usúdiť, že z pohľadu presnosti mal najhoršie výsledky model s aktivačnou funkciou sigmoid a z pohľadu najmenšieho počtu epôch po ktorých nastal overfitting to boli modeli s aktivačnými funkciemi PReLU a LeakyReLU (parameter alfa = 0.1).

4. Optimalizácia

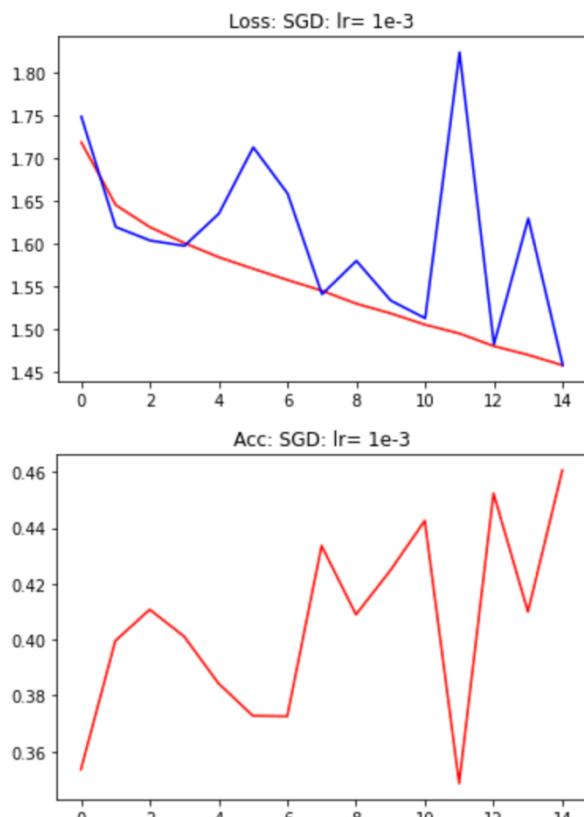
V tejto časti úlohy sme sa zaoberali zmenou optimalizátora Adam s learning rateom 1e-3 v základnom modeli za optimalizátori Adam, SGD, RMSprop, AdamW v rôznych konfiguráciách. Následne sme si ešte vybrali optimalizátor AdamW na ktorom sme vyskúšali 3 rôzne veľkosti minibatchov a to konkrétnie 8, 16, 64. Počas celého priebehu trénovania tejto podúlohy sme zvolili počet epôch 15.

Adam

Ako prvú možnosť sme sa pokúsili zmeniť learning rate v optimalizátore Adam z 1e-3 na 1e-5. Ako môžeme vidieť na obrázku 10, oproti základnému modelu spôsobilo zmenšenie learning rateu to, že aj pri 15tej epoche sme neprišli k overfittingu a trénovacia a validačná chyba sa veľmi pomaly blíži k 0. Presnosť téhoto modelu dosiahla v najlepšom prípade 39.6% čo je o 32.66% menej oproti základnému modelu. Pre daný optimalizátor sme teda zvolili veľmi nízky learning rate.



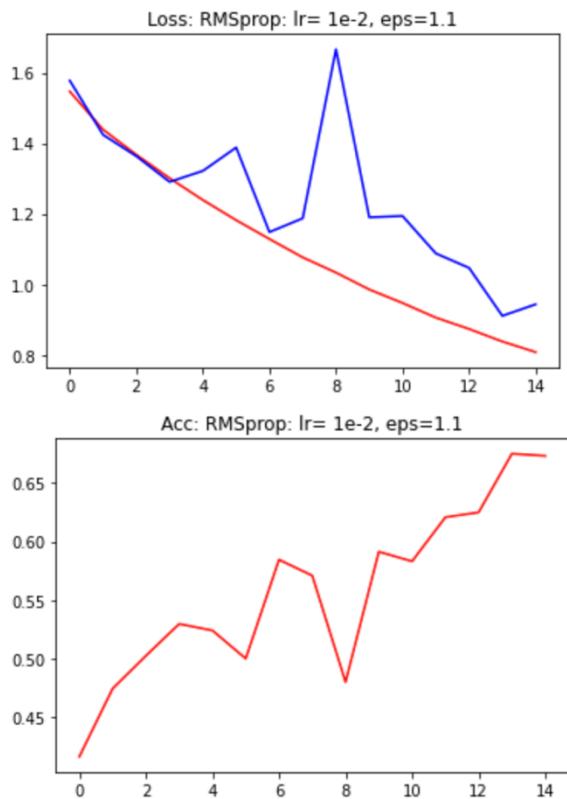
Obrázok 10 Trénovacia a validačná chyba pre optimalizátor Adam (lr=1e-5), Graf 2: Presnosť modelu pre optimalizátor Adam (lr=1e-5)



SGD

Ako druhú možnosť sme sa pokúsili zmeniť optimalizátor Adam na optimalizátor SGD s rovnakým learning rateom ako mal Adam a to 1e-3. Ako môžeme vidieť na obrázku 11, oproti základnému modelu spôsobila zmena optimalizátora veľkú nestabilitu validačnej chyby od tretej epochy a rovnako nestabilitu presnosti modelu. Zaujímavý prípad môžeme vidieť v 10tej epoche kde model dosahoval presnosť cca 44% ale v 11tej epoche už iba cca 35%. V rovnakých epochách môžeme vidieť aj nárast validačnej chyby z cca 1.55 v 10tej epoche na 1.8 v 11 tej epoche.

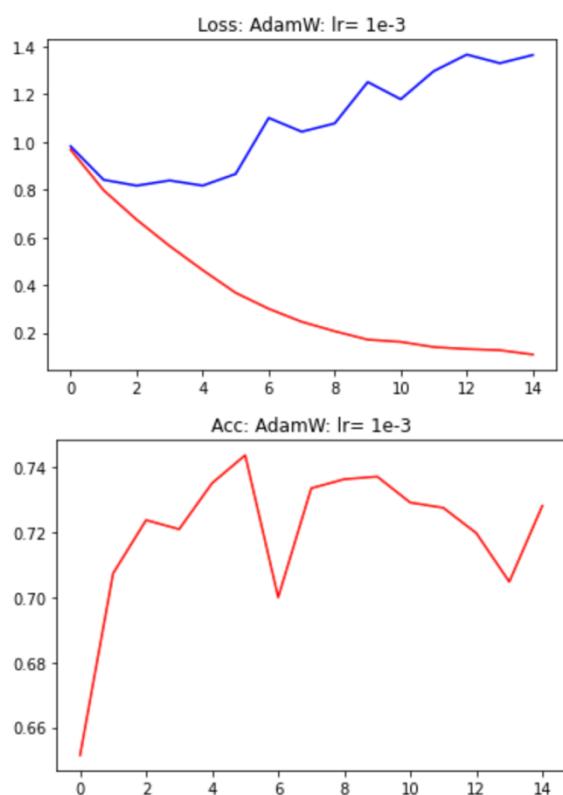
Obrázok 11 Trénovacia a validačná chyba pre optimalizátor SGD lr=1e-3), Graf 2: Presnosť modelu pre optimalizátor SGD (lr=1e-3)



Obrázok 12 Trénovacia a validačná chyba pre optimalizátor RMSprop ($lr=1e-2$, $eps=1.1$), Graf 2: Presnosť modelu pre optimalizátor RMSprop ($lr=1e-2$, $eps=1.1$)

RMSprop

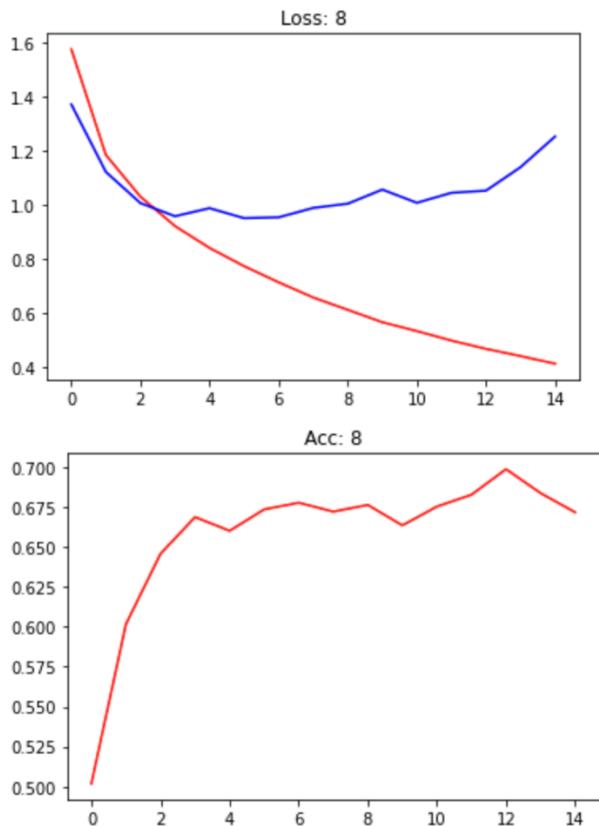
Ako trietu možnosť sme sa pokúsili zmeniť optimalizátor Adam na optimalizátor RMSprop s hodnotou learning rate = $1e-2$ a epsilon = 1.1. Ako môžeme vidieť na obrázku 12, oproti základnému modelu spôsobila táto zmena nestabilitu vo validačnej chybe a presnosti v rozmedzí 7mej aj 9tej epochy. Nestabilitu spôsobila chybná voľba parametra epsilon, ktorý mal byť malá konštantá, ale my sme si zvolili hodnotu 1.1. V najlepšom prípade dosiahol model 67.5%, čo je o 4.76% menej oproti základnému modelu.



Obrázok 13 Trénovacia a validačná chyba pre optimalizátor AdamW ($lr=1e-3$), Graf 2: Presnosť modelu pre optimalizátor AdamW ($lr=1e-3$)

AdamW

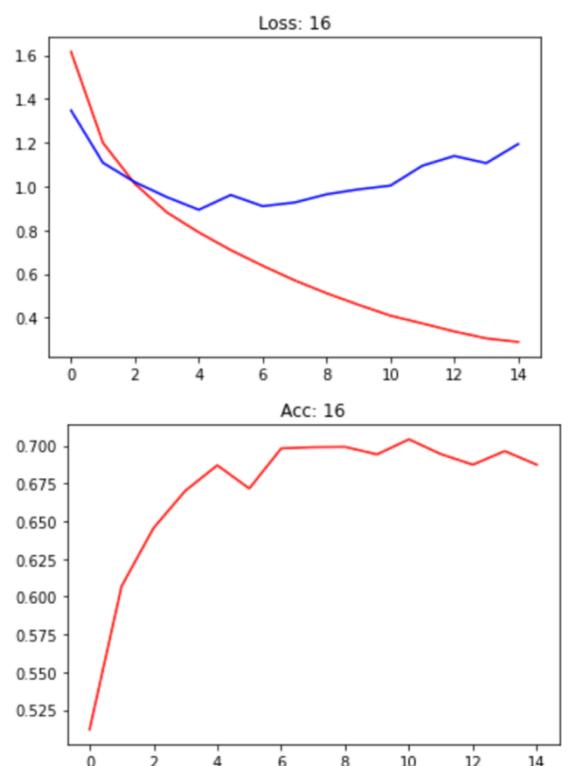
Ako poslednú možnosť sme sa pokúsili zmeniť optimalizátor Adam na optimalizátor AdamW s rovnakou hodnotou learning rateu ako v základnom modeli ($1e-3$). Pre optimalizátor AdamW sme sa rozhodli z dôvodu, že model nátrénovaný s týmto optimalizátorom by mal byť schopný lepšej generalizácie. Ako môžeme na obrázku 13 vidieť model dosahuje v najlepšom prípade presnosť 74.38%, čo je o 2.12% zlepšenie oproti základnému modelu. Zároveň ale môžeme pozorovať, že od 5tej epochy nastáva overfitting. Z tohto dôvodu sa v ďalšej časti tejto podúlohy budeme venovať zmene veľkosti batchu pre optimalizátor AdamW.



Obrázok 14 Trénovacia a validačná chyba pre optimalizátor AdamW ($lr=1e-3$, $batch_size=8$), Graf 2: Presnosť modelu pre optimalizátor AdamW ($lr=1e-3$, $batch_size=8$)

AdamW veľkosť batchu 8

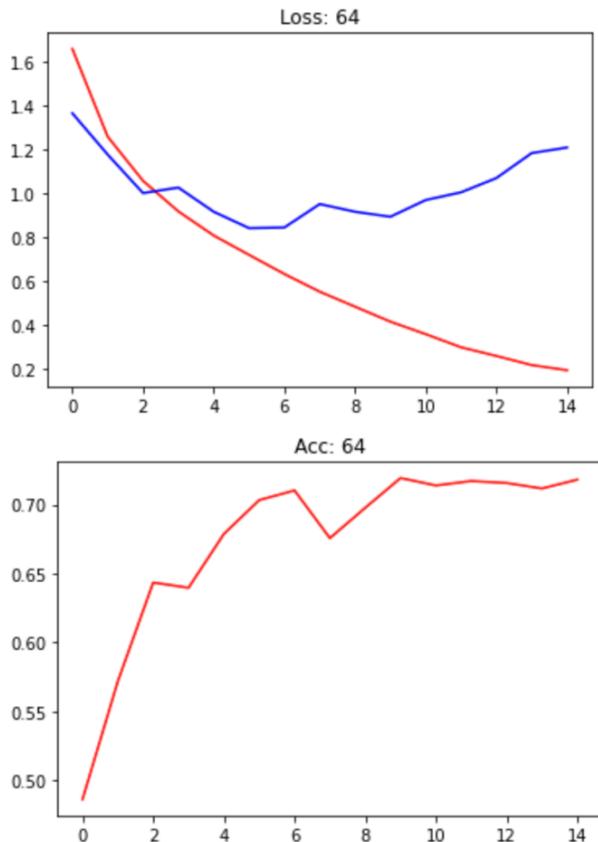
V tejto časti úlohy sme sa rozholi pre model s optimalizátorom AdamW zmeniť veľkosť dávky (batch size) z pôvodných 32 na 8. Ako môžeme na obrázku 14 pozorovať, táto zmena spôsobila to, že sa overfitting výraznejšie prejavil na validačnej chybe až od 12tej epochy. Model ale dosiahol v najlepšom prípade presnosť 69.89%, čo je o 2.37% horšie ako základný model a o 4.49% horšie ako pri veľkosti dávky 32.



Obrázok 15 Trénovacia a validačná chyba pre optimalizátor AdamW ($lr=1e-3$, $batch_size=16$), Graf 2: Presnosť modelu pre optimalizátor AdamW ($lr=1e-3$, $batch_size=16$)

AdamW veľkosť batchu 16

V tejto časti úlohy sme sa rozholi pre model s optimalizátorom AdamW zmeniť veľkosť dávky (batch size) z pôvodných 32 na 16. Ako môžeme na obrázku 15 pozorovať, táto zmena spôsobila to, že sa overfitting výraznejšie prejavil na validačnej chybe až od 8nej epochy. Model ale dosiahol v najlepšom prípade presnosť 70.42%, čo je o 1.84% horšie ako základný model a o 2.12% horšie ako pri veľkosti dávky v predchádzajúcej časti úlohy (veľkosť dávky 32).



AdamW veľkosť batchu 64

V poslednej časti tejto podúlohy sme sa rozholi pre model s optimalizátorom AdamW zmeniť veľkosť dávky (batch size) z pôvodných 32 na 64. Ako môžeme na obrázku 15 pozorovať, táto zmena spôsobila to, že sa overfitting výraznejšie prejavil na validačnej chybe až od 9tej epochy. Model ale dosiahol v najlepšom prípade presnosť 71.92%, čo je o 0.34% horšie ako základný model a o 2.46% horšie ako pri veľkosti dávky v predchádzajúcej časti úlohy (veľkosť dávky 32).

Obrázok 16 Trénovacia a validačná chyba pre optimalizátor AdamW ($lr=1e-3$, $batch_size=64$), Graf 2: Presnosť modelu pre optimalizátor AdamW ($lr=1e-3$, $batch_size=64$)

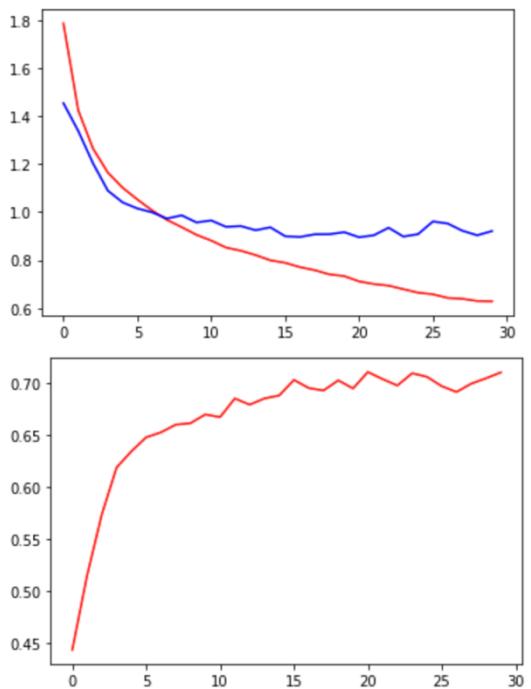
Z predchádzajúcich grafov môžeme usúdiť, že z pohľadu presnosti mal najlepšie výsledky model s optimalizátorom AdamW a veľkosťou dávky 32. Z pohľadu stability bol na tom najhoršie model natrénovaný pomocou optimalizátora SGD s rovnakým learning ratom ako optimalizátor Adam v základnom modeli. Z tejto časti úlohy usudzujeme, že pri vytváraní najlepšieho modelu použijeme optimalizátor AdamW s veľkosťou dávky 32, ktorý zlepší prenosť modleu až o 2.12%.

5. Dropout a Augmentácia

V tejto časti úlohy sme sa zaobrali pridaním dropoutu do základného modelu. Najprv len v plne prepojenej vrstve ($p = 0.5$), potom aj po konvolučných vrstvách ($p = 0.2$) a na záver sme otestovali aj jednu vlastnú možnosť, ktorú si popíšeme v bode **Vlastná možnosť**.

V ďalšej časti úlohy sme sa zaobrali augmentáciou dát, kde sme si zvolili 3 rôzne nastavenia, ktoré si predstavíme v samostaných podčastiach tejto podúlohy.

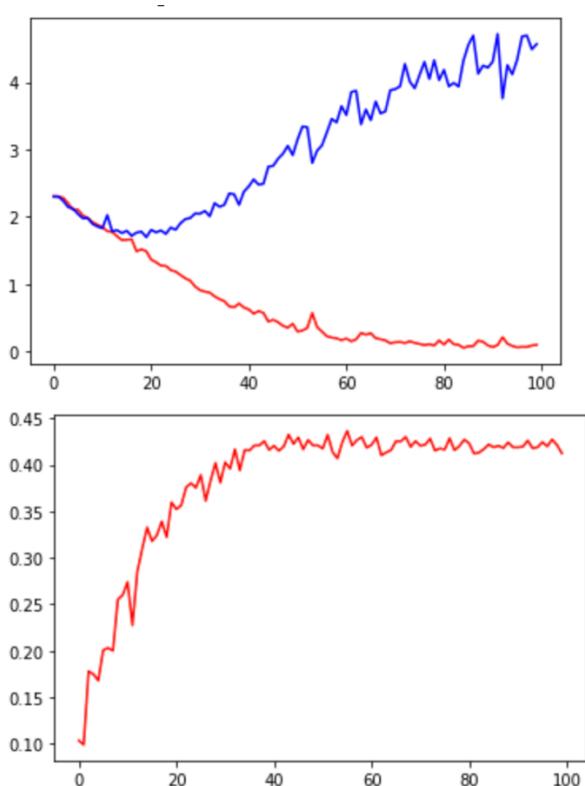
Na záver sme pre všetky spomínané prípady vyskúšali trénovanie s počtom trénovacích dát 1000 (z pôvodných 45 000). Trénovanie prebehlo v 40tich epochách pre pôvodný počet dát a v 100 epochách pre 1000 testovacích obrázkov.



Obrázok 17 Trénovacia a validačná chyba pre dropout v plne prepojenej vrstve ($p = 0.5$) Graf 2: Presnosť modelu pre dropout v plne prepojenej vrstve ($p = 0.5$)

Dropout v plne prepojenej vrstve ($p = 0.5$)

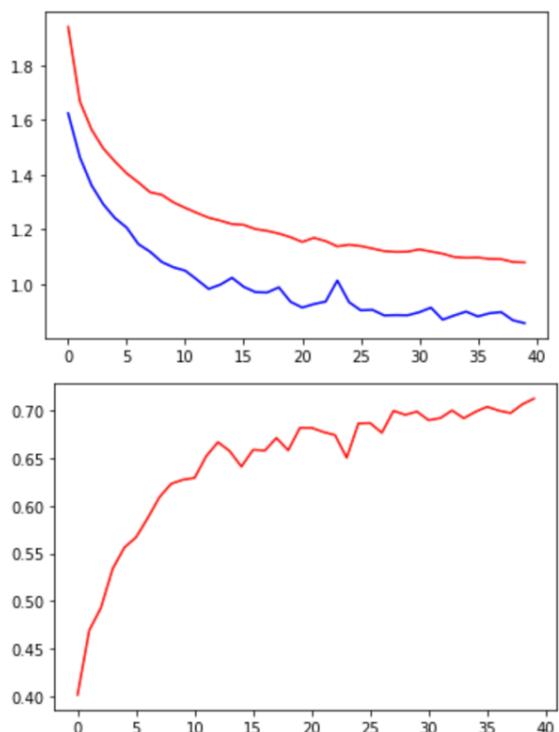
Ako môžeme na obrázku 17 vidieť, pridaním dropoutu ($p=0.5$) po plne prepojených vrstvách (po aktivačnej funkcií ReLU) sa výrazne zvýšil počet epôch trénovania bez overfittingu oproti základnému modelu (čo je aj cieľom dropoutu). Model dosahuje presnosť 70.9%, čo je o 1.36% menej oproti základnému modelu.



Obrázok 18 Trénovacia a validačná chyba pre dropout v plne prepojenej vrstve ($p = 0.5$) pre 1000 trénovacích dát. Graf 2: Presnosť modelu pre dropout v plne prepojenej vrstve ($p = 0.5$) pre 1000 trénovacích dát

Dropout v plne prepojenej vrstve ($p = 0.5$) iba 1000 trénovacích dát

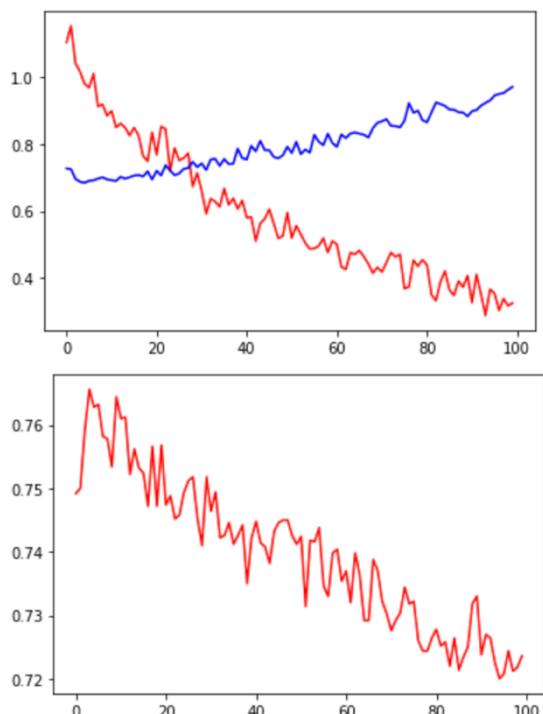
Ako môžeme na obrázku 18 vidieť, pridaním dropoutu ($p=0.5$) po plne prepojených vrstvách (po aktivačnej funkcií ReLU) a následným natrénovaním na zmenšenej trénovacej množine sme od cca 25tej epochy narazili na overfitting, čo považujeme za dobrý výsledok vzhľadom na to, že pri pôvodnom modeli sme na overfitting narazili už po 10tej epoche s výrazne väčším počtom dát. Takto natrénovaný model dosahuje presnosť 42.72%, čo je o 29.54 menej oproti základnému modelu.



Obrázok 19 Trénovacia a validačná chyba pre dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$) Graf 2: Presnosť modelu pre dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$)

Dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$)

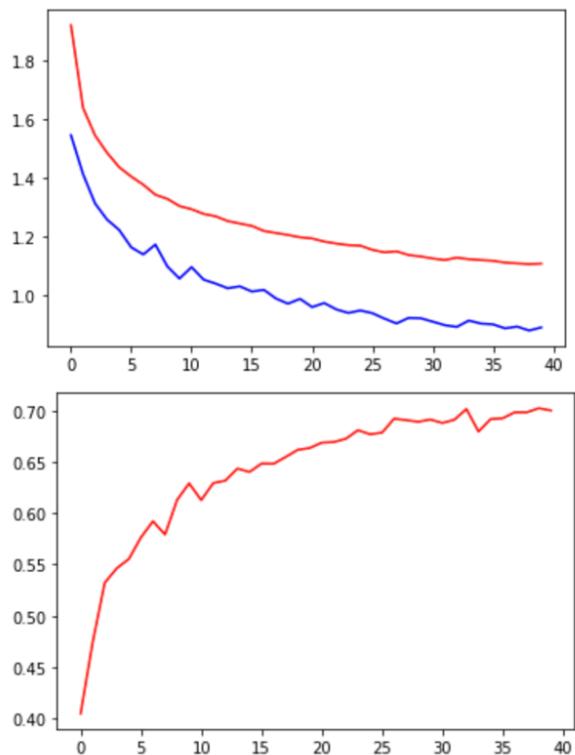
Ako môžeme na obrázku 19 vidieť, pridaním dropoutu ($p=0.5$) po plne prepojených vrstvách a aj po konvolučných vrstvách ($p = 0.2$) (po aktivačnej funkcií ReLU) sa oproti predchádzajúcej možnosti znížila validačná chyba oproti trénovacej chybe a rovnako nenastal ani po 40tich epochách overfitting. Model dosahuje presnosť 71.22%, čo je o 1.04% menej oproti základnému modelu.



Obrázok 20 Trénovacia a validačná chyba pre dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$) pre 1000 trénovacích dát Graf 2: Presnosť modelu pre dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$) pre 1000 trénovacích dát

Dropout v plne prepojenej vrstve ($p = 0.5$) a aj po konvolučných vrstvách ($p = 0.2$) iba 1000 trénovacích dát

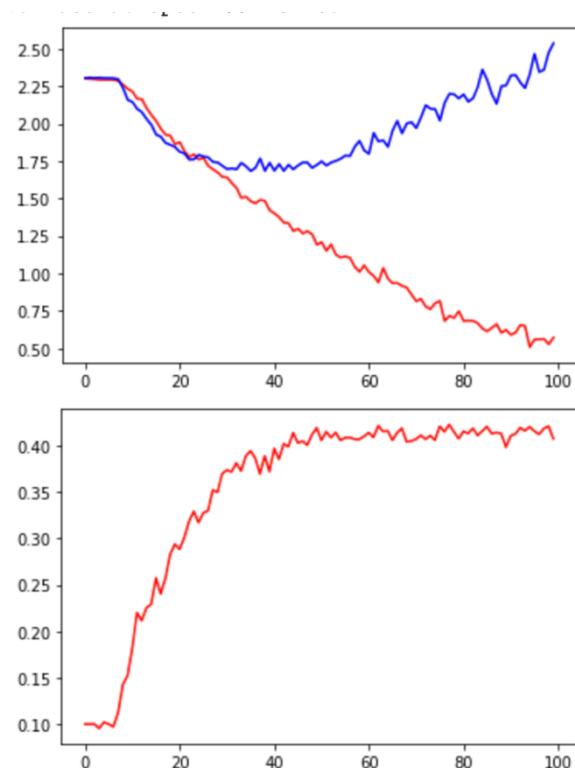
Ako môžeme na obrázku 20 vidieť, v tejto možnosti sme od cca 25tej epochy narazili na overfitting a presnosť modelu sa začala výrazne znížovať už po 15tej epochhe. Z tohto môžeme usúdiť, že pre nás model a znížený počet trénovacích dát nie je vhodné pridať dropout aj po konvolučných vrstvách.



Obrázok 21 Trénovacia a validačná chyba pre vlastnú možnosť
Graf 2: Presnosť modelu pre vlastnú možnosť

Vlastná možnosť

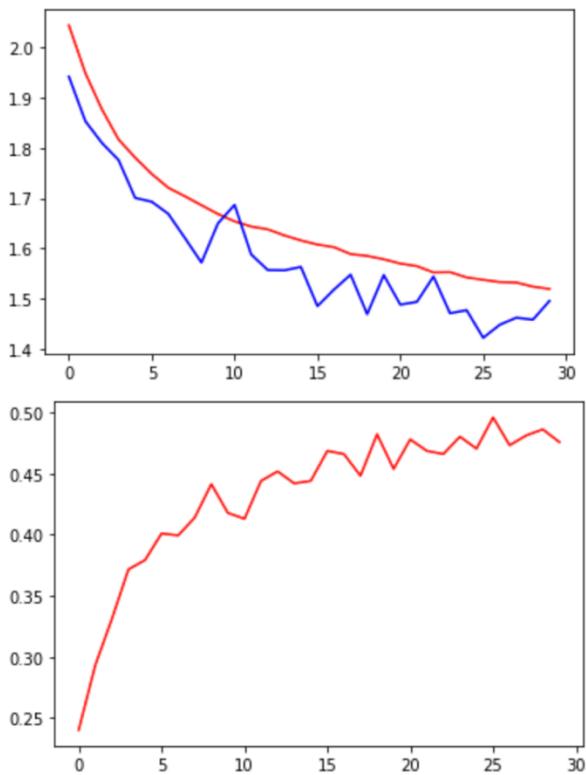
V tejto časti sme sa rozhodli doplniť dropout pre plne prepojené siete a pre konvolučné siete nie po aktivačnej funkcií (v našom prípade ReLU), ale rovno po konvolučných a plne prepojených vrstvach. Zistili sme, že v našom prípade sme nezaznamenali výraznú rozdiel vo výsledkoch oproti modelu, kde sa dropout použil až po aktivačnej funkcií (obrázok 21).



Obrázok 22 Trénovacia a validačná chyba pre vlastnú možnosť a 1000 trénovacích dát Graf 2: Presnosť modelu pre vlastnú možnosť a 1000 trénovacích dát

Vlastná možnosť iba 1000 trénovacích dát

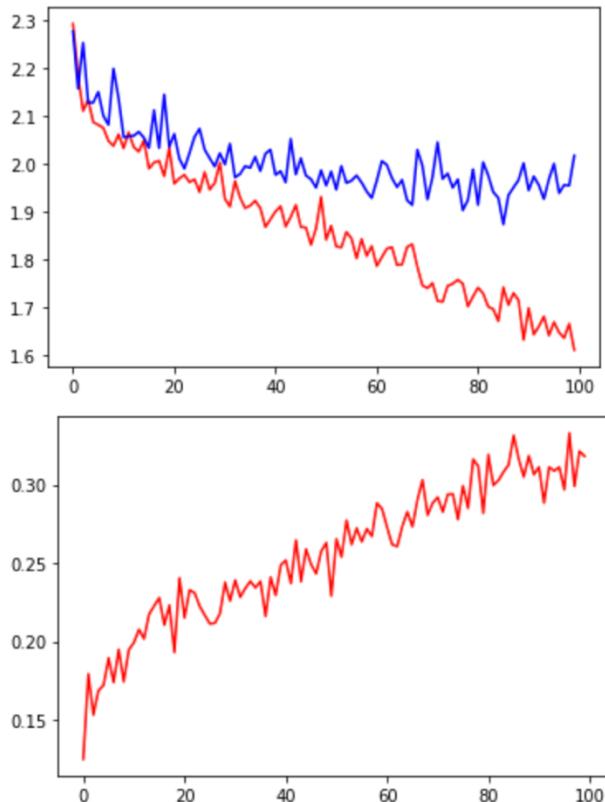
Na menšom počte dát si môžeme všimnúť, že oproti možnosti kde sme použili dropout po aktivačnej funkcií je model stabilnejší a jeho presnosť sa s pribúdajúcim počtom epôch neznižuje (overfitting pozorujeme až po 50tej epoche, v možnosti po aktivačnej funkcií to bolo po 15tej epoche).



Obrázok 23 Trénovacia a validačná chyba pre prvéj augmentáciu
Graf 2: Presnosť modelu pre prvéj augmentáciu

Augmentácia nastavenie 1

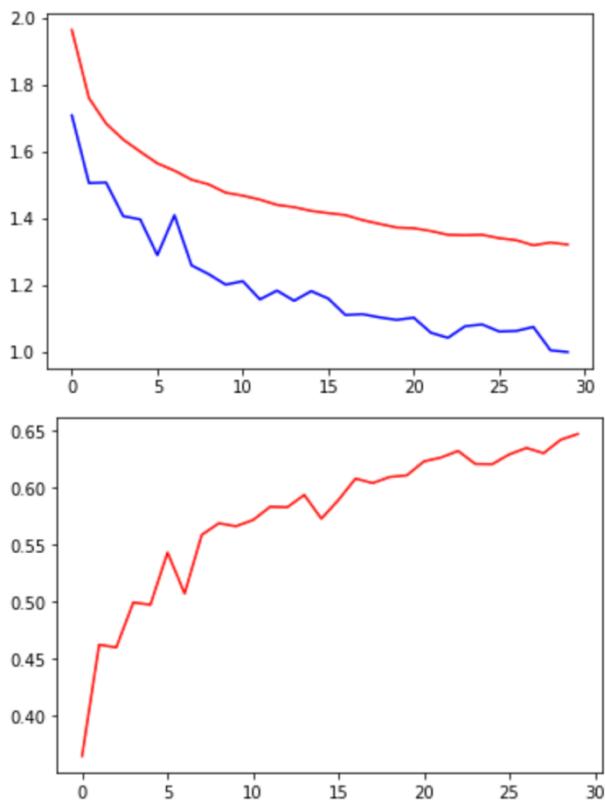
Pri augmentácii sme sa ako prvú možnosť rozhodli pomocou funkcie `Compose`, ktorá spája viaceré transformácie spojiť transformácie pre náhodnú rotáciu obrazu, ďalej pre náhodné vertikálne a horizontálne otočenie a transformáciu pre orezanie náhodnej časti obrázka (s následným zmenením rozlíšenia na pôvodných 32x32). Ako môžeme na obrázku 23 vidieť, pomocou spomenutej augmentácie sme dosiahli na 30tich epochách presnosť modelu 49.6%, čo je o 22.6% menej oproti základnému modelu.



Obrázok 24 Trénovacia a validačná chyba pre prvéj augmentáciu a 1000 trénovacími dátami Graf 2: Presnosť modelu pre prvéj augmentáciu a 1000 trénovacími dátami

Augmentácia nastavenie 1 iba 1000 trénovacích dát

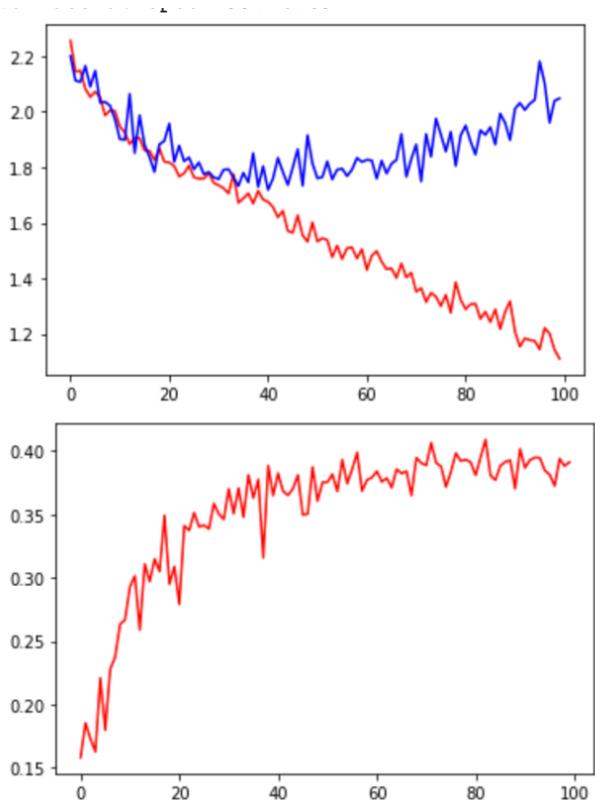
Na menšom počte trénovacích dát môžeme na obrázku 24 vidieť, že nami zvolené transformácie spôsobujú miernu nestabilitu modelu (krivky na grafoch nie sú hladké) a od 40tej epochy nesledujeme ani pokles validačnej chyby. Model natrénovaný na menšej množine dát dosahuje v najlepšom prípade presnosť 32.14% (pri trénovaní na rovnakom počte dát sme pri pridaní dropoutu v rôznych konfiguráciach dosahovali oveľa lepšie výsledky).



Obrázok 25 Trénovacia a validačná chyba pre druhú augmentáciu Graf 2: Presnosť modelu pre druhú augmentáciu

Augmentácia nastavenie 2

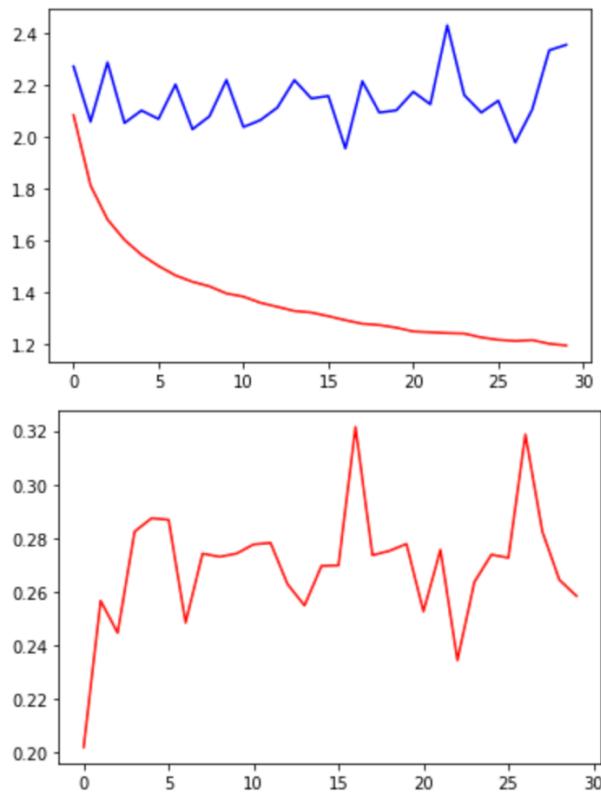
Ako druhú možnosť pre augmentáciu sme sa rozhodli použiť náhodné vertikálne a horizontálne otočenie a Gaussov filter, ktorý redukuje šum v obraze a znižuje detaily. Ako môžeme z obrázka 25 vidieť, takéto nastavenie dostalo oproti predošej verzie augmentácie lepšie výsledky v presnosti modelu a viac stabilnú krivku pre validačnú chybu. Model dosahuje presnosť 64.72%, čo je o 7.54% menej oproti základnému modelu.



Obrázok 26 Trénovacia a validačná chyba pre druhú augmentáciu a 1000 trénovacích dát Graf 2: Presnosť modelu pre druhú augmentáciu a 1000 trénovacích dát

Augmentácia nastavenie 2 iba 1000 trénovacích dát

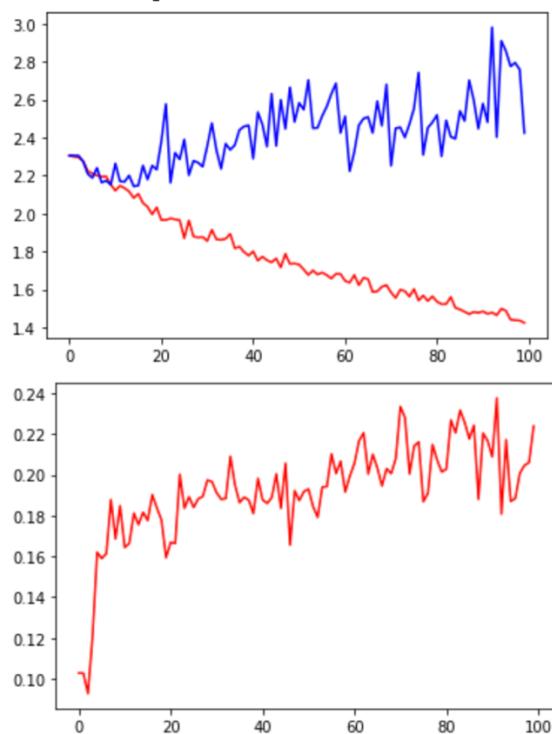
Na menšom počte trénovacích dát môžeme na obrázku 25 vidieť, že nami zvolené transformácie spôsobujú rovnako ako v predchádzajúcim nastavení miernu nestabilitu modelu a zvyšovanie validičnej chyby môžeme pozorovať od 70tej epochy. Model natrénovaný na menšej množine dát dosahuje v najlepšom prípade presnosť 40.18%



Obrázok 27 Trénovacia a validačná chyba pre tretiu augmentáciu Graf 2: Presnosť modelu pre tretiu augmentáciu

Augmentácia nastavenie 3

Pri poslednej možnosti sme sa chceli presvedčiť, že zvolením nesprávnych transformácií sa nedopracujeme k zlepšeniu presnosti modelu a nepredídeme ani overfittingu. Preto sme sa rozhodli použiť funkciu ColorJitter, ktorá náhodne zmení jas, kontrast, sýtosť a odtien obrázka. Následne sme ešte využili funkciu pre náhodnú affinnú transformáciu. Ako môžeme z obrázka 27 vidieť, za pomoci spomenutých transformácií sme dosiahli nestabilitu validačnej chyby hneď od prvej epochy a aj v najlepšom prípade dosahovala presnosť iba 31.86%, čo je najmenšia presnosť modelu s pôvodným počtom dát akú sme v tejto úlohe dosiahli.



Obrázok 28 Trénovacia a validačná chyba pre tretiu augmentáciu a 1000 trénovacích dát Graf 2: Presnosť modelu pre tretiu augmentáciu a 1000 trénovacích dát

Augmentácia nastavenie 3 iba 1000 trénovacích dát

Pri menšom počte dát môžeme z obrázka 28 vidieť nestabilitu validačnej chyby ako aj jej nárast od 10tej epochy a v najlepšom prípade dosahovala presnosť iba 22.38%, čo je rovnako najmenšia presnosť modelu so zmenšeným počtom dát akú sme v tejto úlohe dosiahli.

Z tejto časti úlohy sme sa dozvedeli, že pomocou správneho použitia dropoutu a augmentácie vieme dosiahnuť výrazne stabilnejšiu krivku validačnej chybe a predísť overfittingu aj pri výrazne menšom počte dát. Naopak pri nesprávnej voľbe vieme dosiahnuť vysokú nestabilitu a výrazne znížiť presnosť modelu. Preto sa pri pokuse vytvoriť najlepší model budeme zaoberať výberom vhodného umiestnenia dropoutu a výberu vhodného nastavenia augmentácie.

6. Hlboká siete

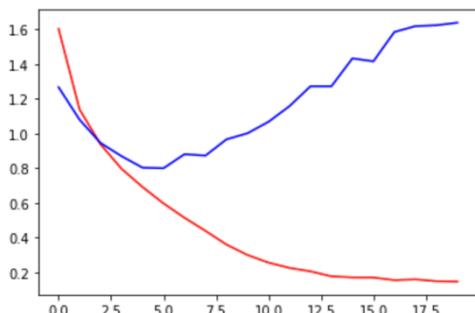
V tejto časti úlohy sme sa zaoberali tvorbou hlbokej siete z konvolučných a pooling vrstiev. Následne sme vytvorili ešte 2 modeli, kde sme v jednom pridali plne prepojené vrstvy a do ďalšieho Batch Norm po konvolučných a plne prepojených vrstvách.

Základná architektúra hlbokej siete

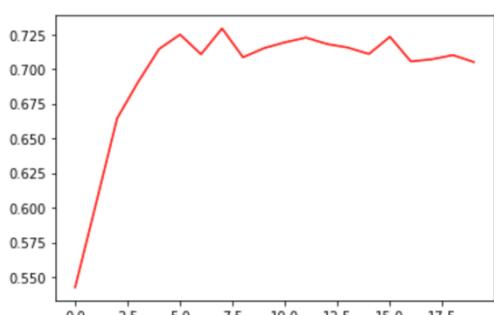
Pri tvorbe základnej architektúry sme sa rozhodli využiť 11 vrstiev a finálnu architektúru môžeme vidieť na obrázku 29.

```
Sequential(  
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (2): ReLU()  
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (5): ReLU()  
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (8): ReLU()  
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (11): ReLU()  
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (14): ReLU()  
    (15): Conv2d(512, 10, kernel_size=(2, 2), stride=(1, 1))  
    (16): Flatten(start_dim=1, end_dim=-1)  
)
```

Obrázok 29 Základná architektúra hlbokej siete



Ako môžeme z obrázka 30 vidieť, základná architektúra hlbokej siete dosahuje v najlepšom prípade presnosť 72.32%, čo je o 0.06% lepšie ako základný model. Overfitting môžeme na validačnej chybe sledovať od 7mej epochy.



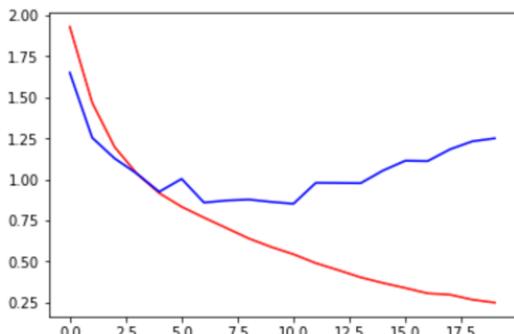
Obrázok 30 Trénovacia a validačná chyba pre základnú architektúru hlbokej siete
Graf 2: Presnosť modelu pre pre základnú architektúru hlbokej

Hlboká sieť s plne prepojenými vrstvami

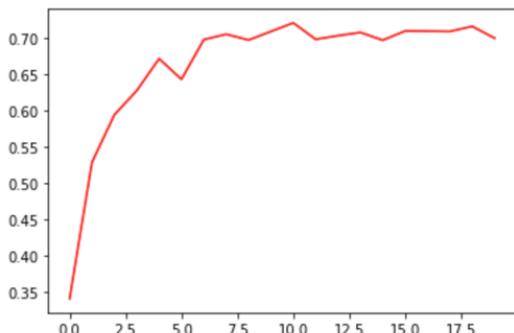
V tejto časti úlohy sme za operáciou Flatten pridali 5 plne prepojených vrstiev s aktivačnou funkciou ReLU. Architektúru siete môžeme vidieť na obrázku 31.

```
Sequential(  
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (2): ReLU()  
    (3): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (5): ReLU()  
    (6): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (8): ReLU()  
    (9): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (11): ReLU()  
    (12): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (13): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (14): ReLU()  
    (15): Flatten(start_dim=1, end_dim=-1)  
    (16): Linear(in_features=2048, out_features=1024, bias=True)  
    (17): ReLU()  
    (18): Linear(in_features=1024, out_features=512, bias=True)  
    (19): ReLU()  
    (20): Linear(in_features=512, out_features=256, bias=True)  
    (21): ReLU()  
    (22): Linear(in_features=256, out_features=128, bias=True)  
    (23): ReLU()  
    (24): Linear(in_features=128, out_features=10, bias=True)  
)
```

Obrázok 31 Architektúra hlbokej siete s plne prepojenými vrstvami



Ako môžeme z grafov na obrázku 32 vidieť, táto architektúra dosahuje v najlepšom prípade presnosť cca 70% a overgitting pri trénovacej chybe vzniká 10tej epochi.



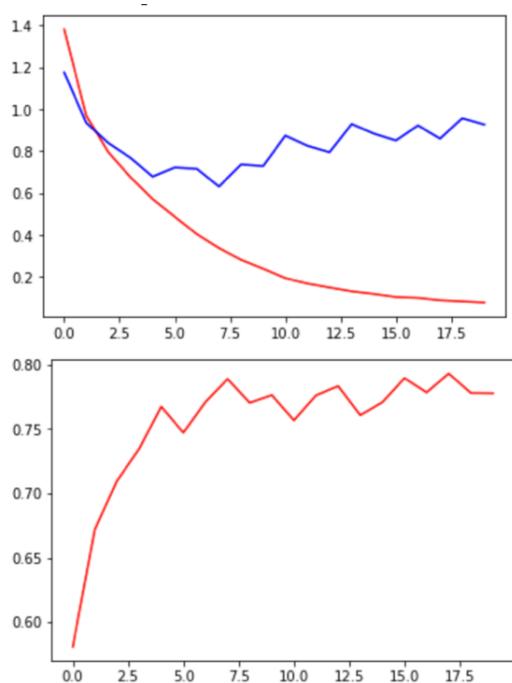
Obrázok 32 Trénovacia a validačná chyba pre hlbokú sieť s plne prepojenými vrstvami Graf 2: Presnosť modelu pre hlbokú sieť s plne prepojenými

Hlboká siet s plne prepojenými vrstvami a Batch normom

V tejto časti úlohy sme sa rozhodli rozšíriť model z predchádzajúcej podúlohy (s plne prepojenými vrstvami) o batch norm po konvolučných (2d verzia) a plne prepojených vrstvách (1D verzia), ktorého úlohou je normalizácia výstupných dát z predchádzajúcej vrstvy pre ďalšiu vrstvu. Finálnu architektúru môžeme vidieť na obrázku 33.

```
Sequential(  
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (3): ReLU()  
    (4): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (6): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (7): ReLU()  
    (8): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (11): ReLU()  
    (12): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (13): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (15): ReLU()  
    (16): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (17): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (18): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (19): ReLU()  
    (20): Flatten(start_dim=1, end_dim=-1)  
    (21): Linear(in_features=2048, out_features=1024, bias=True)  
    (22): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (23): ReLU()  
    (24): Linear(in_features=1024, out_features=512, bias=True)  
    (25): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (26): ReLU()  
    (27): Linear(in_features=512, out_features=256, bias=True)  
    (28): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (29): ReLU()  
    (30): Linear(in_features=256, out_features=128, bias=True)  
    (31): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (32): ReLU()  
    (33): Linear(in_features=128, out_features=10, bias=True)  
)
```

Obrázok 33 Architektúra hlbokej siete s plne prepojenými vrstvami a batch norm



Ako môžeme z grafov na obrázku 34 vidieť, pridaním batch normu sa nám výrazne zlepšila presnosť modelu oproti základnému modelu. Presnosť je 79.32%, čo je o 7.06 percenta lepšia presnosť. Overfitting môžeme sledovať po 10tej epoche na validačnej chybe

Obrázok 34 Trénovacia a validačná chyba pre hlbokú sieť s plne prepojenými vrstvami a batch norm
2: Presnosť modelu pre hlbokú sieť s plne prepojenými vrstvami a batch norm

7. Najlepší model

Pri tvorbe najlepšieho modelu sme sa snažili využiť informácie z predošlých úloch, ako aj nodobudnúť nové informácie z internetu. Na obrázku 35 môžeme vidieť finálnu architektúru, ktorú si v nasledujúcich odstavcoch podrobnejšie popíšeme.

```
Sequential(  
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (3): PReLU(num_parameters=1)  
    (4): Dropout2d(p=0.2, inplace=False)  
    (5): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (8): PReLU(num_parameters=1)  
    (9): Dropout2d(p=0.2, inplace=False)  
    (10): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (13): PReLU(num_parameters=1)  
    (14): Dropout2d(p=0.2, inplace=False)  
    (15): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (16): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (17): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (18): PReLU(num_parameters=1)  
    (19): Dropout2d(p=0.2, inplace=False)  
    (20): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(2, 2))  
    (21): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (22): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)  
    (23): PReLU(num_parameters=1)  
    (24): Dropout2d(p=0.2, inplace=False)  
    (25): Flatten(start_dim=1, end_dim=-1)  
    (26): Linear(in_features=2048, out_features=1024, bias=True)  
    (27): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (28): PReLU(num_parameters=1)  
    (29): Dropout(p=0.5, inplace=False)  
    (30): Linear(in_features=1024, out_features=512, bias=True)  
    (31): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (32): PReLU(num_parameters=1)  
    (33): Dropout(p=0.5, inplace=False)  
    (34): Linear(in_features=512, out_features=256, bias=True)  
    (35): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (36): PReLU(num_parameters=1)  
    (37): Dropout(p=0.5, inplace=False)  
    (38): Linear(in_features=256, out_features=128, bias=True)  
    (39): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
    (40): PReLU(num_parameters=1)  
    (41): Dropout(p=0.5, inplace=False)  
    (42): Linear(in_features=128, out_features=10, bias=True)  
)
```

Obrázok 35 Architektúra najlepšieho modelu

Aktivačná funkcia

Na základe experimentovania v bode 2 sme sa pre najlepší model rozhodli využiť aktivačnú funkciu ReLU.

Optimalizátor

Na základe experimentovania v bode 3 sme sa ako optimalizátor rozhodli použiť AdamW, ktorý dosahoval o 2.12% lepšiu presnosť ako základný model voči ktorému sme ho porovnávali. Pri voľbe learning rateu sme sa inšpirovali funkciou adjust_lr z GitHub repozitára na reidentifikáciu vozidiel [b], ktorý využíval trik Warmup learning rate (ale

s inými hodnotami learning rateu), ktorý bol prezentovaný v článku Bag of Tricks and A Strong Baseline for Deep Person Re-identification [c]. V tomto článku zlepšil warmup learning rate presnosť modelu na reidentifikáciu osôb na datasete Market1501 o 1.2%.

Dropout a Augmentácia

Na základe experimentovania v bode 4 sme sa rozhodli využiť Dropout po každej konvolučnej a plne prepojenej vrstve (po aktivačnej funkcií). Dropout je regularizačná technika, ktorá pomáha predchádzať overfittingu (v našom prípade sme model trénovali na 50tich epochách).

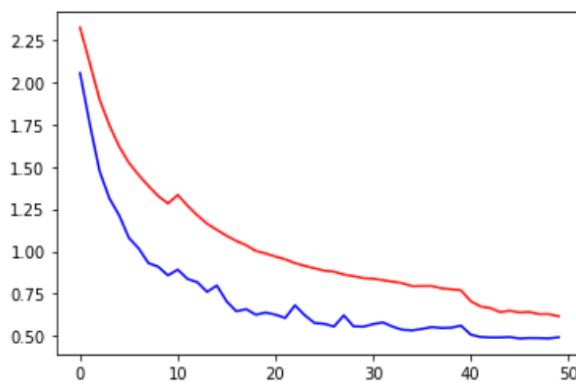
Pri augmentáciu sme sa rozhodli využiť funkciaľnosť automatickej augmentácie [d] , ktorú framework PyTorch ponúka. Pre túto možnosť sme sa rozhodli z dôvodu, že z nášho pohľadu ponúka produkčný framework lepšie voľby transformácií obrazu pre zlepšenie finálnej presnosti modelu ako transformácie, ktoré sme prezentovali my v 4 bode úlohy.

BatchNorm

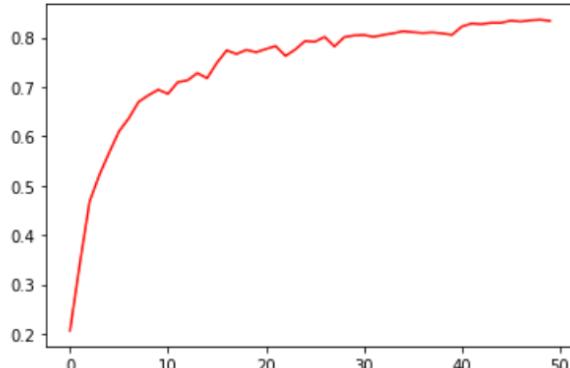
Na základe experimentovania v bode 5 sme sa rozhodli pre pridanie BatchNormu po konvolučných (2d verzia) a plne prepojených vrstvách (1D verzia). Jeho úlohou je normalizácia výstupných dát z predchádzajúcej vrstvy pre ďalšiu vrstvu.

Ďalšie parametre

Na základe skúsenosti pri tvorbe základného modelu sme sa rozhodli pri konvolučných vrstvách využiť veľkosť batchu 32, veľkosť jadra 3x3, stride 1 a padding 2x2 aby sme zachytili informácie aj z okrajov obrazu. Model je natrénovaný pomocou 50tich epoch a výsledný graf presnosti a chýb môžeme vidieť na obrázku 36.



Ako môžeme z grafov vidieť, nás najlepší model dosahuje presnosť 83.36%, čo je oproti základnému modelu výrazne zlepšenie. Konkrétnie ide o zlepšie 11.1% na datasete CIFAR-10. Ďalej môžeme vidieť, že ani pri počte epôch 50 sme pri validčnej chybe nenašli na overfitting.



Obrázok 36 Trénovacia a validačná chyba najlepší model Graf 2: Presnosť modelu pre najlepší model

8. Záver

V tomto dokumente sme si popísali výsledky a nastavenia jednotlivých parametrov a trikov používaných pri trénovaní konvolučných neurónových sieti na datasete CIFAR-10. Na základe informácií nadobudnutých z bodov 1 až 6 a informácií z internetu sme v bode 7 vytvorili architektúru pre najlepší model, ktorý dosahuje presnosť 83.36%, čo je oproti základnému modelu zlepšenie o 11.1%.

Zdroje

- [a] <https://www.cs.toronto.edu/~kriz/cifar.html>
- [b] https://github.com/lxc86739795/vehiclereid_baseline/blob/6eb91630a8c1d767157ef7d1b84c3c5ccfdd7a22/train.py#L176
- [c] https://openaccess.thecvf.com/content_CVPRW_2019/papers/TRMTMCT/Luo_Bag_of_Tricks_and_a_Strong_Baseline_for_Deep_Person_CVPRW_2019_paper.pdf
- [d] https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#autoaugment