Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

PLZEŇ, 2017 JAN HRDLIČKA

Místo této strany bude
zadání práce.

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni 5. dubna 2017

.....

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce Ing. Miroslavu Hlaváčovi za pomoc při řešení této práce.

Zároveň děkuji organizaci MetaCentrum za přístup k výpočetním a úložným zařízením, které patří stranám a projektům přispívajícím k National Grid Infrastructure MetaCentrum poskytovanou programem "Projects of Large Infrastructure for Research, Development, and Innovations"(2015042).

Anotace

Práce se zabývá detekcí a rozpoznáváním pokerových karet pomocí neuronových sítí. V práci je popsán postup získání karty z videozáznamu, metoda srovnávání se vzorem, příprava dat pro neuronovou síť a její následné trénování pomocí modulu Caffe. Na konci práce dochází k vyhodnocení přesnosti klasifikace karet pomocí metody srovnávání se vzorem a neuronové sítě.

Klíčová slova: poker, karty, rozpoznávání, Python, srovnávání se vzorem, neuronová síť, Caffe

Annotation

This thesis deals with detection and recognition of poker cards using neural networks. In this thesis is described process of acquisitioning card from video-sequence, Template matching method, data preparation for neural network and its training with Caffe module. At the end of the thesis, there is a comparison of the classification accuracy between Template matching and neural network.

Key words: poker, cards, recognition, Python, Template matching, neural network, Caffe

Obsah

Seznam obrázků

Kapitola 1

Úvod

Počítačové vidění je věda, která se snaží automatizovat vizuální procesy. Snahou je naučit počítač, aby z nějakého snímku dokázal získat informace - například jaké objekty se na snímku nachází, jaké mají vztahy k ostatním objektům na snímku (je v pozadí, v popředí, nalevo), výšku a šířku objektů, barvu, atd. Díky těmto informacím pak může počítač v některých případech nahradit člověka, rozhodovat za něj a ušetřit ho tak od většinou rutinní práce. Počítačové vidění se však velice rozšiřuje i například do medicíny, kde díky počítači může dojít k záchraně životů.

Počítačové vidění má počátky v 70. letech minulého století, jde tedy o vcelku nový vědní obor. Do té doby neměly počítače dostatečný výpočetní výkon na práci s obrazem a zároveň nebyla rozšířena zařízení, která by tento obraz uchovávala v digitální podobě. Přibližně v tu samou dobu se začínají objevovat první neuronové sítě, zpočátku velice jednoduché. Jelikož se stále zvyšuje výpočetní výkon zařízení, zvyšuje se i složitost těchto sítí, což vede k lepším výsledkům. Pro představu tohoto pokroku, v roce 1957 byla vynalezena jednovrstvá neuronová síť (také perceptron), která je schopna naučit se lineární diskriminační funkci. Na druhé straně síť ResNet od firmy Microsoft z roku 2015 má 152 vrstev a dokáže správně klasifikovat obrázky o velikosti 32×32 pixelů s $3,6\%$ chybovostí, chybovost člověka se pohybuje mezi $5 - 10\%$ [27].

Účelem této práce je pomocí počítače automaticky rozpoznávat pokerové karty z videozáznamu, tedy jak hodnotu (eso, dvojka, ..., král), tak barvu (srdce, káry, kříže, piky). Pokerových karet je 52, 13 karet pro každou barvu. Vzhledem k operacím, které jsou na kartách v práci prováděny, se karty na snímcích nesmí překrývat a musí být na segmentovatelném pozadí. Videozáznamy jsou pořízeny z různých zařízení a ze dvou setů karet. Práce bude řešena v programovacím jazyce Python za použití knihovny OpenCV. Nejprve dojde k vytvoření databáze všech karet pomocí segmentací, rotací a transformací. Z těchto karet se vyberou vhodné vzory pro metodu srovnávání se vzorem, která roztřídí karty podle jejich hodnoty a barvy. Poté se vytvoří soubory h5, pomocí kterých bude na serverech Metacentra trénována neuronová síť. Architektura sítě bude sestavena pomocí modulu Caffe. Očekává se, že přesnost klasifikace natrénované neuronové sítě bude vyšší než u metody srovnávání se vzorem.

Kapitola 2

Metody zpracování obrazu a dat

Tato kapitola se bude zabývat teoretickým rozborem metod, které byly použity pro přípravu dat k trénování a testování neuronové sítě. Nejprve zde bude ukázáno, jak je obraz v počítači reprezentován, poté dojde k představení základních operací s obrazem, metody srovnávání se vzorem a nakonec formátu HDF, který bude sloužit jako vstupní soubor pro trénování sítě.

2.1 Obraz v počítači

Obraz je v počítači reprezentován jako pole o třech dimenzích. Toto pole nese informaci o barevných vlastnostech jednotlivých pixelů - bodů v obraze. Pole, které reprezentuje obraz tedy může vypadat například takto:

$$250 \times 300 \times 3.$$

V tomto případě má obrázek šířku 250 pixelů, výšku 300 pixelů a má tři barevné kanály. Počet kanálů je určen typem barevného modelu. Většina modelů má kanály tři. Nejčastěji používaným barevným modelem je model RGB (červená, zelená a modrá). Každá barva má tedy 250×300 hodnot od nuly do 255, které označují intenzitu jednotlivé barvy tohoto bodu, nula nejméně intenzivní, 255 nejvíce.

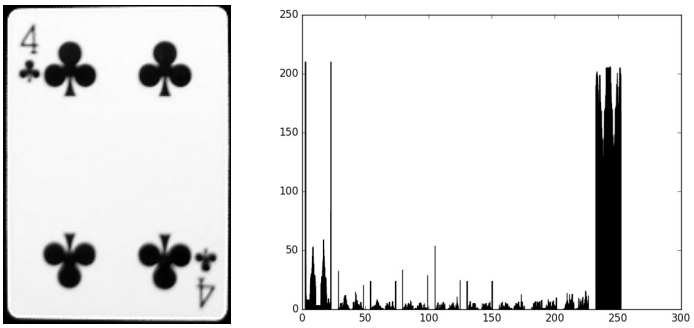
Často se také používá obraz, který je popsán pouze odstíny šedé barvy. Místo třech barevných kanálů je tento obraz reprezentován pouze jedním, který označuje intenzitu šedé, nula pro černou a 255 pro bílou [15]. Nejčastěji používanou metodou pro převod obrazu z RGB do šedotónu je metoda známá pod anglickým názvem luminosity method: [14]:

$$Y = 0,21R + 0,72G + 0,07B, \tag{2.1}$$

kde Y je šedotónový pixel a R, G, B pixely pro dané barevné kanály. Pole, reprezentující šedotónový obraz, vypadá analogicky takto:

$$250 \times 300.$$

Odstíny šedi se pro zpřehlednění a zjednodušení práce zanáší do histogramu. Histogram je graf znázorňující počet pixelů v obrazu pro každou intenzitu. Horizontální osa histogramu reprezentuje intenzity, které se nacházejí v obrazu. Vertikální osa reprezentuje počet pixelů dané intenzity [20]. Z obrázku (2.1) lze vidět, že se v obrazu nachází velký počet černých pixelů:



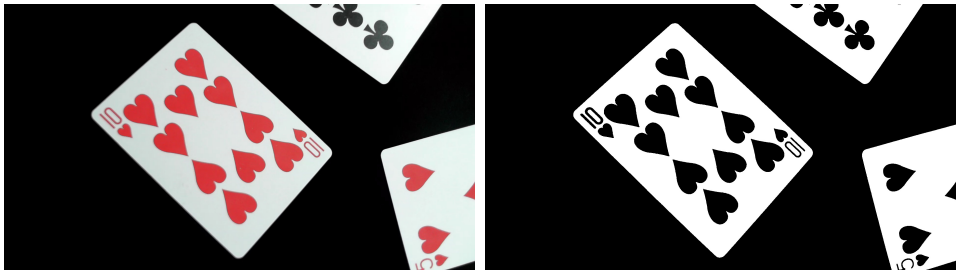
Obrázek 2.1: Obraz a jeho histogram

2.2 Segmentace obrazu

Segmentace se používá k rozdělení obrazu na více oblastí s podobnými vlastnostmi. Tento proces se používá pro zjednodušení identifikace potřebných objektů v obrazu. Tato metoda má dvě možnosti přístupu - nekontextová nebo kontextová segmentace.

Nejjednodušší metodou nekontextové segmentace je **thresholding - prahování**: Princip prahování spočívá v nastavení prahu. Pokud je hodnota pixelu vyšší než zvolený práh, je tato hodnota přepsána na jedničku (nebo také True), pokud ne, je přepsána na nulu (False). Obraz je tedy nyní reprezentován polem nul a jedniček, známý také jako binární obraz. Pro získání správných výsledků je tedy třeba zvolit správný práh. Práh se volí v oblasti histogramu, kde jsou nejmenší četnosti, aby se podobné oblasti oddělily [8].

Problém správného zvolení prahu řeší **adaptivní segmentace:** Zatímco u klasického prahování je nastaven stejný práh pro celý obraz, u adaptivní segmentace se tento práh určuje spočítáním buď středních hodnot, nebo mediánu v dané oblasti obrazu. Tato metoda může také vyřešit problém nerovnoměrného osvětlení v obrazu [31].



Obrázek 2.2: Binární segmentace obrazu

2.3 Metoda srovnávání se vzorem

Metoda srovnávání se vzorem využívá vzorového obrazu, pomocí kterého se snaží v obrazu najít oblasti, které jsou tomuto vzoru podobné. Vzor se v obrazu posouvá pixel po pixelu a hledá se nejvyšší podobnost s obrazem. Míru podobnosti lze získat pomocí následujících vzorců:

suma čtvercových rozdílů

$$\mathbf{R}(x,y) = \sum_{x'y'}(\mathbf{T}(x'y') - \mathbf{I}(x+x', y+y'))^2, \tag{2.2}$$

křížová korelace

$$\mathbf{R}(x,y) = \sum_{x'y'}(\mathbf{T}(x'y') - \mathbf{I}(x+x', y+y')), \tag{2.3}$$

korelační koeficient

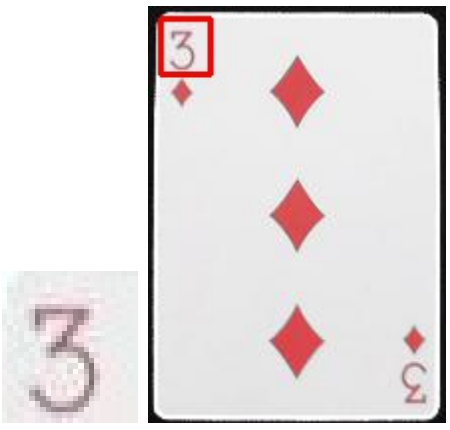
$$\mathbf{R}(x,y) = \sum_{x'y'}(\mathbf{T}'(x'y') - \mathbf{I}(x+x', y+y')), \tag{2.4}$$

přičemž

$$\mathbf{T}'(x',y') = \mathbf{T}(x',y') - 1/(w \cdot h) \cdot \sum_{x''y''}(\mathbf{T}(x''y''), \tag{2.5}$$

kde $\mathbf{T}$ je vzorový obraz a $\mathbf{I}$ testovaný obraz, $\mathbf{R}(x,y)$ je matice výsledků pro danou oblast x, y, w je šířka obrazu a h je výška obrazu. Často se tyto metody pro jednodušší práci s výsledky normalizují [6].

Tato metoda je velice rozšířená pro detekci objektů v obrazu pro svoji jednoduchost. V reálném případě však nedojde k naprosté shodě vzoru s oblastí v obrazu kvůli šumům. Problém také může nastat, pokud se oblast podobná vzoru v obrazu nachází, ale je jinak natočená, proto se buď vzorem při posouvání pixelů otáčí, nebo se použije metoda Feature matching [3], která srovnává společné vlastnosti, nikoliv pixely. Na obrázku níže lze vidět, že po zvolení čísla tři jako vzoru dojde k nalezení tohoto vzoru v levém horním rohu karty, otočené číslo metoda srovnávání se vzorem nedetekuje.



Obrázek 2.3: Nalezení vzoru v obrazu

2.4 Hierarchical Data Format

Hierarchical Data Format je set souborových formátů (file format) určený k ukládání a organizaci velkého množství dat. Jedním z hlavních důvodů proč používat HDF, je rozšíření do mnoha programovacích jazyků a schopnost uchovat velké množství dat v jednom souboru. Současná verze HDF5 zjednodušuje strukturu souboru do dvou hlavních typů objektu:

- Datasety - mnohodimenzionální pole stejného datového typu, povolené datové typy:
  - integer a float
  - string
  - pointer
  - enumerate
- Skupiny - kontejnerové struktury, které mohou obsahovat datasety a další skupiny

Pomocí těchto objektů lze specifikovat relace - hierarchičnost mezi daty. Tento HDF soubor není omezen celkovou velikostí, počtem ani velikostí objektů v souboru [9]. K prohlížení a editaci HDF souborů byl vytvořen program HDFView.

HDF soubory se používají jako vstupní soubory při trénování a testování neuronových sítí hlavně z důvodu minimalizace paměťových a výkonnostních nároků.

2.5 OpenCV

OpenCV je otevřená knihovna, vyvíjená společností Intel, určena zejména pro práci s obrazem v reálném čase. Knihovna je napsána v C++ a běží na mnoha platformách. U osobních počítačů jsou to Windows, Linux, macOS, FreeBSD, NetBSD, OpenBSD. OpenCV lze zároveň používat v telefonech, momentálně knihovna běží na operačních systémech Android, iOS, Maemo či BlackBerry.

OpenCV lze použít v programovacích jazycích C, C++, Python, Java nebo Matlab. Knihovna je optimalizována pro Intel procesory [5]. Zejména díky své obsáhlosti a rozšířenosti napříč mnoha platformami je tato knihovna nejpoužívanější knihovnou pro práci s obrazem v Pythonu. Na stránkách OpenCV je rovněž velice podrobná dokumentace s názornými příklady a tutoriály.

2.6 NumPy

Datové struktury programovacího jazyka Python nejsou příliš vhodné pro složitější numerické operace. Z tohoto důvodu se využívá balíku NumPy, který obsahuje strukturu N-dimenzionálního pole. Největší výhodou je, že na rozdíl od matic může mít toto pole jakoukoliv dimenzi. Zároveň přístup k jednotlivým prvkům pole je mnohem jednodušší. V balíku jsou také obsaženy funkce lineární algebry, Fourierova transformace nebo funkce pro práci s náhodnými čísly [30].

Numpy patří pod otevřenou knihovnu Scipy, která je určena pro vědeckou práci v Pythonu. Scipy obsahuje ještě následující balíky:

- SciPy - zpracování signálu, shlukování, složitější datové struktury
- Matplotlib - obohacené 2D grafy
- IPython - interaktivní prostředí pro Python
- Sympy - symbolická matematika
- pandas - datové struktury a analýza

Knihovna Scipy [13] je jednou z nejpoužívanějších knihoven v Pythonu zejména díky balíku NumPy.

V této práci je ze Scipy rovněž využíván balík Matplotlib pro vykreslení obrazu do grafu a prostředí IPython, konkrétně webová aplikace Jupyter Notebook, která je velice vhodná právě pro práci s obrazem.

2.7 Scikit-image

Tato knihovna, známá také jako skimage, je další otevřenou knihovnou pro práci s obrazem v Pythonu. Knihovna obsahuje např. algoritmy pro segmentaci obrazu, geometrické transformace, filtrování, hranovou detekci, atd. Scikit-image je naprogramováno v Pythonu a je navrženo tak, aby se dalo provázat s knihovnou Scipy [29].

Na rozdíl od OpenCV, které je zaměřeno spíše na počítačové vidění, je tato knihovna používána zejména pro zpracování obrazu. Oproti OpenCV není Scikit-image tak rozšířené, avšak obsahuje mnoho užitečných nástrojů pro úpravu obrazu, které OpenCV neobsahuje.

Kapitola 3

Neuronové sítě

V této kapitole bude v úvodu ukázáno, jak se neuronové sítě liší od klasického programu a kde se vyskytují. V další podkapitole bude vysvětleno, co je to neuron a aktivační funkce. Na modelu perceptronu bude poté vysvětlen princip fungování neuronových sítí. Závěr kapitoly bude o architekturách neuronových sítí, kde dojde k podrobnějšímu popisu konvolučních neuronových sítí.

3.1 Úvod

Neuronové sítě jsou dnes jedním z nejdůležitějších prvků umělé inteligence. V případě klasického programování počítači kódem krok po kroku říkáme, jak má danou úlohu řešit. Můžeme tedy vyřešit jen problémy, kterým rozumíme a víme, jak je vyřešit. Kdežto u neuronových sítí často problému nerozumíme a ani nevíme jak ho řešit a síť na řešení přijde sama. Podobně jako lidé se tyto sítě učí pomocí příkladů a mají schopnost zobecňovat problémy, tzn. pokud jsou síti předložena data, na která síť nebyla konkrétně naučena, může síť přesto zařadit tato data správně. Jak už bylo zmíněno v úvodu této práce, inteligence těchto sítí se rapidně zvyšuje a v některých konkrétních případech dokáže člověka i předčit, musí být však správně natrénovaná. Kvůli vysokým požadavkům na výkon a časové náročnosti zpracování obrovského množství dat je tyto sítě nutno trénovat na výkonnějších zařízeních, než je osobní počítač.

Zásadním pro vývoj těchto sítí byl rok 2006, kdy byly vynalezeny tzv. hluboké neuronové sítě (deep neural networks) [21], které umožnily velice přesné řešení mnoha komplexních problémů. Tyto sítě jsou v širokém měřítku používány firmami jako Microsoft, Google či Facebook a jsou obsaženy v téměř každém chytrém telefonu jako hlasové vyhledávání (Cortana pro zařízení s operačním systémem Windows, OK Google pro Android či Siri pro Apple). Neuronové sítě však nejsou určeny jen pro oblasti řeči a obrazu, používají se například i v následujících odvětvích:

- Finance - předpověď chování akcií, bankrotu, cen...
- Lékařství - určení diagnóz pacientů, odhad nákladů na léčbu, odhad délky

trvání nemoci...

- Personalistika - vybírání a najímání zaměstnanců na základě CV, plánování událostí, profilace zaměstnanců...
- Průmysl - předpověď kvality výrobku, teploty stroje, závady na stroji...
- Ostatní - sportovní sázky, předpověď počasí, umělá inteligence ve hrách, odhad výnosu sklizně...

3.2 Neuron

Biologický neuron je buňka, která je hlavní součástí centrální nervové soustavy a zároveň jedna ze dvou buněk, která se podílí na stavbě mozku. Každý neuron má své tělo, z kterého vedou dendrity, pomocí kterých neuron přijímá informace ve formě elektrického signálu a axon, který slouží k vysílání informací. Axon je na konci rozvětven a připojuje se pomocí synapsí na dendrity ostatních neuronů. Pokud neuron nahromadí do svého těla dostatek elektrického signálu z dendritů, vyšle signál do axonu [15].

Uměle vytvořený neuron funguje obdobně, signál z axonu lze označit jako x_0 a sílu synapse jako W_0, pro další dendrity se bude postupovat analogicky(pro i-tý dendrit x_i a W_i). V těle neuronu x dochází k sumaci všech vstupních signálů:

$$x = \sum_i x_i W_i + b, \tag{3.1}$$

přičemž b je práh, při jehož překročení vyšle neuron signál do axonu. Po této sumě následuje aktivační funkce f a signál je poslán na výstup y:

$$y = f(x) = f(\sum_i x_i W_i + b). \tag{3.2}$$

3.2.1 Aktivační funkce

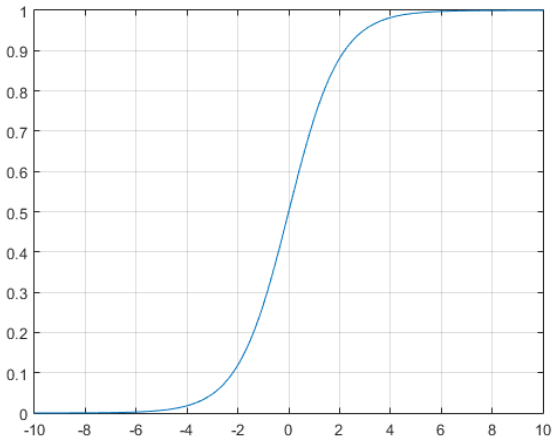
V umělých neuronových sítích se nacházejí aktivační funkce. Tyto funkce, známé také jako funkce přenosové, biologicky reprezentují míru akčního potenciálu vyslání signálu z uzlu. Aktivační funkce mohou být lineární nebo nelineární. Lineární aktivační funkce se používají při řešení velice specifických případů, například pro klasifikaci lineárně oddělitelných obrazů. Díky nelineárním funkcím se neuronové sítě mohou naučit řešit složitější problémy.

Mezi nejrozšířenější nelineární aktivační funkce patří tyto:

Logistická funkce

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{3.3}$$

Tato funkce byla v historii často používána, dnes už je využívána jen vzácně. Nevýhodou této funkce je, že při vysokých hodnotách se výstup blíží jedné a při porovnání dvou odlišných vysokých hodnot bude gradient velmi malý.

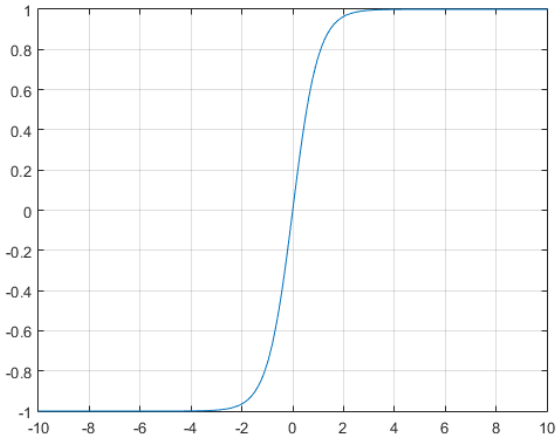


Obrázek 3.1: Logistická funkce

Hyperbolický tangens

$$f(x) = \tanh x. \tag{3.4}$$

Funkce hyperbolického tangensu je podobná logistické funkci, avšak má hodnoty více soustředěné k nule, což je výhodou.

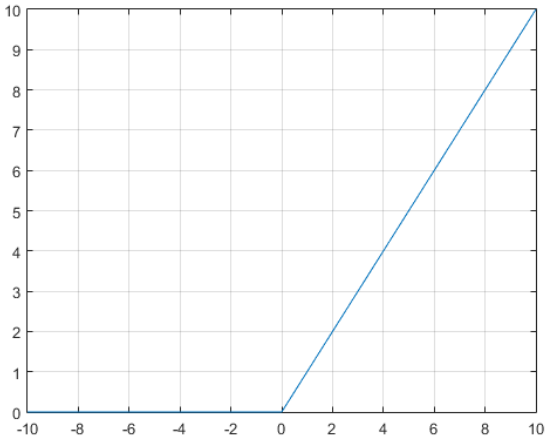


Obrázek 3.2: Hyperbolický tangens

ReLU

$$f(x) = \max(0, x). \tag{3.5}$$

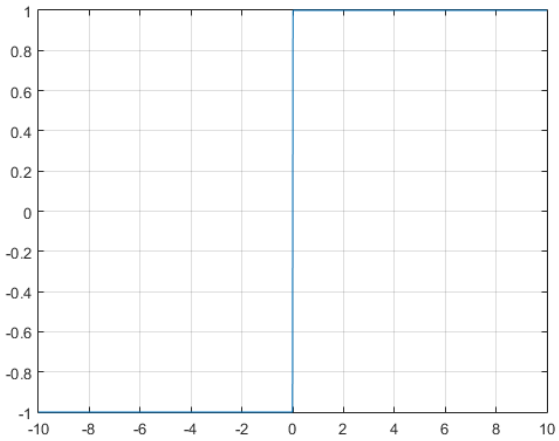
Tato funkce je momentálně používána nejvíce. Pomocí ReLU se hluboké neuronové sítě trénují několikanásobně rychleji než kdyby byla na stejné síti použita funkce hyperbolického tangensu [18]. Důvodem rychlejšího natrénování je snadné počítání gradientu díky tvaru této funkce.



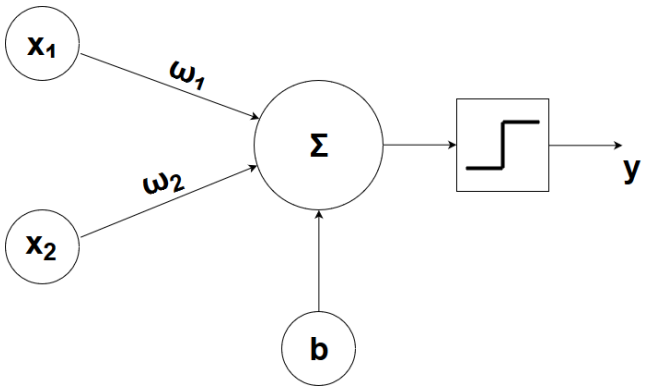
Obrázek 3.3: Funkce ReLU

3.3 Perceptron

Chování neuronových sítí lze ukázat na modelu perceptronu. U složitějších neuronových sítí jsou metody komplikovanější, princip však zůstává stejný. Perceptron je nejjednodušší neuronovou sítí, jelikož obsahuje pouze jeden neuron. Aktivační funkcí perceptronu je znaménková funkce.



Obrázek 3.4: Znaménková funkce



Obrázek 3.5: Model perceptronu se dvěma vstupy

Jako každá neuronová síť má perceptron dvě fáze: fázi učící a fázi predikce.

3.3.1 Učící fáze

Perceptronu se předkládá známá množina dat, podle kterých nastaví své váhy.

1. **Inicializace** - Váhovou matici $\boldsymbol{W}$ a prahový vektor $\mathbf{b}$ inicializujeme náhodnými malými čísly (doporučení $< -1, 1 >$). Zvolíme konstantu učení $c > 0$.

2. **Výpočet výstupní hodnoty** - Podle rovnice 3.2 zjistíme výstup perceptronu.

3. **Výpočet chyby** - srovnáním výstupu perceptronu s informací od učitele získáme podle rovnice níže chybu sítě:

$$E(k) = E(k-1) + 0.5(u(k) - y(k))^T \cdot (u(k) - y(k)), \tag{3.6}$$

   kde $u(k)$ je požadovaný výstup v k-tém kroku, $y(k)$ je skutečný výstup v k-tém kroku a $E(k-1)$ chyba z minulého kroku.

4. **Aktualizace vah**

$$\boldsymbol{W}(k+1) = \boldsymbol{W}(k) + c \cdot (u(k) - y(k)) \cdot \mathbf{x}(k)^T, \tag{3.7}$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + c \cdot (u(k) - y(k)). \tag{3.8}$$

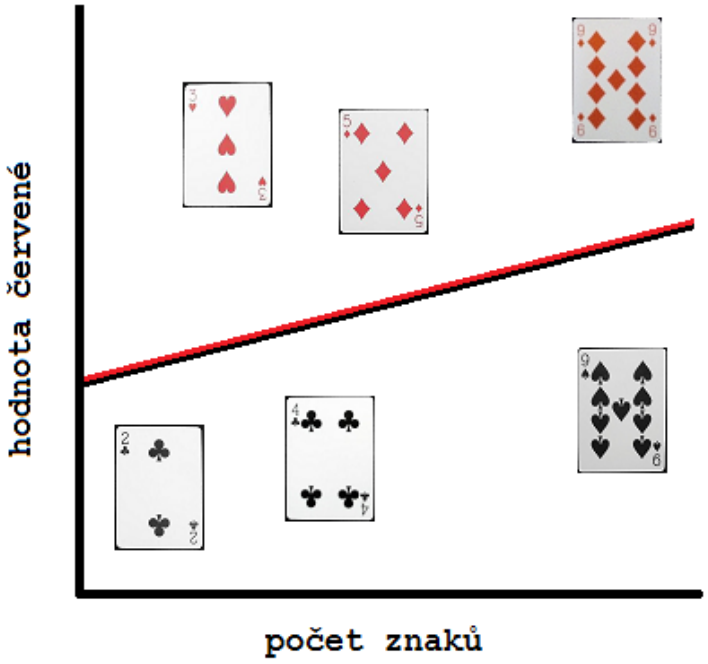
5. **Test vyčerpanosti trénovací množiny** - pokud se v trénovací množině nachází dosud nepoužité vstupy, vracíme se na krok 2.

6. **Konec trénovacího cyklu** - pokud je chyba nulová nebo se dostane pod předem stanovenou hodnotu, algoritmus končí a perceptron je natrénován. Pokud ne, vynulujeme $E(k)$ a vracíme se na krok 2 [32].

Algoritmus je znám také pod názvem backpropagation [11] (zpětná propagace).

3.3.2 Fáze predikce

V této fázi jsou perceptronu předkládána neznámá vstupní data. Pokud byla správně zvolena trénovací množina, má perceptron nastavené váhy tak, že je schopen správně zařadit data do jedné ze dvou tříd. V případě hracích karet by perceptron například dokázal rozlišit, zda se jedná o červenou nebo černou kartu.



Obrázek 3.6: Natrénovaný perceptron

3.4 Optimalizační algoritmy

Optimalizace je proces hledání nastavení parametrů **W** tak, aby ztrátová funkce byla minimální. Způsob, jakým se nastaví parametry **W** pro minimalizaci ztráty určuje optimalizační algoritmus.

3.4.1 Náhodné nastavení

U této metody dochází k nastavování matice **W** náhodně. Z těchto mnoha náhodných nastavení se poté vybere nastavení s nejmenší ztrátou. Metoda je známá také pod anglickým pojmem brute force [4].

3.4.2 Náhodné místní vyhledávání

V této metodě nejprve dojde k náhodnému nastavení váhové matice $\mathbf{W}$ a parametru δ. Parametr se přičte k váhové matici a zkoumá se, zda došlo ke snížení ztrátové funkce. Pokud ne, je δ zvolena náhodně znovu a proces se opakuje. Tato metoda rovněž neposkytuje uspokojivé výsledky.

3.4.3 Gradientní metody

Momentálně nejrozšířenější metodou optimalizace neuronových sítí je optimalizace pomocí gradientů. Hlavní myšlenkou gradientních metod je hledání směru, pomocí kterého se lze dostat do globálního minima ztrátové funkce. Nejrozšířenějším algoritmem tohoto typu je **SGD** (Stochastic gradient descent) [23], tento algoritmus aktualizuje váhovou matici s každým trénovacím obrazem. To má za následek zrychlení trénování sítě a zároveň výkyvy ztrátové funkce. Tento algoritmus zároveň snižuje pravděpodobnost toho, že by nastavením parametrů matice $\mathbf{W}$ skončil pouze v lokálním minimu ztráty a ne v minimu globálním. Chování algoritmu lze vidět z rovnice 3.9:

$$\mathbf{W} = \mathbf{W} - \eta \cdot \nabla_{\boldsymbol{W}} \boldsymbol{L}(\boldsymbol{W}; \boldsymbol{x}_i; y_i), \tag{3.9}$$

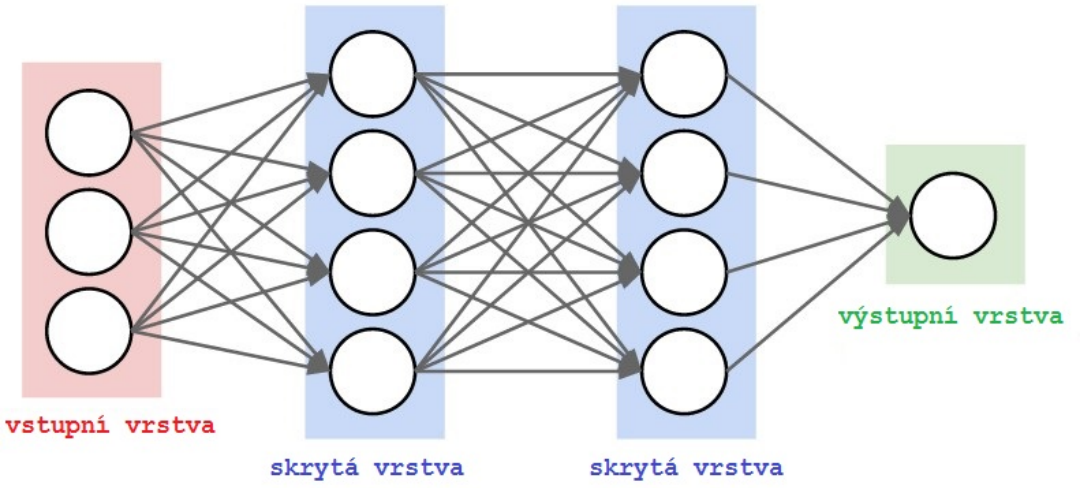
kde η je konstanta učení, $\boldsymbol{x}_i$ je i-tý trénovací obraz a y_i je třída obrazu x_i. V kombinaci s algoritmem zpětné propagace (3.3.1) tvoří tato metoda standardní algoritmus pro trénování neuronových sítí.

Mezi často používané gradientní metody patří také obdoby SGD, které proces optimalizace ještě více urychlí:

- **ADADELTA** [33] - zároveň optimalizuje konstantu učení, parametry optimalizuje pomocí momentové metody [24]
- **RMSprop** [28] - optimalizuje konstantu učení pomocí modifikace algoritmu RPROP (resilient propagation) [22]
- **ADAM** [23] - vylepšená metoda ADADELTA, v porovnání s ostatními adaptivními algoritmy poskytuje lepší výsledky

3.5 Architektury

V drtivé většině případů je pro klasifikaci lineární funkce nedostačující, proto je třeba použít sítě s více vrstvami, tzn. výstupy z neuronů použijeme jako vstupy pro další vrstvy neuronů. Nejčastěji používanou vrstvou je tzv. fully-connected vrstva, ve které jsou neurony mezi sousedními vrstvami spojeny každý s každým. Neurony ve stejné vrstvě spolu spojeny nejsou. Pochopitelně je třeba rovněž použít složitější aktivační funkce než funkci znaménkovou, viz 3.2.1.



Obrázek 3.7: Ukázka dvouvrstvé fully-connected sítě [15]

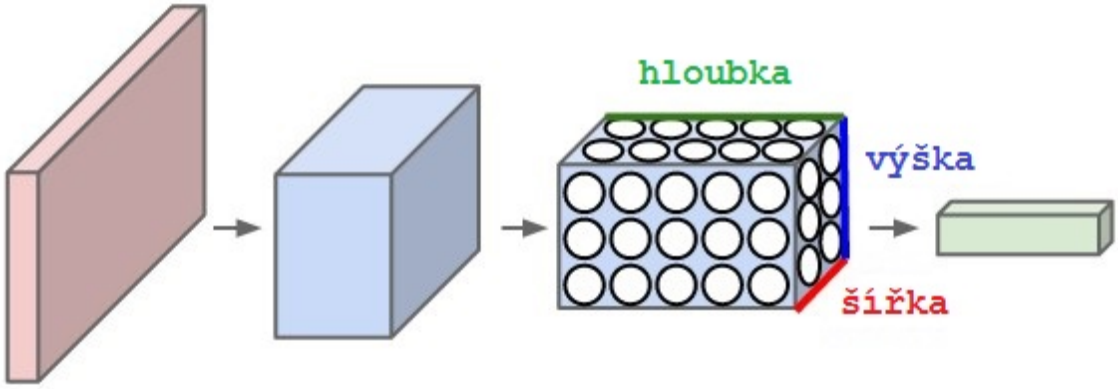
Pokud je počet skrytých vrstev v síti vyšší než jedna, jedná se o tzv. **hlubokou neuronovou síť** (deep neural network).

Výstupy neuronů mohou být také přivedeny na vstupy neuronů z předchozí vrstvy, v takovém případě se jedná o **rekurentní neuronovou síť**, která se využívá spíše k modelování a optimalizaci. Rekurentní neuronová síť se v praxi používá například jako našeptávač ve webových vyhledávačích [10].

Velice často také nejsou data lineárně separovatelná, proto se používají klasifikační metody, nejčastěji metoda **SVM** nebo **Softmax** (kapitola 4.3). Základní myšlenkou je použití tzv. jádrového triku (anglicky kernel trick), s jehož pomocí se provádí transformace dat z původního prostoru příznaků do prostoru vyšší dimenze, ve kterém již jsou lineárně separabilní [26].

3.5.1 Konvoluční neuronová síť

Tyto sítě jsou primárně určeny pro práci s obrazem. Na rozdíl od klasických neuronových sítí, kde je na vstupu příznakový vektor, mají konvoluční neuronové sítě jako vstupní data digitální obraz. Místo vektoru tak síť očekává matici (viz 2.1).



Obrázek 3.8: Ukázka dvouvrstvé konvoluční neuronové sítě, hloubka označuje kanály obrazu [15]

Při klasifikaci obrazu musí mít klasická síť pro každý pixel nastavené váhy $\boldsymbol{W}$. U digitálního obrazu záleží spíše na kontextu okolních pixelů a proto je třeba jich zpracovat více najednou. Došlo by tedy s největší pravděpodobností k tomu, že by síť byla přetrénovaná a postrádala by tak vlastnost generalizace. V konvolučních neuronových sítích jsou proto používány speciální vrstvy, které značně redukují počet parametrů v síti [19].

Vrstvy

- **Konvoluce** - Tato vrstva slouží pro získání lokálních vlastností obrazu. Ve vrstvě se nachází tzv. filtry s vahami, které se posouvají obrazem a konvolvují s ním (provádí maticový součin). Mějme tedy černobílý obrázek o velikosti 5×5 a filtr o velikosti 3×3, posouváním filtru po obrázku vznikne tzv. aktivační mapa:

$$\underset{\text{Vstupní obraz}}{\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}} \quad \underset{\text{Filtr}}{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}} \quad \underset{\text{Aktivační mapa}}{\begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}}$$

  Červeně označené prvky ve vstupním obrazu znázorňují, jakou oblast filtr snímá. Výsledkem součinu oblasti obrazu a váhy filtru je číslo čtyři v aktivační mapě.

Další číslo v aktivační mapě získáme posunutím filtru v obrazu:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Vstupní obraz

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Filtr

$$\begin{bmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{bmatrix}$$

Aktivační mapa

Jak si lze představit z příkladu, výsledek aktivační mapy závisí kromě velikosti filtru na mnoha dalších parametrech:

- **Váhy filtru** se trénují společně se sítí podle zvolené metody (viz 3.4). Nejpoužívanějšími metodami inicializace vah jsou Xavier a Gauss.
- Dalším parametrem filtru je jeho **posun**. V příkladu výše lze vidět, že se filtr posouvá obrazem o jeden pixel doprava, tedy posun je jedna. Pokud by se posun zvýšil na dva, aktivační mapa by měla místo velikosti 3×3 velikost 2×2. V praxi se však doporučuje tento parametr ponechat na hodnotě jedna.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

3×3 filtr s posunem 2

- Zároveň se ve filtru může uvádět jeho **přesah**, v příkladech výše je přesah nulový, jelikož filtr nepřesahuje svou snímací plochou obraz. Pokud by přesah byl jedna, filtr by snímal i oblast, kde obraz nemá žádné číslo, v tomto případě se obraz dodefinuje nulami. Proto je tento parametr znám pod anglickým názvem zero-padding (obalení nulami). Přesah filtru se používá pro změnu velikosti aktivační mapy.

$$\begin{bmatrix} 0 & 0 & 0 & & & \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ & 0 & 0 & 1 & 1 & 1 \\ & 0 & 0 & 1 & 1 & 0 \\ & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

3×3 filtr s přesahem 1

- **Pooling** - Pro zmenšení počtu parametrů a zabránění přetrénování se používá pooling vrstva. Filtr se posouvá po obraze a z dané oblasti vybere pouze příznak, který by ho mohl charakterizovat. Nejpoužívanější metodou vrstvy je max pooling, která z oblasti vybere maximální hodnotu pixelu. Na příkladu níže je pro jeden kanál ukázána funkce této vrstvy s filtry o velikosti 2×2:

$$\begin{bmatrix} 5 & 7 & 8 & 9 \\ 3 & 1 & 4 & 5 \\ 1 & 7 & 1 & 12 \\ 4 & 2 & 4 & 6 \end{bmatrix} \qquad \begin{bmatrix} 7 & 9 \\ 7 & 12 \end{bmatrix}$$

  U těchto filtrů nastavujeme pouze jejich velikost, oproti konvoluční vrstvě se zde nevyskytují parametry posunu a přesahu.

- **Normalizace** - Tyto vrstvy jsou používány z regulačních důvodů. Stejně jako u poolingu, normalizační vrstvy pomáhají zrychlit trénování sítě a vyhnout se jejímu přetrénování.

  - **Batch normalizace** - Často se data před trénováním sítě normalizují. Avšak při průchodu dat vrstvami sítě dochází ke změně jejich velikosti, gradientu atd. To má za následek zpomalení sítě. Z tohoto důvodu se používá batch normalizace, pomocí kterých se na vstupu vrstev zajistí normalizovaná data ve formě malých dávek.
  - **LRN** - Local Response Normalization [18] se používá pro lokální normalizaci pixelů v obraze. Například pokud je nějaký pixel příliš jasný, tak se pomocí této vrstvy normalizuje pomocí svých sousedních pixelů.
  - **Dropout** - Tato vrstva zajišťuje, že se neuronové sítě naučí reprezentovat stejná data různým nastavením vah. To je zajištěno náhodným odpojením neuronů v učící fázi. Díky mnohonásobné reprezentaci dat dochází ke zvýšení robustnosti sítě a menší pravděpodobnosti přetrénování.

- **Fully-connected** - Neurony spolu nejsou spojeny po celé vrstvě, ale jen v jejich okolí, jelikož je velká pravděpodobnost, že pixely, co jsou od sebe více vzdálené, nebudou mít mezi sebou žádný vztah.

Každou konvoluční vrstvu by měla následovat ReLU funkce, poté se většinou zařazuje pooling vrstva. Na výstup se zařazuje fully-connected vrstva. Pokud těchto vrstev chceme zařadit více, je třeba mezi vrstvy opět zařadit ReLU. Architektury předtrénovaných konvolučních sítí jsou k dispozici na internetu, proto místo vymýšlení vlastní architektury bývá často vhodnější pouze upravit model z předchozích výzkumů na svá data.

3.6 Caffe

Caffe [12] je framework určený pro hloubkové učení. Caffe je velice rozšířeno zejména díky své modularitě a rychlosti. Framework je vyvíjen v Kalifornské univerzitě v Berkeley. K vývoji zároveň přispívá rozšířená komunita. Neuronovou síť lze v Caffe vytvořit složením předpřipravených kódů. Framework je naprogramován v C++ a je provázán s Pythonem a Matlabem. Pro vytvoření sítě jsou potřeba dva *.prototxt* soubory:

- **Model** - Zde je uložena architektura sítě.
- **Solver** - Tímto souborem se upravují parametry trénování sítě.

Po natrénování sítě je tato síť uložena pod příponou *.caffemodel*.

3.6.1 Model

V modelu musí být specifikováno, co má dělat (trénování, testování, trénování s testováním), poté je nutné zvolit typ vstupních dat a cestu k nim. Síť se poté skládá stejně, jak bylo naznačeno na konci kapitoly 3.5. Na konci sítě se pro určení chybovosti zavádí ztrátová funkce. V rámečku níže je ukázka vytváření vrstev pro síť:

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 8
    kernel_size: 10
    stride: 1
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
```

V příkladu lze vidět konvoluční vrstvu s osmi filtry o rozměrech 10×10 a posunem jedna, matice filtrů je inicializována jako gaussovská, přesah je nastaven na nulu. Každá vrstva má své jméno, na které se dá odkazovat, viz. *name*. Vrstvy jsou spolu spojené pomocí klíčových slov *bottom* - předcházející vrstva a *top* - současná vrstva. Pod konvoluční vrstvou se v ukázce nachází aktivační funkce ReLU.

3.6.2 Solver

Solver určuje, jak bude probíhat trénování sítě. Pomocí tohoto souboru se nastaví parametry trénování sítě:

```
net: "home/janhrdli/NS/cards_train_val.prototxt"
snapshot_prefix: "cardclassifier"
base_lr: 0.02
momentum: 0.9
weight_decay: 0.005
lr_policy: "step"
stepsize: 1000
gamma: 0.1
display: 1000
max_iter: 10000
snapshot: 5000
solver_mode: GPU
```

Nejprve je třeba spárovat model se solverem - viz první řádek kódu. Důležitými parametry jsou *base_lr*, který udává konstantu učení a *gamma*, která v tomto případě zajišťuje každých 1000 kroků (parametr *stepsize*) zmenšení konstanty učení desetkrát. Natrénované váhy modelu se ukládají každých 5000 iterací (parametr *snapshot)* pod názvem *cardclassifier* (parametr *snapshot_prefix)*. Po dosažení maximálního počtu iterací trénování sítě končí.

Pro testování sítě je nutno nahrát natrénovaný soubor *.caffemodel* a předložit soubor s architekturou sítě. Tento soubor je většinou pojmenován jako *deploy.prototxt*. Model má na vstupu předpokládané dimenze vstupních dat a ztrátová funkce na konci sítě je smazána, až na tyto rozdíly je *deploy.prototxt* shodný s trénovacím modelem.

3.7 Metacentrum

Metacentrum je organizace zabývající se zejména gridovým počítáním, tedy počítáním na více počítačích, které se nachází na různých místech republiky. Díky metacentru lze vypočítat úlohy, které jsou příliš náročné pro klasický počítač. Jelikož je tato organizace pod sdružením CESNET(Czech Education and Scientific NETwork), je metacentrum pro studenty zdarma [25].

Pro zahájení počítání je potřeba SSH klient a FTP klient pro prohlížení a přesun souborů. O úlohu, pomocí které se bude počítat, lze zažádat příkazem v SSH klientovi. Parametry příkazu mohou být délka trvání úlohy, potřebný počet procesorů pro úlohu, potřebná paměť, atd. Zároveň lze specifikovat, zda má být úloha **interaktivní** či **dávková**.

Při zvolení interaktivní úlohy máme možnost sledovat přímý průběh počítání a upravovat výpočet přímo za běhu. Tento typ úlohy je vhodný při práci s grafickým

prostředím nebo při počítání jednodušších úloh, na které je třeba dohled.

Dávková úloha používá tzv. skriptů, ve kterých je napsán algoritmus pro její výpočet. Na rozdíl od interaktivní úlohy nelze výsledek ovlivnit, výsledky jsou poskytnuty až po skončení této úlohy. Tento typ je vhodný při počítání složitých úloh.

V tomto případě byla úloha počítána na clusteru doom. Tento cluster je určen pro výpočty na grafických kartách a obsahuje třicet uzlů, každý z nich má následující hardwarovou specifikaci:

CPU	2× 8-core Intel Xeon E5-2650v2 2.60GHz
RAM	64GB
GPU	2× nVidia Tesla K20 5GB (Kepler)
disk	2× 2x 1TB 10k SATA III, 2x480GB SSD
net	Ethernet 1Gb/s, Infiniband QDR

Tabulka 3.1: Specifikace zařízení

Počítání na grafických kartách (GPU) bylo zvoleno z důvodu rychlejšího trénování sítě než na CPU.

Kapitola 4

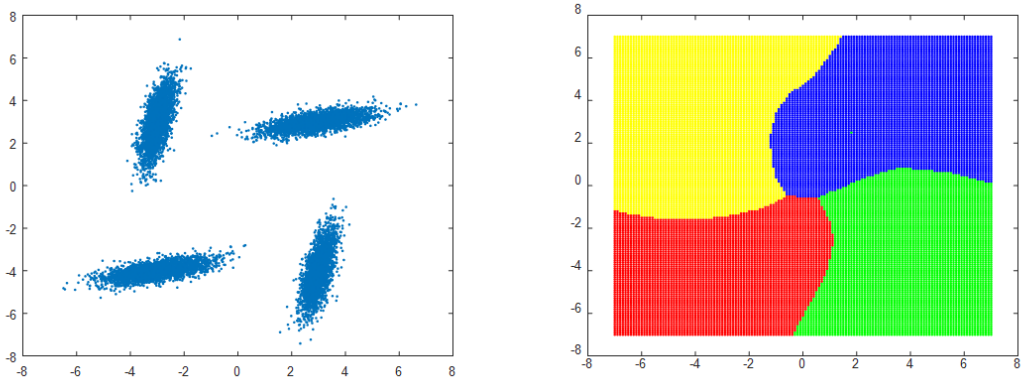
Klasifikace

V této kapitole budou ukázány metody klasifikace. V kapitole 3.3.2 bylo ukázáno, že na základě výsledků natrénované neuronové sítě lze posoudit, do jaké třídy vstupní data zařadit. Způsob klasifikace dat do tříd určuje typ klasifikátoru.

4.1 Klasifikace podle k-nejbližšího souseda

Název této metody napovídá, že neznámý testovací obraz se klasifikuje do stejné třídy jako k-nejbližších známých obrazů. Obrázek 4.1 znázorňuje princip klasifikace podle jednoho nejbližšího souseda, kde vlevo jsou známá trénovací data a vpravo je barvami ukázáno, do jaké třídy by patřila neznámá testovací data.

Klasifikace podle k-nejbližších sousedů funguje obdobně. Pro klasifikaci však musí u k-nejbližších sousedů nastat případ, že neznámý testovací obraz je nejblíže k-známým obrazům, které jsou všechny ze stejné třídy. Pokud nejbližší trénovací obrazy nejsou ze stejné třídy, nelze testovací obraz klasifikovat [16]. Tato metoda není v praxi příliš používána z důvodu vysokých paměťových nároků, jelikož si stroj musí pamatovat všechna trénovací data.



Obrázek 4.1: Klasifikace podle jednoho nejbližšího souseda

4.2 Lineární klasifikace

V tomto způsobu klasifikace se počítá tzv. **výsledková funkce** (score function) s, pomocí které se rozhodne, do jaké třídy testovací obraz patří. Výsledková funkce je vektor o $\mathbf{N}$ prvcích, kde $\mathbf{N}$ je počet tříd, který uchovává skóre podobnosti testovacího obrazu s danou třídou. Toto skóre je spočteno následovně:

$$s(\boldsymbol{x}_i, \mathbf{W}, \mathbf{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}, \tag{4.1}$$

kde $\boldsymbol{W}$ je váhová matice, ve které je pro každou třídu uloženo, jak se mají jednotlivé pixely zvýhodňovat či znevýhodňovat. Pro karty s obrázkem (K,J,Q) by například tato matice měla veprostřed vysoké hodnoty, jelikož se veprostřed těchto karet nachází nejdůležitější část, podle které lze nejsnáze určit, o kterou kartu se jedná. Jak už zaznělo výše $\boldsymbol{x}_i$ je vstupní obraz a $\boldsymbol{b}$ je vektor, pomocí kterého se zvýhodňují jednotlivé třídy. Jednoduchý lineární klasifikátor je ukázán na obrázku . Klasifikace téměř vždy probíhá ve více dimenzích než jsou dvě, to je ale náročné na vizualizaci. Změna váhové matice $\boldsymbol{W}$ by na obrázku 3.6 způsobila změnu sklonu klasifikační přímky. Změna parametru $\boldsymbol{b}$ by naopak způsobila posun přímky po y-ové ose [15].

4.3 Klasifikace pomocí ztrátové funkce

Ztrátové funkce reprezentují rozdíl mezi požadovanou a predikovanou hodnotou [15]. Například pokud by byla síti předložena černá karta a síť by ji klasifikovala jako červenou, byla by ztrátová funkce vysoká. Pokud by došlo ke správné klasifikaci, ztrátová funkce by nabývala nízké hodnoty. Pro klasifikaci využívají neuronové sítě většinou jeden z následujících klasifikátorů:

4.3.1 Support Vector Machine

Tato funkce, známá také jako SVM [26], se snaží o minimalizaci ztrátové funkce připočítáním zisku Δ k výsledkové funkci u požadované třídy. SVM je definována jako:

$$L_i = \sum_{i \neq j} max(0, s_j - s_{y_i} + \Delta), \tag{4.2}$$

kde s_{y_i} označuje výsledkovou funkci pro správnou třídu (y_i označuje index správné třídy) a s_j znamená výsledkovou funkci pro j-tou třídu. Tato funkce je známá pod názvem **hinge loss** [2]. Z rovnice 4.2 lze zjistit, že SVM se snaží o to, aby skóre správné třídy y_i bylo alespoň o Δ větší než skóre chybné třídy. Pokud tomu tak není, dochází k akumulaci ztrátové funkce. Zisk Δ je nastavován jako jedna.

Jak bylo zmíněno v rovnici 4.1, výsledková funkce je spočtena pomocí váhové matice $\boldsymbol{W}$. Může se stát, že je ztrátová funkce nulová pro všechny testovací obrazy (došlo ke správné klasifikaci všech obrazů) a matice $\boldsymbol{W}$ je nastavena na příliš velké hodnoty. Může totiž existovat i jiná matice $\lambda\boldsymbol{W}$ (kde $\lambda > 1$), která by zajistila nulovou ztrátovou funkci. Výsledková funkce by však byla pro obě matice odlišná. Pro

odstranění tohoto problému se používá tzv. regularizační penalizace (regularization penalty) $\boldsymbol{R}(W)$. Nejčastěji se používá tzv. L2 norma:

$$\boldsymbol{R}(W) = \sum_k \sum_l \boldsymbol{W}_{k,l}^2. \tag{4.3}$$

Konečná ztrátová funkce tedy vypadá následovně:

$$\boldsymbol{L} = \frac{1}{N} \sum_l L_i + \lambda R(W), \tag{4.4}$$

kde N je počet trénovacích obrazů a λ je parametr, který je většinou určen pomocí křížové validace [17].

Penalizací vysokých vah matice zároveň dojde ke zlepšení generalizace a tedy ke zmenšení pravděpodobnosti přetrénování sítě [15].

4.3.2 Softmax

Tento klasifikátor normalizuje výsledkovou funkci (rovnice 4.1) tak, aby výstup této funkce představoval pravděpodobnosti klasifikace do daných tříd. Ztrátová funkce je počítána pomocí křížové entropie(**cross entropy**) [7]:

$$L_i = -f_{y_i} + \ln \sum_j e^{f_j}, \tag{4.5}$$

kde funkce f_j představuje funkci softmax:

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}. \tag{4.6}$$

Funkce softmax, po které je klasifikátor pojmenován, zajišťuje hodnoty mezi nulou a jedničkou (převádí je na pravděpodobnosti), suma těchto hodnot dá dohromady jedničku (jak plyne z věty o pravděpodobnosti). Tato funkce se rovněž používá jako aktivační funkce (viz 3.2.1). Konečná ztrátová funkce je spočítána stejně jako u SVM (rovnice 4.4).

Kapitola 5

Aplikace metod a vyhodnocení experimentů

Tato kapitola se bude zabývat praktickým řešením identifikace karet. Vyjmutí karet ze záznamu, vytvoření databáze a metoda srovnávání se vzorem byly naprogramovány v programovacím jazyku Python zejména pomocí knihovny OpenCV. Neuronová síť byla poté trénována na serverech Metacentra za použití modulu Caffe. Ověření přesnosti sítě bylo rovněž naprogramováno v Pythonu.

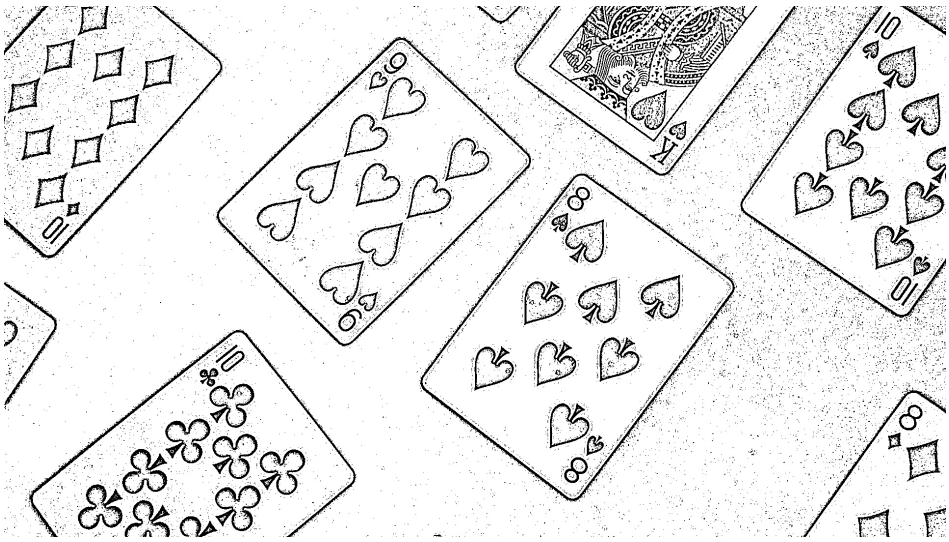
5.1 Získání snímků z videa

Videozáznam byl pořízen z více zařízení s různou záznamovou kvalitou pro získání různorodých dat pro neuronovou síť. Pro nahrání a následné snímkování záznamu pomocí funkce *VideoCapture* z knihovny OpenCV (kapitola 2.5) s formátem *.mp4* je nutné mít v Pythonu nainstalován balík *ffmpeg*.

Obrázek 5.1: Snímek získaný z videozáznamu

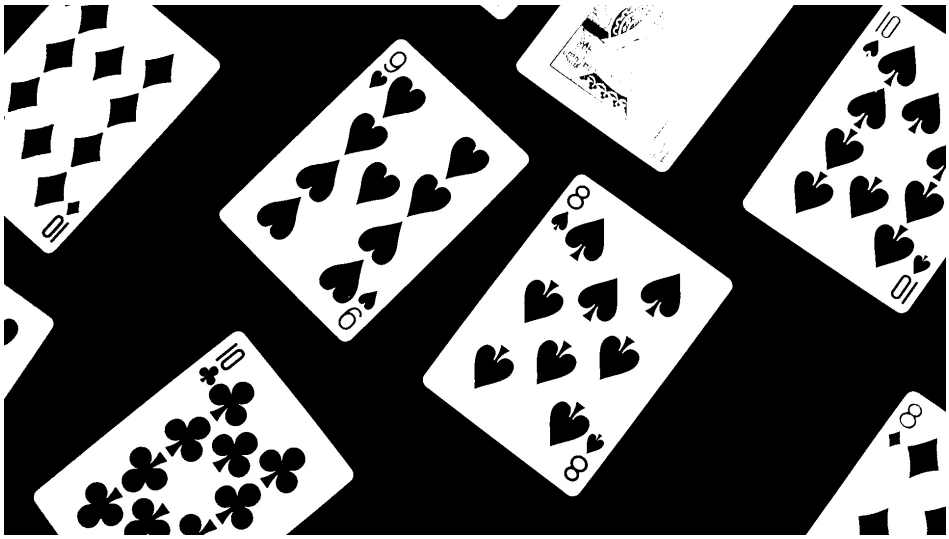
5.1.1 Segmentace

Z obrázku 5.1 lze na kartách vidět, že videozáznam nebyl pořízen za ideálních světelných podmínek a pod správným úhlem, neboť se karty lesknou. Neuronová síť se tak díky této skutečnosti naučí rozlišovat i tyto případy. Následná segmentace probíhala aplikací adaptivní segmentace pomocí funkce *adaptiveThreshold* z OpenCV.



Obrázek 5.2: Segmentovaný snímek

Jak lze vidět z obrázku 5.2, jasové změny ve snímku byly vyrovnány pomocí metody adaptivní segmentace. Obrázek 5.3 ukazuje, co se stane při segmentaci s pevně zvoleným prahem.



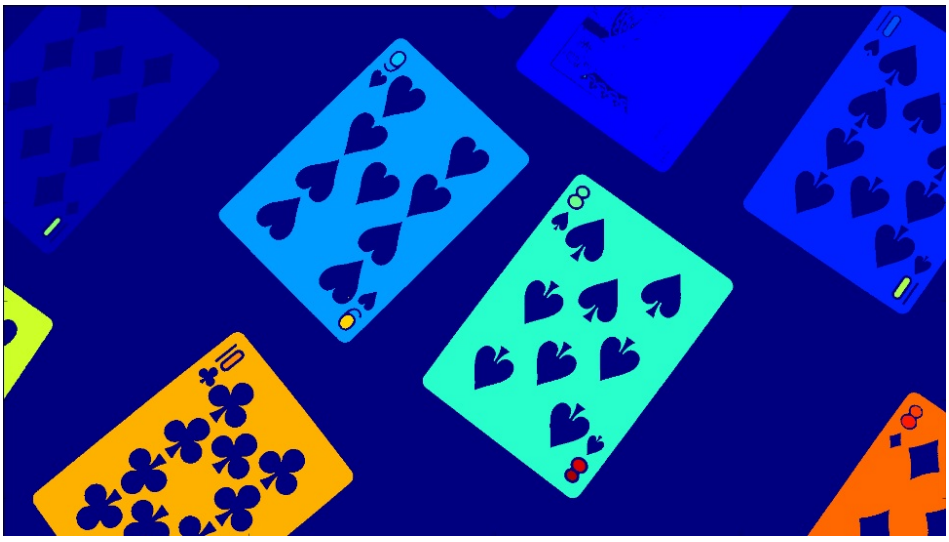
Obrázek 5.3: Binárně segmentovaný snímek

5.2 Získání karty z obrazu

Video je nyní převedeno na sekvenci snímků. Pomocí segmentovaného obrazu z kapitoly 5.1.1 nalezneme objekty v obraze ve formě karet. Na nalezené karty poté aplikujeme kontury, pomocí kterých zaměříme souřadnice karet v obraze. Karty poté otočíme a vyjmeme z obrazu. Nakonec změníme velikost každé karty tak, aby všechny měly stejný počet pixelů.

5.2.1 Labeling

Pomocí metody *measure.label* z knihovny Scikit-image (kapitola 2.7) se tyto segmentované snímky převedly na objekty. Objekty však netvořily pouze karty, ale i čísla a obrázky na nich, viz obrázek 5.4 níže. U těchto objektů se dá zjistit funkcí *measure.regionprops* ze Scikit-image, kolik zabírají pixelů. Proto jsme určili hraniční počet pixelů, pomocí kterého jsme rozlišili, zda se jedná o kartu. Aby se do databáze nedostaly karty, které nejsou na snímku vidět celé, byl vytvořen jednobarevný rámeček kolem všech snímků pro zmenšení počtu kartou obsažených pixelů. Pokud by byly karty natáčeny z velké výšky nebo ze zařízení s jiným rozlišením, bylo by nutné hraniční počet pixelů změnit.



Obrázek 5.4: Labeling snímku

Na obrázku výše jsou objekty rozlišeny barvou, lze vidět, že i mezery v číslech mohou být objekty.

5.2.2 Konturování

Dalším krokem bylo kolem získaných karet nakreslit kontury, které slouží pro ohraničení požadovaného objektu. Pomocí metody *findContours* z OpenCV byly nalezeny vnější kontury karet. Tyto kontury však ohraničovaly kartu pouze pomocí

bodů, které netvořily rovnou přímku. Proto byla na kontury ještě použita aproximace těchto bodů, která způsobila ohraničení karet přímkami. Poté byly kontury zakresleny do obrazu metodou *drawContours* opět z knihovny OpenCV.

Obrázek 5.5: Kontury ohraničující karty

5.2.3 Rotace

Pro zajištění co nejpřesnější rotace se použila metoda *minAreaRect* z OpenCV, která z kontur získala obdélník s minimálním počtem pixelů. Tento obdélník se zakreslil do původního obrazu. Karty mají zaoblené rohy a při použití kontur není zcela jasné, kde se nachází roh karty, při použití obdélníka se pouze vybere minimální x-ová souřadnice a maximální y-ová souřadnice. Požadovaný úhel je poté spočítán jako $\arctan \frac{y}{x}$. Tímto způsobem může mít karta na x-ové ose delší stranu, proto pokud se to stalo, byla ještě otočena o devadesát stupňů.

Obrázek 5.6: Rotace srdcové devítky

5.2.4 Ořez, změna velikosti

Po rotaci kartu lze vyjmout ze snímku. Souřadnice, kde se karta nachází, se zjistily pomocí metody *argwhere* z balíku NumPy (kapitola 2.6). Nakonec se provedla změna velikosti karet tak, aby všechny karty měly stejný počet pixelů, jelikož metoda srovnávání se vzorem i neuronová síť očekávají na vstupu stejné rozměry všech karet.

Obrázek 5.7: Získaná karta

Z obrázku 5.7 lze vidět, že karta není úplně přesný obdélník, to je zapříčiněno tím, že záznam nebyl pořízen přesně kolmo k ploše s kartami. Tato chyba se může projevit horšími výsledky při testování přesnosti u metody srovnávání se vzorem.

Počet získaných karet byl závislý na typu zařízení, jak ukazuje tabulka 5.1:

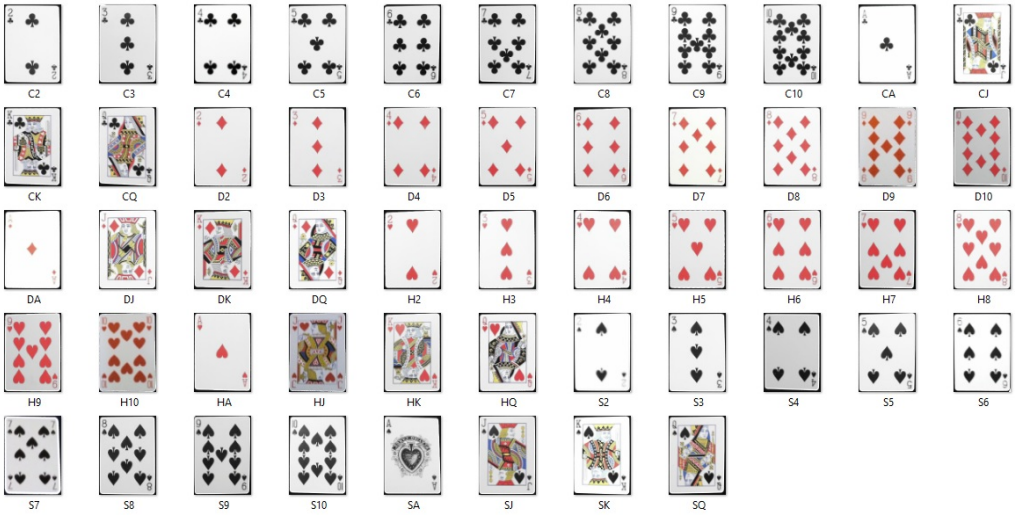
Zařízení	**Canon C-760**	**Honor** 3C	**Iphone 4S**	**LifeCam Studio**
Rozlišení	320×240	1280×720	1280×720	1920×1080
FPS	15	30	27	29
Doba videí	$340s$	$560s$	$370s$	$190s$
Získaných karet	1800	7410	3700	4080

Tabulka 5.1: Nahraná videa a jejich specifikace

Video s nejlepšími světelnými podmínkami bylo pořízeno webkamerou Microsoft LifeCam Studio. Toto video bylo totiž jako jediné nahráváno ve školní laboratoři. Ostatní nahrávky byly pořízeny z domova za horších podmínek. Všechna videa byla natáčena z přibližně stejné výšky. Na pořízených kartách bylo poznat z jakého jsou videa, jelikož každé zařízení zachytilo barvy trošku jinak, což je výhoda pro trénování neuronové sítě.

5.3 Srovnávání se vzorem

Vzory pro tuto metodu mohou být vybrány jako celé karty nebo jako kombinace hodnoty v rohu karty a tvaru barvy (srdce, piky, kříže, káry). Jelikož v reálném případě dochází často k znehodnocení obrazu šumem nebo pohybem kamery, byly pro tuto práci vybrány jako vzory celé karty. Často také mohou vzniknout nepřesnosti při otáčení karty - kapitola 5.2.3, tudíž získání vzoru, který by reprezentoval průměrnou kartu pro danou hodnotu a barvu je velmi obtížné a zdlouhavé.



Obrázek 5.8: Zvolené vzory

Další nevýhodou této metody je, že zvolené vzory mají různé jasové parametry (viz obr. 5.8 výše) a pro videozáznam pořízený s jinými světelnými podmínkami by touto metodou nebylo dosaženo uspokojivých výsledků. Snažit se vytvořit vzory tak, že by se karty nasnímaly jednotlivě za stejných jasových podmínek a pod co nejpřesnějším pravým úhlem by také nepřineslo přijatelné výsledky. Deformace a různý jas karty jsou v těchto vzorech žádoucí, protože pokud například byla karta natáčena pět vteřin, tak je velice nepravděpodobné, že by došlo k výraznějšímu pohybu kamerou a proto se bude tato deformace bude nacházet na všech kartách daného druhu, to samé platí pro jas.

Srovnávání se vzorem je realizováno pomocí funkce *matchTemplate* s parametrem *TM_CCOEFF_NORMED* z OpenCV. Tato funkce vrací hodnotu v rozmezí nula až jedna, kde jedna je maximální podobnost (stejný obraz). Hlavní myšlenkou je srovnat testovaný obraz se všemi vzory a vybrat ten vzor, který získal největší podobnost. Tato podobnost je ještě porovnávána s prahem, pomocí kterého dojde k vyfiltrování většiny chybně zaklasifikovaných karet. Pokud je podobnost větší než práh, testovaná karta se uloží do složky, která je pojmenovaná po názvu vzoru - viz 5.8. Pro karty, kde se nachází složitější obrázek (K,Q,J) je práh nastaven jako menší hodnota, jelikož špatná klasifikace těchto karet je méně pravděpodobná. Karty, které měly podobnost se vzorem menší než nastavený práh jsou přesunuty do složky s nezaklasifikovanými kartami.

Dohromady byly vybrány tři sety vzorů. Na klasifikaci karet z ostatních videí byl vždy použit nejvhodnější z těchto tří setů. Úspěšnost správného zatřídění karty u metody srovnávání se vzorem velice záleží na správně zvolených vzorech a nastavení prahu. Při nastavení vysokého prahu zatřídí metoda většinu karet do garbage složky a v ostatních složkách budou s největší pravděpodobností pouze karty, které tam opravdu patří. Pokud naopak nastavíme práh nízký, v garbage složce bude karet méně, ale zvyšuje se riziko chybného zatřídění karty. U testování úspěšnosti této metody při níže nastaveném prahu se musí kontrolovat, zda jsou karty zařazeny do správných složek. Při zvolení vyššího prahu se bude počet chyb metody blížit počtu karet v garbage složce.

Testování přesnosti probíhalo na kartách z webkamery. Při zvolení prahů podobnosti 85% a 75% pro karty s obrázkem byly všechny chybně zaklasifikované karty v garbage složce, těchto karet bylo 2200, přesnost tedy byla $49,63\%$. Při nastavení prahů podobnosti na 75% a 65% obsahovala garbage složka 156 karet, avšak ve složkách pro karty bylo nalezeno 771 chybně zatříděných karet, výsledná přesnost s tímto nastavením činila $76,83\%$. Důkazem, že je nutné volit vzory velmi pečlivě, je fakt, že v některých složkách s jednotlivou kartou bylo nalezeno až 63 chybně zatříděných karet a v některých naopak ani jedna. Největší úspěšnost $81,81\%$ byla dosažena při nastavení prahů na 78% a 68% pro karty s obrázkem.

5.4 Příprava dat pro neuronovou síť

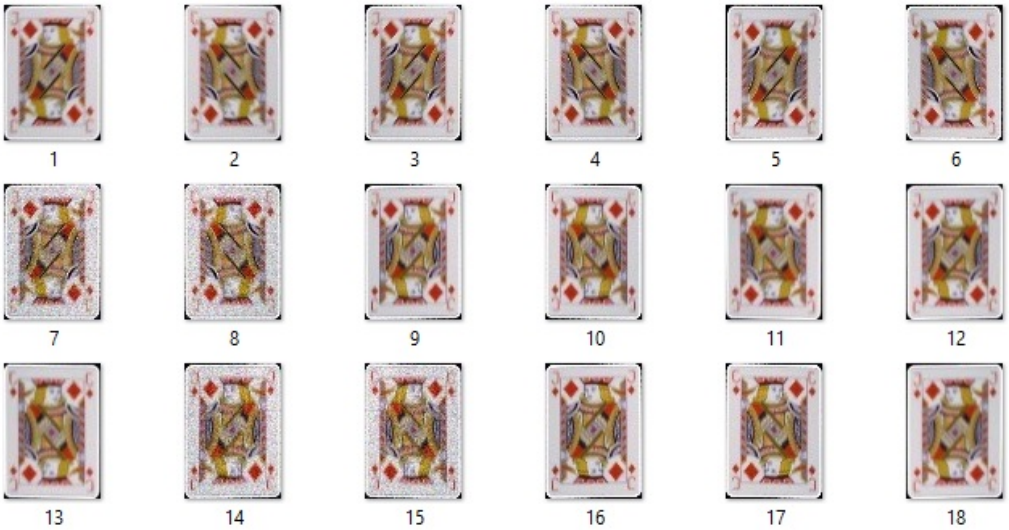
Pro učící fázi sítě byly sloučeny složky s roztříděnými kartami z více videí, vytvořené v kapitole 4.3. Zároveň byl pro každou složku u trénovacích dat zajištěn stejný počet karet v ní, aby se síť nenatrénovala více na určitý typ karty a nezvýhodňovala ji tak při klasifikaci. Třída nezaklasifikovaných karet nebyla pro neuronovou síť použita. Pro větší robustnost sítě a několikanásobně větší počet trénovacích dat jsou na kartách provedeny augmentace:

- **otočení podle x-ové osy** - Jak lze vidět z obrázku 5.7, karta není horizontálně souměrná a jejím otočením získáme odlišnou kartu.
- **otočení podle y-ové osy** - Karta není vertikálně souměrná, také viz 5.7. Zrcadlovým otočením zároveň dojde k otočení čísla na kartě.
- **Rozmazání** - K rozmazání byla použita metoda *GaussianBlur* z OpenCV s maticí vyhlazení 5×5.
- **Zašumění** - Pro tuto augmentaci byl použit gaussovský šum z metody *util.random_ noise* ze Scikit-image se střední hodnotou 0 a rozptylem 0.1.

Augmentací na jedné kartě mohlo být více, nikdy však nebylo použito naráz rozmazání a zašumění, došlo by pak k příliš velkému znehodnocení kvality obrazu. Karty bez augmentací byly pro testování a trénování sítě použity taktéž. Všechna data se vydělila 255, aby bylo zajištěno, že na vstupu neuronové sítě bude preferovaný rozsah od nuly do jedné. Zároveň byly obrazy zmenšeny, jelikož vstupní HDF5

soubor by byl příliš velký (tímto se rovněž snižuje pravděpodobnost přetrénování sítě).

Soubor HDF5 obsahoval dva datasety, v jednom byl uložen obraz, ve druhém číslo, které určuje, do jaké třídy by se měl obraz zaklasifikovat. Číslo bylo odvozeno od abecedního pořadí názvu vzorů. Pro testovací data byl postup stejný. Celkem jsme pro neuronovou síť připravili tři HDF5 soubory - trénovací s augmentacemi a testovací s augmentacemi a bez augmentací.



Obrázek 5.9: Augmentované karty, 1 - rozmazaná, 2 - rozmazaná, otočená podle y-ové osy, 3 - původní, 4 - otočená podle y-ové osy, 5 - otočená podle x-ové osy, 6 - otočená podle x-ové osy, otočená podle y-ové osy, 7 - zašuměná, 8 - zašuměná, otočená podle y-ové osy, 9 - původní, 10 - otočená podle y-ové osy, 11 - otočená podle x-ové osy, rozmazaná, 12 - otočená podle x-ové osy, rozmazaná, otočená podle y-ové osy, 13 - otočená podle x-ové osy, rozmazaná, 14 - rozmazaná, otočená podle y-ové osy, 15 - zašuměná, 16 - zašuměná, otočená podle y-ové osy, 17 - původní, 18 - otočená podle y-ové osy

5.5 Trénování neuronové sítě

V souboru, určenému k trénování sítě, bylo po augmentacích dohromady 12720 karet. Nejprve byla pro trénování sítě přidělena architektura i solver. Tato přidělená architektura měla tři konvoluční vrstvy, mezi kterými byly funkce ReLU a pooling. Na konci těchto vrstev se nacházely dvě fully-connected vrstvy a nakonec Softmax vrstva. Po testování měla síť přesnost kolem 80%, což je sice více než u metody srovnávání se vzorem, ale předpokládali jsme, že by přesnost mohla jít ještě zvýšit úpravou architektury sítě. Všechny sítě byly trénovány s výstupem typu Softmax pomocí optimalizačního algoritmu SGD.

Poté jsme se snažili napodobit architektury známých sítí - AlexNet [18], SegNet [1], ZF Net [34]. Tyto pokusy rovněž neposkytly příliš uspokojivé výsledky zejména kvůli nedostatku přiděleného místa na grafické kartě a možnosti počítání úlohy pouze

po dobu jednoho dne.

Nepřesnost sítí mohla být způsobena i špatnými trénovacími a testovacími daty. Pokusili jsme se tedy ještě karty zmenšit na finální rozlišení 60×84 a snížit na nich sílu augmentací. Po tomto zákroku se přesnost sítí zvýšila. Práci velmi usnadnily skripty (viz 3.7), pomocí kterých se do fronty zařadilo více úloh s odlišnými architekturami a výsledky byly poté uloženy do *.sh* souboru, takže nebylo třeba po každém dokončeném trénování upravovat architekturu sítí.

5.5.1 Výsledky

Nejlepších výsledků dosáhly sítě v tabulkách níže (5.2, 5.3 a 5.4). Konvoluční vrstvy byly vždy tvořeny ze třiceti dvou 3×3 filtrů s posunem jedna a přesahem nula. Konečná fully-connected vrstva(také inner product) měla jako výstup vektor o 52 hodnotách. Tyto hodnoty říkaly, s jakou pravděpodobností testovaná karta spadá do dané třídy karet. Pochopitelně jsme tedy zvolili nejvyšší hodnotu a porovnali pořadí této hodnoty s číslem, které bylo uloženo v HDF souboru, viz 5.4. Pokud jsme našli nejvyšší hodnotu na 34. místě a číslo v HDF souboru bylo rovněž 34, došlo ke správné klasifikaci. Pro klasifikaci se často používají také hodnoty TOP 3 a TOP 5, které uvažují správnou klasifikaci, i když karta je mezi prvními třemi nejpravděpodobnějšími třídami (TOP 3) na výstupu Softmaxu.

Architektura	**Augmentovaná**	**Původní**
Konvoluce, ReLU, konvoluce, ReLU, inner product 4096, ReLU inner product	86,04%	86,31%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	90,54%	90,22%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	92,21%	92,26%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product 2048, ReLU, inner product	92,67%	93,45%

Tabulka 5.2: Výsledky sítí pro TOP 1

Architektura	**Augmentovaná**	**Původní**
Konvoluce, ReLU, konvoluce, ReLU, inner product 4096, ReLU inner product	98,87%	99,06%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	99,18%	98,90%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	98,07%	98,21%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product 2048, ReLU, inner product	99,56%	99,69%

Tabulka 5.3: Výsledky sítí pro TOP 3

Architektura	**Augmentovaná**	**Původní**
Konvoluce, ReLU, konvoluce, ReLU, inner product 4096, ReLU inner product	99,91%	99,96%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	99,54%	99,61%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product	99,60%	99,80%
Konvoluce, ReLU, pool, konvoluce, ReLU, pool, inner product 2048, ReLU, inner product	99,93%	99,93%

Tabulka 5.4: Výsledky sítí pro TOP 5

Tabulka 5.2 zobrazuje úspěšnosti správné klasifikace karty. Tabulky 5.3 a 5.4 zobrazují úspěšnost umístění karty mezi třemi nejpravděpodobnějšími klasifikacemi sítě (tab. 5.3) nebo mezi pěti nejpravděpodobnějšími klasifikacemi (5.4).

Z tabulek je zřejmé, že přesnost sítí je tím vyšší, čím více nejvyšších hodnot Softmaxu prohledáváme. Z tabulek lze také vidět, že augmentovaná testovací data byla pro sítě obtížnější na rozpoznání, vyjímkou je druhá druhá síť, která na těchto datech dosahovala stejných nebo dokonce lepších výsledků. Z přesnosti poslední sítě na původních datech v tabulce 5.2 lze zjistit, že tato síť klasifikovala chybně přibližně každou patnáctou kartu. Přesnost na augmentovaných datech znázorňuje, jak si dokáže síť poradit s různými deformacemi karet. Pro reálné fungování sítě je však důležitá přesnost na neaugmentovaných datech. Síť byla s augmentovanými daty trénována proto, aby se pokrylo co největší množství deformací, které na kartě mohou reálně nastat.

Kapitola 6

Závěr

Cílem této práce bylo vybrat vhodnou metodu pro rozpoznávání jednotlivých karet a poté implementovat algoritmus detekce a rozpoznávání karet pomocí této metody. Nejprve bylo nutné naprogramovat algoritmus pro vyjmutí karet z videozáznamu. Jelikož byla většina záznamů nahrávána v neideálních světelných podmínkách, byly karty jinak jasné. Zároveň nebyly nahrány pod přesným pravým úhlem, proto lze vidět z obrázku 5.7 mírnou deformaci karty v levém horním rohu. Zejména kvůli těmto skutečnostem pak musely být vzory pro metodu srovnávání se vzorem vybírány velice pečlivě tak, aby každý vzor reprezentoval průměrný vzhled dané karty. I přes tuto snahu neposkytla tato metoda příliš uspokojivé výsledky. Jak zaznělo v úvodu, byly natáčeny dva sety karet. Většina pokerových setů si je však velice podobná, tudíž tento fakt nijak identifikaci neztížil.

Roztříděné, augmentované obrazy byly poté použity jako trénovací a testovací data pro neuronovou síť. Jasové rozdíly v kartách byly pro síť naopak prospěšné. Došlo k vyzkoušení mnoha architektur a změn parametrů. Nejlepší natrénovaná neuronová síť měla přesnost $93,45\%$ na neaugmentovaných kartách, což je o $11,64\%$ více, než u metody srovnávání se vzorem s nejvhodněji zvoleným parametrem prahu. K natrénování sítě pomohla různorodost kvality videí.

Práce podle mého názoru proběhla úspěšně, jelikož se splnilo očekávání, že neuronová síť bude přesnější než metoda srovnávání se vzorem. Z výsledků lze vidět, že metoda srovnávání se vzorem vyžaduje velké množství manuální práce při výběru vhodných vzorů pro změny světelných podmínek v kartách. Na rozdíl od metody srovnávání se vzorem je klasifikace neuronovou sítí velice rychlá, protože nevyžaduje výběr nových vzorů při změně zdrojového videa.

Do budoucna by bylo možné síť natrénovat na ještě více setů karet. Zároveň by bylo zajímavé vytvořit novou síť na identifikaci a počítání žetonů. Tato práce by se poté dala aplikovat do systému, který by byl schopný hrát poker.

Literatura

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[2] Peter L Bartlett and Marten H Wegkamp. Classification with a reject option using a hinge loss. *Journal of Machine Learning Research*, 9(Aug):1823–1840, 2008.

[3] Adam Baumberg. Reliable feature matching across widely separated views. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 774–781. IEEE, 2000.

[4] Roberto J Bayardo Jr. Brute-force mining of high-confidence classification rules. In *KDD*, volume 97, pages 123–126, 1997.

[5] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[6] Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*. John Wiley & Sons, 2009.

[7] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[8] Rafael C Gonzalez and Richard E Woods. Thresholding. *Digital Image Processing*, pages 595–611, 2002.

[9] HDF Group et al. Hierarchical data format, version 5. 2014.

[10] Dean J Hachamovitch, Ronald A Fein, and Edward J Fries. Automatic word completion system for partially entered data, April 23 2002. US Patent 6,377,965.

[11] Robert Hecht-Nielsen et al. Theory of the backpropagation neural network. *Neural Networks*, 1(Supplement-1):445–448, 1988.

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

[13] Eric Jones, Travis Oliphant, and Pearu Peterson. {SciPy}: open source scientific tools for {Python}. 2014.

[14] Christopher Kanan and Garrison W Cottrell. Color-to-grayscale: does the method matter in image recognition? *PloS one*, 7(1):e29740, 2012.

[15] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. *Online Course*, 2016.

[16] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585, 1985.

[17] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Stanford, CA, 1995.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[20] André Marion. *Introduction to image processing*. Springer, 2013.

[21] Michael A Nielsen. Neural networks and deep learning. *URL: http://neuralnetworksanddeeplearning. com/.(visited: 01.11. 2014)*, 2015.

[22] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

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

[24] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[25] Z Šustr, J Sitera, M Mulač, M Ruda, D Antoš, L Hejtmánek, P Holub, Z Salvet, and L Matyska. Metacentrum, the czech virtualized ngi. In *EGEE Technical Forum*, 2009.

[26] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[27] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[28] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2), 2012.

[29] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[30] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[31] William M Wells, W Eric L Grimson, Ron Kikinis, and Ferenc A Jolesz. Adaptive segmentation of mri data. *IEEE transactions on medical imaging*, 15(4):429–442, 1996.

[32] Bernard Widrow and Michael A Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

[33] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[34] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.