Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce
Rozpoznávání izolovaných znaků znakového jazyka

Plzeň 2022 **Jakub Honzík**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2021/2022

Studijní program: Kybernetika a řídicí technika
Forma studia: Prezenční
Specializace/kombinace: Umělá inteligence
a automatizace (UIA18bp)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: **Jakub HONZÍK**
Osobní číslo: **A19B0355P**

Téma práce: Rozpoznávání izolovaných znaků znakového jazyka

Téma práce anglicky: Isolated sign language recognition

Vedoucí práce: Ing. Marek Hrúz, Ph.D.
Výzkumný program 1

Zásady pro vypracování:

1. Seznamte se s knihovnou PyTorch pro práci s hlubokými neuronovými sítěmi
2. Zvolte a nastudujte vhodný model rozpoznávání izolovaných znaků
3. Otestujte tento model na vhodné databázi izolovaných znaků
4. Výsledky vyhodnoťte a navrhněte případná vylepšení

Seznam doporučené literatury:

Dodá vedoucí práce

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 23. května 2022

Jakub Honzík

Poděkování

Tímto bych rád poděkoval vedoucímu této bakalářské práce Ing. Marku Hrúzovi, Ph.D. za odbornou pomoc, pedagogické nasazení, poskytování cenných rad a přátelské, trpělivé chování. Dále bych rád poděkoval rodině a přátelům za psychickou podporu.

Abstract

In this thesis I'm pursuing Argentinian sign language recognition using artifitial neural network. Training and testing data are from dataset LSA64. Parameter optimalization of neural network and its evaluation were realised with framework MMAction2. I've tested optimizers SGD and Adam, for which I've found suitable order of learning rate, for SGD it was 0.001 and for Adam $1 \cdot 10^{-5}$. For these two settings I carried out leave-one-out cross-validation. The results show, that the network using SGD finishes training approximately 10 epochs sooner. Accuracy of SGD for cross-validation on test set is 0.947 ± 0.027, Adam has accuracy 0.937 ± 0.018. Experiments suggest, that in this case optimizer SGD is more suitable, if there's more emphasis on accuracy. Optimizer Adam is more suitable, if there's more emphasis on robustness.

Key words

Sign language, neural networks, Temporal segment networks, ResNet, MMAction2, LSA64

Abstrakt

V této práci se věnuji rozpoznávání jednotlivých znaků Argentinské znakové řeči pomocí umělé neuronové sítě. Trénovací a testovací data pocházejí z datasetu LSA64. Optimalizce parametrů neuronové sítě a její vyhodnocení jsem realizoval ve frameworku MMAction2. Testoval jsem optimalizátory SGD a Adam, pro které jsem nalezl vhodný řád konstanty učení, pro SGD to bylo 0,001 a pro Adam $1 \cdot 10^{-5}$. Pro tyto dvě nastavení jsem provedl křížovou validaci s vynecháním jednoho figuranta. Výsledky ukazují, že síť využívající SGD se na datech natrénuje o zhruba 10 epoch dříve. Úspěšnost SGD při křížové validaci na testovací množině je $0,947 \pm 0,027$, Adam má úspěšnost $0,937 \pm 0,018$. Z experimentů vyplývá, že v tomto případě je vhodnější použití optimalizátoru SGD, pokud je větší důraz na přesnost. Optimalizátor Adam je vhodnější, pokud je větší důraz na robustnost.

Klíčová slova

Znaková řeč, neuronové sítě, Temporal segment networks, ResNet, MMAction2, LSA64

Obsah

Úvod

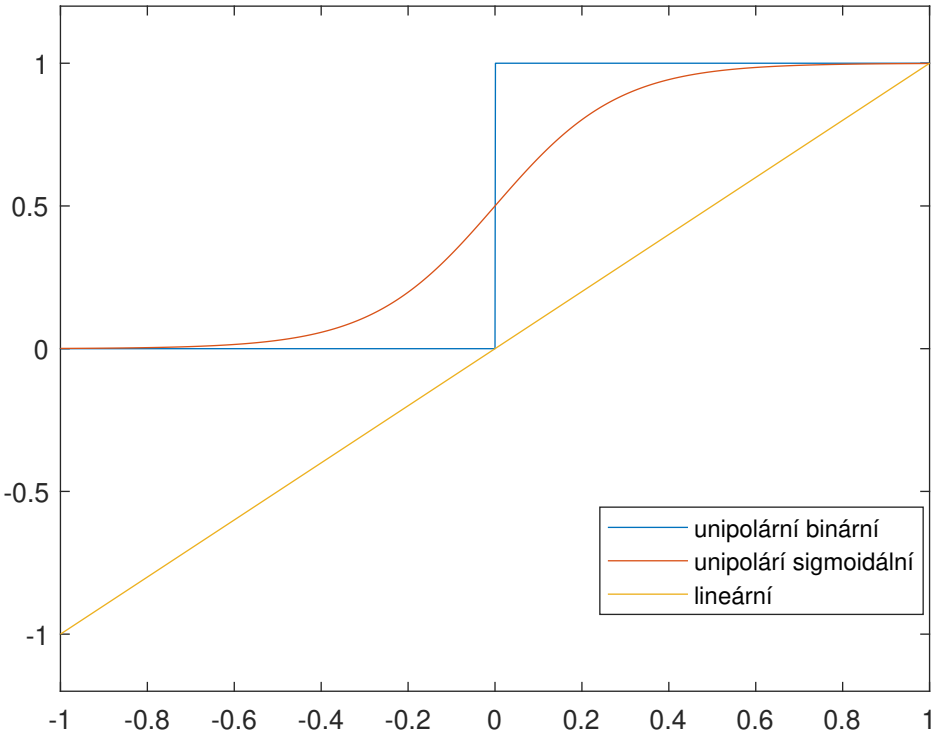
Část lidské populace trpí poruchou sluchu na takové úrovni, že nedokážou rozumět mluvené řeči. Mnozí z nich jsou dokonce takto znevýhodněni dříve než se naučí mluvit. Alternativou místo mluvené řeči je pro ně znaková řeč. Motivací této práce je uzpůsobit více technologie pro to, aby je mohly lépe využívat tito lidé. Konkrétně dokázat z nahrávky znakové řeči rozpoznat jednotlivé znaky, které řečník ukazuje. Pro rozpoznávání využívám umělou neuronovou síť. V rámci synergie s katedrou kybernetiky jsem využil otevřený software pro porozumění videí znázorňujících akce MMAction2 [7], který je součástí projektu OpenMMLab [4]. Vzhledem k velkým zkušenostem na katedře kybernetiky s architekturami I3D [3] a TimeSformer [2] jsem po domluvě s vedoucím práce zvolil architekturu Temporal segment networks (TSN) [33], která je zaběhnutá, není v experimentální fázi a zároveň není tak prozkoumaná na katedře, tudíž tato práce bude mít i přínos k poznání na katedře. V době psaní této práce je TSN na 17. místě v přesnosti 5 hypotéz na datasetu Moments in Time [20], chtěl bych v této práci zjistit jeho vlastnosti pro rozpoznávání znaků znakové řeči. Síť je trénována na datasetu LSA64 [25], který obsahuje 64 rozdílných znaků, má tedy vhodnou obtížnost pro seznamování se s umělými neuronovými sítěmi. Cílem práce je nalézt pro tento dataset vhodné hyperparametry jako jsou optimalizátor a konstanta učení, a zvolené nastavení otestovat. Zdrojové kódy použité pro experimenty této práce a návod na jejich použití jsou k nalezení na stránce: https://github.com/honzikjak/BP-rozpoznavani-znakove-reci-LSA64

1 Neuronová síť

Neuronová síť je složená funkce vzniklá propojením jednotlivých matematických modelů neuronů [26]. Tyto neurony jsou na sebe napojovány se snahou napodobit aktivitu lidského mozku. Funkce jednoho neuronu je definovaná rovnicí 1.1

$$y = f\left(\sum_{i=1}^{n}(w_i \cdot x_i + b)\right), \tag{1.1}$$

kde f je aktivační funkce, w_i jsou váhy neuronu, b je práh neuronu, x_i jsou vstupy a n je počet vah neuronu. Jako aktivační funkce se mohou použít například binární, sigmoidální, nebo lineární funkce. Jejich průběh je zobrazen na Obrázku 1.



Obrázek 1.1: aktivační funkce

Tyto neurony bývají uspořádány do vrstev, které na sebe navazují. Do vstupní

vrstvy se načte informace, například vektor, nebo matice. Vrstva tuto informaci zpracuje a vyšle novou informaci do další vrstvy, která obdobně tato data zpracuje. Poslední vrstva se nazývá výstupní, její výstup je odpověď sítě na vstupní data. Vrstvám, které nejsou vstupní ani výstupní se říká skryté vrstvy. Tato práce se bude zabývat dopřednou sítí určenou pro klasifikaci. V úloze klasifikace je cílem správné zařazení vstupních dat do třídy. Obvykle má výstupní vrstva stejný počet neuronů jaký je počet tříd do kterých se klasifikuje. Pro neuron odpovídající správné třídě by měl být výstup roven 1, pro všechny ostatní 0. Ve výstupní vrstvě se často využívá funkce Softmax

$$f = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}. \tag{1.2}$$

Výstup z funkce Softmax se dá interpretovat jako hodnota důvěry, že vstup náleží do třídy i. Samotná uspěšnost správného zařazení se však může od této důvěry lišit.

1.1 Trénování

Pro trénování neuronové sítě je zapotřebí mnoho dat. Čím složitější síť, tím více dat. Data jsou rozdělována na trénovací, validační a testovací množinu. Trénovací množina je určena pro nastavování parametrů sítě tak, aby správně reagovala na trénovací data. Validační množina je určená pro průběžné kontrolování, jestli se síť nenastavuje moc konkrétně na trénovací data, čímž by ztrácela obecnou přesnost. Testovací data jsou využita po dokončení trénování pro konečné vyhodnocení sítě. Parametry sítě jsou měněny pouze pomocí trénovací množiny. Síť určená pro rozpoznávání se trénuje učením s učitelem. Do síti vstupují data z trénovací množiny a odezva sítě na tyto data je porovnána se správným označením od učitele. Soubor s odkazem na data a jejich označením se nazývá anotace. Trénování probíhá pomocí metody backpropagation [14]. Hodnoty vah a prahů neuronů v síti se upravují na základě chyby, která se zpětně šíří z výstupu. Parametry se mění ve směru záporného gradientu chyby. Chyba se počítá pomocí ztrátové funkce. Při klasifikaci se jako ztrátová funkce nejčastěji používá křížová entropie

$$L = -\sum_{i=1}^{c} P_i \log Y_i, \tag{1.3}$$

kde P_i je požadovaná hodnota, na pozici cílové třídy je rovna 1, jinde 0, Y_i je výstup i-tého neuronu a c je počet tříd. Parametry sítě jsou pak z chyby modifikovány pomocí optimalizátoru, který určuje, jak jsou upraveny na základě zpětně propagovaného gradientu.

SGD [27] je základní optimalizátor. Parametr p se aktualizuje podle rovnice 1.4

$$p_{t+1} = p_t - lr \cdot \mathbf{g_t}, \tag{1.4}$$

kde lr je konstanta učení a $\mathbf{g_t}$ je gradient. Někdy se k výsledku přičítá předchozí změna parametru vynásobená momentem α.

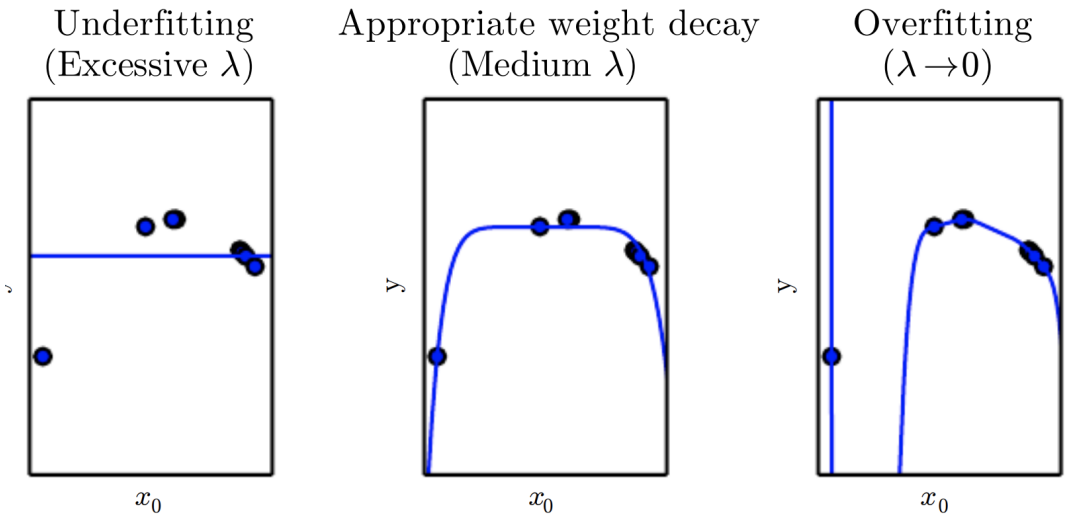
Adam [15] si ukládá exponenciálně rozkládající se střední hodnotu $\mathbf{a_t}$ a varianci $\mathbf{u_t}$ předchozích gradientů

$$p_{t+1} = p_t - \frac{lr}{\sqrt{\mathbf{u_t} + \epsilon}}\mathbf{a_t}, \tag{1.5}$$

kde ϵ je vektor malých čísel aby se zabránilo dělení nulou. Konstanta učení určuje jak moc se změní parametry sítě ve směru záporného gradientu. S moc nízkou hodnotou konstanty učení se sít trénuje moc pomalu. S moc vysokou hodnotou konstanty učení se parametry sítě nedostanou do optima, mohou se od něj dokonce vzdalovat. Hodnota konstanty učení se může v průběhu trénování měnit, například snížením o řád po určitém počtu trénovacích epoch. U otimalizéru se také nastavuje úbytek vah (weight decay), které upravuje hodnotu ztrátové funkce pomocí hodnot vah, jak je ukázáno v rovnici 1.6

$$L_{nov} = L_{pvodn} + \lambda w^T \cdot w, \tag{1.6}$$

kde λ je úbytek vah. Tímto způsobem se dá zabránit přetrénování.



Obrázek 1.2: Nastavení sítě v úloze regrese. Vlevo pro vysokou hodnotu úbytku vah. Uprostřed pro ideální hodnotu úbytku vah. Vpravo pro nízkou hodnotu úbytku vah [31].

Na Obrázku 1.2 je vidět, že pro moc nízkou hodnotu úbytku vah se síť nedostatečně přenastaví, pro ideální hodnotu úbytku vah se síť nastaví správně, pro moc nízkou hodnotu úbytku vah se síť přetrénuje.

Gradient je do hlubších vrstev propagován pomocí řetězového pravidla derivování

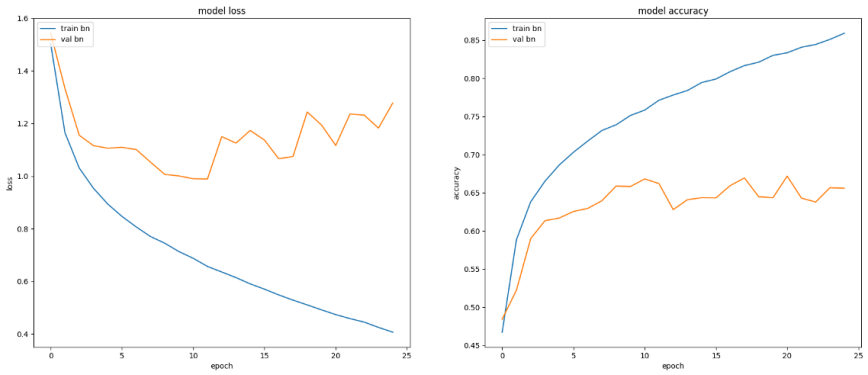
$$\begin{aligned} y &= f(u_1(x,t), u_2(x,t), ..., u_p(x,t)) \\ \frac{\partial y}{\partial x} &= \sum_{j=1}^{p} \frac{\partial f}{\partial u_j} \frac{\partial u_j}{\partial x}. \end{aligned} \tag{1.7}$$

V případě hlubokých sítí, kde se vyskytují vnořené funkce uvnitř vnořených funkcí $f_1(f_2(...(f_n(x)))$ se derivuje podle rovnice 1.8

$$\frac{df_1}{dx} = \frac{df_1}{df_2}\frac{df_2}{df_3}...\frac{df_n}{dx}. \tag{1.8}$$

Platí tedy, že čím hlouběji je měněný parametr, tím vícekrát se násobí předchozí gradienty. To způsobuje problém mizejícího gradientu (násobí se mezi sebou čísla menší než 1) či explodujícího gradientu (násobí se mezi sebou čísla vyšší než 1). Při dlouhém trénování může dojít k přetrénování sítě. Síť přesně nastaví tak, aby

rozpoznala přímo konkrétní data a ztrácí informace o obecných vlastnostech dat. Pro předcházení tohoto problému se používají validační data, pomocí kterých se kontroluje, zda na ně síť postupně nereaguje hůře, zatímco se stále zlepšuje její odezva na trénovací data.



Obrázek 1.3: Ukázka přetrénování. Vlevo ztrátová funkce. Vpravo přesnost klasifikace [6].

Pokud se síť přetrénuje použije se stav sítě, kdy reagovala na validační data nejlépe. Dalším problémem robustnosti je málo obecná reprezentace dat. Například u rozpoznávání obrázků by mohla síť selhat na jinak velikých a jinak natočených obrázkách. Řešením problému je augmentace trénovacích dat. Data se upraví tak, aby reprezentovala i rozdílně získaná data. Mezi možné augmentace patří například změna velikosti, rotace, převrácení, odtexturování, oříznutí, změna měřítka. Také se pro robustnost zavádí dropout, který při trénování způsobí vynechání části neuronů, aby se zabránilo zbytečným spojením při kterých je jeden neuron zcela závislý na jiném.

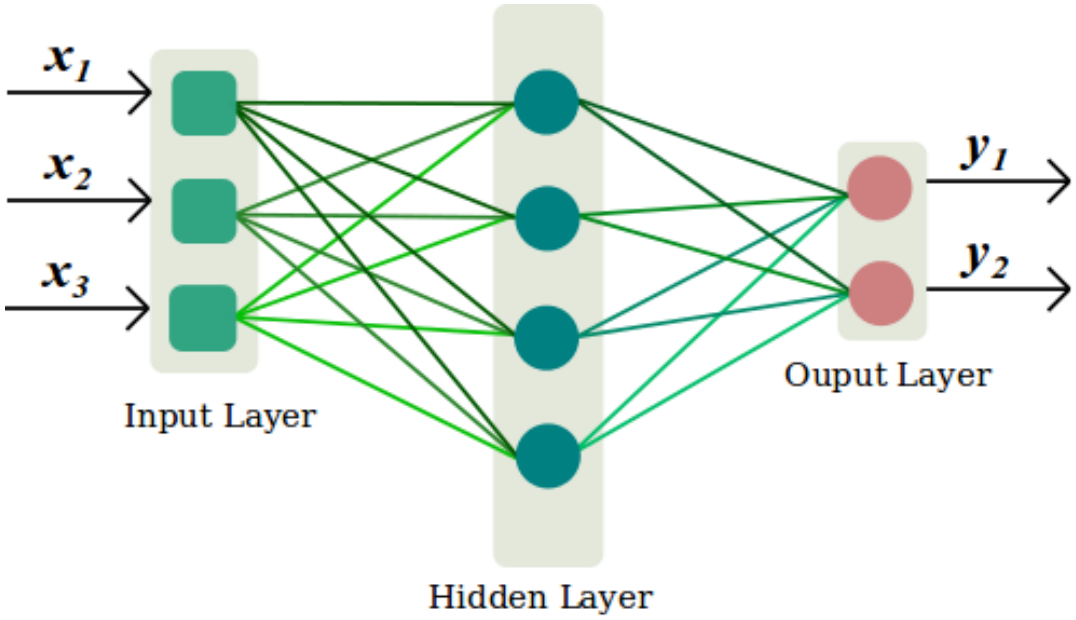
Při spuštění trénování mohou být počáteční parametry sítě nastaveny náhodně, nebo podle kontrolního bodu (checkpoint), do kterého se ukládají parametry sítě v průběhu trénování.

1.2 Typy propojení sítí

Existuje více způsobů jak na sebe napojit dvě vrstvy. Mezi základní patří plně propojené vrstvy a konvoluční vrstvy. V celkové síti se mohou tyto propojení prolínat, například základ sítě může být tvořen konvolučními vrstvami a poslední výstupní vrstvy plně propojenými vrstvami.

1.2.1 Plně propojené vrstvy

Základní způsob propojení je propojit všechny výstupy předchozí vrstvy se všemi vstupy následující vrstvy. Každé propojení má svou vlastní váhu a každý vektor v následující vrstvě má svůj vlastní práh. Tento způsob se nazývá plně propojené vrstvy (fully connected layers).



Obrázek 1.4: Příklad plně propojené vrstvy [30].

1.2.2 Konvolučí vrstvy

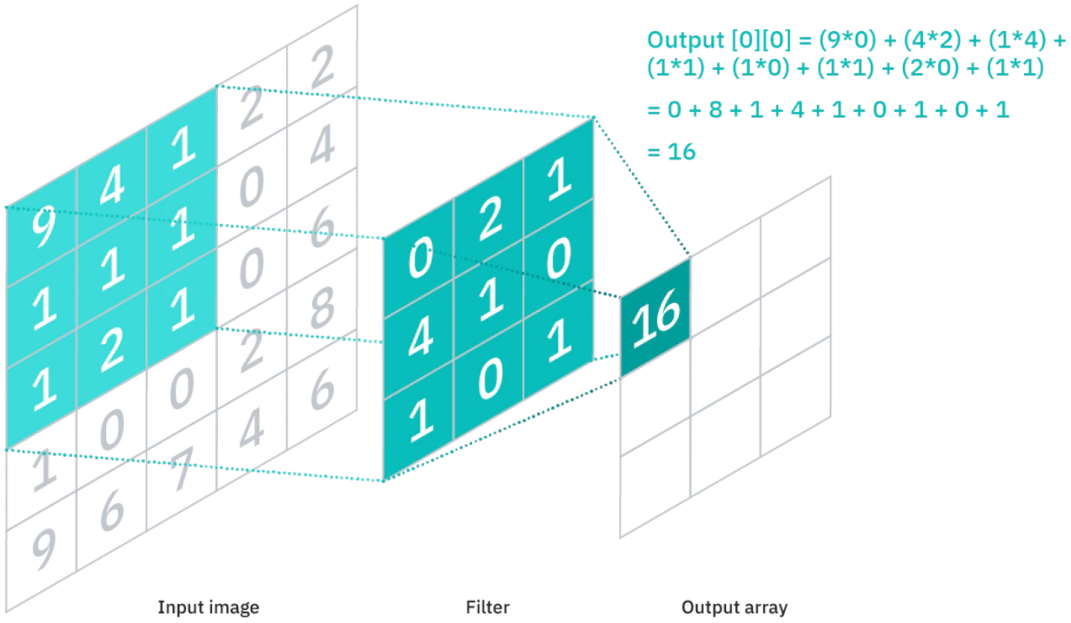
Pro rozpoznávání obrázků nebo videí se často používají konvoluční vrstvy (convolutional layers). Obrázek je reprezentován maticí jejíž dimenze reprezentují počet barev, šířku obrázku, výšku obrázku. Konvoluční vrstvy využívají matici jádra, která

konvolučně prochází obrázek podle rovnice 1.9

$$y[m,n] = x[m,n] \cdot h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j], \tag{1.9}$$

kde x je vstup a h je jádro.

Do jednoho neuronu další vrstvy vždy vstupují výstupy obsažené uvnitř jádra, tedy se získá malý vzorek z obrazu upravený filtrem převrácených hodnot jádra.



Obrázek 1.5: Příklad konvoluční vrstvy [12].

Hodnoty filtru jádra jsou dány parametry sítě. V další vrstvě vzniká nová matice, jejíž velikost je oproti matici předchozí vrstvy změněna parametry sítě: velikost jádra (kernel size), velikost kroků (stride) a velikost obalu (padding). Velikost jádra určuje z kolika hodnot okolo vzorkovaného bodu se získávají hodnoty. Velikost kroků určují jak často se vzorkuje. Velikost obalu určuje jak velký může mít v daném vzorku jádro přesah mimo vstup. Tyto parametry ovlivňují, jak veliká bude matice na výstupu. Při omezeném přesahu jádra se nenavzorkují některé krajní body ze vstupní matice a zmenší se tím velikost výstupu, jehož velikost je ještě podělena velikostí kroku. Tímto způsobem se často vzorkuje stejný vstup vícekrát s rozdílnými filtry, čímž se přidá nová dimenze do matice následující vrstvy.

2 Technologie

2.1 Dataset LSA64

Dataset LSA64 [25] byl v této práci použit pro všechny experimenty. Obsahuje 3200 videí s 64 rozdílnými argentinskými znaky. Znaky jsou na videích ukazovány 10 lidmi, každý 5 videí pro každý znak. Jsou poskytovány 3 verze datasetu. Prvotní dataset, který obsahuje videa tak jak byla natočena. Oříznutý dataset, který obsahuje videa začínající a končící pohybem rukou. Předzpracovaný dataset, u kterého je provedena segmentace rukou a extrahována pozice rukou a hlavy. V této práci byl využit prvotní dataset. Dataset nemá žádné oficiální rozdělení na trénovací, validační a testovací data. Videa byla natáčena RGB kamerou.

Obrázek 2.1: Ukázka smímků z datasetu LSA64.

2.2 Natáčení

Dataset byl natáčen na dvou místech. Prvních 23 jednoručních znaků bylo nahráno venku s přirozeným světlem. Zbylých 41 znaků, ze kterých je 22 obouručních a 19 jednoručních, bylo nahráno v místnosti s umělým světlem. Na Obrázku 2.1 jsou vlevo 2 snímky natáčené venku a zbylé 4 snímky natáčené v místnosti. Figurant 10 se neúčastnil druhého natáčení a byl nahrazen jiným figurantem. Na všech videích mají figuranti černé oblečení a pozadí je bílé. Znaky byly nahrávány vestoje i vsedě. Figuranti měli na rukou barevné rukavice aby se usnadnilo sledování a segmentace videa. Vše bylo natočeno kamerou Sony HDR-CX240. Stojan kamery byl postaven

2 metry od zdi s nastavenou výškou na 1,5 metrů.

2.3 PyTorch

PyTorch [24] je framework pro hluboké učení. Byl vydán v roce 2016 firmou Facebook. Spouští se v jazyce python, ale běh probíhá v jazyce c++. Byl tvořen tak, aby byl intuitivní a snadno použitelný Podporuje běh síti na GPU a automatickou differenciaci. Je využíván pro nahrazení knihovny numpy pro využití rychlejších GPU výpočů a pro výzkum v hlubokém učení, kde nabízí vysokou flexibilitu a rychlost. PyTorch poskytuje řadu postupů s tenzory pro zrychlení matematických výpočtů. Využívá unikátní způsob stavění neuronových sítí pomocí páskového autogradu, který si pamatuje provedené operace a dokáže díky tomu získávat gradienty parametrů ve zpětném módu. Pro tuto techniku má PyTorch jednu s nejlepších implementací. PyTorch využívá knihovny na zrychlení výpočtů, jako například intel MKL [32], NVIDIA cuDNN [5] a NVIDIA NCCL [21]. PyTorch velice efektivně využívá pamět pomocí vlastních paměťových alokátorů pro GPU, čímž umožnujě trénování větších sítí.

2.4 TSN

Temporal Segment Networks [33] je architektura neuronových sítí na klasifikaci videí. Liší se od ostatních především tím, že místo využití průtoku z celého videa rozděluje video na více snippetů navzorkovaných z jeho částí. Z každého snippetu vyjde predikce zařazení do třídy. Dohromady se z nich odvodí výsledné zařazení do třídy. Nejprve se video rovnoměrně rozkouskuje na K segmentů. Z každého segmentu se náhodně navzorkuje snippet. Vznikne posloupnost K snippetů: $(T_1, T_2, ..., T_k)$. Každý snippet pak projde konvoluční sítí $F(T_k; \mathbf{W})$ kde T_k je snippet a $\mathbf{W}$ jsou parametry sítě. Následně je všech K výstupů spojeno aggregační funkcí G a výsledek je převeden na pravděpodobnosti pomocí funkce softmax H. Celková funkce vypadá takto:

$$\mathrm{TSN}(T_1, T_2, ..., T_k) = H(G(F(T_1; \mathbf{W}), F(T_2; \mathbf{W}), ..., F(T_K; \mathbf{W}))) \tag{2.1}$$

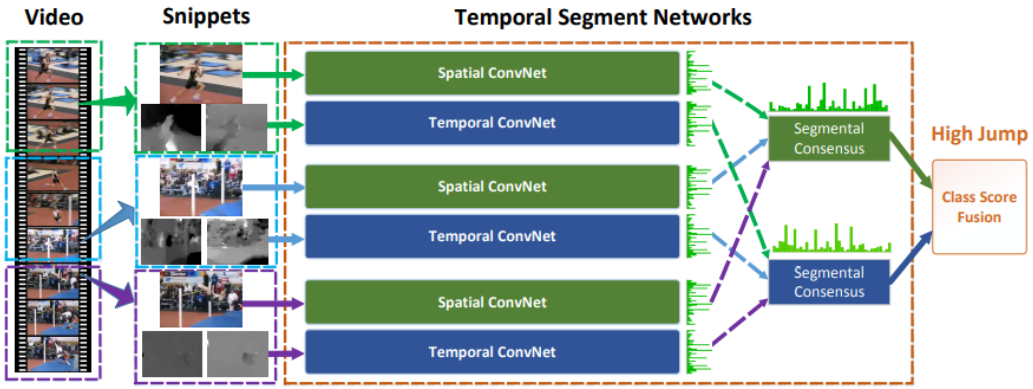
Jako aggregační funkce se experimentálně hodnotily funkce, jako například průměrování, maximum, vážené průměrování. Používá se z nich průměrování. Z vý-

sledku se počítá loss funkce takto:

$$L(y, \mathbf{G}) = -\sum_{\mathbf{i=1}}^{\mathbf{C}} \mathbf{y_i}(\mathbf{G_i} - \log \sum_{\mathbf{j=1}}^{\mathbf{C}} \exp \mathbf{G_j}) \tag{2.2}$$

kde C je počet tříd, y_i je požadovaný výstup vzhledem ke třídě i. Parametry $\mathbf{W}$ jsou optimalizovány pomocí metody back-propagation. Gradienty $\mathbf{W}$ jsou derivovány:

$$\frac{\partial L(y, \mathbf{G})}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{G}} \sum_{k=1}^{K} \frac{\partial G}{\partial F(T_k)} \frac{F(T_k}{\partial \mathbf{W}} \tag{2.3}$$



Obrázek 2.2: Schéma architektury TSN. Video se rozdělí na snippety, ty projdou konvolučními sítěmi, hodnoty z výstupů se shodnou na hodnotě která je výsledkem [33].

TSN dosáhl v roce 2016 nejlepších výsledků na datasetech HMDB51 [16] (69,4%) a UCF101 [29] (94,2%). V dnešní době si TSN udržuje 17. místo v přesnosti 5 hypotéz na datasetu Moments in time [20].

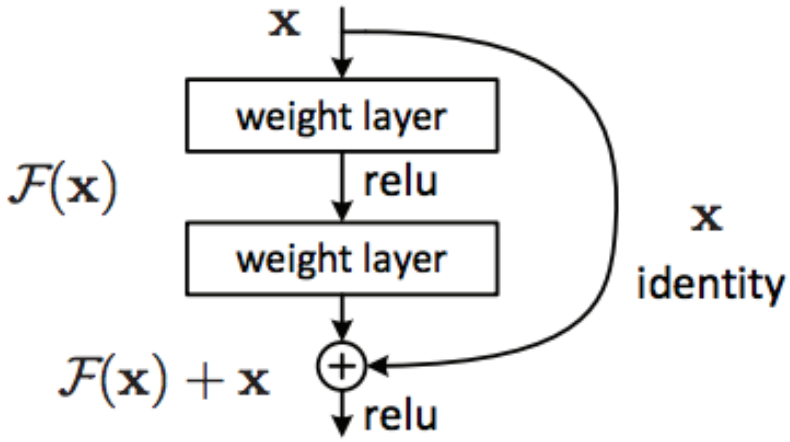
2.5 MMAction2

MMAction2 [7] je otevřený software poskytující sadu nástrojů pro zpracování videí. Je postaven na knihovně mmcv [8]. Porozumění je rozděleno narozdílné komponenty, které se dají propojit pro širší porozumění. Software je postavený na frameworku PyTorch. Současně podporuje 27 různých algoritmů, které řeší mnoho rozdílných úloh z nichž hlavní jsou rozpoznávání, časoprostorová detekce, lokalizace a

detekce akce pomocí skeletizace. Také podporuje 20 různých datasetů určených pro všechny čtyři hlavní úlohy. Nástroje byly důkladně testovány a zdokumentovány. MMAction2 je stále rozvíjející se software. Pro TSN používá MMAction2 jako základ nejčastěji síť ResNet50 [10], ale také například ResNeXt101 [34], Densenet-161 [11], nebo Swin Transformer Base [18]. Modely bývají předtrénované na datasetech ImageNet [9], Kinetics400 [13], nebo IG-1B [19]. TSN model využitý v této práci má jako základ ResNet50 předtrénovaný na datasetu ImageNet.

2.6 ResNet

Residual Network [10] je architektura konvolučních sítí. Síť je navržená tak, aby pro určitý počet vrstev namísto odhadnutí optimální funkce $H(x)$, odhaduje residuální funkci $F(x)$, pro kterou platí $F(x) + x = H(x)$. V samotné síti se tento koncept využívá pomocí přeskakování spojení, umožňující výstupu z jedné vrstvy vstupovat nejen do své následující vrstvy, ale i do pozdějších vrstev. Celková síť je pak tvořena z bloků, které představují jednotlivé residuální funkce.

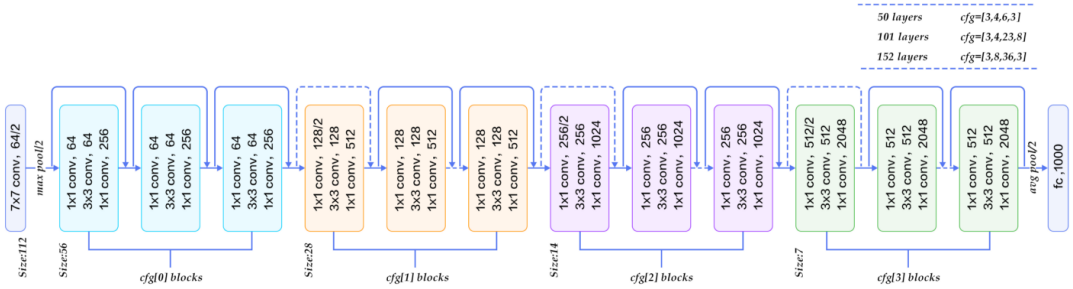


Obrázek 2.3: Blok reziduální sítě

Dimenze $F(x)$ a x musí být stejné. Pokud nejsou, aplikuje se lineární projekce W_s na přesakující spojení, jak je ukázáno v rovnici 2.4

$$y = F(x) + W_s x. \tag{2.4}$$

Díky přeskakování spojení se značně zmírňuje problém mizejícího a explodujícího gradientu a umožňuje tím zavádět hlubší sítě. ResNet je navrhnutý pro více hloubek, může mít hloubku až po 152 vrstev. Sítě hloubky 50 a více mají bloky o třech vrstvách, jejich obecné schéma je zobrazené na Obrázku 2.4. V této práci byl využit ResNet o hloubce 50.



Obrázek 2.4: Schéma architektury ResNet s velkými hloubkami

3 Příprava experimentů

3.1 Rozdělení dat

Data jsem rozdělil na trénovací validační a testovací množiny následovně. Z 10 figurantů se vybral jeden, jehož všechny nahrávky byly určeny pouze pro testování. Ze zbytku videí se vždy od každého figuranta vybralo jedno z 5 videí, které bylo určeno pro validaci a 4 videa na trénování.

train	train	train	train	train	train	train	train	train	test
train	train	train	train	train	train	train	train	train	test
train	train	train	train	train	train	train	train	train	test
train	train	train	train	train	train	train	train	train	test
val	val	val	val	val	val	val	val	val	test

Tabulka 3.1: Rozdělení nahrávek pro jeden znak. Každý sloupec reprezentuje 5 nahrávek od 1 figuranta

3.2 Konfigurace

V knihovně MMaction2 jsem vycházel z přednastaveného modelu TSN pro rozpoznávání videí z datasetu kinetics400. Jedná se o model typu Recoginzer2D. Jako základ využívá architekturu ResNet50 a jako hlavičku pro výstup využívá TSNHead. Vstup do hlavičky je z 2048 neuronů. Výstup jsem nastavil na 64, což je počet tříd do kterých se klasifikuje. Dále je v hlavičce nastavení, jakou agregační funkcí se spojí výstupy ze základu sítě, pravděpodobnost na vynechání neuronu (dropout), hodnota počáteční směrodatné odchylky a nastavení hyperparametrů TSN.

Dále se v konfiguracích nastavují cesty k datům a k anotačním souborům, normalizace obrázků, která je předem vypočítána, typ datasetu optimalizátor a jeho parametry, změna konstanty učení závisle na epoše, počet epoch, intervaly ukládání kontrolních bodů, intervaly zapisování do logů, intervaly a způsob vyhodnocování validace, startovní kontrolní bod, pracovní postup, počet videí na GPU, počet pracovníků na předpřípravu dat pro GPU, počet snímků v klipu, interval sousedních

snímků, počet vzorkovaných klipů . Také se zde dají nastavit volitelné parametry argumentů.

Nastavení pro používaný model jsou popsána ostatních částech práce. Část nastavení pro používaný model jsou zobrazena v Tabulce 3.2

Parametr	**Hodnota**
agregační funkce	průměr
dropout	0,4
typ datasetu	VideoDataset
změna konstany učení	nevyužita
vyhodnocování validace	přesnost *k* hypotéz
startovní kontrolní bod	100e_kinetics400_rgb
pracovní postup	trénování, validace
počet videí na GPU	3 (test: 1)
počet pracovníků	1
počet snímků	1
interval snímků	1
počet klipů	8

Tabulka 3.2: Nastavené hodnoty v konfiguraci

3.3 Dataset

MMAction2 dokáže zpracovat 2 typy formátů dat pro rozpoznávání. První je RawframeDataset, kde je pro každé video složka, ve které jsou uloženy všechny jeho snímky. Další je VideoDataset, kde jsou videa ukládána ve formátu mp4. Takto jsou data ukládána do složek rozdělujících je na trénovací, validační a testovací data. U obou těchto datasetů je anotace zapsána do textového souboru. Pro rawframe dataset jsou na každé řádce mezerou odděleny relativní cesta k složce s videem, počet snímků v souboru a označení třídy videa. Pro video dataset jsou obdobně na každé řádce odděleny cesta k mp4 souboru a označení třídy videa. Z těchto formátů jsem si zvolil VideoDataset a třídy jsem označil čísly 0-63 podle

znaků které na nich byly ukazovány. Pro lepší zacházení z daty jsem vytvořil skript data_distribution_setup_lsa64, který přesouvá videa do složek lsa64_train, lsa64_val a lsa64_test a zároveň přepisuje anotační soubory. Skript byl později využit pro křížovou validaci.

3.4 Spuštění trénování

Trénování v MMAction2 vyžaduje v argumentech cestu na konfigurační soubor. Zbytek argumentů je volitelný. Většinu z nich lze nastavit i v konfiguraci. Pokud jsou nastaveny i v konfiguraci i v argumentech, jsou použity ty v argumentech. Patří mezi ně nastavení validace, nastavení testování, pracovní adresář, výchozí kontrolní bod, počet využitých GPU a jejich ID, seed na náhodné generování a nastavení determinističnosti cuDDN [5] algoritmů.

3.5 Zpracování dat

Pro logování se v configu využívá mmcv třída TextLoggerHook.Při trénování se po každých 20 iteracích zapisovaly informace o průběhu trénování do logů a do json souboru. Mezi logované informace patří: mód (trénovací, validační nebo testovací), epocha, iterace, konstanta učení, stav paměti, přesnost první hypotézy, přesnost 5 hypotéz, gradientová norma, čas iterace. Po každé epoše probíhala validace, ze které se také zapisovali informace.

Pro zpracování dat z trénování jsem vytvořil skript train_resoults, který z json záznamu trénování zobrazuje graf ztrátové funkce a přesnosti pro trénovací a validační data. Vyhodnocuje také, při které epoše je nejnižší hodnota ztrátové funkce pro validační data a vyhodnotí pro tuto epochu přesnosti pro trénovací a validační data. Dále ukládá hodnoty ztrátové funkce a přesnosti pro validační data jako vektory.

3.6 Hardware

Experimenty probíhali na PC jehož vlastnosti jsou ukázány v Tabulce 3.3

Název zařízení	LAPTOP-1NMNCGNE
Procesor	Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
Nainstalovaná paměť	RAM 8,00 GB (použitelné: 7,85 GB)
Typ systému	64bitový operační systém, procesor pro platformu x64
Graficá karta	NVIDIA GeForce GTX 1050
Paměť GPU	4,0 GB
Verze ovladače	471.41

Tabulka 3.3: Vlastnosti použitého PC.

3.7 Software

Na PC s operačním systémem Windows11 byla pro běh MMAction2 využita NVIDIA CUDA 10.1 [22] a Python prostředí s kihovnami ukázanými v Tabulce 3.4

python	3.7.9
torch	1.8.0+cu101
torchvision	0.9.0+cu101
opencv	3.4.2
mmcv-full	1.3.9
mmaction2	0.20.0
matplotlib	3.4.3

Tabulka 3.4: Vezce Python Knihoven.

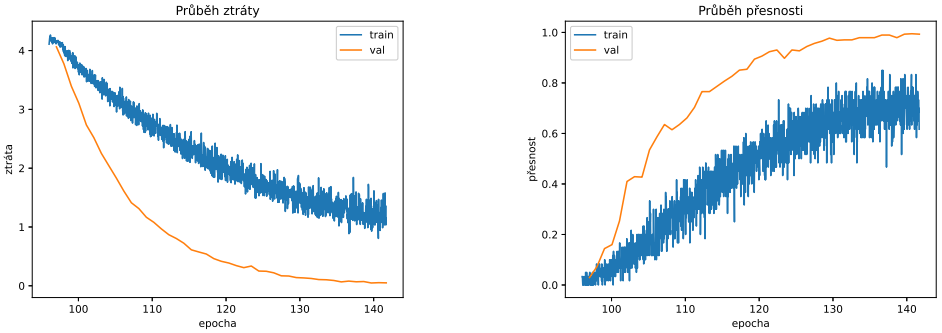
4 Experimenty

V experimentech jsem Začínal trénovat z kontrolního bodu TSN natrénovaného pro dataset kinetics400 [13]. Pro načtení jsem využil argument resume-from, který nahraje parametry sítě a číslo epochy, všechny experimenty proto začínají 96. epochou. Alternativně lze použít argument load-from, který načte pouze parametry sítě, a trénování začíná od 1. epochy.

Experimenty začínají 96. epochou a končí epochou 140.

4.1 Základní experiment

V prvním experimentu jsem zjišťoval chování sítě při nezměněných parametrech. Trénování probíhalo pomocí optimalizátoru SGD s konstantou učení 0,0001; momentem 0,9 a úbytkem vah 0,0001.

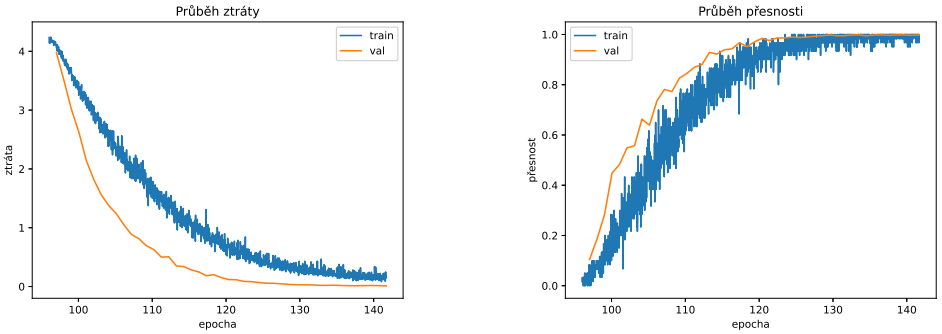


Obrázek 4.1: Základní nastavení pro SGD.

Z grafů na Obrázku 4.1 je vidět, že síť reaguje na validační množinu lépe než na trénovací, což je neobvyklé. Možným důvodem by mohla být augmentace aplikovaná na trénovací množinu. Dalším důvodem může být menší množství dat ve validační množině, při které se může chyba projevit méně.

4.2 Augmentace

V prvním Experimentu byla nastavena augmentace změny měřítka na 1; 0,875; 0,75; 0,66 původního měřítka a pravděpodobností na zrcadlové otočení obrazu 0,5. S tímto nastavením se značně zpomalilo trénování. Cílem druhého experimentu bylo zjistit chování sítě při zrušení augmentace. Všechny ostatní parametry zůstaly stejné.

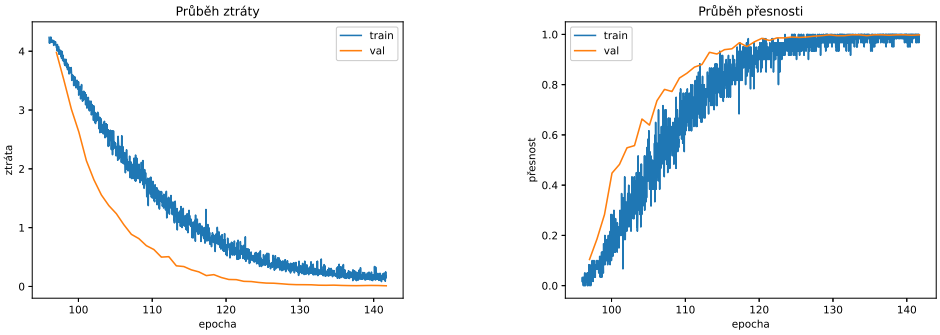


Obrázek 4.2: Ztrátová funkce a přesnost při zrušené augmentaci

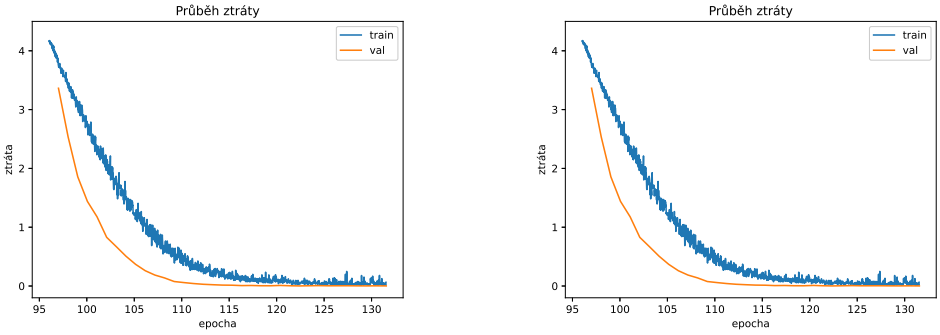
Jak je vidět na Obrázku 4.2, síť stále dosahuje lepších výsledků pro validační množinu, nicméně jsou si výsledky pro obě množiny významně blíže než při nastavené augmentaci. Při porovnání grafů na Obrázkách 4.1 a 4.2 je vidět, že se bez augmentace dostává síť rychleji k optimálnímu nastavení, dokonce i pro validační množinu. Pro účel této práce nebylo nutné rozpoznávat videa s rozdílnými měřítky a augmentace zrcadlového otočení je pro znakovou řeč nežádoucí. Pro zbytek experimentů se tedy již tyto augmentace přestali využívat.

4.3 Optimalizátory

V kapitole 1 jsme si ukázali optimalizátory SGD a Adam. Nelze u nich obecně říct, který je lepší, zkouším tedy ve třetím experimentu, jak se budou optimalizátory chovat pro jejich základní nastavení. Z grafů na Obrázku 4.3 je vidět chování sítě využívající optimalizátor SGD s konstantou učení 0,0001; momentem 0,9 a úbytkem vah 0,0001. V grafech na Obrázku 4.4 je zobrazeno chování sítě využívající optimalizátor Adam s konstantou učení 0,00001; průběžnou střední hodnotou 0,9; průběžnou variancí 0,999 a ϵ pro zlepšení stability $1 \cdot 10^{-8}$.



Obrázek 4.3: SGD



Obrázek 4.4: Adam

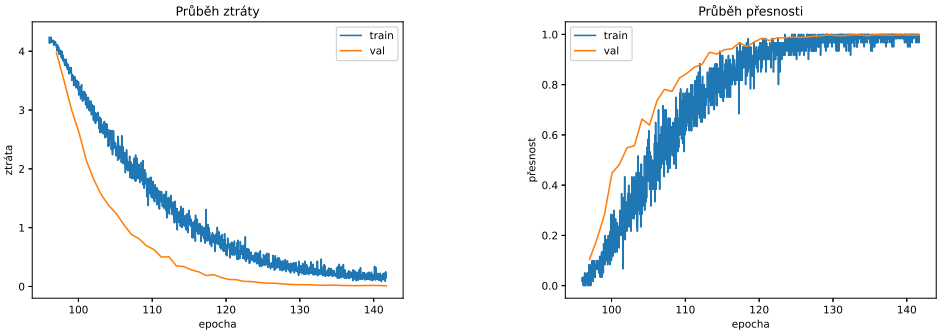
Z experimentů není jasné, který optimalizátor je vhodnější na využívaná data. Využijí se tedy oba.

4.4 Konstanta učení

Cílem čtvrtého experimentu bylo nalézt vhodnou konstantu učení pro vybrané optimalizátory. Hodnoty byly měněny vždy o jeden řád.

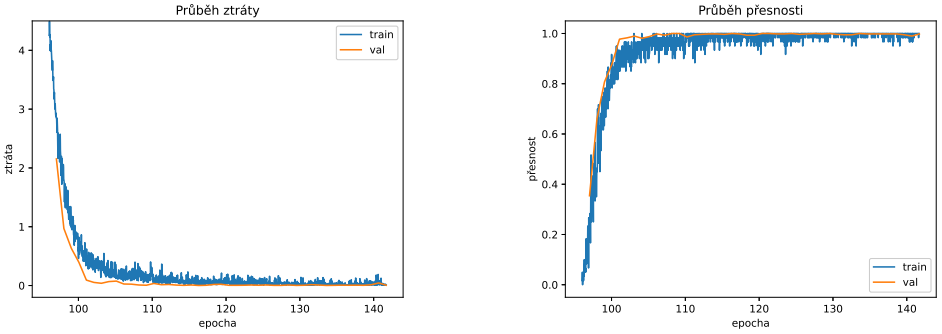
4.5 SGD

Začal jsem na hodnotě 0,0001 a po každém experimentu jsem ji zvýšil o řád výš.



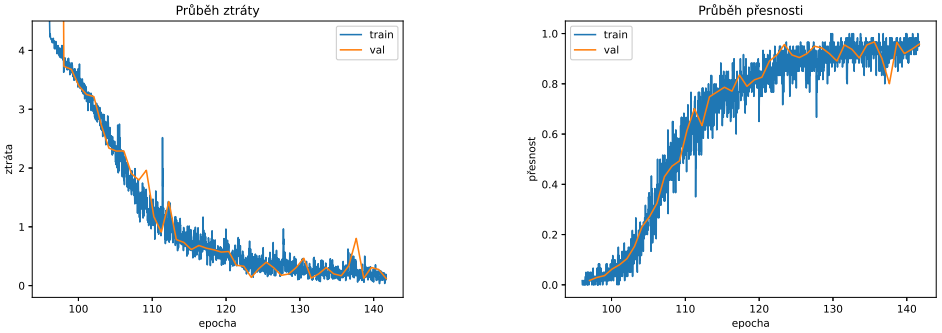
Obrázek 4.5: sgd, konstanta učení = 0,0001

V grafech na Obrázku 4.5 je vidět, že ztrátová funkce při konstantě učení 0,0001 po celou dobu klesá a pomalu konverguje k nule. Přesnost klasifikace se blíží k 1.



Obrázek 4.6: sgd, konstanta učení = 0,001

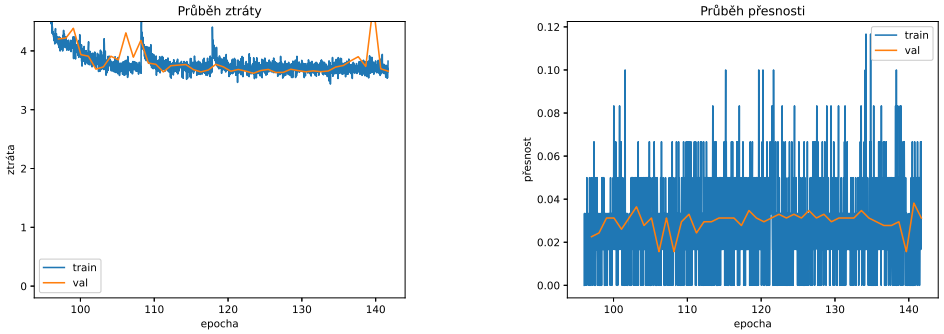
V grafu na Obrázku 4.6 je vidět, že se ztrátová funkce při konstantě učení 0,001 dostane stabilně k 0 již po 20 epochách. Obdobně se přesnost dostává k 1.



Obrázek 4.7: sgd, konstanta učení = 0,01

Jak je vidět na Obrázku 4.7, pro konstantu učení 0,01 se síť dostává pomaleji a

méně stabilně k optimálnímu nastavení.

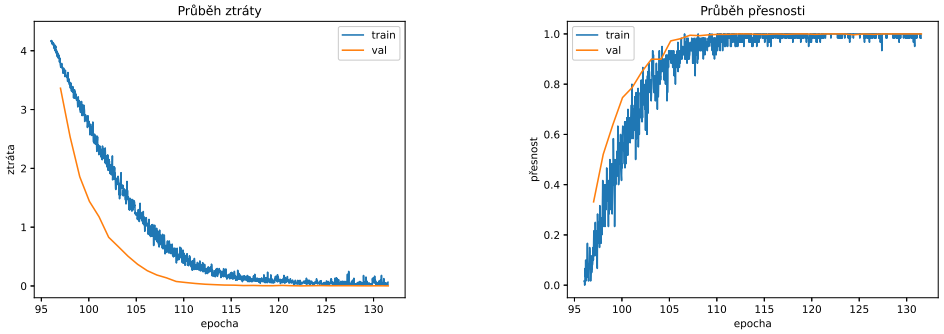


Obrázek 4.8: sgd, konstanta učení = 0,1

V grafech na Obrázku 4.8 je vidět chování sítě pro konstantu učení 0,1. Ztrátová funkce se nedostane pod hodnotu 3. Přesnost sítě zůstává kolem 0,03 (což je přibližně dvakrát úspěšnější, než vybrání náhodně z 64 tříd) Výsledek je tedy nepoužitelný.

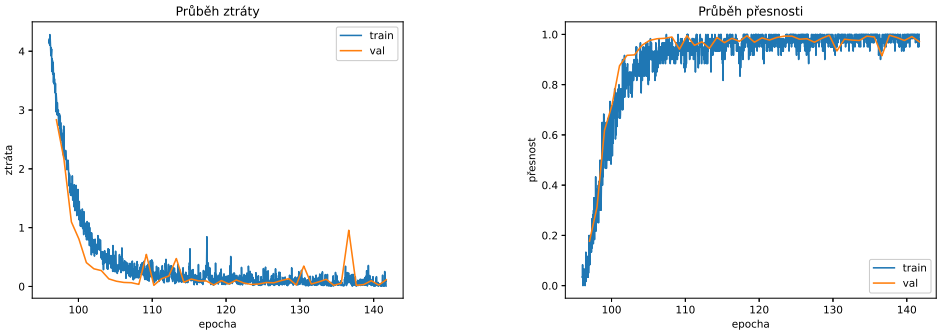
4.6 Adam

Pro optimalizátor Adam jsem začal na konstantě učení $1 \cdot 10^{-5}$ a poté jsem ji zvyšoval stejně jako u SGD.



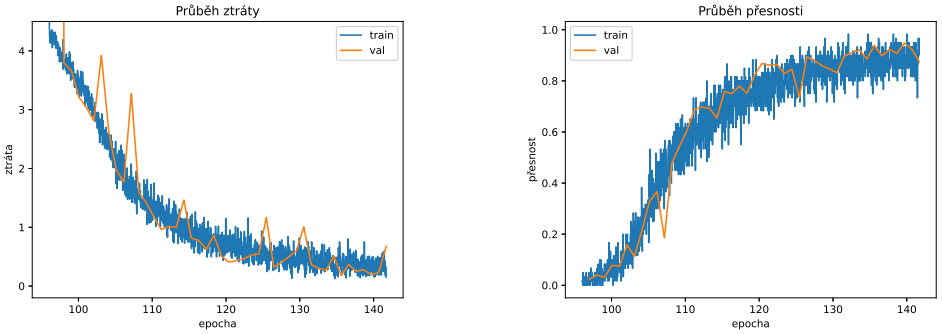
Obrázek 4.9: Adam, konstanta učení = $1 \cdot 10^{-5}$

V grafech na Obrázku 4.9 je vidět, že pro konstantu učení $1 \cdot 10^{-5}$ se ztrátová funkce dostane stabilně k 0 za 30 epoch. Přesnost se také dostane k 1.



Obrázek 4.10: Adam, konstanta učení = $1 \cdot 10^{-4}$

V grafech na Obrázku 4.10 je vidět, že pro zvýšení konstanty učení na $1 \cdot 10^{-4}$ se při trénování dostane ztrátová funkce k nule dříve, než při $1 \cdot 10^{-4}$, ale ztrácí se stabilita.



Obrázek 4.11: Adam, konstanta učení = $1 \cdot 10^{-3}$

V grafech na Obrázku 4.11 je vidět chování sítě pro konstantu učení $1 \cdot 10^{-3}$. Parametry sítě se s takto vysokou konstantou učení nedokážou dostat k optimálnímu nastavení s dostatečnou přesností.

4.7 Vyhodnocení konstanty učení

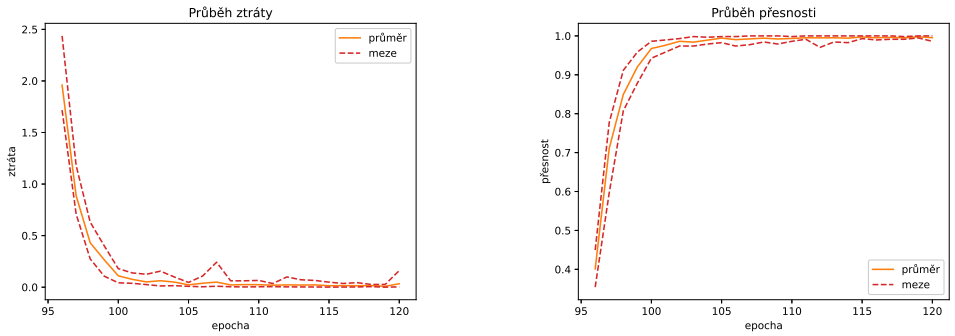
Nejlepší z vyzkoušených konstant učení se zdá být pro optimalizátor SGD 0,001 a pro optimalizátor Adam $1 \cdot 10^{-5}$. Díky dobrému předtrénování se síť dostane k dobrým výsledkům pro oba optimalizátory a není třeba v průběhu trénování snižovat hodnotu konstanty učení.

4.8 Křížová validace

Pro dvě nejlépe se chovající nastavení jsem provedl křížovou validaci. Do testovací množiny jsem vždy vynechával jiného figuranta. Od zbylých devíti figurantů jsem vybral jedno video náhodně na validaci a zbylé čtyři na trénování. S takto rozdělenými daty se provedlo trénování. Poté se ze všech epoch vybrala ta s nejnižší hodnotou ztrátové funkce pro validační data. Na tuto epochu bylo provedeno testování pro testovací data s figurantem vynechaným při trénování.

4.9 SGD

Jak je vidět na Obrázku 4.6, ztrátová funkce i přesnost s optimalizátorem SGD a konstantou učení 0,001 dokonverguje již před 120. epochou. Kvůli omezené výpočetní kapacitě jsem pro SGD ukončoval křížovou validaci již na 120. epoše. Na Obrázku 4.12 jsou zobrazeny dosažené meze a průměr ztrátové funkce a přesnosti pro validační množinu při křížové validaci.



Obrázek 4.12: SGD, robustnost při křížové validaci.

V grafech na Obrázku 4.12 je vidět, že při trénování dochází k případům, při kterých se dočasně hodnota ztráty začne zvyšovat a přesnost snižovat. To může být způsobeno tím, že se parametry sítě vydají špatným směrem a vzdálí se kvůli tomu optimálnímu nastavení.

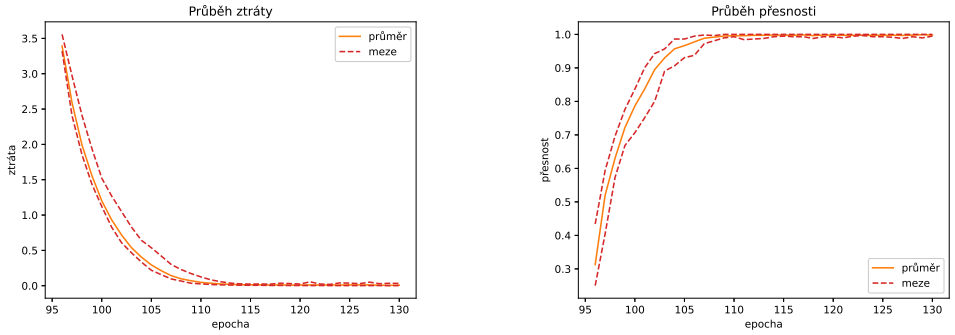
V Tabulce 4.1 jsou pro všechny vynechané osoby zobrazeny přesnosti na trénovací, validační a testovací množině pro jejich vybranou epochu.

		Trénovací data		Validační data		Testovací data	
osoba	epocha	top_1	top_5	top_1	top_5	top_1	top_5
1	120	1,0	1,0	0,998	1,0	0,9313	1,0
2	116	0,967	1,0	1,0	1,0	0,975	1,0
3	119	1,0	1,0	1,0	1,0	0,9062	1,0
4	115	0,983	1,0	1,0	1,0	0,9437	0,9938
5	119	0,983	1,0	0,998	1,0	0,9844	1,0
6	113	0,983	1,0	1,0	1,0	0,9406	0,9969
7	120	1,0	1,0	0,998	1,0	0,9875	1,0
8	109	0,983	1,0	0,995	0,998	0,9250	1,0
9	113	0,983	1,0	0,998	1,0	0,9406	0,9875
10	119	1,0	1,0	1,0	1,0	0,9313	1,0
průměr	116,3	0,988	1,0	0,999	1,0	0,947	0,999
odchylka	3,743	0,0112	0	$1{,}64 \cdot 10^{-3}$	$5{,}50 \cdot 10^{-4}$	0,027	$4{,}17 \cdot 10^{-3}$

Tabulka 4.1: Křížová validace SGD.

4.10 Adam

Na základě chování křivek v grafech na Obrázku 4.9 zobrazujícím průběh trénování s optimalizátorem Adam a konstantou učení $1 \cdot 10^{-5}$, jsem prováděl křížovou validaci do 130. epochy. Na Obrázku 4.13 jsou zobrazeny dosažené meze a průměr ztrátové funkce a přesnosti pro validační množinu při křížové validaci pro optimalizátor Adam.



Obrázek 4.13: Adam, robustnost při křížové validaci.

V grafech na Obrázku 4.13 je vidět, že při využití optimalizátoru Adam se při konvergenci paramerametry udržují u optimálního nastavení při všech 10ti trénováních.

Výsledky přesnosti pro vybranou epochu jsou zobrazeny v Tabulce 4.2.

		Trénovací data		Validační data		Testovací data	
osoba	epocha	top_1	top_5	top_1	top_5	top_1	top_5
1	125	1,0	1,0	1,0	1,0	0,938	1,0
2	119	1,0	1,0	1,0	1,0	0,953	1,0
3	129	1,0	1,0	1,0	1,0	0,906	0,997
4	121	1,0	1,0	1,0	1,0	0,928	0,991
5	121	0,983	1,0	1,0	1,0	0,956	1,0
6	121	1,0	1,0	0,998	1,0	0,941	0,997
7	130	0,983	1,0	0,998	1,0	0,959	1,0
8	129	1,0	1,0	1,0	1,0	0,944	1,0
9	127	1,0	1,0	1,0	1,0	0,925	0,997
10	130	0,983	1,0	1,0	1,0	0,916	1,0
průměr	125,2	0,995	1,0	1,0	1,0	0,937	0,998
odchylka	4,34	$8{,}05 \cdot 10^{-3}$	0	$7{,}34 \cdot 10^{-4}$	0	0,0176	$3{,}02 \cdot 10^{-3}$

Tabulka 4.2: Křížová validace Adam.

4.11 Vyhodnocení

Při srovnání optimalizátorů SGD a Adam pro dataset LSA64 má SGD lepší výsledky a rychleji se s ním trénuje. Pro testovací množinu je s optimalizátorem SGD úspěšnost $0,947 \pm 0,027$, zatímco s optimalizátorem Adam je úspěšnost $0,937 \pm 0,018$. Výhodou využití optimalizéru Adam je větší trénovací robustnost. Pokud je tedy cílem dosáhnout co nejlepších výsledků a nevadí nejistota, je vhodnější optimalizátor SGD. Pokud je důležitější mít jistotu, že se síť natrénuje na nějakou úroveň a nevadí trochu menší přesnost, je vhodnější optimalizátor Adam.

Závěr

Na dataset znakové řeči LSA64 [25] jsem aplikoval umělou neuronovou síť TSN se základem ResNet50 za účelem rozpoznávat jednotlivé znaky. Využil jsem na práci otevřený software MMAction2, který je součástí projektu OpenMMLab. Síť byla předtrénována na datasetu kinetics400. V experimentech jsem vyzkoušel optimalizátory SGD a Adam, a u obou jesm nalezl vhodnou konstantu učení pro dotrénování sítě. Pro SGD 0,001 a pro Adam $1 \cdot 10^{-5}$. Pro tyto nastavení jsem provedl křížovou validaci, kde jsem do testovací množiny vždy vynechával jednoho z figurantů. Výsledky ukazují, že se síť s SGD pro dataset LSA64 natrénuje rychleji než síť s Adamem, ale Adam zajišťuje větší jistotu správného natrénování. Úspěšost s SGD byla $0,947 \pm 0,027$. Úspěšnost s Adamem byla $0,937 \pm 0,018$. Na dataset LSA64 se tedy síť dokáže naučit rozpoznávat poměrně dobře a snadno. Dataset však obsahuje pouze 64 znaků. V budoucnu bych se mohl zabývat rozsáhlejšími datasety, například datasety WLASL [17] nebo AUTSL [28]. Během práce jsem získal mnoho teoretických znalostí i praktických zkušeností ohledně umělých neuronových sítí. Implementace experimentů jsou uloženy na stránce: https://github.com/honzikjak/BP-rozpoznavani-znakove-reci-LSA64

Literatura

[1] Bera, S. – Shrivastava, V. K. Analysis of various optimizers on deep convolutional neural network model in the application of hyperspectral remote sensing image classification. *International Journal of Remote Sensing*. 2020, 41, 7, s. 2664–2683.

[2] Bertasius, G. – Wang, H. – Torresani, L. Is space-time attention all you need for video understanding. *arXiv preprint arXiv:2102.05095*. 2021, 2, 3, s. 4.

[3] Carreira, J. – Zisserman, A. Quo vadis, action recognition? a new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, s. 6299–6308, 2017.

[4] Chen, K. et al. MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*. 2019.

[5] Chetlur, S. et al. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*. 2014.

[6] Cochard, D. Prevent overfitting using dropout. `https://medium.com/axinc-ai/prevent-overfitting-using-dropout-53041b5a3797`, 2020. Accessed: 2018-05-14.

[7] Contributors, M. OpenMMLab's Next Generation Video Understanding Toolbox and Benchmark. `https://github.com/open-mmlab/mmaction2`, 2020.

[8] Contributors, M. MMCV: OpenMMLab Computer Vision Foundation. `https://github.com/open-mmlab/mmcv`, 2018.

[9] Deng, J. et al. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, s. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[10] He, K. et al. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, s. 770–778, 2016.

[11] Huang, G. et al. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, s. 4700–4708, 2017.

[12] IBM Cloud Education. What Are Convolutional Neural Networks? https://www.ibm.com/cloud/learn/convolutional-neural-networks, 2020. Accessed: 2018-05-14.

[13] Kay, W. et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*. 2017.

[14] Kelley, H. J. Gradient theory of optimal flight paths. *Ars Journal*. 1960, 30, 10, s. 947–954.

[15] Kingma, D. P. – Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

[16] Kuehne, H. et al. HMDB: a large video database for human motion recognition. In *2011 International conference on computer vision*, s. 2556–2563. IEEE, 2011.

[17] Li, D. et al. Word-level Deep Sign Language Recognition from Video: A New Large-scale Dataset and Methods Comparison. In *The IEEE Winter Conference on Applications of Computer Vision*, s. 1459–1469, 2020.

[18] Liu, Z. et al. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, s. 10012–10022, 2021.

[19] Mahajan, D. et al. Exploring the limits of weakly supervised pretraining. In *Proceedings of the European conference on computer vision (ECCV)*, s. 181–196, 2018.

[20] Monfort, M. et al. Moments in time dataset: one million videos for event understanding. *IEEE transactions on pattern analysis and machine intelligence*. 2019, 42, 2, s. 502–508.

[21] Nvidia. Nvidia/NCCL: Optimized Primitives for collective multi-gpu communication. https://github.com/NVIDIA/nccl, 2020. Accessed: 2018-05-21.

[22] NVIDIA – Vingelmann, P. – Fitzek, F. H. CUDA, release: 10.2.89, 2020. Dostupné z: https://developer.nvidia.com/cuda-toolkit.

[23] Nwankpa, C. et al. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*. 2018.

[24] Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H. et al. (Ed.) *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019. s. 8024–8035. Dostupné z: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[25] Ronchetti, F. et al. LSA64: an Argentinian sign language dataset. In *XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016).*, 2016.

[26] Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, 65, 6, s. 386.

[27] Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. 2016.

[28] Sincan, O. M. – Keles, H. Y. Autsl: A large scale multi-modal turkish sign language dataset and baseline methods. *IEEE Access*. 2020, 8, s. 181340–181355.

[29] Soomro, K. – Zamir, A. R. – Shah, M. UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*. 2012.

[30] Unknown. Fully-Connected Neural Network. https://www.gabormelli.com/RKB/Fully-Connected_Neural_Network, 2022. Accessed: 2018-05-14.

[31] Unknown. Papers with code - weight decay explained. https://paperswithcode.com/method/weight-decay, 2020. Accessed: 2018-05-19.

[32] Wang, E. et al. Intel math kernel library. In *High-Performance Computing on the Intel® Xeon Phi™*. Springer, 2014. s. 167–188.

[33] Wang, L. et al. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, s. 20–36. Springer, 2016.

[34] Xie, S. et al. Aggregated Residual Transformations for Deep Neural Networks. *CoRR*. 2016, abs/1611.05431. Dostupné z: http://arxiv.org/abs/1611.05431.