Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Detekce specifických objektů v digitálním snímku pro potřeby určení druhu obsahu scény

Plzeň 2017 Tereza Štanglová

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2017

Tereza Štanglová

Abstract

The main objective of this master thesis is to create a convolutional neural network for detection of specific objects in digital images. The dataset is divided in two classes (target and non–target) and each image has to fit into one class. Target images (class *1*) should contain pornography, specifically naked women's breasts, non–target images (class *0*) should not. In the first part, basic features of convolutional neural networks (CNN) are presented. That includes structure of nets, description of layers and learning algorithm. The second part examines various architectures of CNNs. These architectures are implemented using CNTK framework. The most promising results were achieved with architecture with three and five convolutional layers and approximately eight thousand training samples. Also a web page was created for user testing.

Abstrakt

Hlavním cílem této práce je vytvoření konvoluční neuronové sítě, která bude spolehlivě detekovat specifické objekty v digitálních snímcích. Data, na kterých byla síť trénována, jsou rozdělena do dvou skupin. Cílové snímky (třída *1*) jsou snímky, které obsahují pornografii, konkrétně odhalená ženská prsa. Necílové snímky (třída *0*) pornografii neobsahují. V první části práce je vysvětlena základní problematika konvolučních neuronových sítí. To zahrnuje například popis jejich struktury, jednotlivých vrstev a algoritmu učení. V praktické části je popsáno několik různých architektur konvolučních sítí. Pro implementaci byl vybrán framework CNTK. Nejlepších výsledků dosahovaly sítě se třemi a pěti konvolučními vrstvami, které byly natrénovány na množině s přibližně osmi tisíci vzorky. V rámci práce byly vytvořeny i webové stránky, které slouží k uživatelskému testování.

Obsah

1 Úvod

Tato diplomová práce navazuje na stejnojmennou práci, která byla vytvořena v rámci oborového projektu. Cílem bylo prozkoumání existujících přístupů k detekci specifických objektů v digitálních snímcích. Bylo zjištěno, že pro detekci objektů ve snímcích se nejlépe hodí konvoluční neuronové sítě. Vytvořená konvoluční síť však tehdy nedosáhla požadovaných výsledků z důvodu nedostatečného porozumění problematice. Účelem této práce je prohloubení znalostí o konvolučních sítích, jejich architektuře a principu učení.

Konvoluční neuronové sítě jsou speciálním druhem vícevrstvých dopředných neuronových sítí. Byly navrženy pro rozpoznávání snímků přímo z pixelů s minimálním předzpracováním. Topologie sítí je inspirována vizuálním kortexem mozku. Rozpoznávají objekty nezávisle na jejich deformaci, změně polohy a velikosti. Obsahují vrstvy, z nichž každá má svou specifickou funkci.

Cílem bylo vytvořit konvoluční síť, která bude klasifikovat snímek do dvou tříd. Cílový (*target*) snímek třídy *1* je snímek obsahující pornografii. Ostatní, necílové (*non–target*), snímky mají klasifikační třídu *0*. Automatická detekce pornografie je v dnešní době nutností. Například na sociální sítě lidé nahrávají několik milionů obrázků denně, což není možné kontrolovat ručně. Klasifikátor pro detekci pornografie slouží především jako ochrana uživatele proti nevyžádanému obsahu.

Aby se zabránilo případnému morálnímu ohrožení čtenáře, je nutné avizovat, že realizační část obsahuje několik ukázek **pornografických** snímků z trénovací množiny dat.

2 Teoretická část

2.1 Umělé neuronové sítě

Umělé neuronové sítě jsou inspirovány biologickými strukturami neuronů u živých organismů. Základní stavební jednotkou je umělý neuron. Jedná se o matematický model biologického neuronu[1]. Umělý neuron má libovolný počet vstupů a jeden výstup. Neurony jsou vzájemně propojeny a navzájem si předávají signály. Umělé neuronové sítě jsou vhodné pro úlohy z oblasti klasifikace, aproximace a predikce.

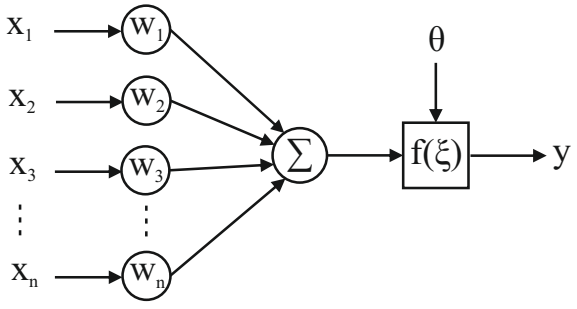
Patří do skupiny algoritmů, které využívají učení s učitelem, kdy se trénovací i testovací sada dat skládá ze dvojic, které jsou tvořeny příznakovým vektorem a požadovaným výstupem.

2.1.1 Model umělého neuronu

McCulloch–Pittsův model neuronu, viz obr. 2.1, se skládá ze tří částí – vstupní, funkční a výstupní. Je formálně popsaný jako uspořádaná trojice

$$(\boldsymbol{w}, \theta, f),$$

kde $\boldsymbol{w} = (w_1, ... \ w_n) \in \mathbb{R}^n$ je vektor synaptických vah, $\theta \in \mathbb{R}$ je práh a $f : \mathbb{R} \to \mathbb{R}$ je aktivační funkce [26] [15].



Obrázek 2.1: Model umělého neuronu.

[1] `https://cs.wikipedia.org/wiki/Neuron`

Vstupní hodnoty jsou reprezentovány vektorem $\boldsymbol{x} = (x_1, \ldots x_n) \in \mathbb{R}^n$. Každá vstupní hodnota má svoji váhu, která udává, jak je daný vstup významný. Vstupy upravené váhami se sečtou a společně s prahem reprezentují potenciál neuronu ξ podle 2.1 [25].

$$\xi = \sum_{i=1}^{n} x_i w_i + \theta. \tag{2.1}$$

Aplikováním aktivační funkce na potenciál neuronu podle (2.2) je získán výstup neuronu.

$$y = f(\xi). \tag{2.2}$$

Celý vzorec pak vypadá takto:

$$y = f\left(\sum_{i=1}^{n} x_i w_i + \theta\right). \tag{2.3}$$

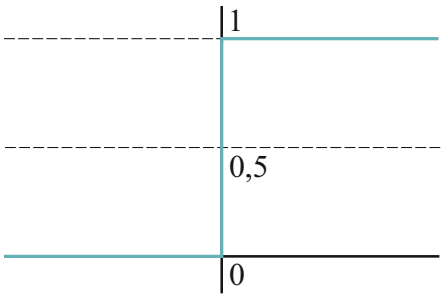
2.1.2 Aktivační funkce

Výběr funkce závisí na povaze vstupních dat. Nejjednodušší forma aktivační funkce, kterou lze použít, je binární. Neuron buď je aktivován, nebo není. Příkladem je skoková funkce, obr. 2.2, jejíž výstup lze určit podle (2.4).

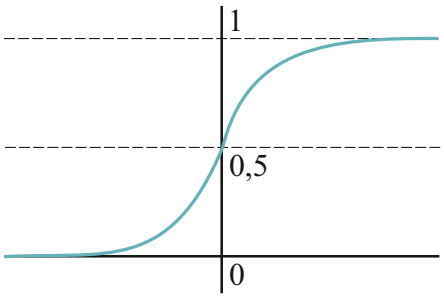
$$f(\xi) = \begin{cases} 0 \; pro \; \xi \leq 0, \\ 1 \; pro \; \xi > 0. \end{cases} \tag{2.4}$$

Nejčastěji jsou ale voleny hladké spojité monotónní funkce, které mají derivaci v každém bodě kvůli průběhu algoritmu zpětného šíření [5] popsaného v kapitole 2.1.4. Mezi takové funkce patří například sigmoida (2.5), viz obr. 2.3, nebo hyperbolická tangenta (2.6), viz obr. 2.4.

$$f(\xi) = \frac{1}{1 + e^{-\alpha\xi}}. \tag{2.5}$$

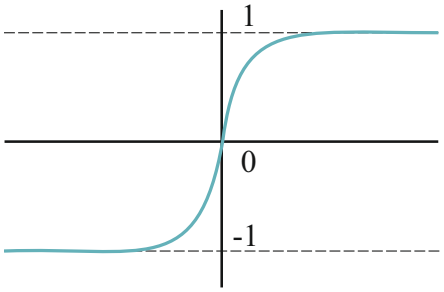


Obrázek 2.2: Skoková funkce.



Obrázek 2.3: Sigmoida.

$$f(\xi) = tanh(\xi) = \frac{2}{1 + e^{-2\xi}} - 1. \tag{2.6}$$

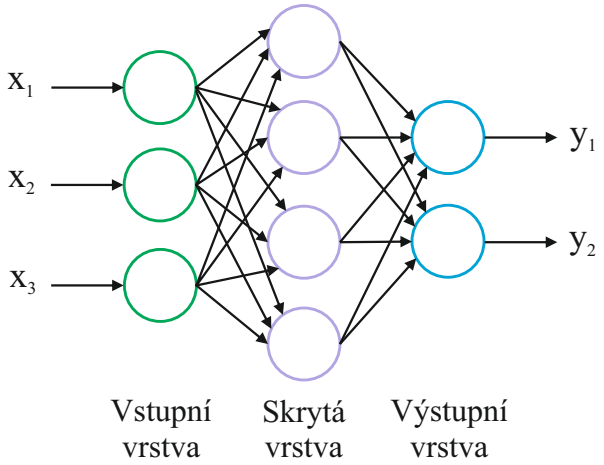


Obrázek 2.4: Hyperbolická tangenta.

2.1.3 Vícevrstvé neuronové sítě

Propojením jednotlivých neuronů a uspořádáním do vrstev vznikne vícevrstvá neuronová síť. Základním typem je dopředná neuronová síť (angl.

Feedforward neural network), která je složená ze vstupní vrstvy, jedné či více skrytých vrstev a výstupní vrstvy [21] [19], viz obr. 2.5. Neurony jedné vrstvy jsou propojeny se všemi neurony vrstvy následující. Za aktivační funkci se volí funkce lineární nebo sigmoida.



Obrázek 2.5: Dopředná neuronová síť.

Počet neuronů vstupní vrstvy je stejný jako počet prvků vstupního vektoru. Například pro úlohu rozpoznávání ručně psaných číslic jsou v databázi MNIST[2] vzorky o velikosti 28×28 obrazových bodů, tj. vstup je vektor o velikosti $28 \times 28 = 784$ prvků. Počet neuronů ve skrytých vrstvách je volen v závislosti na složitosti úlohy experimentálně. Při použití velmi malého počtu neuronů se může stát, že síť nemá kapacitu k naučení daného problému, a při použití velmi vysokého počtu neuronů se může stát, že se výrazně prodlouží doba učení nebo dojde k přeučení sítě, které je popsáno v kapitole 2.1.5. Počet neuronů výstupní vrstvy je ovlivněn kódováním výstupu. Z příkladu rozpoznávání číslic vyplývá, že výstupní vrstva bude mít deset neuronů odpovídajících číslicím od 0 do 9.

2.1.4 Učení sítě a algoritmus zpětného šíření

Učení neuronové sítě probíhá iterativně od nejnižší (vstupní) vrstvy směrem k nejvyšší vrstvě (výstupní). Nejčastěji se používá algoritmus zpětného šíření (angl. *Backpropagation*). Jedná se o gradientní metodu učení s učitelem.

[2] `http://yann.lecun.com/exdb/mnist/`

Algoritmus je dělen do tří kroků – dopředné šíření vstupního signálu, zpětné šíření chyby a aktualizace vah a prahů jednotlivých neuronů. Provádění těchto částí se cyklicky opakuje, dokud není chyba sítě dostatečně malá nebo dokud se nedosáhne mezního počtu iterací. Algoritmus nezaručuje konvergenci, tudíž nemusí požadované chyby nikdy dosáhnout. Cílem je nastavit váhy a prahy tak, aby byl skutečný výstup co nejpodobnější požadovanému [5].

Celkovou chybu sítě lze vypočítat, pokud jsou známy hodnoty skutečných a požadovaných výstupů [29], podle (2.7).

$$E = \frac{1}{2} \sum_{i=0}^{n} \sum_{j=0}^{m} \left(y_{i,j} - t_{i,j} \right)^2 , \tag{2.7}$$

kde n je počet trénovacích vzorů, m je počet výstupů sítě, $y_{i,j}$ je hodnota skutečného výstupu a $t_{i,j}$ je očekávaná výstupní hodnota pro daný trénovací vzor. Zpětným šířením je chybová (resp. cenová) funkce minimalizována.

Po zjištění hodnoty chyby se iterativně upraví hodnota každé váhy na základě její parciální derivace podle (2.8).

$$w_{k,l}(t+1) = w_{k,l}(t) - \alpha \frac{\partial E}{\partial w_{k,l}(t)}, \tag{2.8}$$

kde $w_{k,l}$ je váha hrany vedoucí z neuronu k do neuronu l, parametr α je regularizační parametr, na kterém závisí rychlost učení, a t je index iterace [14]. $-\frac{\partial E}{\partial w_{k,l}}$ je přírůstek váhy přispívající k minimalizaci chyby E.

Výpočet přírůstku pro výstupní vrstvu:

$$\begin{aligned} -\frac{\partial E}{\partial w_{k,l}} &= -\frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial \xi_l} \frac{\partial \xi_l}{\partial w_{k,l}} = -\frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial \xi_l} \frac{\partial}{\partial w_{k,l}} \sum_p w_{pl} y_p \\ &= -\frac{\partial E}{\partial y_l} \frac{\partial y_l}{\partial \xi_l} y_k - \frac{\partial E}{\partial \xi_l} f'(\xi_l) y_k = -(y_l - t_l) f'(\xi_l) y_k = \delta_l y_k. \end{aligned} \tag{2.9}$$

Výpočet přírůstku pro skryté vrstvy:

$$\begin{aligned} -\frac{\partial E}{\partial w_{k,l}} &= -\left(\sum_p \frac{\partial E}{\partial \xi_p}\frac{\partial \xi_p}{\partial y_l}\right)\frac{\partial y_l}{\partial \xi_l}y_k = -\left(\sum_p \frac{\partial E}{\partial \xi_p}\frac{\partial}{\partial y_l}\sum_q w_{q,p}y_q\right)\frac{\partial y_l}{\partial \xi_l}y_k \\ &= -\left(\sum_p \frac{\partial E}{\partial \xi_p}w_{l,p}\right)\frac{\partial y_l}{\partial \xi_l}y_k = -\left(\sum_p \delta_p w_{l,p}\right)f'(\xi_l)y_k = \delta_l y_k, \end{aligned} \tag{2.10}$$

kde δ_p je vypočtený chybový faktor neuronu. Chybový faktor je část chyby neuronové sítě šířené z daného neuronu ke všem neuronům nižší vrstvy.

Aktivační funkce musí obecně splňovat několik kritérií. Musí být spojitá, diferencovatelná a monotónně neklesající.

Pro derivaci sigmoidy platí:

$$f'(\xi_l) = f(\xi_l)(1 - f(\xi_l))\alpha. \tag{2.11}$$

Počet iterací, po kterém dojde k uložení modifikovaných vah, se nazývá **epocha**.

2.1.5 Problémy učení

Uváznutí v lokálním minimu

Tento problém může nastat při použití gradientní metody, kdy se při uváznutí v lokálním minimu algoritmus předčasně ukončí. Proto je vhodné volit monotónní aktivační funkce. Tento problém může být vyřešen změnou topologie sítě – jinými hodnotami počátečních vah, přidáním či odebráním neuronů skryté vrstvy.

Přeučení sítě

K přeučení (angl. *overfitting*) dochází, pokud má síť příliš velký počet neuronů. Neuronová síť se naučí přesně rozpoznávat vzory ze sady trénovacích

dat (velmi malá hodnota chyby E) a ztrácí tak schopnost generalizace. Při klasifikaci dat neobsažených v trénovací množině může častěji docházet k chybám. Řešením je zvětšení trénovací sady nebo opět změna topologie – např. odebrání neuronů ze skryté vrstvy.

Nedoučení sítě

K nedoučení (angl. *underfitting*) dochází, pokud má síť naopak velmi malý počet neuronů. Hodnota chyby E je pak velmi vysoká. Řešením může být přidání neuronů do skryté vrstvy. V zásadě se doporučuje začít s menším množstvím neuronů a postupně jejich počet zvyšovat.

2.2 Konvoluční neuronové sítě

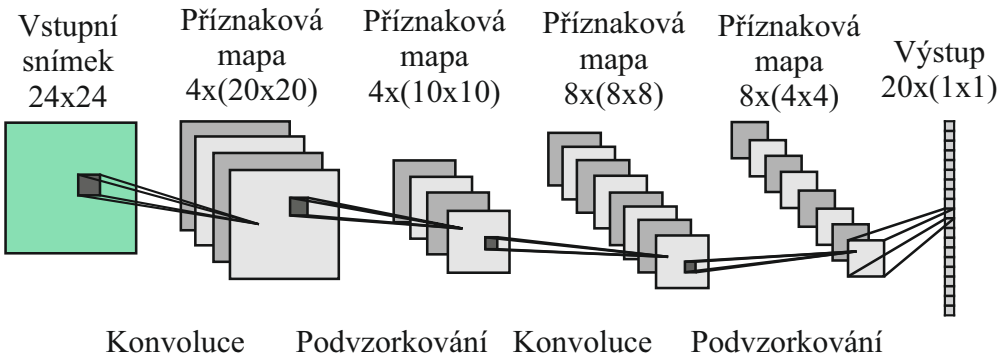
Konvoluční neuronové sítě jsou speciálním druhem vícevrstvých dopředných neuronových sítí. Byly navrženy pro rozpoznávání snímků přímo z pixelů s minimálním předzpracováním. Topologie sítě je inspirována vizuálním kortexem mozku. Cílem sítě je rozpoznávání objektů nezávisle na jejich deformaci, změně polohy a velikosti.

Konvoluční sítě obsahují vrstvy, z nichž každá má svou specifickou funkci. Příklad architektury je ilustrován na obr. 2.6. Z obrázku je patrné, že například druhá vrstva obsahuje čtyři příznakové mapy o velikosti 20×20 pixelů. K charakteristickým vlastnostem patří například získávání příznaků z recepčních polí a technologie sdílení vah [2] [27]. Recepční pole je plocha vstupního snímku vymezená konvolučním filtrem. Pole je znázorněno na obr. 2.7 zeleně.

2.2.1 Vrstvy konvoluční sítě

Vstupní vrstva

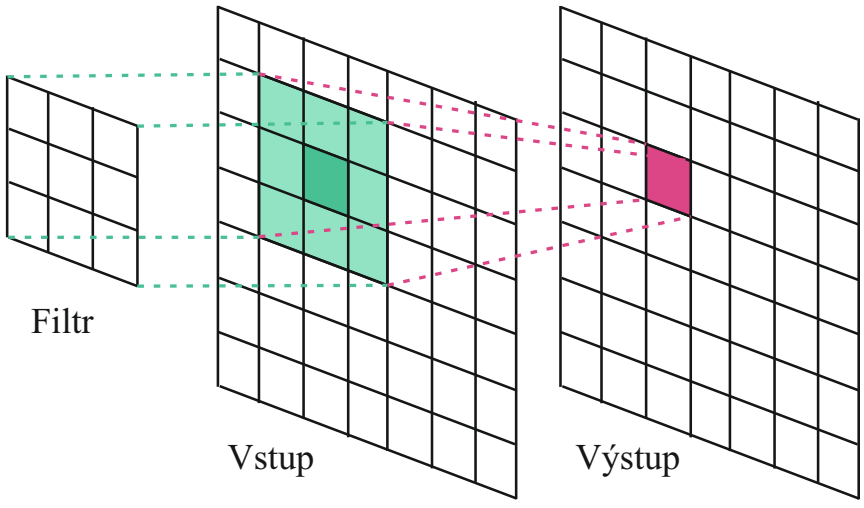
Vstupem je matice hodnot obrazových bodů snímku ve formátu *výška snímku* $\times$ *šířka snímku* $\times$ *hloubka*. Hloubkou je myšlen počet barevných kanálů. Pro snímky ve formátu RGB je hloubka rovna třem kanálům. Černobílé snímky obsahují pouze jeden kanál.



Obrázek 2.6: Příklad architektury konvoluční neuronové sítě.

Konvoluční vrstva

Konvoluční vrstva slouží k extrahování příznaků ze snímku. Obsahuje sadu konvolučních filtrů (angl. *kernels*). Tyto filtry jsou velice malé, většinou je jejich velikost 5×5 obrazových bodů se třemi barevnými kanály. Při dopředném kroku je provedena operace konvoluce[3] mezi pixely vstupního snímku a filtry. Filtry jsou vždy aplikovány na každý obrazový bod, jak je vidět na obr. 2.7. Výstupem po aplikaci je dvojdimenzionální *příznaková mapa* [11].



Obrázek 2.7: Aplikace konvolučního filtru.

Konvoluce ve spojitém prostoru je definována vztahem (2.12).

[3] `https://en.wikipedia.org/wiki/Convolution`

$$(f * g)(x) = \int_{-\infty}^{\infty} f(\alpha)g(x-\alpha)d\alpha, \tag{2.12}$$

kde funkce $g(x)$ se nazývá konvoluční jádro. Hodnota konvoluce funkce f s jádrem g v bodě x je integrál ze součinu funkce f s otočenou funkcí konvolučního jádra [28].

Při zpracování snímků se používá diskrétní model konvoluce. Výpočet hodnoty v bodě x, y je definován vztahem (2.13) [28].

$$(f * g)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j) \cdot g(i, j), \tag{2.13}$$

kde $f(x, y)$ je hodnota pixelu vstupního snímku $\mathbf{f}$ na souřadnicích (x, y), $g(i, j)$ je hodnota váhy konvolučního filtru g na souřadnicích (i, j) a $k \times k$ je velikost konvolučního filtru.

Síť se naučí, že se pro snímky se stejnými příznaky aktivují stejné filtry. Příznakem mohou být například různě orientované hrany či oblasti specifické barvy v první konvoluční vrstvě. Při použití více konvolučních vrstev dostávají příznaky srozumitelnější charakter, například příznaky tvarů jako kruhovitost či obdélníkovitost.

Vstupní a výstupní dimenze konvoluční vrstvy

Pokud je na vstupu neuronové sítě digitální snímek, není z výpočetního hlediska vhodné, aby byla vstupní vrstva s konvoluční plně propojena. Místo toho se zavedl pojem *lokální konektivita*, kdy je neuron konvoluční vrstvy propojen pouze částí vstupu [11]. Pokud by například na vstupu byl snímek o velikosti $32 \times 32 \times 3$ a velikost filtru byla 5×5, pak by jeden neuron konvoluční vrstvy byl propojen pouze s oblastí o velikosti $5 \times 5 \times 3$, tj. celkem 75 vah namísto 3072.

Velikost výstupu ovlivňuje několik faktorů, které jsou:

- **Hloubka** – Hloubkou je v tomto případě myšlen počet filtrů konvoluční vrstvy. Počet *příznakových map* bude stejný jako počet aplikovaných filtrů.

- **Krok konvoluce** – Krok udává, o kolik bodů se konvoluční filtr posouvá.
- **„Zero padding“** – Před konvolucí je možné vstupní snímek rozšířit o nulové prvky. Tyto prvky se přidají kolem snímku tak, aby bylo možné aplikovat konvoluční filtr i na okrajové pixely.

Počet neuronů na výstupu:

$$N = \frac{W - F + 2P}{S + 1}, \tag{2.14}$$

kde W je velikost vstupu, F je velikost konvolučního filtru, P je velikost okraje složeného z nulových prvků a S je velikost kroku. Pro snímek velikosti $227 \times 227 \times 3$, filtr 11×11, žádný nulový okraj a krok velikosti 4 je výsledek $(227 - 11)/4 + 1 = 55$.

Velikost výstupní dimenze O konvoluční vrstvy lze vypočítat jako:

$$O = N \times N \times K, \tag{2.15}$$

kde K je počet konvolučních filtrů.

Aktivační vrstva

V této vrstvě se aplikuje nelineární aktivační funkce na výstup každé konvoluční vrstvy. V některých zdrojích se aplikace aktivační funkce uvádí přímo jako součást konvoluční vrstvy a ne jako samostatná vrstva.

U klasických neuronových sítí se používá sigmoida [11], protože na vstupu přijímá reálné hodnoty (součet signálů v těle neuronu). U konvolučních sítí se jako aktivační funkce používá *Rectified Linear Unit* neboli *ReLU*, která je definována vztahem (2.16). Průběh funkce je znázorněn na obr. 2.8.

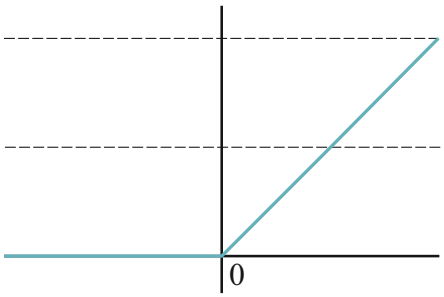
$$f(\xi) = max(0, \xi). \tag{2.16}$$

Z rovnice vyplývá, že každá záporná hodnota příznakové mapy je nahrazena nulou.

Pro derivaci ReLU platí:

$$f'(\xi) = \begin{cases} 1 \; pro \; \xi > \theta, \\ 0 \; pro \; \xi \leq \theta, \end{cases} \tag{2.17}$$

kde θ je práh, jehož hodnota je obvykle 0.



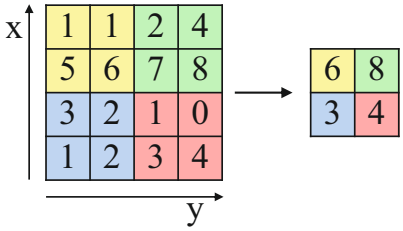
Obrázek 2.8: Aktivační funkce konvoluční sítě – ReLU.

I u konvolučních sítí lze použít jako aktivační funkci sigmoidu nebo hyperbolickou tangentu, ale experimentálně bylo zjištěno, že při použití ReLU je trénování sítě několikanásobně rychlejší [9].

Pooling vrstva

Po konvolučních vrstvách se obvykle vkládají tzv. *pooling* (podvzorkovací) vrstvy. Tyto vrstvy slouží k redukci dimenze pro snížení počtu parametrů sítě a výpočetní náročnosti, což pomáhá i proti přetrénování sítě. Počet příznakových map je stejný jako u předchozí konvoluční vrstvy. Nejpoužívanější metodou podvzorkování je *max–pooling*. Na příznakovou mapu je aplikován filtr o určité velikosti, který si lze představit jako *okénko* vymezující část mapy. Z okénka je pak zachován prvek s nejvyšší hodnotou [23], viz obr. 2.9.

Vstupem je matice o velikosti $W_1 \times H_1 \times D_1$, kde W_1 je šířka, H_1 výška příznakových map a D_1 jejich počet. Pokud bude dále zadána velikost vzorkovacího filtru F a krok S, lze velikost dimenze výstupu vypočítat podle (2.18) [11].



Obrázek 2.9: Podvzorkování.

$$\begin{aligned} W_2 &= \frac{W_1 - F}{S + 1}, \\ H_2 &= \frac{H_1 - F}{S + 1}, \\ D_2 &= D_1. \end{aligned} \tag{2.18}$$

Při podvzorkování se nepoužívají nulové okraje jako u konvoluce.

Plně propojená vrstva

Po sérii konvolučních a *pooling* vrstev následuje jedna nebo více plně propojených (angl. *fully–connected*) vrstev, kdy je každý neuron této vrstvy spojený se všemi neurony vrstvy předchozí. Příznaky získané z konvolučních vrstev by pro většinu klasifikačních úloh stačily, ale vložením plně propojené vrstvy se síť může snadno naučit i jejich nelineární kombinace [10].

Výstupní vrstva

Poslední vrstva konvoluční sítě je také plně propojená s předchozí. Obsahuje stejný počet neuronů jako je počet klasifikačních tříd. Pokud je potřeba klasifikovat do více než dvou tříd, je jako aktivační funkce volena funkce nazvaná *softmax*. *Softmax* funkce [11] [23] je zobecnění logistické regrese. Výstupem jsou pravděpodobnosti příslušností k jednotlivým třídám, které jsou definovány podle (2.19). Součet těchto pravděpodobností se rovná 1.

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}, \tag{2.19}$$

kde z_j je součet hodnot neuronů předchozí vrstvy násobených vahami vedoucími k neuronu j a je definován podle (2.20).

$$z_j = w_1^j x_1 + ... \; w_n^j x_n = \sum_{i=1}^{n} w_i^j x_i. \tag{2.20}$$

2.2.2 Vizualizace

Jak již bylo zmíněno, při použití každé další konvoluční vrstvy dostávají hledané příznaky jasnější charakter, což je vidět z obr. 2.10. Nejčastěji jsou vizualizovány naučené filtry. Vizualizace je důležitá k lepšímu pochopení, jak se konvoluční sítě učí. Obrázek 2.10 obsahuje příklad vizualizací konvolučních filtrů a k nim příslušných snímků. Je zde vidět, že v první konvoluční vrstvě jsou rozpoznávány hrany objektů, ve třetí podobné textury a v páté celé objekty [6].

2.2.3 Učení konvoluční sítě

Jako u klasických neuronových sítí, i konvoluční sítě k učení využívají pro plně propojené vrstvy algoritmus zpětného šíření, který je popsán v kapitole 2.1.4. Pro ostatní vrstvy je mírně upraven [1].

Konvoluční vrstva

Každá konvoluční vrstva l je následovaná podvzorkovací vrstvou $l+1$, kdy je jeden neuron podvzorkovací vrstvy spojen s blokem neuronů v konvoluční vrstvě. Pro výpočet chyby ve vrstvě l bude zvětšena mapa chyb z vrstvy $l+1$ na velikost mapy z vrstvy l. Pro chyby jednotlivých příznakových map ve vrstvě l platí:

$$\delta_j^l = \beta_j^{l+1} \left(f'(\xi_j^l) \circ \text{UP}(\delta_j^{l+1}) \right), \tag{2.21}$$

kde j je index příznakové mapy, β_j^{l+1} jsou váhy v podvzorkovací vrstvě, $f'(\xi_j^l)$ je derivace aktivační funkce a UP(...) představuje zvětšovací operaci, která

Obrázek 2.10: Vizualizace konvolučních filtrů [6].

každý obrazový bod ukládá horizontálně i vertikálně n–krát a která je definována jako:

$$\text{UP}(x) = x \otimes 1_{n \times n}, \tag{2.22}$$

kde operace $\otimes$ značí Kroneckerův součin matic[4].

Gradient prahu b je definován jako součet chyb jednotlivých příznakových map podle (2.23).

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^l)_{u,v}. \tag{2.23}$$

Gradient vah podle (2.24) je definován podobně jako u metody zpětného šíření s tím rozdílem, že váhy u konvolučních sítí jsou sdílené přes více spojení.

$$\frac{\partial E}{\partial k_{ij}^l} = \sum_{u,v} (\delta_j^l)_{uv} (p_i^{l-1})_{uv}, \tag{2.24}$$

kde $(p_i^{l-1})_{uv}$ je oblast z x_i^{l-1}, která byla vynásobena hodnotou k_{ij}^l při konvoluci pro výpočet hodnoty bodu (u, v) na výstupu konvoluční mapy x_j^l.

Pooling vrstva

Jak již bylo zmíněno, v podvzorkovací vrstvě probíhá operace zmenšení vstupních map, kterou lze formálně zapsat jako:

$$x_j^l = f\left(\beta_j^l \cdot \text{DOWN}(x_j^{l-1}) + b_j^l\right), \tag{2.25}$$

kde DOWN(...) je podvzorkovací funkce, β_j^l je multiplikativní práh pro každou mapu a b_j^l je aditivní práh. Pokud za podvzorkovací vrstvou následuje plně propojená vrstva, probíhá algoritmus zpětného šíření tak, jak je popsán v kapitole 2.1.4. Pokud následuje vrstva konvoluční, je nutné zjistit, které neurony aktuální vrstvy odpovídají neuronům následující vrstvy. Váhy, které násobí

[4] `https://en.wikipedia.org/wiki/Kronecker_product`

spojení mezi vstupem a výstupem, odpovídají vahám konvolučního filtru, což lze definovat jako:

$$\delta_j^l = f'(u_j^l) \circ \text{CONV2}\left(\delta_j^{l+1}, \text{ROT180}(k_j^{l+1}), full\right), \tag{2.26}$$

kde CONV2(...) je operace konvoluce a ROT180(...) je operace, kdy je konvoluční jádro reprezentované polem rotované o 180°. Parametr *full* říká, že příznakové mapy budou rozšířeny o nulové prvky, aby bylo možné provést konvoluci i pro okrajové prvky. Následně je možné vypočítat gradient pro multiplikativní a aditivní prahy. Pro aditivní práh b se gradient vypočítá podle (2.27).

$$\frac{\partial E}{\partial b_j} = \sum_{u,v} (\delta_j^l)_{u,v}. \tag{2.27}$$

Pro výpočet derivace multiplikativního prahu β je potřeba mít uloženy originální podvzorkované mapy d_j^l z dopředného průchodu sítí definované podle (2.28).

$$d_j^l = \text{DOWN}(x_j^{l-1}). \tag{2.28}$$

Pak lze gradient vypočítat podle (2.29).

$$\frac{\partial E}{\partial \beta_j} = \sum_{u,v} (\delta_j^l \circ d_j^l)_{uv}. \tag{2.29}$$

2.2.4 Regularizace

Základní problém strojového učení je vytvoření algoritmu, který bude dobře vyhodnocovat nejen trénovací a testovací data, ale zejména data neznámá. Regularizační techniky slouží k tomu, aby se chyba sítě při testování zmenšovala i za cenu zvětšování chyby při trénování [20], tj. je nutné zamezit přeučení sítě, které je popsáno v kapitole 2.1.5. Dále budou uvedeny jen některé z mnoha regularizačních technik.

L2 regularizace

Tato regularizační technika spočívá v přidání tzv. regularizačního parametru k chybové funkci [15]. Pro kvadratickou chybu bude vzorec (2.7) z kapitoly 2.1.4 upraven podle (2.30).

$$E = \frac{1}{2}\sum_{i=0}^{n}\sum_{j=0}^{m}(y_{i,j} - t_{i,j})^2 + \frac{\alpha}{2}\sum_{w} w^2, \tag{2.30}$$

kde w jsou hodnoty synaptických vah sítě. Obecně lze regularizovanou chybovou funkci zapsat jako:

$$E = E_0 + \frac{\alpha}{2}\sum_{w} w^2. \tag{2.31}$$

Pokud je hodnota parametru α malá, bude síť preferovat minimalizaci chybové funkce. Pokud je hodnota parametru α velká, bude síť upřednostňovat malé váhy.

Umělé zvětšování trénovací sady dat

Pro natrénování neuronových sítí je potřeba mít velké množství trénovacích dat. Pokud je k dispozici pouze omezené množství, je vhodné použít právě techniku umělého zvětšování trénovací množiny. Na originální snímky se aplikují jednoduché obrazové transformace [11]. Mezi tyto transformace patří například:

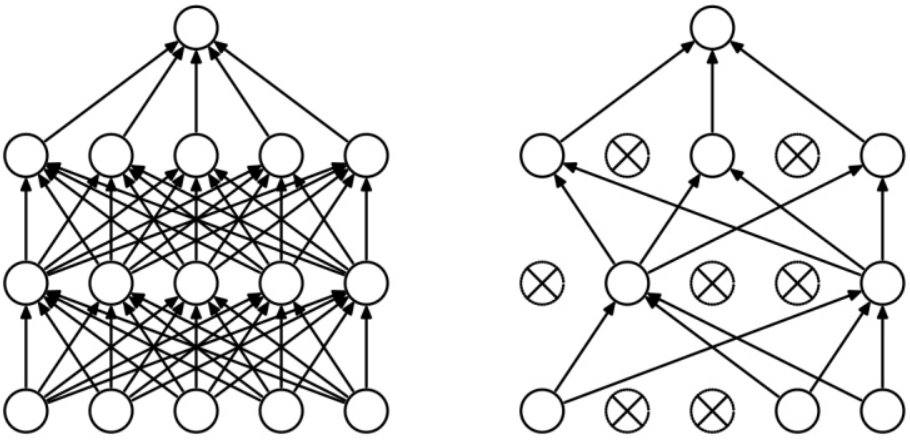
- Rotace.
- Translace.
- Změna měřítka (angl. *scaling*).
- Oříznutí.
- Změna barev (angl. *color jittering*).

Je možné aplikovat i kombinace těchto transformací.

Dropout

Pokud není dán časový limit pro naučení sítě, je nejlepší model ten, který vznikne kombinací různých modelů s různě nastavenými parametry. Kombinovat modely je efektivní pouze v případě, že mají jinou architekturu nebo byly trénovány na jiných datech. Optimalizace parametrů pro různé architektury konvolučních sítí je časově extrémně náročná a většinou není k dispozici dostatek dat, aby se stejný model dal natrénovat na odlišných trénovacích množinách. Použití regularizační techniky *dropout* zabraňuje přeučení sítě a poskytuje způsob, jak efektivně kombinovat různé architektury neuronových sítí [18].

Vybrané neurony vstupní a skrytých vrstev se ze sítě dočasně odstraní, jak je vidět na obr. 2.11. Na levé straně obrázku je vyobrazena plně propojená neuronová síť a na pravé straně je neuronová síť po aplikaci metody *dropout*. Výběr neuronů, které se nebudou podílet na učení sítě, je náhodný. Neuron bude neaktivní s pravděpodobností p. U neuronů vstupní vrstvy je pravděpodobnost p nastavena obvykle na 0,2 a u neuronů skrytých vrstev na 0,5. Neuronové sítě implementující tuto techniku dosahují prokazatelně lepších výsledků [17].



a) Standardní neuronová síť. b) Po aplikaci metody dropout.

Obrázek 2.11: Dropout [18].

2.2.5 Architektury

Existuje mnoho různých architektur konvolučních neuronových sítí. Architektury se liší například počtem neuronů ve vrstvách, pořadím vrstev nebo použitou aktivační funkcí. Mezi nejznámější patří:

LeNet

První úspěšnou implementaci konvoluční neuronové sítě vytvořil Yann LeCun. Tato síť byla primárně určena pro rozpoznávání znaků v dokumentech. Poslední verze, *LeNet–5*, je složena z osmi vrstev – vstupní vrstva, dvakrát konvoluční vrstva následovaná podvzorkovací, dvě plně propojené vrstvy a výstupní vrstva [8].

AlexNet

V roce 2012 byla představena síť *AlexNet*. Měla více vrstev a neuronů než *LeNet*. Síť s touto architekturou vyhrála soutěž *ImageNet Large Scale Visual Recognition Challenge*[5] (ILSVRC), což byl zásadní průlom pro výzkum konvolučních neuronových sítí. Architektura je speciální tím, že měla i několik po sobě jdoucích konvolučních vrstev. Do té doby bylo obvyklé, že po konvoluční vrstvě následovala podvzorkovací vrstva [9].

ZF Net

Síť s architekturou *ZF Net* vyhrála *ILSVRC* v roce 2013. Byla založena na architektuře *AlexNet*. Rozdíl byl v nastavení parametrů sítě, velikosti prostřední konvoluční vrstvy a velikosti kroků filtru v první konvoluční vrstvě [7].

GoogLeNet

Síť s architekturou *GoogLeNet* vyhrála *ILSVRC* v roce 2014. Zásadním přínosem této architektury byl tzv. *Inception Module*, díky kterému se dramaticky

[5] `http://www.image-net.org/challenges/LSVRC/`

snížil počet parametrů sítě [16]. Pro porovnání architektura *AlexNet* měla šedesát milionů parametrů, *GoogLeNet* měla jen čtyři miliony.

VGGNet

Síť *VGGNet* vyvinutá na Oxfordu využívala v konvolučních vrstvách filtry velikosti 3×3, tedy mnohem menší než *AlexNet*. Velkým přínosem pro výzkum konvolučních sítí bylo, že se podařilo prokázat, že hloubka sítě, tj. počet vrstev, má zásadní vliv na rychlost [30]. Síť měla celkem šestnáct vrstev.

ResNet

Síť s architekturou *ResNet* vyhrála *ILSVRC* v roce 2015. Architektura sítě je specifická v tom, že na konci nemá žádné plně propojené vrstvy. Vrstev bylo celkem 152 [22]. *ResNet* je považována za nejmodernější mezi architekturami konvolučních sítí.

2.2.6 Software

Následující výčet zahrnuje frameworky pro práci s konvolučními neuronovými sítěmi. Framework je sada knihoven, které implementují časté programátorksé konstrukce a usnadňují tím rutinní práci. Cílem frameworku je sjednocení architektury aplikací a usnadnění práce programátora.

Theano

Knihovna *Theano* je psaná v programovacím jazyce Python. Umožňuje definovat, optimalizovat a vyhodnocovat matematické výrazy zahrnující multidimenzionální pole. Výhody knihovny jsou například [24]:

- Integrace s matematickou knihovnou NumPy pro jazyk Python. Tato knihovna usnadňuje práci s numerickými daty.
- Náročné výpočty mohou být prováděny grafickým akcelerátorem (GPU).

- Rychlost a stabilita.
- Dynamické generování kódu v jazyce C pro rychlejší vyhodnocování výrazů.
- Rozsáhlá podpora automatického testování.

Caffe

Framework *Caffe* je psaný v programovacím jazyce C++. Poskytuje rozhraní pro Python a Matlab. Součástí jsou i hotové modely a testovací příklady. Výhody tohoto frameworku jsou například [4]:

- Konvoluční síť lze definovat konfiguračním skriptem.
- Rozšiřitelný kód – lze provádět vlastní úpravy kódu.
- Rychlost – *Caffe* dokáže zpracovat až 60 milionů snímků denně s využitím GPU.
- Komunita – používá se jak pro akademické účely, tak i v komerční sféře.

CNTK

Microsoft Cognitive Toolkit (CNTK) je sada nástrojů vyvinutá společností Microsoft. Výhody *CNTK* jsou například [31]:

- Poskytuje rozhraní pro Python a C++.
- Síť lze definovat konfiguračním skriptem ve speciálním jazyce BrainScript.
- Podporuje operační systémy Linux i Windows.
- Pro použití lze nainstalovat připravené binární soubory, které lze stáhnout z webových stránek projektu[6], nebo si stáhnout zdrojové kódy a vytvořit vlastní binární soubor.
- Rozsáhlá sada hotových modelů, manuálů a testovacích příkladů.
- Komunita.

[6] `https://github.com/Microsoft/CNTK/releases`

TensorFlow

TensorFlow[7] je knihovna, kde je konvoluční síť reprezentována grafem. Uzly jsou matematické operace a hrany představují multidimenzionální pole. Struktura grafu závisí na charakteru vstupních dat.

DLib

DLib je sada nástrojů psaná v jazyce C++, pomocí které lze vytvářet komplexní programy pro řešení problémů strojového učení. Výhody *DLib* jsou například [12]:

- Rozsáhlá dokumentace všech implementovaných funkcí.
- Poskytuje rozhraní pro Python.
- Rozsáhlá sada ukázkových příkladů.
- Knihovna se nemusí nijak instalovat či dále nastavovat. Pro použití stačí stáhnout zdrojové kódy a zkompilovat.
- Implementuje i jiné algoritmy strojového učení než hluboké sítě. Součástí jsou například i algoritmy pro SVM, shlukovací algoritmy a další.

Z uvedeného výčtu byla k realizaci práce vybrána sada nástrojů *CNTK* od společnosti Microsoft. Důvody pro zvolení *CNTK* jsou:

- Živý projekt, který je neustále vyvíjen a vylepšován.
- Aktivní komunita uživatelů.
- Přehledná dokumentace projektu.
- Rozsáhlá sada testovacích příkladů.
- Snadné napojení na webovou aplikaci.

[7] `https://www.tensorflow.org/`

3 Realizační část

3.1 CNTK

Jednou z výhod CNTK je, že se jedná o stále vyvíjený projekt, který je neustále vylepšován a optimalizován. Toto může být na druhou stranu i jeho nevýhoda. Za dobu vývoje a testování diplomové práce se několikrát změnily konvence zápisu pro konfigurační skript, který je popsán v kapitole 3.2, a některé součásti jsou již zastaralé (angl. *deprecated*). Z toho vyplývá, že informace v následující části nemusí být v době čtení již aktuální. Poslední verze CNTK, která je použitá pro tuto práci, je verze 2.0 beta 11[1] z 10. února 2017.

3.1.1 Instalace

Podporovány jsou operační systémy Windows (64–bit) i Linux (64–bit). V rámci práce byla otestována instalace na obou operačních systémech (dále jen OS). Pro použití *CNTK* je možné buď stáhnout již připravené binární soubory nebo stáhnout zdrojové kódy projektu.

Instalace pro OS Windows

Pro tento systém byla zvolena možnost instalace přímo s použitím zdrojových kódů projektu, jelikož počítač, na kterém byl tento typ instalace prováděn, byl určen k vývoji. Pro sestavení projektu je v současné době nutné mít nainstalováno *Microsoft Visual Studio 2015 update 3*. Pro stažení zdrojových kódů[2] z repozitáře projektu je potřeba mít nainstalovanou aplikaci pro práci s verzovacím systémem Git, například SourceTree, GitHub Desktop Client nebo TortoiseGit. Pak je již možné stáhnout zdrojové kódy na lokální disk.

Seznam dalších potřebných knihoven:

- *Intel Math Kernel Library* (MKL) – Knihovna pro urychlení matema-

[1] `https://github.com/Microsoft/CNTK/releases/tag/v2.0.beta11.0`
[2] `https://github.com/Microsoft/CNTK`

tických výpočtů neuronových sítí[3].

- *Microsoft MPI v7* (Message Passing Interface) – Rozhraní pro vývoj aplikací založené na zasílání zpráv mezi jednotlivými uzly[4].
- *Boost* – Sada knihoven pro programovací jazyk C++, které poskytují podporu generování pseudonáhodných čísel, zpracování obrazu, vícevláknové programování a další[5].
- *Protobuf* (Protocol Buffers) – Knihovna slouží pro serializaci strukturovaných dat. Je nezávislá na platformě i programovacím jazyku[6].
- *SWIG* – Nástroj pro spojení programů psaných v C nebo C++ s vyššími programovacími jazyky (PHP, Ruby, Python atd.)[7].
- *OpenCV* (Open Source Computer Vision) – Knihovna pro manipulaci s digitálními snímky. Je zaměřena na počítačové vidění a zpracování digitálních snímků v reálném čase[8]. Tato knihovna je potřebná, pokud vstupem sítě budou právě digitální snímky.
- *zlib & libzip* – Knihovny zlib[9] a libzip[10] slouží pro vytváření, čtení a úpravám archivních souborů ve formátu `zip`. Tyto knihovny jsou potřebné, pokud vstupem sítě budou digitální snímky.
- *Anaconda* – Distribuce programovacího jazyka Python. Obsahuje balíky pro práci s velkými daty[11].

Jelikož součástí vývojového počítače není grafická karta s čipy od společnosti NVIDIA, nebylo možné využít výhod, které tyto karty poskytují k akceleraci výpočetních operací.

[3] `https://software.intel.com/en-us/intel-mkl`
[4] `https://www.microsoft.com/en-us/download/details.aspx?id=49926`
[5] `http://www.boost.org/`
[6] `https://github.com/google/protobuf`
[7] `http://swig.org/`
[8] `http://opencv.org/`
[9] `http://zlib.net/`
[10] `https://nih.at/libzip/`
[11] `https://www.continuum.io/`

Instalace pro OS Linux

Pro tento systém byla zvolena možnost instalace z připraveného binárního souboru. Celkem se instalovalo na dva počítače, z nichž jeden byl osazen grafickou kartou s čipem NVIDIA, druhý ne.

Pro využití grafické karty na akceleraci výpočtů je nutné mít nainstalované následující knihovny:

- *NVIDIA CUDA* (Compute Unified Device Architecture) – Prostředí pro vývoj aplikací v C a C++ akcelerovaných na GPU. Obsahuje matematické knihovny a nástroje pro ladění a optimalizaci vyvíjené aplikace[12].
- *cuDNN* – Knihovna speciálně určená pro výpočty v hlubokých neuronových sítích. Poskytuje implementaci pro dopřednou i zpětnou konvoluci, podvzorkování, normalizaci atd.[13]
- *CUB*[14] – Knihovna poskytující vysoce výkonná primitiva pro paralelní programování a další nástroje pro vývoj aplikací v prostředí CUDA.
- Poslední verze ovladače grafické karty.

3.2 Konfigurační skript

Příkazové bloky a architektura neuronové sítě se definují konfiguračním skriptem. Skript je psán ve speciálním jazyce BrainScript. Konfigurační příkazy jsou specifikovány dvojicí *název–hodnota*, kde hodnota může být číslo, řetězec, pole nebo vnořený blok. Následující ukázky kódu jsou převzaty ze souboru `config.cntk`, který lze nalézt na přiloženém CD.

3.2.1 Bloky skriptu

Příkazy, které se budou vykonávat, jsou sdruženy do pojmenovaných bloků. Pojmenování bloků záleží na uživateli. Pořadí, v jakém se budou bloky vy-

[12] `https://developer.nvidia.com/cuda-downloads`
[13] `https://developer.nvidia.com/cuDNN`
[14] `https://github.com/NVlabs/cub`

konávat, je určeno parametrem `command`.

Příklad zápisu:

```
command = Train:Test
```

Ukázka kódu 3.1: Příklad zápisu příkazu `command`.

Každý blok je určen svou akcí. Například ze zápisu

```
LibovolnyNazev = {
  action="train"
  ...
}
```

Ukázka kódu 3.2: Příklad zápisu bloku akcí.

vyplývá, že blok s názvem `LibovolnyNazev` sdružuje příkazy pro trénování (angl. *train*) neuronové sítě.

Blok Train

Blok s akcí *train* slouží pro natrénování modelu neuronové sítě. Musí obsahovat alespoň tři základní bloky. Tyto bloky jsou:

- `reader` – Tento blok slouží k načítání vstupních dat a bude detailněji popsán v kapitole 3.3.
- `SGD` (Stochastic Gradient Descent) – SGD znamená v češtině stochastický gradientní sestup. Před popisem bloku je nutné zavést tyto termíny:
  - *Epocha* (angl. *epoch*) – Jedna epocha je kompletní, pokud sítí prošly všechny trénovací vzorky.
  - *Velikost dávky* (angl. *batch size*) – Počet trénovacích vzorků, které prochází sítí. Čím je velikost dávky větší, tím více operační paměti je využito.
  - *Počet iterací* – Počet průchodů sítí jednou dávkou. Průchod sítí se skládá z jednoho dopředného a jednoho zpětného průchodu.

  Příklad: Máme sadu trénovacích dat obsahující 10 vzorků, velikost dávky je 2 a algoritmus běží po tři epochy. Z toho vyplývá, že v každé

epoše bude pět dávek (10/2 = 5). Každá dávka musí projít sítí, takže na jednu epochu připadá pět iterací. Pokud má být trénování zastaveno po třech epochách, proběhne celkem patnáct iterací algoritmu.

Blok gradientního sestupu má následující parametry:

```
SGD = {
    epochSize = 0
    # velikost dávky
    minibatchSize = 64
    # rychlost učení
    learningRatesPerMB = 0.01*20:0.003*12:0.001*28:0.0003
    # moment
    momentumPerMB = 0.9
    # maximální počet epoch
    maxEpochs = 30
    # L2 regularizace
    L2RegWeight = 0.0005
    # dropout
    dropoutRate = 0.5
}
```

Ukázka kódu 3.3: Blok parametrů gradientního sestupu.

Parametr `epochSize` je počet vzorků, po kterých jsou provedeny speciální akce, například uložení momentálního stavu modelu (angl. *checkpoint*). Pro velké sady dat je někdy žádoucí uložit stav modelu po určitém časovém intervalu, například každých 30 minut. Hodnota tohoto parametru bude odpovídat počtu vzorků, jejichž zpracování trvá daný čas. Pokud je nastaven na 0, provedou se speciální akce[15] po zpracování celé sady dat.

Parametr `momentumPerMB` udává moment algoritmu zpětného šíření. Jedná se o modifikaci algoritmu, kdy se během výpočtu zohledňují nejen změny vah ve směru gradientu, ale i předešlá změna vah, tzv. *moment*. Určuje tedy míru vlivu předchozí změny. Obvykle se volí 0,9 [25].

Parametry `L2RegWeight` a `dropoutRate` slouží k regularizaci a jsou popsány v kapitole 2.2.4.

- `BrainScriptNetworkBuilder` – Tento blok slouží k definování neuronové sítě. Definici lze provést přímo v konfiguračním souboru nebo v externím souboru s příponou `bs`. Definice sítě v externím souboru se hodí, když jsou testovány různé návrhy sítě.

[15] `https://github.com/Microsoft/CNTK/wiki/BrainScript-epochSize-and-Python-epoch_size-in-CNTK`

Blok pro definici neuronové sítě:

```
BrainScriptNetworkBuilder = {
  model = Sequential (
    # definice modelu neuronové sítě
  )
  # vstupní snímek o velikosti 224x224 pixelů se třemi barevnými kanály
  features = Input{224:224:3}
  # normalizace snímku
  featNorm = features - Constant (128)
  # počet klasifikačních tříd
  labels = Input{2}

  # aplikace modelu na normalizovaný snímek
  z = model (featNorm)

  # aplikace funkce Sotfmax
  ce = CrossEntropyWithSoftmax (labels, z)
  # výpočet chyby predikce pro každou třídu
  errs = ClassificationError (labels, z)

  # deklarace speciálních uzlů
  featureNodes    = (features)
  labelNodes      = (labels)
  criterionNodes  = (ce)
  evaluationNodes = (errs)
  outputNodes     = (z)
}
```

Ukázka kódu 3.4: Blok pro definici neuronové sítě.

Ze speciálních uzlů jsou nejdůležitější `outputNodes`, které reprezentují výstup sítě. V případě binární klasifikace budou výstupní uzly dva.

Pomocí parametru `model` lze definovat architekturu sítě. Základní vrstvy jsou:

- `ConvolutionalLayer` – Vytvoří konvoluční vrstvu. Základní parametry funkce jsou počet konvolučních filtrů, velikost filtrů a krok filtru (angl. *stride*).
- `MaxPoolingLayer` – Vytvoří podvzorkovací vrstvu s operací `MAX`, tj. z okénka je vybrán prvek s nejvyšší hodnotou, jak je popsáno v kapitole 2.2.1. K dispozici je také funkce `AveragePoolingLayer`, kde je z prvků okénka použitý průměr hodnot. Základní parametry funkcí jsou velikost okénka a krok posunutí.
- `DenseLayer` – Vytvoří plně propojenou vrstvu. Základní parametry funkce jsou vybraná aktivační funkce (ReLU, softmax, sigmo-

ida atd.) a počet neuronů vrstvy.

– `LinearLayer` – Vytvoří výstupní plně propojenou vrstvu. Počet neuronů vrstvy je stejný jako počet klasifikačních tříd.

Zbývající vrstvy a jejich dokumentaci lze nalézt na webových stránkách projektu CNTK[16].

Následující ukázka kódu pro definici modelu neobsahuje pro lepší přehlednost všechny parametry vrstev. První číslo u konvolučních vrstev udává počet filtrů, druhý parametr je velikost filtru a třetí velikost kroku, o který se filtr posouvá. U podvzorkovací vrstvy je první parametr velikost podvzorkovacího filtru a druhý je krok. U plně propojené vrstvy (`DenseLayer`) je první parametr počet neuronů a druhý zvolená aktivační funkce. Jednotlivé vrstvy jsou odděleny dvojtečkou.

```
model = Sequential (
  ConvolutionalLayer{96,(11:11),stride=(4:4)}:
  ReLU:
  MaxPoolingLayer{(3:3),stride=(2:2)}:
  ConvolutionalLayer{192,(5:5),pad=true}:
  ReLU:
  MaxPoolingLayer{(3:3),stride=(2:2)}:
  ConvolutionalLayer{384,(3:3),pad=true}:
  ReLU:
  ConvolutionalLayer{384,(3:3),pad=true}:
  ReLU:
  ConvolutionalLayer{256,(3:3),pad=true}:
  ReLU:
  MaxPoolingLayer{(3:3),stride=(2:2)}:
  DenseLayer{4096,activation=ReLU}:
  Dropout:
  DenseLayer{4096,activation=ReLU}:
  Dropout:
  LinearLayer{2}
)
```

Ukázka kódu 3.5: Příklad definice modelu.

Blok Test a Eval

Blok s akcemi `test` a `eval` slouží k testování/ohodnocení přesnosti modelu.

[16] `https://github.com/Microsoft/CNTK/wiki/BrainScript-Full-Function-Reference`

Obsahuje tyto parametry:

- `modelPath` – Cesta k modelu sítě, jenž byl vytvořen v bloku s akcí `train`. Model může být také definován přímo parametrem `BrainScriptNetworkBuilder` stejně jako u bloku s akcí `train`.
- `reader` – Tento blok slouží k načítání vstupních dat a bude detailněji popsán v kapitole 3.3.
- `minibatchSize` – Velikost dávky (počet testovacích vzorků, které prochází sítí).
- `epochSize` – Počet vzorků, po jejichž průchodu sítí se provedou speciální akce, viz parametr `epochSize` u bloku s akcí `train`.

3.3 Načítání snímků

K načítání vstupních dat slouží blok `reader`. Parametry bloku jsou:

- `readerType` – Určuje typ dat, která budou na vstupu. Pro vstup v podobě digitálních snímků má tento parametr hodnotu `ImageReader`. Další možné hodnoty jsou například `CNTKTextFormatReader` (textová data) nebo LUSequenceReader (pro zpracování přirozeného jazyka).
- `file` – Textový soubor, ve kterém jsou zapsány cesty ke vstupním snímkům. Řádky tohoto souboru musí mít následující formát:

```
1 cesta_k_souboru<tab>index_tridy
```

Ukázka kódu 3.6: Formát řádky souboru pro načtení snímku pomocí komponenty `ImageReader`.

  `cesta_k_souboru` je relativní nebo absolutní cesta vstupního snímku, `index_tridy` je index třídy, ke které daný snímek přísluší. Tyto hodnoty jsou odděleny tabulátorem. V názvech souborů není povolena diakritika.

- `labels` – Vnořený blok obsahující jeden parametr – `labelDim`, který udává počet klasifikačních tříd. Pro binární klasifikaci bude hodnota tohoto parametru 2.

- `features` – Vnořený blok obsahující základní informace o snímku, případně i informace pro umělé zvětšení sady dat. Příklad bloku:

```
features = {
  # šířka snímku
  width = 224
  # výška snímku
  height = 224
  # počet barevných kanálů
  channels = 3
    # transformace pro umělé zvětšení sady dat
    transforms = (
      {
        # provede náhodný výřez ze snímku
        type = "Crop"
        cropType = "RandomSide"
        sideRatio = 0.8
        # přidá do snímku chvění
        jitterType = "UniRatio"
      }:
      {
        # změna velikosti snímku a vyhlazení pomocí lineární interpolace
        type = "Scale"
        width = 224
        height = 224
        channels = 3
        interpolations = "linear"
      }:
      {
        # transpozice snímku z HWC na CHW
        # C je počet kanálů, H je výška snímku a W je šířka snímku
        type = "Transpose"
      }
    )
}
```

Ukázka kódu 3.7: Blok popisu vstupního snímku a transformací pro umělé zvětšení datové sady.

3.4 Příprava dat

Snímky nemusí být před vstupem do sítě nijak upravovány. Transformace, které budou aplikovány na snímek za účelem umělého zvětšování sady dat, jsou definovány v konfiguračním skriptu. Protože se v případě této práce

bude jednat o binární klasifikaci dat, byla data rozdělena do čtyř adresářů, které jsou:

- **train_target** – Adresář obsahující cílová (angl. *target*) data pro trénovací fázi.
- **train_nontarget** – Adresář obsahující necílová (angl. *non–target*) data pro trénovací fázi.
- **test_target** – Adresář obsahující cílová data pro testovací fázi.
- **test_nontarget** – Adresář obsahující necílová data pro testovací fázi.

Za cílová data jsou považovány snímky, které obsahují odhalená ženská prsa. Tato data mají klasifikační třídu 1, protože obsahují pornografický materiál. Necílová data jsou snímky, které žádnou pornografii neobsahují. Tato data mají klasifikační třídu 0.

Pro načtení snímků v bloku `reader` musí být vytvořen soubor, který obsahuje cesty ke snímkům na lokálním disku a jejich klasifikační třídu. Formát řádky souboru je popsán v kapitole 3.3. K vytvoření souboru slouží skript `ImagePreprocessing.py`, který je k dispozici na přiloženém CD. Skript je psaný v programovacím jazyce Python. Pro úspěšné spuštění je nutné mít nainstalovanou aktuální verzi interpretu jazyka Python. Příklad spuštění skriptu a popis jeho parametrů:

```
1 python ImagePreprocessing.py -t target -n non-target -m akce
```

Ukázka kódu 3.8: Příklad spuštění skriptu pro přípravu dat.

- `target` – Cesta k adresáři, který obsahuje snímky s pornografickým obsahem.
- `non-target` – Cesta k adresáři, který obsahuje snímky bez pornografie.
- `akce` – Tento parametr má pro pro trénovací fázi hodnotu *train* a pro testovací fázi hodnotu *test*.

Výstupní textový soubor může mít název například `train_map.txt`.

3.5 Trénování sítě

Trénování neuronových sítí probíhá experimentálně. Nalezení parametrů sítě, které by zajistily hodnotu chyby učení pod kýženou hranicí, je výsledkem mnoha různých testovacích scénářů. Vlastnosti neuronových sítí, které jsou během fáze trénování optimalizovány, jsou například:

- Počet vstupních dat – K natrénování neuronových sítí, které budou poskytovat uspokojivé výsledky, je potřeba velké množství vstupních dat. Pro představu lze uvést, že databáze MNIST, jež byla zmíněna již v kapitole 2.1.3, obsahuje 60 tisíc trénovacích vzorků a 10 tisíc testovacích.

  V závislosti na počtu vstupních dat může docházet k nedoučení nebo přeučení sítě. Příklad nedoučené sítě je uveden v kapitole 3.6.1. Příklad přeučené sítě je uveden v kapitole 3.6.2.

- Počáteční hodnoty vah – Na první pohled by se mohlo zdát, že iniciovat počáteční hodnotu všech vah na nulu je nejlepším řešením. Toto je však běžný omyl. Z rovnice (2.1) vyplývá, že pokud by byly váhy nulové, měly by všechny neurony skryté vrstvy nulový potenciál. Obecně lze říct, že pokud budou všechny váhy inicializovány na stejnou hodnotu, budou mít všechny neurony skryté vrstvy stejný potenciál. V takovém případě by učení sítě nekonvergovalo.

  Algoritmy pro učení neuronových sítí patří mezi hladové, tj. není zaručeno, že bude nalezeno optimální řešení. Proto je vhodné počáteční hodnotu vah volit náhodně v intervalu závislém na použité aktivační funkci. Například pro hyperbolickou tangentu je vhodný interval $(-1, 1)$, pro funkci softmax je vhodný interval $(0, 1)$. Nalezením optimálních hodnot vah se zabývají práce [3] a [13].

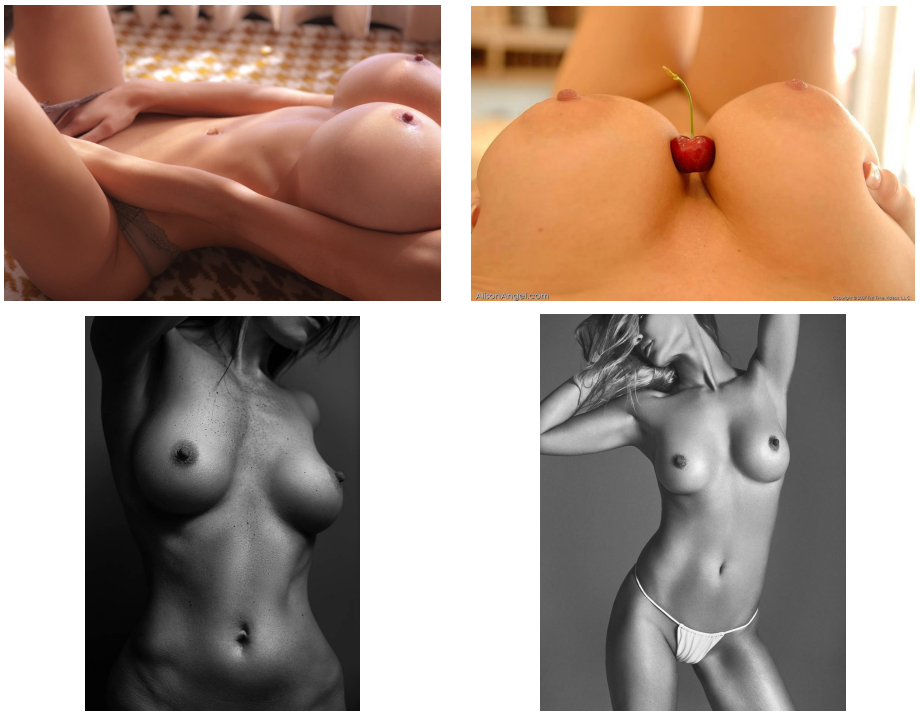
- Práh – Počáteční hodnotu prahů lze na rozdíl od počátečních vah zvolit nulovou. Další často používané hodnoty jsou *1*, *0,1* nebo *0,01*.

- Architektura sítě – Při experimentování s architekturou sítě zůstávají neměnné počty neuronů vstupní a výstupní vrstvy. Měnit lze například pořadí a počet skrytých vrstev, velikosti konvolučních filtrů, kroky filtrů nebo počet neuronů skrytých vrstev.

3.5.1 Sběr dat

Data pro trénování a testování neuronové sítě byla sbírána v průběhu celé práce. Cílová data, tedy data obsahující pornografii, byla stahována z různých internetových zdrojů a byla ručně zkontrolována, zda se jedná o dostatečně reprezentativní vzorky. Příklady cílových snímků zobrazuje tabulka 3.1.

Tabulka 3.1: Ukázky cílových snímků s pornografickým obsahem.



Necílová data pocházejí z internetového portálu *Flickr*[17], což je komunitní web pro sdílení fotografií. Tato data nebyla nijak tříděna a byla vybírána náhodně. Necílové snímky mohou obsahovat cokoliv kromě pornografického obsahu, na který jsou zaměřeny snímky cílové.

3.6 Výsledky

V této kapitole bude uvedeno několik různých použitých architektur ve formě tabulky, kde budou uvedeny vrstvy a jejich parametry seřazené od vstupní po výstupní vrstvu. Vysvětlivky k parametrům vrstev ilustruje tabulka 3.2.

[17] `https://www.flickr.com/`

Tabulka 3.2: Vysvětlivky k vrstvám sítě a jejich parametrům.

Vrstva	Parametry
Vstupní	velikost vstupního snímku
Konvoluční	velikost filtru – počet filtrů – krok filtru – hodnota vah – hodnota prahu – aktivační funkce
Podvzorkovací	velikost podvzorkovacího okénka – krok
Plně propojená	počet neuronů – hodnota vah – hodnota prahu
Výstupní	počet klasifikačních tříd

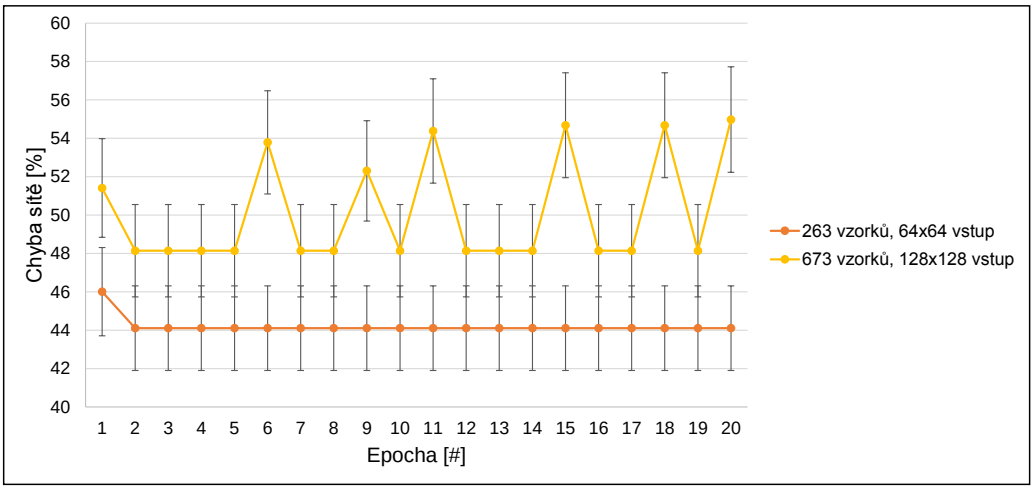
3.6.1 Nedoučená síť

Pokud došlo při trénování neuronové sítě k nedoučení, algoritmus učení nekonverguje. Toto je vidět z grafu na obr. 3.1. Na ose x grafu je uvedeno číslo epochy a na ose y je hodnota chyby v procentech. Architekturu sítě popisuje tabulka 3.3. Síť přijímá na vstupu snímky o velikosti 64×64 obrazových bodů a obsahuje dvě konvoluční, dvě podvzorkovací a jednu plně propojenou vrstvu. V prvním experimentu bylo použito 263 vstupních vzorků a velikost vstupních snímků byla 64×64 pixelů. V druhém experimentu bylo použito 673 vstupních vzorků a velikost vstupních snímků byla 128×128 pixelů.

Tabulka 3.3: Architektura nedoučené sítě.

Vrstva	Parametry
Vstupní	64×64 (128×128)
Konvoluční 1	5×5 – 16 – 1 – 0,0043 – 1 – ReLU
Podvzorkovací 1	2×2 – 2
Konvoluční 2	5×5 – 32 – 1 – 10 – 1 – ReLU
Podvzorkovací 2	2×2 – 2
Plně propojená	128 – 12 – 0
Výstupní	2

Z křivky chyby sítě je zřejmé, že jen zvětšení dimenze vstupní vrstvy a zvětšení trénovací sady dat, nepřineslo v tomto případě žádné podstatné zlepšení. Z toho vyplývá, že dále bude modifikována architektura sítě (přidání vrstev, jiné počáteční hodnoty vah a prahů).



Obrázek 3.1: Graf křivky hodnoty chyby nedoučené sítě.

3.6.2 Přeučená síť

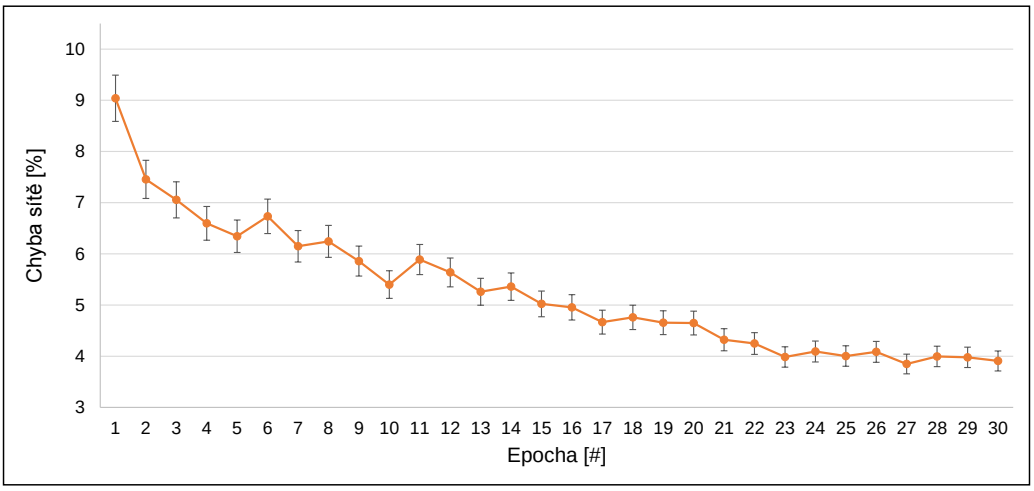
Pokud došlo k přeučení sítě, bude síť téměř bezchybně klasifikovat trénovací vzorky, ale bude selhávat při klasifikaci neznámých. K tomuto jevu může docházet například při dysbalanci datové množiny. V praxi to znamená, že počet trénovacích vzorků jedné třídy byl několikanásobně větší než počet vzorků druhé. Architekturu sítě popisuje tabulka 3.4.

Tabulka 3.4: Architektura přeučené sítě.

Vrstva	Parametry
Vstupní	224×224
Konvoluční 1	$11 \times 11 - 64 - 4 - 0{,}95 - 0 -$ ReLU
Podvzorkovací 1	$3 \times 3 - 2$
Konvoluční 2	$5 \times 5 - 192 - 1 - 2 - 1 -$ ReLU
Podvzorkovací 2	$3 \times 3 - 2$
Konvoluční 3	$3 \times 3 - 192 - 1 - 2.07 - 0 -$ ReLU
Konvoluční 4	$3 \times 3 - 384 - 1 - 2{,}9 - 1 -$ ReLU
Konvoluční 5	$3 \times 3 - 256 - 1 - 2{,}4 - 1 -$ ReLU
Plně propojená 1	$4096 - 6{,}4 - 1$
Plně propojená 2	$4096 - 3{,}2 - 1$
Výstupní	2

Trénovací množina dat obsahovala přibližně 2 000 cílových snímků a 25 000

necílových. Z grafu na obr. 3.2 je vidět, že křivka hodnoty chyby sítě klesá až na čtyři procenta. Tento výsledek byl velice slibný, ale síť ztratila schopnost generalizace.

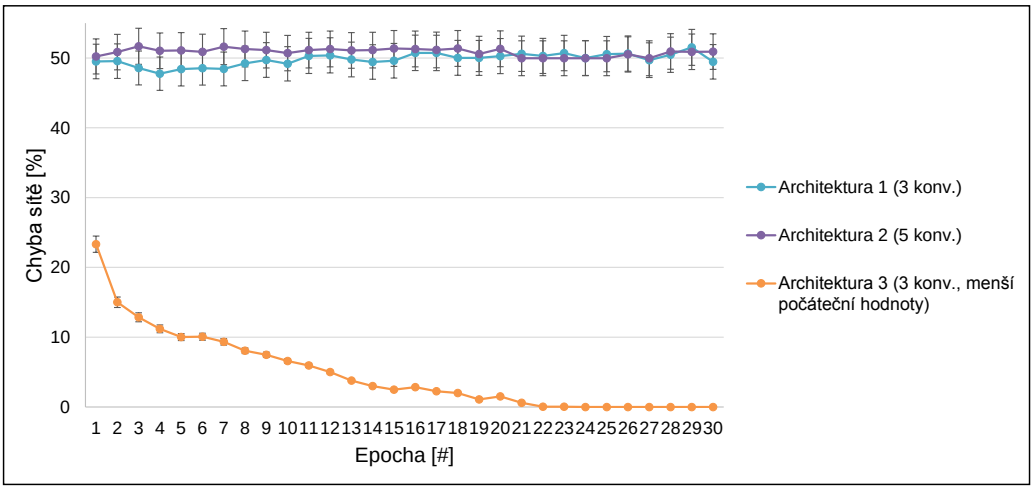


Obrázek 3.2: Graf křivky hodnoty chyby přeučené sítě.

3.6.3 Vliv počátečních hodnot vah a prahů

Malá změna počátečních hodnot parametrů sítě může vést k velice odlišnému průběhu učení. Následující příklad ilustruje změnu křivky hodnoty chyby sítě pro tři odlišné architektury sítě, které popisují tabulky 3.5, 3.6 a 3.7. Velikost vstupního snímku je vždy 224×224 pixelů a na výstupu jsou vždy dvě třídy. Architektury se liší počtem vrstev a počátečními hodnotami vah a prahů. Trénovací množina dat obsahovala 4199 cílových a 4204 necílových snímků.

Křivky učení pro jednotlivé architektury ilustruje graf na obr. 3.3. Z grafu je zřejmé, že algoritmus učení pro *architekturu 1* nekonverguje, protože se hodnota chyby stále pohybuje okolo 50%. Přidání dvou konvolučních vrstev v *architektuře 2* nepřineslo žádné výrazné zlepšení. To přineslo až zmenšení počátečních hodnot prahů a vah v *architektuře 3*. Z grafu je vidět, že algoritmus učení třetí architektury konverguje.



Obrázek 3.3: Graf křivky hodnoty chyby sítě pro různé hodnoty počátečních vah a prahů.

Tabulka 3.5: Architektura 1 – tři konvoluční vrstvy.

Vrstva	Parametry
Konvoluční 1	11×11 – 64 – 4 – 0,1 – 1 – ReLU
Podvzorkovací 1	3×3 – 2
Konvoluční 2	5×5 – 192 – 1 – 0,1 – 1 – ReLU
Podvzorkovací 2	3×3 – 2
Konvoluční 3	3×3 – 256 – 1 – 0,1 – 1 – ReLU
Podvzorkovací 3	3×3 – 2
Plně propojená 1	4096 – 0,01 – 0,1
Plně propojená 2	4096 – 0,01 – 0,1

Tabulka 3.6: Architektura 2 – pět konvolučních vrstev.

Vrstva	Parametry
Konvoluční 1	11×11 – 64 – 4 – 0,1 – 1 – ReLU
Podvzorkovací 1	3×3 – 2
Konvoluční 2	5×5 – 192 – 1 – 0,1 – 1 – ReLU
Podvzorkovací 2	3×3 – 2
Konvoluční 3	3×3 – 384 – 1 – 0,1 – 1 – ReLU
Konvoluční 4	3×3 – 256 – 1 – 0,1 – 1 – ReLU
Konvoluční 5	3×3 – 256 – 1 – 0,1 – 1 – ReLU
Podvzorkovací 3	3×3 – 2
Plně propojená 1	4096 – 0,01 – 0,1
Plně propojená 2	4096 – 0,01 – 0,1

Tabulka 3.7: Architektura 3 – tři konvoluční vrstvy, menší počáteční hodnoty parametrů.

Vrstva	Parametry
Konvoluční 1	11×11 – 64 – 4 – 0,01 – 0,1 – ReLU
Podvzorkovací 1	3×3 – 2
Konvoluční 2	5×5 – 192 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 2	3×3 – 2
Konvoluční 3	3×3 – 256 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 3	3×3 – 2
Plně propojená 1	4096 – 0,005 – 0,1
Plně propojená 2	4096 – 0,005 – 0,1

Všechny tři architektury byly testovány na množině o velikosti 1329 vzorků. Výsledky testovací fáze ilustruje tabulka 3.8.

Tabulka 3.8: Porovnání chyby sítě v testovací fázi při volbě odlišných vstupních parametrů pro 1329 testovacích vzorků.

Název	Chyba sítě v testovací fázi [%]
Architektura 1 – 3 konvoluční vrstvy	49.737
Architektura 2 – 5 konvolučních vrstev	49.737
Architektura 3 – menší počáteční hodnoty	8.954

3.6.4 Vliv velikosti trénovací množiny

Následující příklady ukazují křivky učení pro dvě architektury s různě velkými trénovacími množinami dat. Použité architektury popisují tabulky 3.9 a 3.10. Opět platí, že velikost vstupního snímku je 224×224 pixelů a výstupní vrstva má dva neurony. Architektury se liší počtem konvolučních vrstev a počtem konvolučních filtrů.

Tabulka 3.9: Architektura 1 – pět konvolučních vrstev.

Vrstva	Parametry
Konvoluční 1	11×11 – 96 – 4 – 0,01 – 0,1 – ReLU
Podvzorkovací 1	3×3 – 2
Konvoluční 2	5×5 – 192 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 2	3×3 – 2
Konvoluční 3	3×3 – 384 – 1 – 0,01 – 0,1 – ReLU
Konvoluční 4	3×3 – 384 – 1 – 0,01 – 0,1 – ReLU
Konvoluční 5	3×3 – 256 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 3	3×3 – 2
Plně propojená 1	4096 – 0,005 – 0,1
Plně propojená 2	4096 – 0,005 – 0,1

Tabulka 3.10: Architektura 2 – tři konvoluční vrstvy.

Vrstva	Parametry
Konvoluční 1	11×11 – 64 – 4 – 0,01 – 0,1 – ReLU
Podvzorkovací 1	3×3 – 2
Konvoluční 2	5×5 – 192 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 2	3×3 – 2
Konvoluční 5	3×3 – 256 – 1 – 0,01 – 0,1 – ReLU
Podvzorkovací 3	3×3 – 2
Plně propojená 1	4096 – 0,005 – 0,1
Plně propojená 2	4096 – 0,005 – 0,1

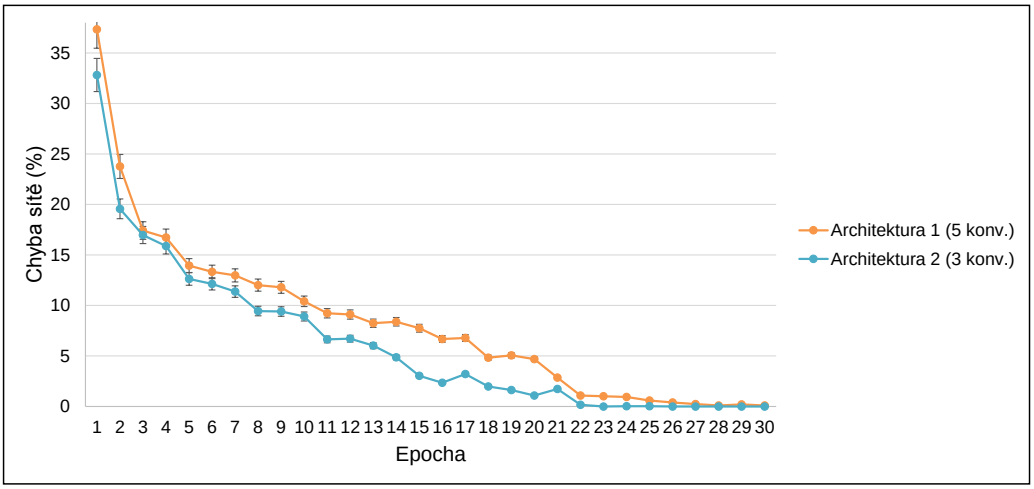
Grafy na obr. 3.4, obr. 3.5 a obr. 3.6 ilustrují křivku učení pro obě architektury vždy pro stejný počet trénovacích vzorků. Z grafů je vidět, že pro obě architektury křivky učení konvergují. Architektura s třemi konvolučními vrstvami konverguje dokonce rychleji. Srovnání architektur pro různě velké

trénovací sady dat obsahuje tabulka 3.11. V tabulce jsou uvedeny počty cílových a necílových vzorků. Počet viděných vzorků reprezentuje velikost datové množiny po jejím umělém zvětšení, viz kapitola 2.2.4.

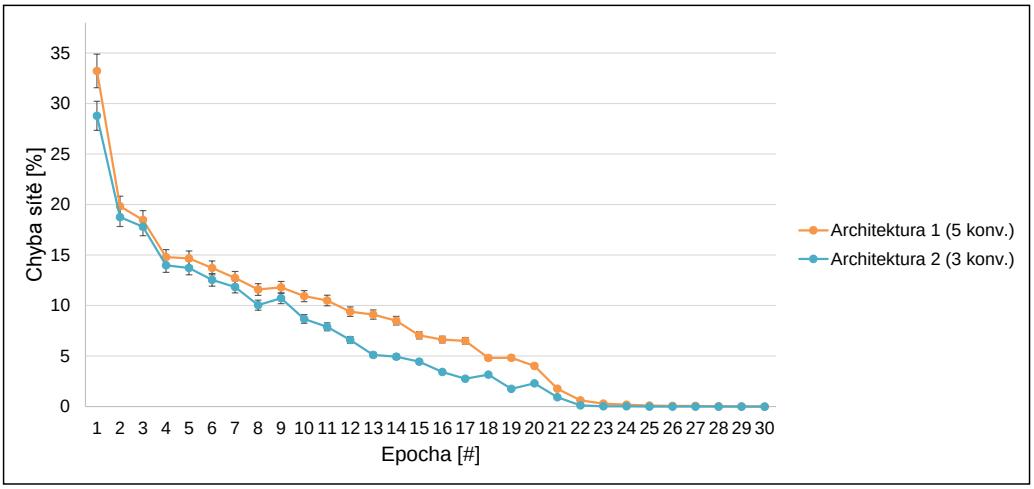
Tabulka 3.11: Srovnání použitých architektur pro různě velké sady trénovacích dat.

Architektura	Počet celkem	Počet cílových	Počet necílových	Počet viděných	Chyba v testovací fázi [%]
Architektura 1	4052	2036	2016	121560	8.202
Architektura 2					8.954
Architektura 1	6257	3108	3149	187710	3.085
Architektura 2					3.311
Architektura 1	8403	4199	4204	252090	3.085
Architektura 2					3.010

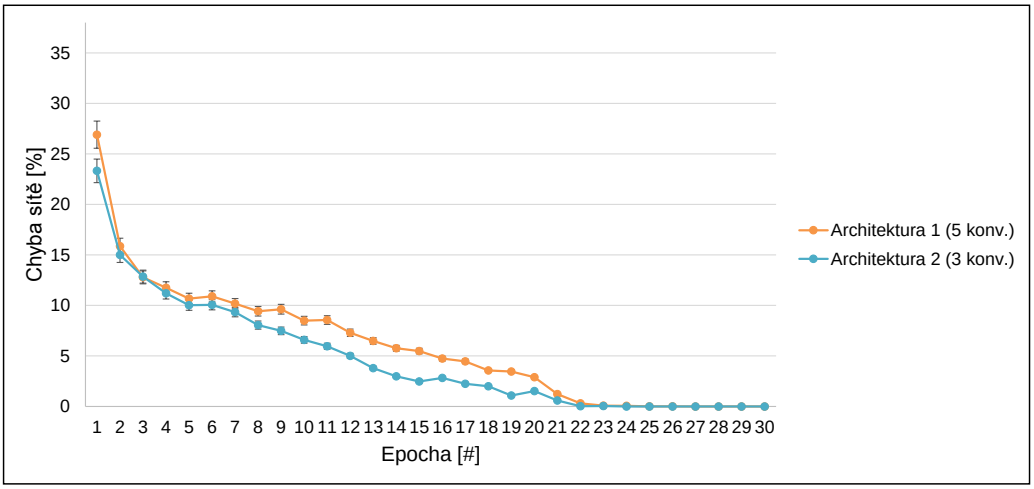
Testovací sada obsahuje vždy 1329 vzorků, z nichž je 661 cílových a 668 necílových. Při porovnání křivek učení v grafech a chyby v testovací fázi je vidět, že pro sady dat o velikosti 4052 a 6257 trénovacích vzorků je chyba testování pro architekturu s pěti konvolučními sítěmi menší, ačkoliv její učení konvergovalo pomaleji. Pro 8403 trénovacích vzorků vyšla chyba testování menší jen o několik setin pro architekturu se třemi konvolučními sítěmi.



Obrázek 3.4: Graf křivky hodnoty chyby sítě pro 4052 trénovacích vzorků.



Obrázek 3.5: Graf křivky hodnoty chyby sítě pro 6257 trénovacích vzorků.



Obrázek 3.6: Graf křivky hodnoty chyby sítě pro 8403 trénovacích vzorků.

3.7 Webové stránky

Součástí práce jsou i webové stránky, které byly vytvořeny za účelem testování modelu. Web je vytvořen ve stylu prezentace. Obsahuje jen jednu statickou stránku, která je rozdělena do logických sekcí podle obsahu. Pro oslovení většího počtu uživatelů jsou k dispozici dvě jazykové mutace webu – anglická a česká.

Server

Pro potřeby práce byl na katedrálním počítači s pracovním názvem *Blackhole*, umístěném v místnosti UC 361, zprovozněn HTTP server Apache, který nyní stránky poskytuje. Na stanici byl nainstalován následující software:

- *Apache 2.4.18* – Softwarový HTTP server. Výhodou je dostupnost pro všechny platformy (Windows, Unix) a otevřený zdrojový kód.
- *PHP 7.0.15* – Podpora pro skriptovací programovací jazyk PHP. Slouží pro programování dynamických webových stránek. PHP skripty jsou prováděny na straně serveru.
- *CNTK verze 2.0 beta 11*[18] – Sada nástrojů pro práci s konvolučními neuronovými sítěmi. Instalace proběhla pomocí již připraveného binárního souboru bez podpory GPU, neboť tato stanice nedisponuje grafickou kartou s čipem NVIDIA. Instalace CNTK a potřebné prerekvizity byly popsány v kapitole 3.1.1. Součástí binárního souboru byla i *Anaconda*, distribuce jazyka Python.

3.7.1 Struktura webových stránek

Soubory webu jsou rozděleny do adresářů podle obsahu. V kořenovém adresáři je umístěn soubor `index.php`. Ostatní soubory jsou rozděleny do následujících adresářů:

- **assets** – Obsahuje soubory, které upravují vzhled webu. Jedná se například o soubory s kaskádovými styly, fonty a soubory v jazyce JavaScript

[18] `https://github.com/Microsoft/CNTK/releases/tag/v2.0.beta11.0`

pro tvorbu interaktivních prvků. Adresář obsahuje i soubory projektu *Lightbox*[19] pro zobrazení obrázkové galerie.

- **cgi–bin** – Adresář pro model neuronové sítě a skript pro klasifikaci obrázku.
- **images** – Adresář pro obrázky. Obsahuje pozadí stránek, ikony a obrázky galerie.
- **lang** – Adresář pro soubory obsahující text, který je zobrazován na stránkách. Soubory jsou vytvořeny ve dvou jazykových mutacích.
- **log** – Adresář pro soubory s logy. Tyto soubory jsou ukládány po dnech. Obsahují například informace o výsledku klasifikace obrázku nebo validační chyby formulářů.
- **scripts** – Adresář obsahující skripty pro zpracování hodnot z formulářů. Jedná se například o skript pro kontaktní formulář, formulář pro zpětnou vazbu nebo skript pro logování validačních chyb.
- **upload** – Adresář pro uložení obrázků nahrávaných uživateli.
- **xml** – Adresář pro soubory ve formátu `xml` pro načítání nejčastěji pokládaných otázek (FAQ – angl. Frequently Asked Questions) a novinek. Soubory jsou vytvořeny ve dvou jazykových mutacích.

3.7.2 Testování modelu uživateli

Pro testování modelu konvoluční neuronové sítě je vytvořený jednoduchý formulář. Uživatel vybere přes pole pro vkládání souboru soubor ze svého lokálního disku a odešle ho ke zpracování na server. Pro zpracování obrázku slouží CGI skript `pdetect_linux.py` psaný v jazyce Python.

CGI

CGI (angl. Common Gateway Interface) je protokol pro propojení externích aplikací s webovým serverem. Klient pošle na server požadavek, server spustí skript nebo binární spustitelný soubor a výstup pošle zpět klientovi. CGI skripty mohou být psané například v jazycích Perl, Python, Bash nebo C.

[19] `http://lokeshdhakar.com/projects/lightbox2/`

Technika CGI je součástí serveru Apache. Nebylo tedy nutné přidávat žádný další modul. CGI skripty musí být uloženy ve zvláštním adresáři `cgi-bin`. V konfiguračním souboru lze ale nastavit direktivu *ScriptAlias*, kterou lze změnit výchozí umístění tohoto adresáře. CGI skripty musí mít tyto náležitosti:

- Cesta k interpretu – První řádek CGI skriptu musí být vždy cesta k interpretu. Při použití interpretu jazyka Python, má cesta tvar například:

```
1 #!/opt/anaconda3/envs/cntk-py35/bin/python
```

Ukázka kódu 3.9: Příklad cesty k interpretu jazyka Python.

V ukázce kódu 3.9 je uvedena cesta k interpretu Pythonu z distribuce *Anaconda*. Tato distribuce je součástí CNTK a byla uvedena v kapitole 3.1.1. Aby bylo možné správně volat funkce CNTK v CGI skriptu, je nutné, aby cesta odkazovala právě na tuto distribuci.

- HTTP hlavička – Říká, co je výsledkem skriptu. Výstupní data skriptu pro zpracování obrázku jsou typu *JSON*[20]. Znaky `\n\n` jsou součástí zápisu, protože HTTP protokol vyžaduje prázdný řádek mezi hlavičkou a tělem zprávy.

```
1 print("Content-type:application/json\n\n")
```

Ukázka kódu 3.10: Příklad HTTP hlavičky.

Zpracování vloženého obrázku

Jak již bylo řečeno, skript pro zpracování obrázku je psán v jazyce Python. CNTK poskytuje rozhraní pro Python, díky kterému lze snadno volat potřebné funkce. Výhodou Pythonu je snadné napojení na web. Skript je spuštěn na serveru po odeslání dat formuláře. Parametr formuláře `enctype` musí mít hodnotu `multipart/form-data`, aby bylo možné odesílat soubory.

```
1 <form action="/cgi-bin/pdetect_linux.py" method="POST"
2   enctype="multipart/form-data">
3   ...
4 </form>
```

Ukázka kódu 3.11: Parametry značky pro formulář s vkládáním souboru.

[20] `https://cs.wikipedia.org/wiki/JavaScript_Object_Notation`

Na serveru se k datům z formuláře přistupuje přes objekt `FieldStorage`. Úsek kódu pro přístup k datům a volání funkce pro klasifikaci:

```
# přístup k hodnotám odeslaných formulářem
form = cgi.FieldStorage()
# objekt nahraného obrázku
fileitem = form['inputImageUpload']
# uložení obrázku na server
full_path = save_image(fileitem)
# volání metody pro klasifikaci
top_class = cntk_prediction(full_path)
```

Ukázka kódu 3.12: Část CGI skriptu pro přístup k datům z formuláře a volání funkce pro klasifikaci.

Funkce pro klasifikaci dat využívá tři důležité moduly, které je nutné přiložit ke skriptu příkazem `import`. Tyto moduly jsou:

- `Image` z balíku `PIL` – Poskytuje třídu stejného jména, která reprezentuje objekt snímku.
- `numpy` – Knihovna pro práci s numerickými daty. Je vhodný pro manipulaci s vícerozměrnými poli.
- `cntk` – Knihovna poskytující rozhraní pro volání funkcí CNTK.

Klasifikovaný snímek musí splňovat několik náležitostí. Na vstupu mohou být pouze snímky, jejichž pixely mají 24–bitovou barevnou hloubku, kde na každou složku z RGB připadá 8 bitů. Pokud je snímek černobílý (mód `L`), musí být převeden na mód `RGB`[21]. Snímek musí mít velikost 224×224 pixelů. Na tuto velikost byly zmenšeny i snímky v trénovací a testovací fázi. Pokud klasifikovaný snímek nemá požadovanou velikost, bude jeho velikost změněna na 224×224 pixelů před klasifikací automaticky. Následující kód ilustruje klasifikaci snímku pomocí CNTK knihovny v jazyce Python.

```
# import modulů
from PIL import Image
import numpy as np
import cntk as ct
from cntk import load_model

def cntk_prediction(pathToImage):
```

[21] `http://pillow.readthedocs.io/en/3.4.x/handbook/concepts.html\#modes`

```
    # otevření souboru
    im = Image.open(pathToImage)
    # odstranění transparentní vrstvy
    im = remove_transparency(im)
    # převod na RGB mód
    im = RGBmode(im)
    # změna velikosti
    im = im.resize((224,224))

    # převod na pole a odečtení střední hodnoty
    rgb_image = np.asarray(im, dtype=np.float32 - 128
    bgr_image = rgb_image[..., [2,1,0]]
    # načtení pole do paměti
    pic = np.ascontiguousarray(np.rollaxis(bgr_image,2))

    # načtení modelu
    z = load_model("model.dnn")
    # načtení výstupních uzlů modelu
    z_out = ct.combine([z.outputs[3].owner])
    # aplikace funkce softmax
    y = ct.ops.softmax(z_out)
    # predikce příslušností ke třídám
    predictions = np.squeeze(y.eval({y.arguments[0]:[pic]}))
    # vrácení třídy s vyšší pravděpodobností
    top_class = np.argmax(predictions)
```

Ukázka kódu 3.13: Část CGI skriptu pro klasifikaci vstupního snímku.

Výsledek klasifikace, tj. index třídy, ke které snímek přísluší, se přidá za název souboru. Například soubor s názvem `obrazek.jpg`, který obsahuje pornografii, bude mít po klasifikaci název `obrazek_P1.jpg`. Písmeno *P* značí predikovanou třídu.

Výstupem CGI skriptu jsou data ve formátu *JSON* ve tvaru:

```
response = {
  'server_filepath': <cesta>, # cesta k obrázku na serveru
  'predicted_class': <trida>, # predikovaná třída
  'message': <zprava>}
```

Ukázka kódu 3.14: Výstup CGI skriptu.

Parametr `message` obsahuje zprávu, která bude uživateli zobrazena. Zpráva se týká výsledku klasifikace a informuje uživatele, zda jím nahraný obrázek obsahuje či neobsahuje pornografii. Společně s touto zprávou se zobrazí jednoduchý formulář, kde uživatel může odpovědět, zda je výsledek predikce správný nebo ne. Zpětná vazba se opět odrazí v názvu souboru. Například

kdyby soubor `obrazek.jpg` z výše uvedeného příkladu skutečně obsahoval pornografii a uživatel by toto potvrdil, měl by soubor konečný název `obrazek_P1_R1.jpg`. Písmeno R značí reálnou třídu.

Sdílení mezi uživateli

Webové stránky mají adresu `http://147.228.64.42/pdetect/`. Tento odkaz s popisem projektu byl sdílen prostřednictvím sociálních sítí, aby se o projektu dozvědělo co možná nejvíce lidí.

3.7.3 Výsledky uživatelského testování

V období od spuštění webových stránek 15.3.2017 do dne 20.4.2017 bylo celkem evidováno 473 nahraných souborů. Výsledky ilustruje tabulka 3.12. Správně byly klasifikovány vzorky, u kterých se predikovaná třída shoduje s třídou označenou uživatelem.

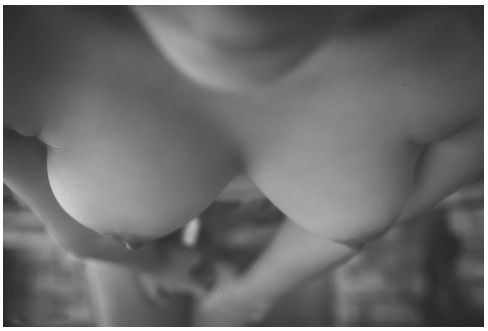
Tabulka 3.12: Výsledky uživatelského testování.

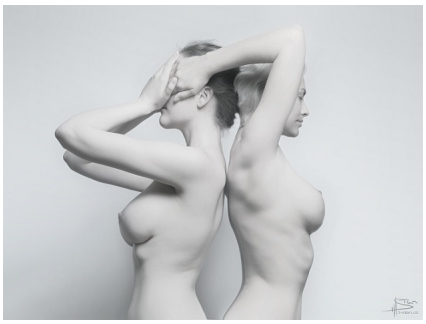
		Odpověď uživatele	
		1	0
Predikce	1	20	66
	0	15	372

Snímky, které obsahuje tabulka 3.13, byly uživatelem označeny, že obsahují pornografii, ale klasifikátor je vyhodnotil negativně. Důvodem, proč klasifikátor vyhodnotil tyto snímky negativně, může být například nedostatečné zastoupení podobných dat v trénovací množině. Pro spolehlivější výsledky by bylo vhodné zařadit do trénovací množiny cílových dat více černobílých snímků. Snímek vpravo nahoře se řadí mezi sporné, protože každý má subjektivní názor na to, co není a co už je pornografie.

Snímky, které obsahuje tabulka 3.14, byly uživatelem označeny jako neobsahující pornografii, ačkoliv odpověď klasifikátoru byla opačná. U prvního a druhého obrázku není vůbec zřejmé, proč byl označen za pornografii. U třetího obrázku, na kterém je žena s nahou figurínou, mohla být figurína sítí interpretována jako nahá osoba. Na čtvrtém obrázku je zase vysoké zastoupení pleťových odstínů. Do trénovací množiny pro necílová data by mělo být zahrnuto více snímků zobrazujících ženy v plavkách.

Tabulka 3.13: Ukázky falešně negativních snímků.



Tabulka 3.14: Ukázky falešně pozitivních snímků.




Problémy při testování

Problémy při testování se vyskytly dvojího druhu – technické problémy a problémy při pochopení účelu práce.

Technické problémy se objevovaly většinou, když si uživatel dostatečně nepřečetl, jak aplikace funguje. Výpočetní doba klasifikace se pohybuje kolem tří sekund. Uživateli se však zdálo, že se dlouhou dobu nic neděje a snažil se odeslat obrázek několikrát, což způsobilo chybový stav a aplikace se jevila nefunkční. Toto bylo vyřešeno přidáním komponenty *waitbar*, která uživateli indikuje, že je jeho požadavek zpracováván.

Někteří uživatelé si sice informace přečetli, ale při zmínce o pornografii raději na odkaz neklikali z obavy, že budou vystaveni nevhodnému obsahu. V takovém případě nezbývá nic jiného, než vysvětlit účel práce uživatelům individuálně, pokud mají zájem.

4 Závěr

Cílem práce bylo seznámit se s metodami pro detekci objektů v digitálních snímcích, především s technikou konvolučních neuronových sítí, a navrhnout a implementovat klasifikační systém, který bude spolehlivě identifikovat pornografické scény.

V teoretické části jsem vysvětlila základní problematiku umělých neuronů a neuronových sítí, jejich strukturu a princip činnosti. Následně jsem popsala konvoluční neuronové sítě, čím se odlišují od obyčejných neuronových sítí, jednotlivé vrstvy a princip jejich učení. Pro implementaci sítě jsem po pečlivém zvážení z několika možných alternativ vybrala framework CNTK od společnosti Microsoft.

V praktické části popisuji, jak práce vznikala. Vytvořila jsem datové sady pro cílové a necílové digitální snímky. Množinu cílových snímků obsahujících pornografii jsem ručně vybrala a zkontrolovala, zda se jedná o dostatečně reprezentativní vzorky vhodné k natrénování sítě. Pro nasbíraná data jsem navrhla a implementovala několik různých architektur konvolučních neuronových sítí, které jsou detailněji popsány v kapitole 3.6. Vytvořené neuronové sítě jsem vyhodnotila a řádně zdokumentovala dosažené výsledky. Nejlepší výsledky měla konvoluční síť s třemi konvolučními vrstvami natrénovaná na přibližně osmi tisících snímcích, kde přibližně polovina byly cílové snímky a polovina necílové. V testovací fázi vykazovala chybu jen 3,010%. Testovací množina obsahovala 1329 vzorků.

Součástí práce bylo také vytvoření webových stránek za účelem uživatelského testování modelu neuronové sítě. U webových stránek byl kladen důraz na jednoduchost pro rychlou orientaci uživatele. Pro oslovení většího počtu uživatelů byly vytvořeny dvě jazykové mutace – anglická a česká. Stránky poskytuje HTTP server Apache, který je zprovozněn na katedrálním počítači. Adresa stránek je `http://147.228.64.42/pdetect/`. Uživatel nahraje vybraný snímek z lokálního disku a pomocí formuláře ho odešle ke zpracování na server. Pro testování funkčnosti modelu jsem vytvořila CGI skript, který provede jednoduché předzpracování nahraného snímku (změna rozměrů, odstranění transparentní vrstvy) a detekuje, zda snímek obsahuje či neobsahuje pornografii. Výsledek klasifikace je odeslán zpět uživateli.

Literatura

[1] Jake Bouvrie. Notes on convolutional neural networks. Technická zpráva, listopad 2006.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Philipp Krähenbühl, Carl Doersch, Jeff Donahue, Trevor Darrell. Data-dependent initializations of convolutional neural networks. *CoRR*, abs/1511.06856, 2015.

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[5] Laurene Fausett. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice–Hall, 1994.

[6] Matthew D. Zeiler, Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[7] Matthew D. Zeiler, Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[8] Yann Lecun, Léon Bottou, Yoshua Bengio, Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[9] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] Ujjwal Karn. An intuitive explanation of convolutional neural networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, 2016. [Online; citováno 3.4.2017].

[11] Andrej Karpathy. Convolutional neural networks for visual recognition. http://cs231n.github.io/, 2016. [Online; citováno 3.4.2017].

[12] Davis E. King. Dlib-ml: A machine learning toolkit. *J. Mach. Learn. Res.*, 10:1755–1758, December 2009.

[13] Dmytro Mishkin, Jiri Matas. All you need is a good init. *CoRR*, abs/1511.06422, 2015.

[14] Iveta Mrázová. Neuronové sítě. Prezentace k přednášce. Karlova univerzita, Matematicko-fyzikální fakulta, 2014.

[15] Michael Nielsen. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com/, 2017. [Online; citováno 5.4.2017].

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[17] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[19] Daniel Shiffman. The nature of code. http://natureofcode.com/book/chapter-10-neural-networks/, 2012. [Online; citováno 8.4.2017].

[20] Sargur Shihari. Deep learning. http://www.cedar.buffalo.edu/~srihari/CSE676/index.html, 2016. [Online; citováno 5.4.2017].

[21] Christos Stergiou, Dimitrios Siganos. Neural networks. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html. [Online; citováno 8.4.2017].

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[23] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, Sameep Tandon. Deep learning tutorial. `http://deeplearning.stanford.edu/tutorial/`. [Online; citováno 3.4.2017].

[24] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[25] Eva Volná. Neuronové sítě 1. Učební texty. Ostravská univerzita, Přírodovědná fakulta, 2002.

[26] Wikipedia. Artificial neuron — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Artificial_neuron`, 2017. [Online; citováno 8.4.2017].

[27] Wikipedia. Convolutional neural network — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Artificial_neuron`, 2017. [Online; citováno 8.4.2017].

[28] Wikipedia. Konvoluce — Wikipedia, the free encyclopedia. `https://cs.wikipedia.org/wiki/Konvoluce`, 2017. [Online; citováno 7.4.2017].

[29] Daniel E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[30] Karen Simonyan, Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[31] Amit Agarwal, Eldar Akchurin, Chris Basoglu, Guoguo Chen, Scott Cyphers, Jasha Droppo, Adam Eversole, Brian Guenter, Mark Hillebrand, T. Ryan Hoens, Xuedong Huang, Zhiheng Huang, Vladimir Ivanov, Alexey Kamenev, Philipp Kranen, Oleksii Kuchaiev, Wolfgang Manousek, Avner May, Bhaskar Mitra, Olivier Nano, Gaizka Navarro, Alexey Orlov, Hari Parthasarathi, Baolin Peng, Marko Radmilac, Alexey Reznichenko, Frank Seide, Michael L. Seltzer, Malcolm Slaney, Andreas Stolcke, Huaming Wang, Yongqiang Wang, Kaisheng Yao, Dong Yu, Yu Zhang, Geoffrey Zweig. An introduction to computational networks and

the computational network toolkit. *Microsoft Technical Report, Num. MSR–TR–2014–112*, 2014.

Seznam zkratek

CGI ..... Common Gateway Interface

CNN ..... Convolutional Neural Networks

CNTK ..... The Microsoft Cognitive Toolkit

CUDA ..... Compute Unified Device Architecture

FAQ ..... Frequently Asked Questions

GPU ..... Graphics Processing Unit

ILSVRC ..... ImageNet Large Scale Visual Recognition Challenge

MKL ..... Intel Math Kernel Library

MPI ..... Message Passing Interface

OS ..... Operační Systém

SGD ..... Stochastic Gradient Descent

SVM ..... Support Vector Machines

ReLU ..... Rectified Linear Unit

Seznam obrázků

Seznam tabulek

Seznam ukázek kódu