Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

DIPLOMOVÁ PRÁCE

Automatická analýza stavu hry z vizuálních dat

PLZEŇ, 2020 **JAN HRDLIČKA**

Místo této strany bude
zadání práce.

PROHLÁŠENÍ

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

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

V Plzni 5. dubna 2020

.....

PODĚKOVÁNÍ

Děkuji panu Ing. Marku Hrúzovi, Ph.D. a panu Ing. Miroslavovi Hlaváčovi, Ph.D. za cenné rady a pomoc při řešení této práce.

Tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Anotace

Tato práce se věnuje automatické analýze hracího stolu karetní hry poker ve variantě Texas hold'em. Cílem práce je vyhodnotit stav hry ze snímku stolu: kolik hraje hráčů, kdo má jaké karty a vypočítat pravděpodobnosti výhry jednotlivých hráčů. Pro identifikaci karet byla natrénována neuronová síť s přesností 99,99% na testovacích datech. Výsledkem práce je aplikace, ve které lze buď analyzovat snímek reálného hracího stolu nebo vygenerovat a analyzovat syntetický snímek.

Klíčová slova: poker, obraz, identifikace, Python, simulace, konvoluční neuronová síť, tensorflow, pravděpodobnost

Annotation

This work is dedicated to game state analysis of the Texas hold'em card game. The aim of this work is to evaluate the game from digital data: how many players are playing, which cards the players have and calculate the probabilities of winning for every player. A neural network with an accuracy of 99.99% on test data was trained to identify the cards. The result of the work is an application in which it is possible to either analyze a real game table image or to generate and analyze a synthetic image.

Key words: poker, image, recognition, Python, convolutional, neural network, tensorflow, probability

Obsah

1 Úvod

Poker je momentálně nejhranější karetní hra na světě. V zemích, kde je poker více rozšířen, je prohlášen za dovednostní hru, kde je nutné uplatnit psychologii, statistiku a ekonomii. To, že je poker v oblasti strojového učení aktuálním tématem, dokazuje například případ umělé inteligence Libratus z roku 2018, která proti čtyřem nejlepším hráčům světa vyhrála v jediném turnaji dohromady 43 milionů korun [1].

Cílem této práce je ze snímku pokerového hracího stolu analyzovat stav hry na stole. Tedy rozpoznat pokerové karty, zjistit, komu karty patří, spočítat pravděpodobnosti hráčů na výhru a zároveň simulovat další stavy hry, které mohou ve hře nastat. K tomu jsou použity metody zpracování obrazu, konvoluční neuronové sítě a Monte-Carlo simulace. Celá práce je zpracována v programovacím jazyku Python. Ke klasifikaci karet slouží natrénovaná konvoluční neuronová síť, která je vytvořena v knihovně tensorflow. Kromě reálných karet a snímků hracích stolů používám i počítačem vygenerované pokerové karty a syntetické snímky herních stolů (syntetická data poslouží k rozšíření trénovacího datasetu pro neuronovou síť a zároveň na nich bude aplikace testována). Výsledky práce jsou vizualizovány v grafickém prostředí PyQt5.

Práce navazuje na předchozí výzkum v této oblasti [2] a je dělena do tří kapitol. První kapitola obsahuje teoretický rozbor použitých metod a současný stav poznání. Nejprve jsou představena základní pravidla Texas hold'em pokeru a způsob výpočtu pravděpodobnosti výhry jednotlivých hráčů. Jelikož se v této práci zabývám pokerem z pohledu počítače, není zde vysvětlena strategie hry ani herní psychologie. Další sekce je věnována digitálnímu obrazu a jeho zpracování. V této sekci jsou popsány nejčastěji používané barevné modely obrazu a způsoby segmentace. Poslední sekce teoretické kapitoly pojednává zejména o konvolučních neuronových sítích pro klasifikaci digitálních obrazů. Na klasických neuronových sítích jsou nejprve vysvětleny základní principy trénování neuronových sítí, které platí zároveň i pro konvoluční neuronové sítě. Konvoluční neuronové sítě se od klasických neuronových sítí odlišují zejména svými vrstvami, jejichž fungování je v této sekci popsáno. Na závěr sekce jsou zmíněny některé základní obecné předpoklady a doporučení pro klasifikaci obrazu pomocí konvolučních neuronových sítí.

Druhá kapitola je věnována praktickému návrhu pokerové aplikace pro analýzu pokerové hry z digitálního obrazu. Prvním krokem je získání hracích karet ze snímku. V druhé sekci je popsán princip návrhu syntetických hracích karet a stolů. Syntetické hrací karty společně s reálnými kartami tvoří trénovací dataset pro konvoluční neuronovou síť. O přípravě trénovacích dat pojednává třetí sekce. Pomocí těchto dat je ve čtvrté sekci natrénováno několik konvolučních neuronových sítí rozdílných architektur. V další sekci jsou popsány principy analýzy snímku pokerové hry, tzn. zjištění komu patří hrací karty a výpočet pravděpodobností hráčů na výhru a remízu dané hry. Pro vizualizaci výsledků a ukázku funkce navržených algoritmů slouží

grafické uživatelské rozhraní, jehož komponenty jsou popsány v předposlední šesté sekci. Nakonec jsou vyhodnoceny výsledky přesnosti výpočtu pravděpodobností a přesnosti klasifikace konvoluční neuronovou sítí.

Závěrečná kapitola je určena ke zhodnocení navržených metod a k úvahám o tom, zda byly metody zvolené správně. Na konci závěrečné kapitoly jsou zmíněna další možná budoucí použití této práce.

2 Teoretická část

Tato kapitola se věnuje teoretické části práce. Po vysvětlení základních pravidel Texas hold'em pokeru bude většina pozornosti směřována ke zpracování digitálního obrazu a ke strojovému učení v podobě konvolučních neuronových sítí.

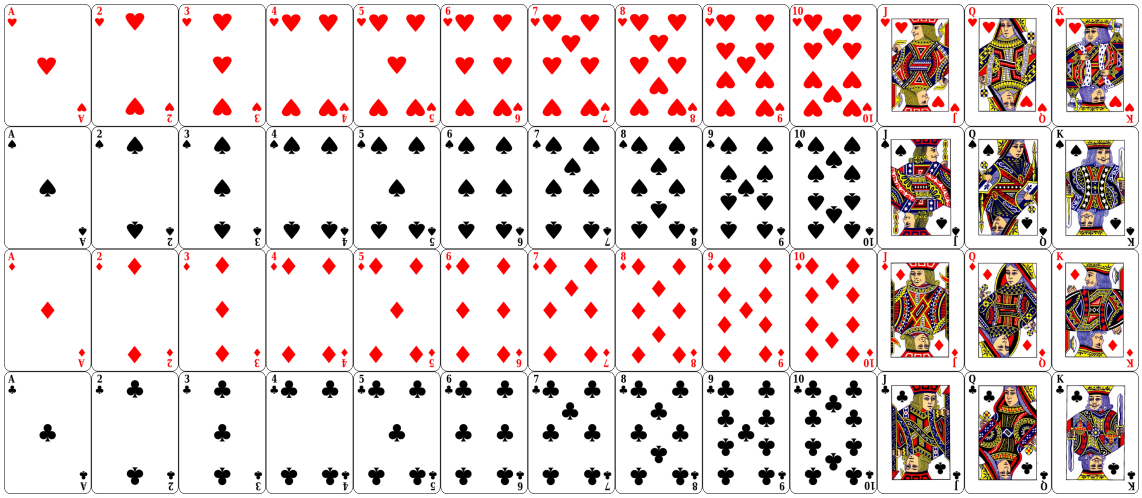
2.1 Texas hold'em poker

Poker se hraje v mnoha variantách, které se od sebe často silně odlišují. Zdaleka nejrozšířenější z těchto variant je Texas hold'em poker. Předpokládám, že čtenář tuto práci nečte primárně kvůli pravidlům pokeru a proto zde budou zmíněny pouze informace, které používám v praktické části této práce při analýze stavu pokerové hry. Cílem této sekce je podat základní informace o skládání pokerových kombinací, díky kterým jsou počítány šance hráčů na výhru. Není zde popsán systém vsázení, počet žetonů či herní strategie. Z hlediska výzkumu pokeru jako hry jsou stěžejní tyto jeho vlastnosti [3]:

1. **Hra nedokonalých informací** (tzv. Bayesovská hra): Hráči nemají kompletní informace o situaci, ve které se nacházejí.
2. **Nedeterministický průběh hry**: Karty jsou rozdávány náhodně.
3. **Částečná pozorovatelnost**: Karty protivníků jsou hráči neznámé.
4. **Hra více hráčů**: Tato hra je určená pro dva a více hráčů. Standardně se hraje v počtu deseti hráčů.
5. **"Zero sum"hra**: Hra s nulovým součtem. Peníze či žetony, které jeden hráč získá, druhý hráč ztrácí.

2.1.1 Hrací karty

Hrací balík obsahuje 52 karet. Celkem se v hracím balíku nachází 4 druhy barev a 13 hodnot. Každá karta má svou unikátní kombinaci barvy a hodnoty. Barvou se rozumí symboly na kartě; kříže ♣, káry ♢, srdce ♡ a piky ♠. Sílu karty ovlivňuje pouze její hodnota. Seřazené hodnoty od nejslabší po nejsilnější vypadají následovně: 2, 3, ..., 10 (častěji označována jako T), kluk J, dáma Q, král K a eso A.



Obrázek 2.1: Všechny hrací karty

2.1.2 Cíl hry

Na začátku každé hry se každému hráči rozdají 2 náhodné karty, které si hráči mezi sebou neukazují. Doprostřed stolu se postupně vyloží 5 náhodných karet. Tyto karty hráči sdílí. Karty se vykládají na stůl v průběhu hry, který je následující:

1. **Preflop**: na stole není žádná karta
2. **Flop**: vyloží se první tři společné karty
3. **Turn**: vyložení čtvrté společné karty
4. **River**: je vyložena poslední společná karta

Hráči soutěží o množství peněz nebo žetonů, které vsázejí na začátku zmíněných stavů hry do tzv. banku (anglicky pot). Hráči mají ve většině případů na výběr ze tří strategií: vsadit, skončit ve hře či nedělat nic (tato strategie je možná pouze pokud nikdo z ostatních hráčů nevsadil). Na konci každé hry - po riveru, se z kombinace hráčových karet a karet na stolu (tedy z dohromady sedmi karet) vybere pět karet, které tvoří nejsilnější kombinaci (tzv. ruka). Hráč s nejsilnější kombinací karet vyhrává bank (speciálními případy mohou být stejně silné kombinace více hráčů či pokud ostatní hráči složili karty ještě před vyhodnocením hry). Po rozdělení banku začíná další hra. Hráčům se rozdají nové karty, na stůl se vyloží 5 neznámých karet a proces se opakuje. Celkovým vítězem je hráč, který získal od ostatních hráčů všechny jejich žetony či peníze.

Cílem zkušenějších hráčů není vyhrát každou jednotlivou hru, ale spíše učinit matematicky a psychologicky nejlepší rozhodnutí o tom, kdy a kolik vsadit, kdy sázku dorovnat a kdy karty složit. Zajímavostí pokeru je právě jeho psychologická stránka. Hráč nemusí volit strategii podle síly své ruky. Svým chováním však může ostatní přesvědčit, že jeho kombinace karet je silná a vyhrát hru tak, že výší své sázky donutí ostatní hráče složit karty.

2.1.3 Karetní kombinace

V předchozí sekci 2.1.2 bylo zmíněno, že každý hráč dostane dvě karty a navíc je na stůl postupně rozdáno pět společných karet. Karty v hráčově ruce a karty na stole tvoří kombinace s určitou silou. Vyhodnocuje se vždy nejsilnější kombinace pěti karet. Karty lze kombinovat jakkoli. Může tedy nastat i případ, že hráčovou nejsilnější kombinací je právě pět karet na stole. Počet možných kombinací závisí na stavu hry. Ve stavu flop sdílí hráči pouze 3 karty, tzn. lze vytvořit pouze jednu kombinaci o velikosti pěti karet (3 karty na stole a 2 hráčovy karty). V dalším stavu turn už lze vytvořit 6 kombinací (výběr pěti karet z celkových šesti). Síly kombinací se vyhodnocují vždy po posledním stavu river, kde hráči mají pro tvorbu své nejsilnější kombinace k dispozici celkem 7 karet (5 karet na stole a 2 hráčovy karty). Níže je vypočten počet kombinací pěti karet, které lze ze sedmi celkových karet vytvořit:

$$C(5,7) = \frac{7!}{(7-5)! \cdot 5!} = 21. \tag{2.1}$$

Čím nižší je pravděpodobnost, že daná kombinace nastane, tím je karetní kombinace hodnotnější a silnější. V tabulce 2.1.3 níže uvádím všechny výherní kombinace spolu s pravděpodobnostmi jejich výskytu.

Název	**Příklad**	**Výskyt [%]**
Královská postupka	10 ♡, J ♡, Q ♡, K ♡, A ♡	0,00015
Postupka v barvě	4 ♣, 5 ♣, 6 ♣, 7 ♣, 8 ♣	0,0013
Čtveřice	7 ♣, 7 ♡, 7 ♢, 7 ♠, 3 ♣	0,024
Full house	K ♣, K ♡, K ♢, 8 ♠, 8 ♣	0,144
Flush	A ♢, 8 ♢, 3 ♢, 5 ♢, Q ♢	0,197
Postupka	2 ♢, 3 ♣, 4 ♠, 5 ♠, 6 ♡	0,395
Trojice	K ♣, K ♢, K ♠, J ♠, 9 ♡	2,17
Dva páry	A ♡, A ♢, 4 ♢, 7 ♠, 6 ♠	5
Pár	5 ♠, 5 ♣, Q ♡, T ♢, 3 ♢	73,53
Vysoká karta	5 ♢, 2 ♠, Q ♠, T ♣, A ♢	100

Tabulka 2.1: Sestupně seřazené karetní kombinace podle jejich síly

Aby mohly některé kombinace nastat, je zapotřebí všech pěti karet. Tak je to například u postupky, kde je potřeba pěti po sobě jdoucích hodnot. Pokud pro výskyt některé kombinace není zapotřebí všech pěti karet, poté je tato kombinace doplněna kartami s nejvyšší hodnotou, aby se maximalizovala síla ruky. Pro představu uvádím v tabulce několik příkladů:

Hráč	**Karty na stole**	**Nejsilnější kombinace**	**Popis**
J ♡, 6 ♣	2 ♠, Q ♢, J ♢, 3 ♠, 9 ♣	J ♡, 6 ♣, Q ♢, J ♢, 9 ♣	Pár
J ♡, 6 ♣	6 ♢, 6 ♠, A ♢, K ♠, 2 ♠	6 ♣, 6 ♢, 6 ♠, A ♢, K ♠	Trojice
J ♡, 6 ♣	A ♢, K ♠, A ♡, K ♢, Q ♠	A ♢, K ♠, A ♡, K ♢, Q ♠	Dva páry
J ♡, 6 ♣	2 ♠, 7 ♢, A ♢, 3 ♠, 9 ♣	J ♡, 6 ♣, 7 ♢, A ♢, 9 ♣	Vysoká karta

Tabulka 2.2: Příklady tvorby karetních kombinací

V prvním řádku tabulky 2.1.3 je hráčovou nejsilnější kombinací pár kluků $J\heartsuit$, $J\diamondsuit$. Tento pár je doplněn třemi kartami s nejvyššími hodnotami, tzn. 6 ♣, $Q\diamondsuit$, 9 ♣. Na druhém řádku je využita pouze jedna karta z hráčovy ruky 6 ♣, která společně s dvěma šestkami na stole 6 ♠, $6\diamondsuit$ tvoří trojici. Nejsilnější rukou je kombinace této trojice s kartami $A\diamondsuit$, K ♠. Třetí řádek reprezentuje případ, kdy nejsilnější kombinaci tvoří pouze karty na stole. Ani jedna hráčova karta netvoří kombinaci s kartami na stole a zároveň nemají hráčovy karty dost vysokou hodnotu a tudíž v nejsilnější kombinaci nejsou. Na posledním řádku lze vidět nejslabší karetní kombinaci vysoké karty. V tomto případě je vybráno 5 karet s nejvyšší hodnotou.

Pokud mají hráči stejně silné kombinace, potom vyhrává hráč s nejvyšší kartou, která se v jeho kombinaci vyskytuje (pokud jsou i tyto karty stejné, rozhoduje druhá nejvyšší karta atd.). Bank se dělí v případě, že hráči mají stejně silné kombinace a zároveň stejné hodnoty karet.

Eso (A) může v postupce figurovat i jako hodnota 1 (jinak je eso nejvyšší možnou hodnotou), tedy lze z něj složit i následující postupku: $A\heartsuit$, $2\diamondsuit$, 3 ♣, 4 ♠, 5 ♠.

2.1.4 Počítání pravděpodobností na výhru

Přesné tabulkové vyjádření pravděpodobností na výhru zatím neexistuje ani pro hru dvou hráčů (tzv. heads-up) [4]. Je to dáno tím, že stavový prostor této hry je příliš rozsáhlý (stavovým prostorem jsou myšleny všechny stavy hry, které mohou nastat). Pomocí abstrakcí a všeobecných pouček lze stavový prostor podstatně zúžit. Ani to však nestačí k přesnému popsání hry.

	A	K	Q	J	T	9	8	7	6	5	4	3	2
A	85%	68%	67%	66%	66%	64%	63%	63%	62%	62%	61%	60%	59%
K	66%	83%	64%	64%	63%	61%	60%	59%	58%	58%	57%	56%	55%
Q	65%	62%	80%	61%	61%	59%	58%	56%	55%	55%	54%	53%	52%
J	65%	62%	59%	78%	59%	57%	56%	54%	53%	52%	51%	50%	50%
T	64%	61%	59%	57%	75%	56%	54%	53%	51%	49%	49%	48%	47%
9	62%	59%	57%	55%	53%	72%	53%	51%	50%	48%	46%	46%	45%
8	61%	58%	55%	53%	52%	50%	69%	50%	49%	47%	45%	43%	43%
7	60%	57%	54%	52%	50%	48%	47%	67%	48%	46%	45%	43%	41%
6	59%	56%	53%	50%	48%	47%	46%	45%	64%	46%	44%	42%	40%
5	60%	55%	52%	49%	47%	45%	44%	43%	43%	61%	44%	43%	41%
4	59%	54%	51%	48%	46%	43%	42%	41%	41%	41%	58%	42%	40%
3	58%	54%	50%	66%	45%	43%	40%	39%	39%	39%	38%	55%	39%
2	57%	53%	49%	47%	44%	42%	40%	37%	37%	37%	36%	35%	51%

Obrázek 2.2: Bucketing: přibližné vyjádření pravděpodobnosti na výhru [5]

Pokeroví hráči využívají jako metodu abstrakce tzv. bucketing. Bucketing seskupuje strategicky stejné kombinace karet do tříd (bucketů) podle jejich síly. Například,

hráčovy karty $A\heartsuit$, $A\clubsuit$ jsou na začátku hry strategicky naprosto stejné, jako kdyby místo těchto karet dostal $A\spadesuit$, $A\diamondsuit$. Tímto způsobem lze hru zjednodušit bez ztráty nejdůležitějších informací [6]. Obrázek 2.2 výše ukazuje příklad použití bucketingu při začátku hry. Obrázek 2.2 popisuje přibližné šance na výhru, dostane-li hráč určitou dvojici karet. Prvním záznamem s hodnotou 85% je případ, že hráč dostal dvakrát eso. Pokud hráč dostane eso a krále, jeho pravděpodobnost na výhru je 66%. Zmíněný obrázek nebere v úvahu možnost, že se barvy hráčových karet shodují, v takovém případě se šance na výhru zvyšuje (je to zapříčíněno tím, že může pravděpodobněji nastat případ pěti stejných barev - viz. pátý řádek v tabulce 2.1.3). Zároveň jsou často používány obecné poučky typu: "pravděpodobnost, že vyhraji, pokud na začátku hry dostanu pár a protivník má pár s nižší hodnotou, je přibližně 80%" [5].

V pozdějších stavech hry se současně s bucketingem používá počítání tzv. outs. Mezi outs se řadí každá karta, která dokáže vylepšit sílu hráčovy kombinace tak, že se bude jednat o nejsilnější možnou karetní kombinaci. Taková karta se nazývá overcard. Díky znalosti počtu zbývajících karet a počtu outs může hráč spočítat pravděpodobnost, s jakou v dalším odkrytí karet dostane do kombinace kartu, která zvyšuje sílu jeho ruky a tím zlepšuje jeho herní pozici.

Jak bylo zmíněno dříve v této sekci, stavový prostor je i přes tyto abstrakce velice rozsáhlý. V tabulce 2.1.4 je znázorněn počet možných stavů pro hru dvou hráčů. Lze vidět, že i po použití bucketingu dosahuje stavový prostor v pozdějších stavech hry velice mnoha stavů. Se stoupajícím počtem hráčů se zároveň exponenciálně rozšiřuje stavový prostor hry.

Stav hry	**Dva hráči**	**Jeden hráč**	**Jeden hráč - bucketing**
Preflop	1 624 350	1 326	169
Flop	28 094 757 600	25 989 600	1 286 792
Turn	1 264 264 092 000	1 221 511 200	55 190 538
River	55 627 620 048 000	56 189 515 200	2 428 287 420

Tabulka 2.3: Stavový prostor pro hru dvou hráčů v Texas hold'em, převzato z [7]

Z výše zmíněných poznatků je zřejmé, že se pravděpodobnosti na výhru musí odhadovat. Pro odhad pravděpodobnosti v pokeru pomocí počítače se používá široké spektrum přístupů [4]. Například:

- **Znalostní či expertní systémy**: Používají se 2 základní přístupy. Aplikace If-then pravidel nebo ocenění síly kombinací karet numerickou hodnotou (velice podobné bucketingu). K tvorbě pravidel je zapotřebí pokerový expert [8].
- **Neuronové sítě a evoluční algoritmy**: Použití umělé inteligence pro zjištění dalšího průběhu hry s využitím samoučících se algoritmů [9].
- **Monte-Carlo simulace**: Simulace mnoha náhodných her, které mohou pro současný stav v budoucnu nastat. Výpočet pravděpodobnosti probíhá tak, že je simulováno několik stovek až tisíců her, ve kterých je vyhodnocováno kdo

vyhraje. Celková pravděpodobnost na výhru se po provedení mnoha simulací spočítá následovně:

$$\text{Hráčova pravděpodobnost výhry} = \frac{\text{Počet výher hráče}}{\text{Počet simulací}} \tag{2.2}$$

Časová náročnost je oproti ostatním přístupům vyšší. Monte-Carlo simulace využívá drtivá většina online poker kalkulátorů [10].

- **Bayesovské sítě**: Vytvoření pravděpodobnostního modelu, který využívá grafovou reprezentaci pro zobrazení pravděpodobnostních vztahů mezi jednotlivými jevy [11].
- **Hledání optimální strategie pomocí Nashovy rovnováhy**: Nashova rovnováha je optimální strategií pro hru dvou hráčů. Tzn. pokud se hráč nedrží optimální strategie, pohorší si (nebo v nejlepším případě na tom bude stejně). Každá konečná hra dvou hráčů má alespoň jednu Nashovu rovnováhu (konečná hra dvou hráčů s nulovým součtem má vždy právě jednu Nashovu rovnováhu) [12]. Optimální strategie se hledá pomocí algoritmu, který hraje sám se sebou (algoritmus je často označován jako CFR či Counterfactual regret minimization) [13]. Metodu CFR+ používá například v úvodu zmíněný superpočítač Libratus [1]. Pro hru třech a více hráčů může existovat více Nashových rovnováh, oproti hře dvou hráčů však rovnováhy nezaručují nejlepší výsledek. Zatím není známo, jestli pro poker se třemi a více hráči existuje "perfektní strategie" [14].

Obtížnost návrhu výše zmíněných přístupů je ve většině případech ovlivněna spíše programátorskými a matematickými schopnostmi než pokerovými znalostmi řešitele. Vyjímkou z výše uvedených přístupů mohou být expertní systémy, při jejichž návrhu je pro tvorbu rozhodovacích pravidel zapotřebí výborných pokerových znalostí. Z hlediska programátorské a matematické obtížnosti se jako nejjednodušší jeví Monte-Carlo simulace, u kterých je pouze zapotřebí správným způsobem naprogramovat simulaci hry.

2.2 Digitální obraz a jeho zpracování

V této sekci bude nejprve vysvětleno, jak je obraz v počítači uložen a poté se představí metody jeho zpracování, které budou použity v praktické části za účelem vyjmutí hrací karty ze snímku.

2.2.1 Barevné modely obrazu

Digitální obraz je tvořen mnoha pixely (pixel je elementární prvek digitálního obrazu). Pixely jsou typicky organizovány ve dvourozměrném pravoúhlém poli. Šířka a

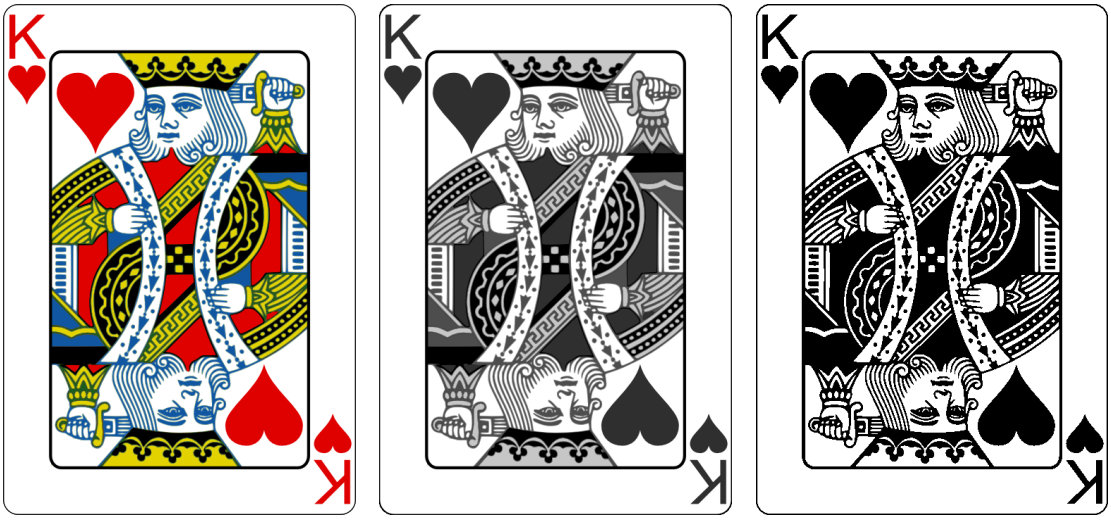
výška obrazu určují velikost tohoto dvourozměrného pole. Šířka obrazu je reprezentována počtem sloupců pole a výška je dána počtem řádek pole. Souřadnice (0,0) se v digitálním obrazu nachází v levém horním rohu, tzn. řádky pole jsou číslovány shora dolů.

Každý pixel v obraze má určitou intenzitu jasu. Intenzita rozhoduje o tom, jak bude pixel vypadat (jakou bude mít barvu). Pro šedotónový obraz se intenzita uvádí jako číslo od nuly do 255, kde nula reprezentuje černou barvu a 255 bílou barvu. Čísla mezi nulou a 255 jsou odstíny těchto barev. Například čísla blíže nule budou mít tmavý odstín. Speciálním případem šedotónového obrázku je binární obraz, u kterého mohou mít pixely pouze dvě hodnoty intenzit; černou a bílou. Pro barevný obrázek se intenzita pixelu uvádí nejčastěji pomocí tří barevných kanálů RGB (červená, zelená, modrá), kde je pro každý barevný kanál vyhrazeno jedno dvourozměrné pole. Smícháním intenzit z těchto tří kanálů vznikne výsledná barva pixelu (intenzity se pro RGB model míchají aditivně).

Váženou sumou intenzit barevných kanálů jednotlivých pixelů lze barevný snímek převést na snímek šedotónový [15] následujícím předpisem:

$$Y = 0,21R + 0,72G + 0,07B \tag{2.3}$$

kde Y je výsledný šedotónový pixel a R, G, B jsou intenzity barevných kanálů pixelu. Lidské oko je citlivější na zelenou barvu a proto je barevný kanál G zvýhodněn vyšší vahou. Binární obraz lze z šedotónového obrazu vytvořit tzv. prahováním, které je zmíněno v další sekci věnované segmentaci obrazu.



(a) Barevný obraz (b) Šedotónový obraz (c) Binární obraz

Obrázek 2.3: Digitální obraz v různých barevných modelech

Další možností reprezentace barevného obrazu je model RGBA, který kromě intenzit tří barevných kanálů obsahuje i tzv. alfa kanál, který uchovává informaci o průhlednosti pixelu (digitální obraz je tedy definován dohromady čtyřmi poli, tři pole uchovávají intenzity barev kanálů a čtvrté pole definuje průhlednost pixelu).

Hodnoty v alfa kanálu mohou nabývat velikosti od 0 do 1, kde 0 označuje maximální transparentnost a 1 neprůhlednost pixelu. Obraz v populárním formátu JPEG transparentnost nepodporuje. Pro zachování transparentnosti se obraz nejčastěji ukládá do formátů PNG či GIF [16].

2.2.2 Segmentace obrazu

Segmentace obrazu se používá k rozdělení obrazu do oblastí s podobnými vlastnostmi. Hlavním cílem segmentace je upravit obraz tak, aby se dal později snáze analyzovat [17]. Analýzou je zde myšleno například zjištění počtu objektů v obraze, zjištění délky určitého objektu, zjištění pixelů náležícím pozadí atp.

Segmentace obrazu může být obecně kategorizována dvěma přístupy:

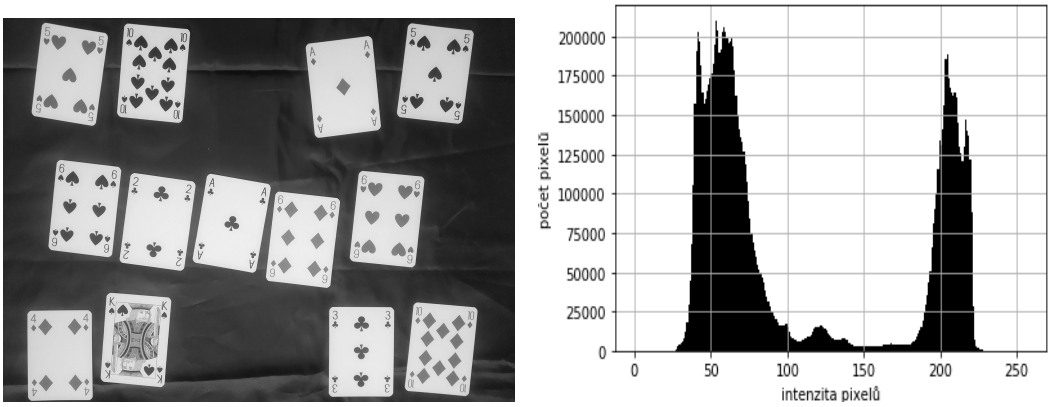
1. **Detekce nespojitostí** (boundary-based): Oblasti jsou rozděleny podle nespojitostí v obraze. Nespojitostí v obraze je myšleno místo, kde se skokově mění jas, barva, textura či hloubka. Do této kategorie spadá například hranová detekce.

2. **Detekce podobností** (region-based): Rozdělení obrazu do oblastí s podobnými vlastnostmi (místa mohou být podobná jasem, barvou, texturou či hloubkou). Tento přístup využívá například metoda prahování či segmentace narůstáním oblastí.

Segmentace prahováním

Prahování je nejjednodušší metodou segmentace obrazu. Princip prahování spočívá v analýze intenzit pixelů. Intenzity pixelů v obrazu se porovnávají s konstatní intenzitou (tato konstanta je nazývána prahem). Pokud je intenzita pixelu nižší než práh, je pixel prohlášen za součást pozadí. Pokud je intenzita pixelu vyšší než práh, pixel patří objektu v popředí [17]. Tedy defaultně jsou tmavé pixely prohlášeny za pozadí a světlé pixely jsou objekty. Prahování však funguje i pro případ, kdy je pozadí světlé a objekty v obraze mají tmavé pixely. Výsledkem segmentace takového obrazu bude snímek s inverzními hodnotami k snímku s tmavým pozadím a světlými objekty, tzn. pro úspěšnou segmentaci závisí pouze na tom, aby byly intenzity pixelů objektů dobře odlišitelné od intenzit pixelů pozadí.

Četnost intenzity pixelů v obrazu přehledně zobrazuje histogram. Pro každou intenzitu je do histogramu vnesena přímka, jejíž výška je dána tím, kolikrát se daná intenzita v obraze vyskytuje. Histogram, kde se pravděpodobně nachází třída pozadí a popředí, se nazývá bimodální a je tvořen dvěma shluky intenzit, podobně jako na obrázku 2.4(b). Levý shluk představuje tmavé pixely, v tomto případě to jsou tmavé pixely pozadí a znaky na kartách. Pravý shluk jsou světlé pixely karet.



(a) Šedotónový obraz (b) Histogram šedotónového obrazu

Obrázek 2.4: Šedotónový obraz a jeho histogram

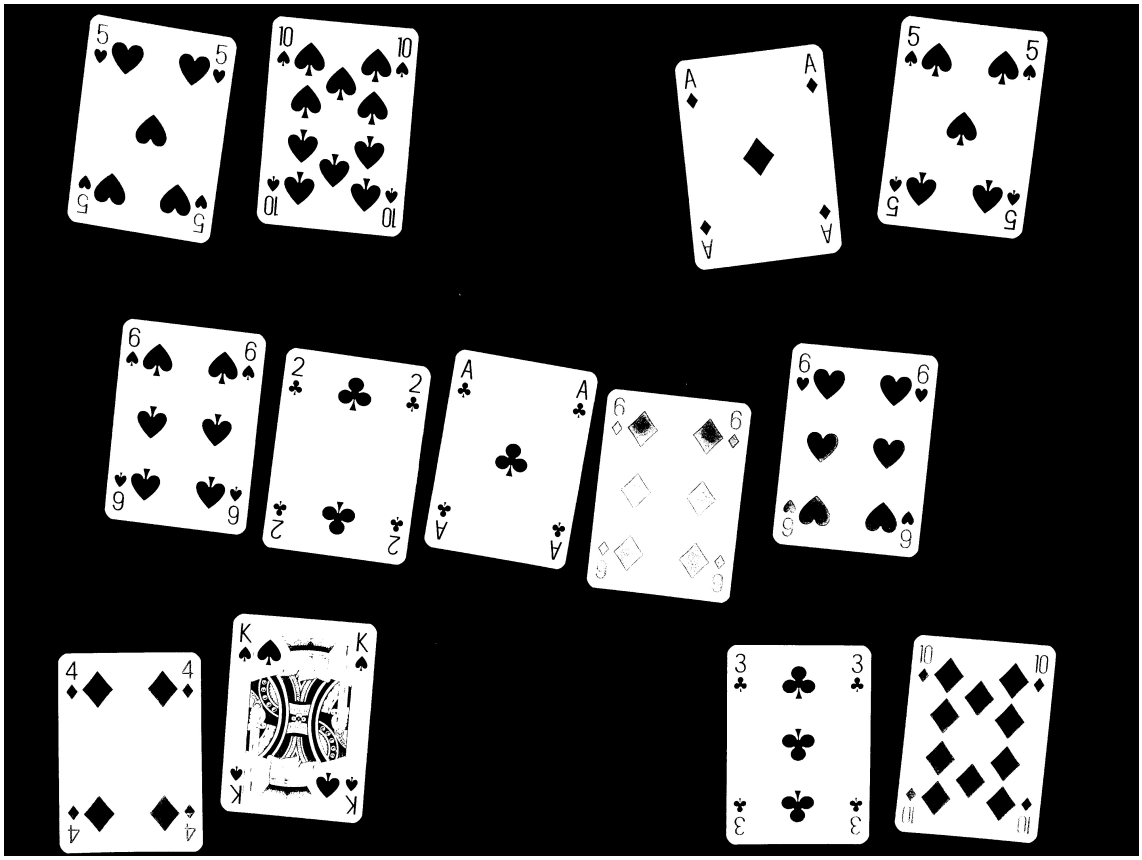
Prahování lze rozdělit podle způsobu nastavování prahu:

1. **Globální prahování**: Nastavení konstatního prahu pro celý snímek. Jako nejlepší se pro obrazy s bimodálním histogramem jeví metoda Otsu (také známa jako metoda optimálního prahování) [18]. Algoritmus nalezne automaticky práh, který z histogramu minimaluzuje vážený rozptyl dvou tříd jasů. Na obrázku 2.4(b) by metoda Otsu nastavila práh na intenzitu 132.

2. **Prahování s proměnným prahem**: Metoda je známá také pod názvem adaptivního prahování. Tato metoda posouvá obrazem okno a počítá práh pouze pro oblast v okně. Prahování poté probíhá v obrazu pouze lokálně, jelikož pro každý posun okna mohl být nalezen jiný práh. Tato metoda je používána v případěch, kde se kvůli světelným podmínkám mění intenzity pixelů v obrazu [19].

3. **Několikanásobné prahování**: Touto metodou lze rozdělit obraz do více než dvou vrstev. Vrstvy se vytvoří tak, že se obraz postupně prahuje od nejnižšího až po nejvyšší práh. Metoda se používá pro obrazy s tzv. multimodálním histogramem [20]. Takový histogram obsahuje 3 a více shluků, které obsahují vysoký počet pixelů.

Hlavní výhodou metody prahování je její jednoduchost a rychlost. Nevýhodou prahování je závislost na tvaru histogramu. Pokud by například segmentovaný obraz měl plochý histogram bez "peaků", metoda prahování by pravděpodobně neposkytla dobré výsledky kvůli neoptimálně zvolenému prahu.



Obrázek 2.5: Segmentovaný snímek metodou Otsu

Segmentace detekcí hran

Hrany jsou místa v obrazu, kde dochází k ostré nespojitosti, většinou v intenzitě jasu. Hrany v obraze většinou korespondují s hranami reálných objektů (hrana v digitálním obraze však může být způsobena například i odrazem světla či stínem). Hrany v obraze se nachází mezi dvěma homogenními regiony [21]. Pro hranovou detekci je důležitá definice gradientního operátoru (pojem gradient bude rovněž používán v sekci 2.3.6).

Gradient obrazu $f(x, y)$ v souřadnicích (x, y) je ve spojitém případě vektor podílu parciálních derivací:

$$\nabla f = [\frac{G_x}{G_y}] = [\frac{\delta f/\delta_x}{\delta f/\delta_y}]. \tag{2.4}$$

Gradientní vektor ∇f směřuje do místa s maximální změnou funkce $f(x, y)$. Velikost této změny je počítána jako:

$$|\nabla f| = \sqrt{{G_x}^2 + {G_y}^2}, \tag{2.5}$$

nebo často její aproximací:

$$|\nabla f| = |G_x| + |G_y|. \tag{2.6}$$

Pro nalezení hran v digitálním obraze se používají tzv. masky, pomocí kterých se derivace aproximují diferencemi (je to dáno tím, že obraz v počítači je reprezentován diskrétně a ne spojitě). Element v masce ukazuje, jakou váhu má pixel v obrazu. Maska provádí s obrazem tzv. konvoluci.

Příklad níže by měl lépe vysvětlit funkci masky. V tomto případě se jedná o detekci svislé hrany. To se dá obecně poznat podle toho, jakým způsobem jsou v masce umístěny nuly. Zde jsou nuly umístěny svisle.

$$\underbrace{\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}}_{\text{Pixely obrazu}} \qquad \underbrace{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}}_{\text{Maska}}$$

Výsledek konvoluce masky s obrazem by pro tento případ vypadal následovně:

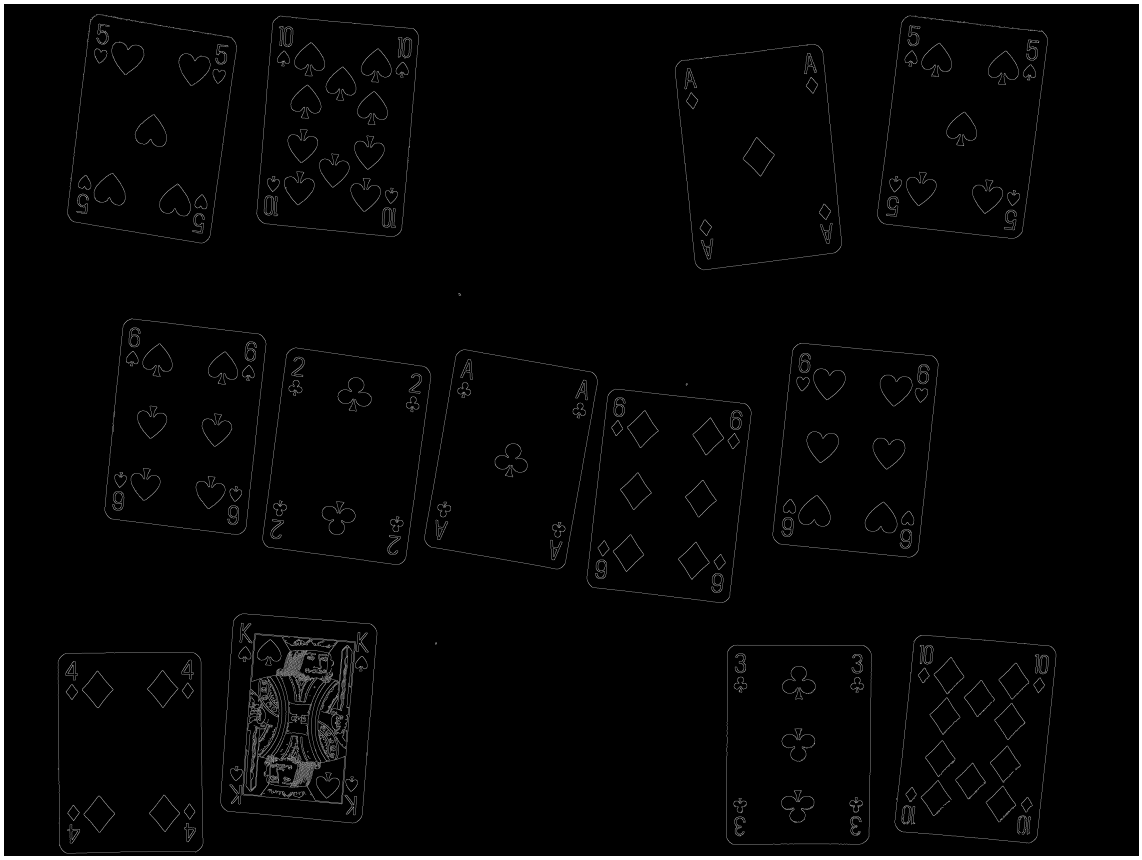
$$R = -x_1 + x_3 - 2x_4 + 2x_6 - x_7 + x_9,$$

Pixely digitálního obrazu jsou prohlášeny za hranové pokud:

$$|R| > T,$$

kde T je předem určený práh. Maska se pixel po pixelu posouvá obrazem. Tím se spočítají čísla R pro všechny regiony v obraze. Porovnáním těchto čísel s prahem se získají informace o všech vertikálních hranách v obrazu. Pro detekci horizontálních hran by se maska otočila o 90° tak, aby nuly byly horizontálně a proces by se opakoval. Typ masky výše se nazývá Sobelův operátor. Dalšími často používanými operátory jsou například Roberts či Prewitt [21].

Pro detekci hran v zašuměném obrazu se používá Canny hranový detektor [21], který před aplikací hranových operátorů obraz vyhladí (vyhlazení obrazu je inverzním postupem k doostření obrazu). Obraz se vyhlazuje pomocí filtru, který se opět posouvá obrazem a konvolvuje s ním stejným způsobem, jako je ukázáno v příkladu výše, maska však obsahuje jiná čísla. Tím je spočtena nová intenzita, jejíž velikost je ovlivněna okolními pixely. Nejčastěji používaným filtrem je Gaussův filtr [21].



Obrázek 2.6: Canny hranová detekce

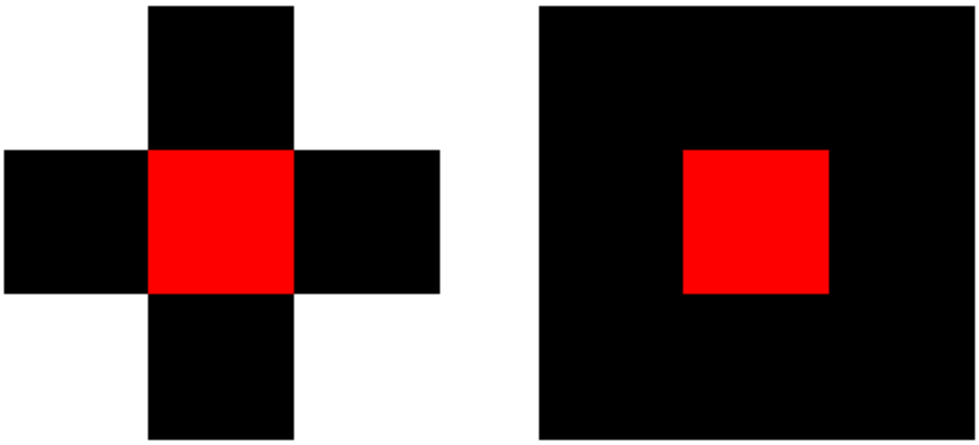
Hranové detektory jsou vhodné pro obrazy, na kterých je mezi objekty vysoký kontrast. Na rozdíl od metody prahování není pro detekci hran důležitý tvar histogramu. Nevýhodou hranových detektorů je vyšší citlivost na šum, což lze mírně vidět i z obrázku 2.6.

Segmentace podle oblastí

Tato metoda segmentuje obraz do regionů s podobnými charakteristikami (nejčastěji opět do regionů s podobnými intenzitami). Existují dva přístupy segmentace podle oblasti:

1. **Segmentace narůstáním oblastí** (region growing): Oblasti narůstají z počátečních pixelů v obrazu, které jsou buď zvoleny manuálně (pokud jsou k dispozici informace o obrazu) nebo automaticky. Pokud počáteční pixel sousedí s pixely, které mají stejné vlastnosti, sloučí se tyto pixely do spojité oblasti. Pro každý pixel v této oblasti se poté znovu kontrolují vlastnosti sousedních pixelů. Výsledkem narůstání oblasti je homogenní oblast s maximálním možným obsahem [22]. Určení, jak spolu pixely sousedí se nazývá sousednost či okolí. Okolí v tomto algoritmu určuje, jakým způsobem budou regiony růst. Nejčas-

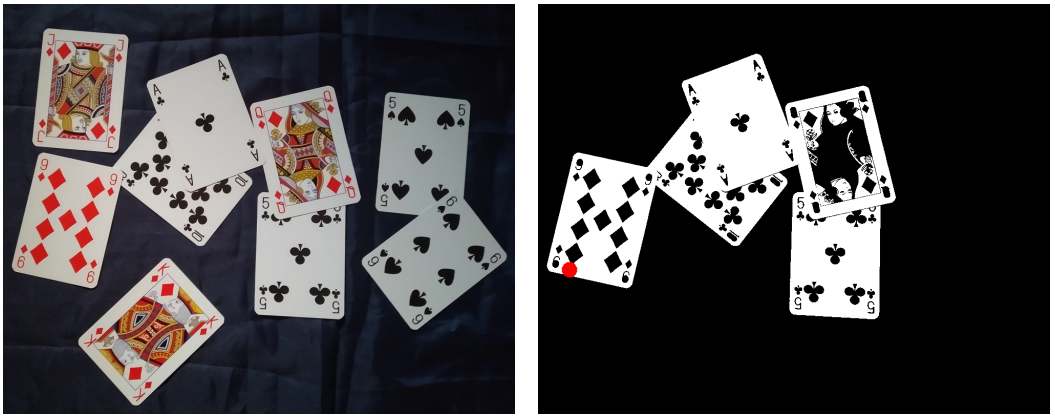
těji se používá tzv. 8-okolí, tzn. zkoumaný pixel (na obrázcích 2.7 vyznačen červeně) sousedí s 8 pixely.



(a) 4-okolí (b) 8-okolí

Obrázek 2.7: Okolí pixelů

Červená značka na obrázku 2.8(b) označuje místo počátečního pixelu, ze kterého oblast narůstala. Segmentovalo se pouze 5 spojených karet tvořící maximální možnou homogenní oblast. Zbylé karty neobsahují pixel, který sousedí s touto oblastí, proto nejsou ve snímku přítomny.



(a) Digitální obraz (b) Obraz po segmentaci narůstáním oblastí

Obrázek 2.8: Segmentace narůstáním oblastí

2. **Segmentace štěpením a sjednocením** (Split and merge): Tato metoda iterativně štěpí obraz do stejně velkých regionů a poté, pokud mají sousední regiony podobné vlastnosti, jsou tyto regiony sjednoceny. Regiony se štěpí do té doby, dokud už není možné regiony alespoň jednou sjednotit.

Metody segmentace podle oblastí jsou účinné, pokud se dají jednoduše definovat kritéria podobnosti regionů. Oproti metodám prahování a hranové detekce není

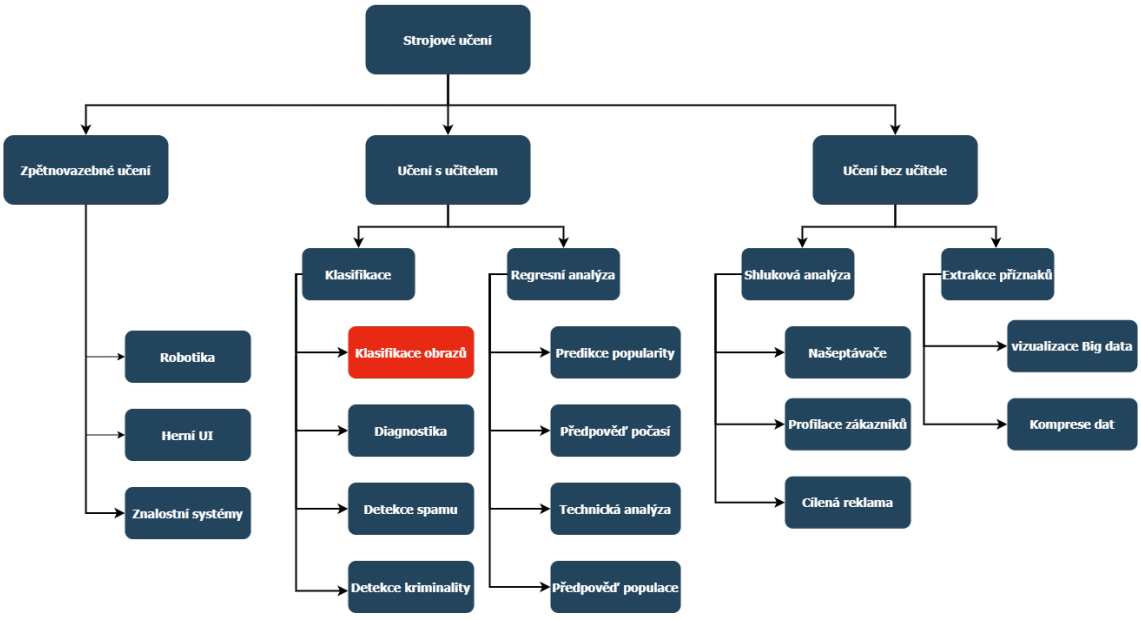
segmentace podle oblastí tak citlivá na šum. Segmentace narůstáním oblastí je využívána zejména v medicíně pro detekci nádorů a analýzu snímků z počítačové tomografie [23] [24]. Pro obrazy s vyšším rozlišením může být tento způsob segmentace zdlouhavý a paměťově náročný [21].

Segmentace pomocí konvolučních neuronových sítí

Speciálním odvětvím segmentace obrazů je segmentace pomocí konvolučních neuronových sítí. Umělá inteligence dokáže lépe generalizovat řešený problém než pevně naprogramované algoritmy a tudíž je segmentace těmito metodami vhodná téměř pro každý složitější úkol [25]. Nadcházející sekce je věnována právě tématu neuronových sítí.

2.3 Neuronové sítě

Hlavní myšlenkou neuronových sítí je modelovat chování lidského mozku pomocí matematiky. Často používaným pojmem je tzv. strojové učení, kde se umělá inteligence adaptuje podle řešené úlohy. Obrázek 2.9 znázorňuje široké spektrum využití metod strojového učení, ve kterých samozřejmě hrají velice důležitou roli právě neuronové sítě.



Obrázek 2.9: Mapa použití strojového učení

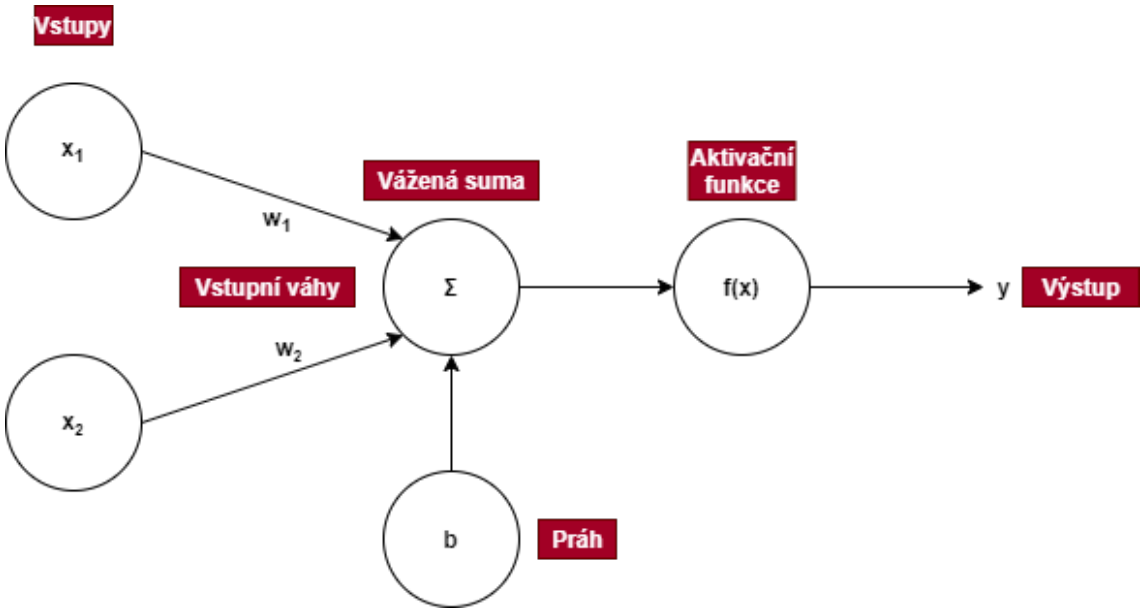
Neuronové sítě dokáží díky jejich podobnosti s lidským mozkem řešit nelineární a složité problémy, na které by bylo použití klasických počítačových algoritmů často nedostačující. S tím však souvisí složitost tohoto tématu. Vzhledem k obrovskému množství parametrů nelze ze sítí snadno získat další informace, než ty, které síť

poskytne na svém výstupu. Kvůli tomu jsou neuronové sítě často označovány za tzv. black-box, se kterým je třeba manipulovat specifickým způsobem.

V této části práce se zaměřím zejména na sítě určené hlavně pro klasifikaci digitálního obrazu, tedy na konvoluční neuronové sítě. Jak lze vidět z obrázku 2.9, jedná se o disciplínu učení s učitelem, kde se konvoluční neuronové síti předkládají předem známé vzory, kvůli kterým síť mění své matematické nastavení a tím dokáže dojít k řešení dané úlohy. Úvod této kapitoly bude věnován klasickým neuronovým sítím, ze kterých konvoluční neuronové sítě z části vychází.

2.3.1 Perceptron

Jedná se o nejjednodušší neuronovou síť tvořenou pouze jedním umělým neuronem. Obrázek 2.10 ukazuje strukturu perceptronu. Vstupy x_1 a x_2 reprezentují dendrity reálného neuronu. Síly synapsí reálného neuronu jsou vyjádřeny vahami w_1 a w_2. Vážená suma Σ přestavuje tělo neuronu, ve kterém je soustředěna energie ze vstupů. Po překročení prahu b vyšle neuron ze svého těla signál (v reálném neuronu je elektrický signál šířen po axonu) na výstup ve formě aktivační funkce. V praxi používané neuronové sítě obsahují stovky milionů těchto neuronů (neuronové sítě se stejným počtem neuronů jako obsahuje lidský mozek, tedy 86 bilionů neuronů, by podle Moorova pravidla mohly být navrženy kolem roku 2035 [26]).



Obrázek 2.10: Model perceptronu se dvěma vstupy

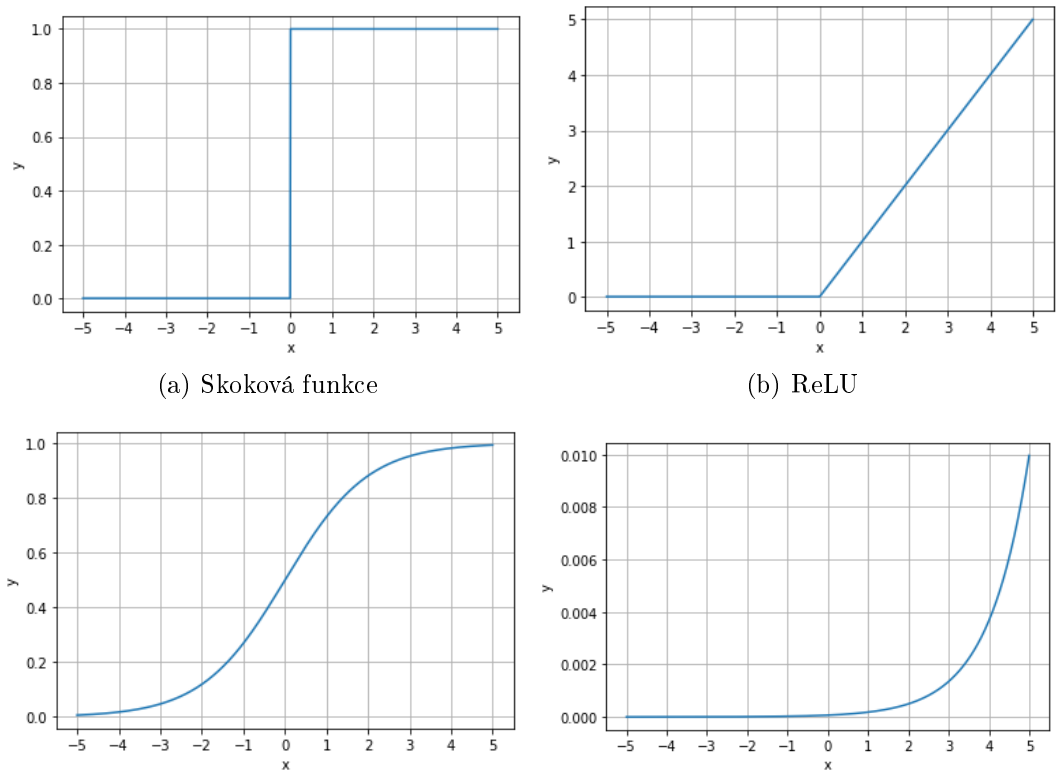
Výstup perceptronu může být popsán následující rovnicí:

$$y = f(x) = f(\sum_i x_i w_i + b) = f(x_1 \cdot w_1 + x_2 \cdot w_2 + b), \tag{2.7}$$

kde se suma vážených vstupů mapuje do aktivační funkce $f(x)$. Některé aktivační funkce budou představeny v následující sekci.

2.3.2 Aktivační funkce

Aktivační funkce označuje funkci, která je aplikována na váženou sumu, tedy na sumu hodnot vstupů vynásobených jejich vahami. Třídy ve většine praktických úlohách nejsou dobře lineárně separabilní, proto je důležité, aby aktivační funkce měla nelineární průběh [27].



(a) Skoková funkce

(b) ReLU

(c) Sigmoida

(d) Softmax

Obrázek 2.11: Přehled aktivačních funkcí

(a) **Skoková funkce**: Tato lineární funkce se kvůli své jednoduchosti často používá právě ve zmíněném perceptronu. Skoková funkce není v praxi používána a je určena spíše pro výukové účely [28].

$$f(x) := \begin{cases} 0 & \text{pokud } x \leq 0, \\ 1 & \text{pokud } x > 0. \end{cases} \tag{2.8}$$

(b) **ReLU** (Rectified Linear Unit): ReLU je momentálně nejpoužívanější aktivační funkcí. Jedná se o nelineární funkci. ReLU zrychluje proces konvergence sítě díky její konstantní derivaci (pokud je derivace konstantní, nemusí se stále

dokola přepočítávat, více v sekci 2.3.6) [2]. Konvergencí sítě je myšlen průběh, kdy se výstup sítě přibližuje k požadovanému výstupu.

$$f(x) = \max(0; x) \tag{2.9}$$

Záporné hodnoty jsou podle rovnice 2.9 mapovány na nulu (to může způsobit tzv. dying ReLU problém, který souvisí se záporným gradientem ztrátové funkce, která bude představena v dalších sekcích). Místo funkce ReLU se pro prevenci této situace může rovněž používat funkce LReLU (Leaky ReLU), která mapuje záporný vstup na nenulové záporné hodnoty. Funkce je dána následujícím předpisem [29]:

$$f(x) = \max(0, 1 \cdot x; x) \tag{2.10}$$

(c) **Sigmoida**: Hlavní důvod, proč byla tato funkce nahrazena funkcí ReLU je tzv. vanishing gradient problem. Tento problém je zapříčiněn tvarem této funkce (viz obrázek 2.11(c)), kde se pro vysoké hodnoty mění gradient velmi nepatrně, což má za následek to, že síť pro takové hodnoty neupravuje své váhy (pro pochopení tohoto problému je nutné porozumět gradientním metodám v sekci 2.3.6) [30].

$$f(x) = \frac{e^x}{e^x + 1} \tag{2.11}$$

(d) **Softmax**: Tato funkce mapuje reálné hodnoty vážených sum na pravděpodobnosti. Aktivační funkce softmax se používá při klasifikaci jako výstupní vrstva.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{2.12}$$

Výpočet pravděpodobností lze lépe pochopit z příkladu. Jedná se o úlohu klasifikace do čtyř tříd. Předpokládejme následující hodnoty $x = \{-1; 0; 3; 5\}$. Poté lze spočíst jmenovatel rovnice 2.12: $\sum_{j=1}^{n} e^{x_j} = e^{-1} + e^0 + e^3 + e^5 = 169,87$

Třída i	x_i	**Čitatel** e^{x_i}	**Pravděpodobnost** $f(x_i)$
1	-1	$0,368$	$0,002$
2	0	1	$0,006$
3	3	$20,09$	$0,118$
4	5	$148,41$	$0,874$

Tabulka 2.4: Příklad spočtení vektoru pravděpodobností softmaxu

Výstup ze softmaxu je tedy vektor pravděpodobností, kde index nejvyšší hodnoty ve vektoru značí příslušnost klasifikovaného obrazu třídě s tímto indexem. Vstup podle příkladu výše náleží do třídy $i = 4$. Z tabulky 2.4 lze vidět, že suma skrz všechny pravděpodobnosti bude podle pravděpodobnostní distribuce rovna jedné [31].

Z této sekce tedy plyne, že nejpoužívanější aktivační funkcí je funkce ReLU. Funkce softmax se zařazuje na konec sítí pro výstup ve tvaru pravděpodobností příslušnosti vstupního obrazu třídám.

2.3.3 Učení s učitelem

V předchozích sekcích bylo vysvětleno, jakým způsobem je počítán výstup sítě (rovnice 2.7) a na jakých parametrech závisí jeho hodnota (váhy w, práh b a typ aktivační funkce). Učení s učitelem je iterativní algoritmus, který mění parametry sítě tak, aby se výstup sítě co nejvíce podobal požadovanému výstupu. Požadovaným výstupem je myšlena správná klasifikace vstupních obrazů do tříd. Učení s učitelem má dvě fáze: fázi učení a fázi predikce.

Fáze učení

Tento proces je také znám pod pojmem "trénování sítě". Síti jsou (většinou v dávkách) na vstup předkládány předem známé obrazy - trénovací data. S těmito daty počítá síť své výstupní hodnoty, které jsou porovnávány s požadovanými výstupními hodnotami. Pokud se výstup sítě dostatečně nepodobá požadovanému výstupu, musí síť změnit své parametry. To se provádí do doby, než bude výstup dostatečně podobný požadovanému výstupu.

Rozdíl mezi požadovanými hodnotami u a výstupními hodnotami sítě y reprezentuje chyba E. Níže je popsán učící algoritmus perceptronu s aktivační funkcí ve tvaru skoku (na obrázku 2.11(a)).

(a) **Inicializace**: Vektor vstupních vah w a práh b jsou inicializovány náhodnými malými čísly (doporučení $< -1, 1 >$). Inicializována je rovněž učící konstanta $c > 0$, která bude rozhodovat o míře změny parametrů sítě.

(b) **Výpočet výstupní hodnoty**: Zjištění výstupu perceptronu $y(k)$ podle rovnice 2.7. Proměnná k zde označuje o kolikátou iteraci se jedná.

(c) **Výpočet chyby**: Chyba je vypočtena srovnáním výstupu perceptronu $y(k)$ se známými výsledky $u(k)$ a chybou z předchozího kroku $E(k-1)$ (pro první krok algoritmu $k = 1$ se předpokládá, že předchozí chyba $E(0) = 0$ [32]).

$$E(k) = E(k-1) + 0.5 \cdot (u(k) - y(k))^T \cdot (u(k) - y(k)) \tag{2.13}$$

(d) **Aktualizace parametrů**: Velikost změny je dána učící konstantou c a rozdílem požadovaného výstupu $u(k)$ a výstupu perceptronu $y(k)$.

$$w(k+1) = w(k) + c \cdot (u(k) - y(k)) \cdot x(k)^T \tag{2.14}$$

$$b(k+1) = b(k) + c \cdot (u(k) - y(k)) \tag{2.15}$$

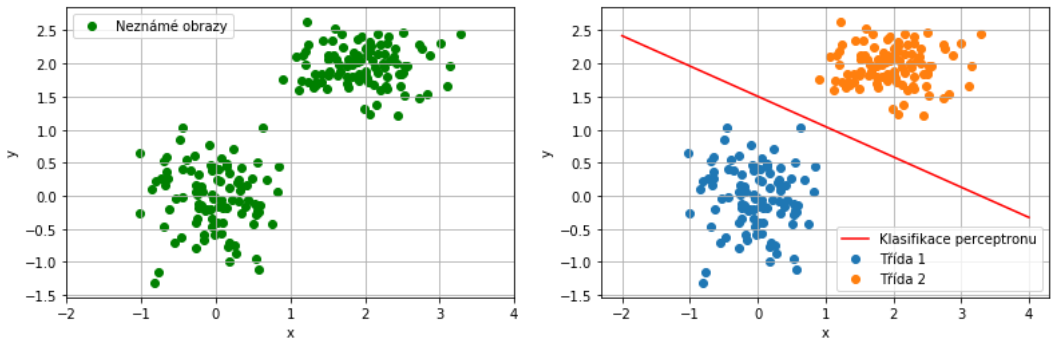
(e) **Test vyčerpanosti trénovací množiny**: Pokud se v trénovací množině nachází dosud nepoužité vstupní obrazy, použijí se tyto obrazy a algoritmus se vrací na krok (b).

(f) **Konec trénovacího cyklu**: V tuto chvíli byly použity všechny dostupné trénovací vzory. Pokud je chyba nižší než předem stanovená hranice, algoritmus končí. Pokud ne, chyba $E(k)$ je vynulována a algoritmus se vrací na krok (b).

Tento algoritmus je kvůli propagaci chyby z výstupu na vstup nazývaný též jako algoritmus zpětného šíření (anglicky backpropagation [33]).

Fáze predikce

Perceptron je v tuto chvíli natrénován. Nastavil své váhy tak, aby dokázal klasifikovat neznámé vstupní obrazy. Fungování klasifikace natrénovaným perceptronem může přiblížit obrázek 2.12 níže, kde dochází ke klasifikaci obrazů se dvěma příznaky (příznakem je myšlena charakteristická vlastnost obrazu, díky které lze jednotlivé obrazy rozlišit) do dvou tříd. Váhový vektor w ovlivňuje sklon křivky, práh b její posun.



(a) Vstupní obrazy (b) Klasifikace perceptronem

Obrázek 2.12: Klasifikace neznámých vstupních obrazů pomocí perceptronu

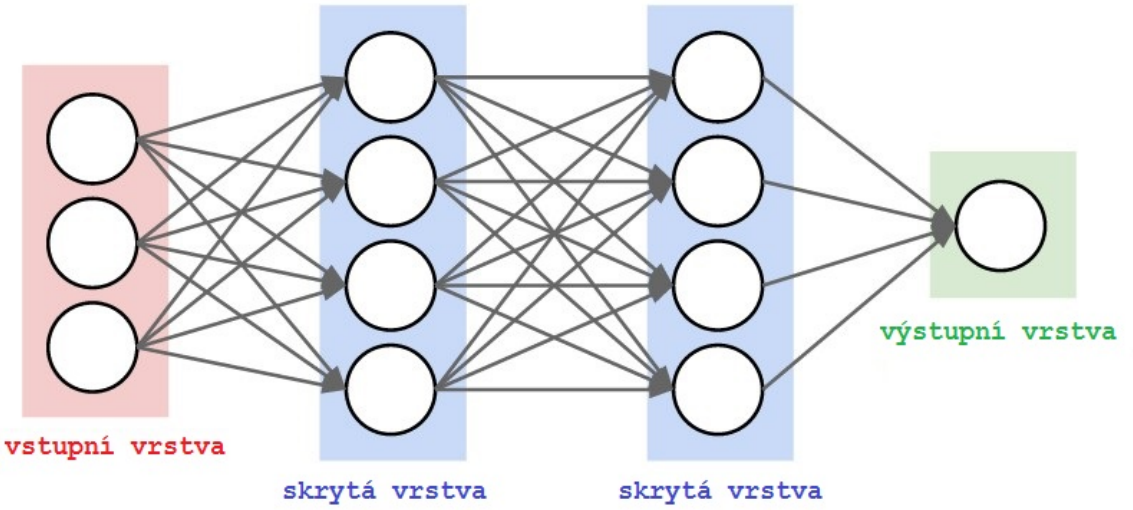
Hlavním cílem této sekce bylo ukázat, že řízenou změnou parametrů sítě dochází k jejímu trénování. Díky tomu se perceptron naučil klasifikovat obrazy do dvou tříd. Většina praktických úloh však neobsahuje lineárně separabilní třídy jako na obrázku 2.12 a jejich vstupní obrazy jsou popsány více než dvěma příznaky (tzn. původní stavový prostor už nelze jednoduše graficky zobrazit). Takové úlohy je nutné řešit pomocí složitějších sítí s více vrstvami. Základní myšlenka tohoto algoritmu je však zachována i pro vícevrstvé sítě.

2.3.4 Vícevrstvé neuronové sítě

Vícevrstvé sítě jsou tvořeny vrstvami neuronů, jejichž výstupy mohou být použity jako vstupy pro další vrstvy neuronů.

Na obrázku 2.13 se nachází tři neurony ve vstupní vrstvě. Vstupní vrstvu následují dvě dense vrstvy. Vrstvy jsou také často označovány jako plně propojené. Výraz plně propojená vrstva znamená, že neurony těchto vrstev jsou s neurony sousedních vrstev

spojeny každý s každým. Pokud má neuronová síť více než jednu skrytou vrstvu, lze o ní říct, že je to hluboká neuronová síť (anglicky deep neural network) [34]. Neuronová síť na obrázku 2.13 obsahuje 2 skryté vrstvy a proto se o ní může mluvit jako o hluboké neuronové síti.



Obrázek 2.13: Příklad dvouvrstvé neuronové sítě [2]

Mezi jednotlivými dense vrstvami se často zařazují aktivační funkce. Jak zaznělo v sekci 2.3.2, nejčastěji používanou aktivační funkcí mezi skrytými vrstvami je ReLU. Ve výstupní vrstvě se ve většině případů vyskytuje softmax 2.11(d).

2.3.5 Ztrátové funkce

Ztrátové funkce definují rozdíl mezi požadovanou hodnotou výstupu a skutečnou hodnotou výstupu (podobně jako v kroku (c) v sekci učení s učitelem). Momentálně nejpoužívanější ztrátovou funkcí pro úlohu klasifikace je tzv. kategorická křížová entropie.

Kategorická křížová entropie

Tato ztrátová funkce se používá pro úlohu klasifikace do více než dvou tříd v sítích s aktivační vrstvou softmax na výstupu. Kategorická křížová entropie (categorical cross-entropy) porovnává skutečné a predikované pravděpodobnostní funkce pro jeden obraz.

Ztrátová funkce J se dá vyjádřit matematicky takto [35]:

$$J = \sum_{j=0}^{M} \sum_{i=0}^{N} (u_{ij} \cdot \log(y_{ij})), \tag{2.16}$$

kde M je počet tříd, N je počet obrazů, u požadovaný výstup sítě a y reprezentuje skutečný výstup sítě. Pro úlohu klasifikace obrazů do dvou tříd se používá speciální verze kategorické křížové entropie (kde $M = 2$), která se nazývá binární křížová entropie (binary cross-entropy).

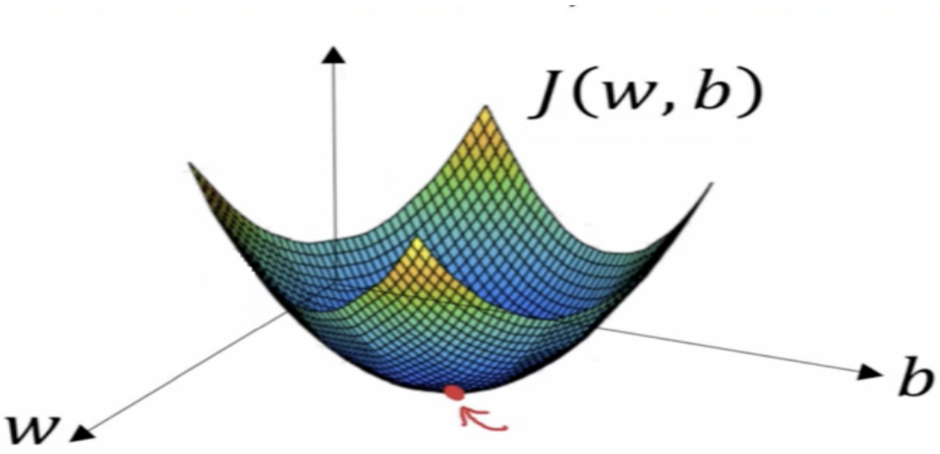
Standardně využívanou funkcí v oblasti regresní analýzy je střední kvadratická chyba MSE [36] (regresní analýza je stejně jako klasifikace disciplína v oblasti učení s učitelem, viz. obrázek 2.9).

2.3.6 Nastavení parametrů sítě

Jak bylo ukázáno v sekci 2.3.3, řízeným nastavováním parametrů sítě lze docílit požadovaných výstupů ze sítě. Ve vícevrstvých neuronových sítích jsou váhy sítí reprezentovány maticemi (na rozdíl od vektorů v perceptronu).

Optimalizační algoritmy

Ke správnému nastavení parametrů sítě a minimalizaci ztrátové funkce J slouží právě optimalizační algoritmy. Několikrát již bylo zmíněno, že správné nastavení parametrů vede k úspěšnému natrénování sítě.



Obrázek 2.14: Optimalizace parametrů je úloha nalezení minima (nejlépe globálního) ztrátové funkce J

Ne vždy se optimalizačním algoritmům povede najít právě globální minimum ztrátové funkce J. Často je neuronová síť dobře funkční i v případě nalezení pouze lokálního minima ztrátové funkce. Přístupů k optimalizaci hodnot vah je mnoho [37] [38]. Momentálně nejpoužívanější a nejefektivnější algoritmy nastavují váhy podle směru růstu průběhu chyby (gradientu). Princip úlohy optimalizačních algoritmů lze připodobnit k hraní kuliček, kde výška hrací plochy odpovídá ztrátě, tzn. důlek je místo

s nejnižší ztrátou, souřadnice na hrací ploše reprezentují nastavení vah a kulička reprezentuje současný stav, tedy nachází se v určité výšce v určitých souřadnicích. Úkolem těchto algoritmů je docvrnkat kuličku do důlku. Jediným rozdílem oproti hraní kuliček je to, že algoritmy nemusí najít globální minimum (jako na obrázku 2.14), nýbrž pouze lokální minimum.

Momentálně nejčastěji používaným optimalizačním algoritmem je ADAM (Adaptive Moment Estimation) [39]. ADAM zároveň optimalizuje učící konstantu tak, aby nebylo minuto globální minimum (což postrádá například další populární algoritmus Adadelta [40]).

Inicializace parametrů

Inicializace parametrů ovlivňuje startovní pozici zmíněné kuličky. Správná inicializace parametrů dokáže zrychlit konvergenci sítě [41]. Standardně používánou inicializační metodou je rovnoměrné rozdělení Xavierem (metoda je často také označována jako Glorot) [42]:

$$\mathrm{var}(w) = \frac{2}{n_{in} + n_{out}}, \tag{2.17}$$

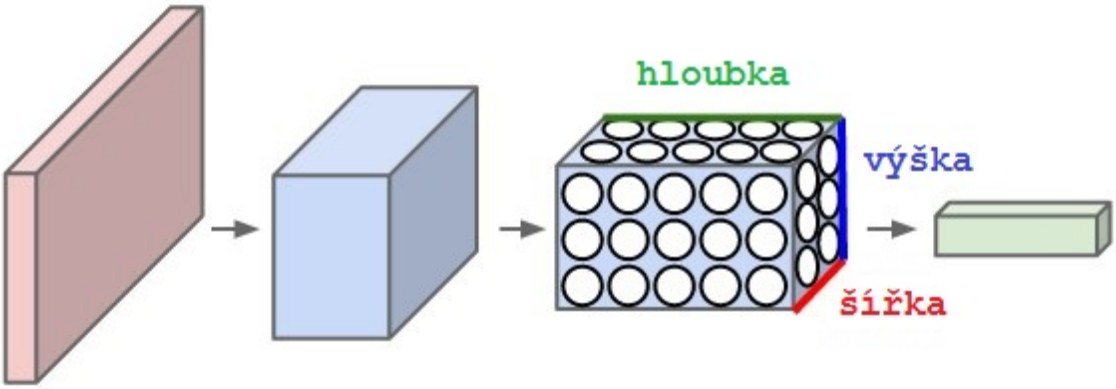
kde n_{in} je počet neuronů předchozí vrstvy a n_{out} počet neuronů nadcházející vrstvy.

Podle [34] je však pro neuronové sítě s aktivační funkcí ReLU doporučeno používat inicializaci He s následujícím předpisem [43]:

$$\mathrm{var}(w) = \frac{2}{n_{in}}. \tag{2.18}$$

2.3.7 Konvoluční neuronové sítě

V předchozích sekcích byly přiblíženy základní principy trénování neuronové sítě. Síť mění pomocí optimalizačních algoritmů své parametry za cílem minimalizace ztrátové funkce, která udává informaci o tom, jak dobře síť vyhovuje předem stanoveným požadavkům. Zmíněné principy platí i pro konvoluční neuronové sítě. Konvoluční neuronové sítě jsou zvláštním druhem neuronových sítí, který předpokládá na svém vstupu digitální obraz [34]. Je tedy pochopitelné, že se tento typ sítí používá zejména v oblasti počítačového vidění. Sítě však nacházejí uplatnění v mnoha odvětvích mimo obor počítačového vidění [44] [45] [46].



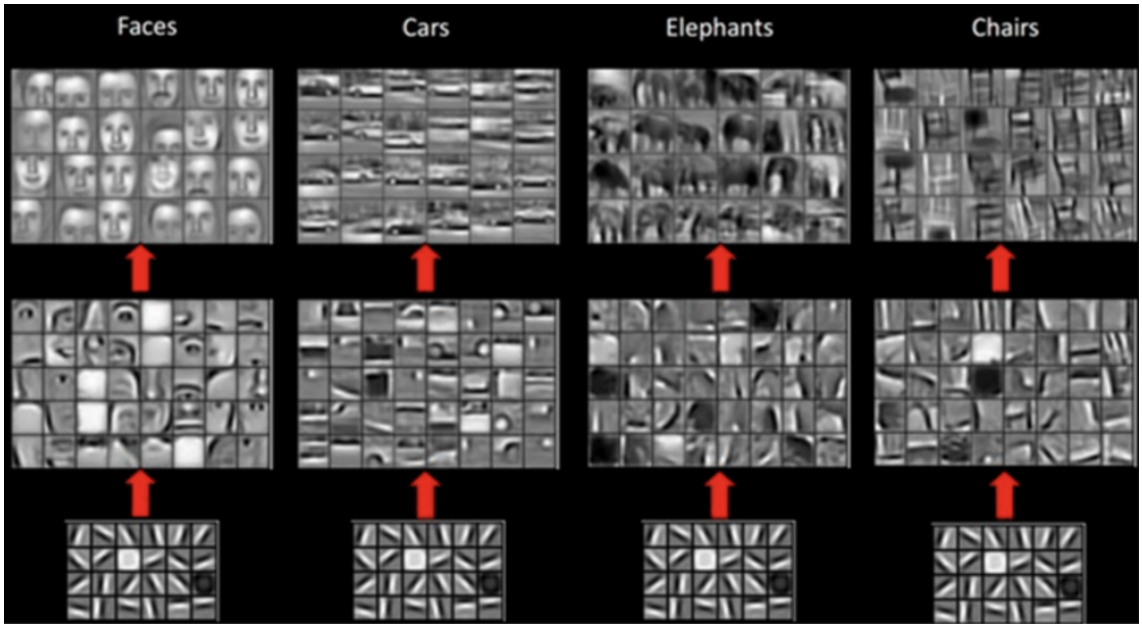
Obrázek 2.15: Ukázka dvouvrstvé konvoluční neuronové sítě, hloubka označuje kanály obrazu (RGB) [2]

Z obrázku 2.15 může být vidět hlavní rozdíl mezi klasickými a konvolučními neuronovými sítěmi. Vrstvy konvolučních neuronových sítí mohou obsahovat neurony ve třech dimenzích. To je zapříčiněno strukturou digitálního obrazu (barevný obraz je uchováván v trojrozměrném poli, viz 2.2.1). V konvolučních neuronových sítích zároveň neplatí pravidlo, že každý pixel má svůj vlastní neuron. U digitálního obrazu záleží hlavně na kontextu okolních pixelů. Proto neurony v konvolučních neuronových sítích reprezentují oblasti obrazů (několik pixelů), nikoliv jednotlivé pixely. Díky tomu lze drasticky snížit počet nastavitelných parametrů sítě a tím zrychlit její trénování [47].

S důrazem na kontext okolních pixelů souvisí i fakt, že neurony dense vrstev konvolučních neuronových sítí jsou spojeny pouze v jejich okolí a ne každý s každým, jak je tomu v klasických neuronových sítích (jak ukazuje obrázek 2.13).

Vrstvy konvolučních neuronových sítí

Díky vrstvám v konvoluční neuronových sítích dochází k redukci parametrů sítě. Nejčastěji pracují konvoluční vrstvy s maticemi, které reprezentují určitou podoblast obrazu. Na tyto matice jsou aplikovány matematické operace, pomocí kterých je obraz dekomponován. Průběh dekompozice obrazu konvolučními neuronovými sítěmi může ukázat obrázek 2.16 na další straně.



Obrázek 2.16: Ukázka dekompozice obrazu konvoluční neuronovou sítí [48]

Nejprve jsou z obrazu extrahovány tzv. příznaky nízké úrovně (na obrázku 2.16 jsou tyto příznaky na třetí řádce). Mezi příznaky nízké úrovně patří například čáry či intenzity pixelů. Průběhem sítí se příznaky postupně seskupují do obsáhlejších a složitějších regionů.

Vrstvy konvolučních neuronových sítí se podle jejich funkce mohou rozdělit na několik typů:

1. **Konvoluční vrstva**: Díky této vrstvě dochází k extrakci příznaků z obrazu. Konvolucí obrazu s vhodně zvolenou maskou filtru lze detekovat nespojitosti v obraze (jak bylo ukázáno na příkladu v sekci 2.2.2). Masky se v konvoluční vrstvě označují jako filtry. V konvoluční vrstvě se nachází desítky až stovky filtrů. Na rozdíl od masek pro detekci hran obsahují filtry nastavitelné váhy, které se v průběhu trénování mění optimalizačním algoritmem (stejně jako klasické neurony jsou i filtry inicializovány pomocí inicializačních algoritmů - viz. 2.3.6). Kromě velikosti filtrů se jako parametr uvádí jeho posun. Příklad níže ukazuje filtr s velikostí 3×3 a posunem 2 [2]. Červenou barvou je v obrazu označen filtr, který se obrazem postupně posouvá. Druhá matice ukazuje další krok, kde je filtr posunut o dva pixely doprava.

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

Počáteční poloha filtru v obrazu Poloha filtru v dalším kroku

Posledním parametrem konvoluční vrstvy je její přesah. Přesah se používá pro změnu velikosti výstupní matice konvoluční vrstvy (tato matice je často nazývána aktivační či příznakovou mapou). Z příkladu níže lze vidět, že pokud byl zvolen přesah roven jedné, zasahuje filtr i do oblastí mimo obraz. Oblasti mimo obraz jsou dodefinovány nulou (anglicky se tento proces nazývá zero-padding, tzn. v přímém překladu obalení nulami). V příkladu ukázky posunu konvolučního filtru se přesah rovná nule.

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

Filtr o velikosti 3×3 s přesahem 1

2. **Pooling vrstva**: Filtr se posouvá po obraze a z oblasti, kterou překrývá, vybere pouze příznak, který ho nejlépe charakterizuje. Nejpoužívanější metodou poolingu je extrakce nejvyšší hodnoty z oblasti filtru. Funkce pooling vrstvy lze nejlépe pochopit z příkladu [2]:

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

Vstupní obraz — Výstup max pooling vrstvy

Barvy ve vstupním obraze reprezentují oblasti, které filtr překrývá. Filtr se posouvá vstupním obrazem a pro každou oblast je vybrána pouze maximální hodnota, která je stejnou barvou vyznačena ve výstupu pooling vrstvy. V tomto příkladu má pooling vrstva velikost 2×2. Posun filtru není příliš často specifikován. Standardně je posun filtru roven velikosti filtru [34] jako v příkladu výše. Tato vrstva se používá pro redukci parametrů sítě.

3. **Aktivační funkce**: Jak bylo zmíněno na konci sekce 2.3.2, za konvoluční vrstvy se zařazují ReLU funkce pro vnesení nelinearity do sítě. Na výstupu konvoluční neuronové sítě se nejčastěji nachází softmax vrstva, mapující reálné hodnoty na pravděpodobnosti.

4. **Normalizační a regulační vrstvy**: Tyto vrstvy pozitivně ovlivňují stabilitu a rychlost konvergence sítě [47].

   (a) **Vstupní normalizace**: Tato vrstva normalizuje vstupní obraz. Použitím rovnice (2.19) níže se hodnoty intenzit pixelů v každém barevném kanálu

upraví z rozsahu $0 - 255$ na rozsah $0 - 1$:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{2.19}$$

kde x_{norm} je normalizovaný pixel, x je originální pixel, x_{max} maximální možná hodnota intenzity 255 a x_{min} minimální možná hodnota intenzity 0. Díky normalizovanému rozsahu vstupních obrazů lze snáze inicializovat váhy sítě [49].

(b) **Dávková normalizace**: Princip této vrstvy je podobný jako u vstupní normalizace. Normalizace však probíhá uvnitř sítě. Ve většině případů se síti předkládají trénovací obrazy v tzv. dávkách. Jedna dávka označuje počet trénovacích obrazů, které jsou naráz přivedeny na vstup sítě. Střední hodnoty a variance těchto dávek jsou normalizovány, což má za následek snazší hledání minima ztrátové funkce a tudíž zrychlení konvergence sítě [50].

(c) **Dropout vrstva**: Dropout vrstva funguje pouze ve fázi učení. Vrstva náhodně odpojuje spojení mezi neurony předcházející vrstvy s neurony následující vrstvy. Parametrem dropout vrstvy je číslo od nuly do jedné, které ovlivňuje poměr odpojených neuronů (dropout s hodnotou nula neodpojí žádný neuron, dropout s hodnotou jedna odpojuje všechny neurony, které do dropoutu vstupují). Podle [51] se doporučuje volit hodnoty v rozmezí $0,5$ - $0,8$. Po dokončení trénování sítě již dropout vrstva neodpojuje vstupní neurony. Zařazením této vrstvy se může díky zvýšené schopnosti generalizace zlepšit přesnost při klasifikaci neznámých obrazů (odpojováním neuronů v trénovací fázi však může dojít ke ztrátě přesnosti na trénovacích datech a ke zdvojnásobení iterací algoritmu zpětného šíření konvergujícího k minimální ztrátě). Dropout vrstva může být zařazena jak za dense vrstvu, tak za konvoluční vrstvu následovanou aktivační funkcí ReLU [51] (v takovém případě je podle [52] vhodné volit hodnoty dropoutu v rozmezí $0,1 - 0,2$).

5. **Flatten vrstva**: Tato vrstva převede pole o třech dimenzích (viz. obrázek 2.15) na vektor. Vrstva se zařazuje na konec sítě pro možnost připojení konvoluční vrstvy na dense vrstvu.

6. **Dense (plně propojená) vrstva**: Na rozdíl od dense v klasické neuronové síti dochází ke spojením pouze v omezených oblastech a nikoliv po celé vrstvě. Od ostatních zmíněných vrstev se dense vrstva odlišuje tím, že na svém vstupu očekává vektor (nikoliv pole o třech dimenzích).

Popis architektury

Architektura konvolučních neuronových sítí se obecně skládá z několika konvolučních vrstev s velikostí filtru v rozmezí 3×3 až 7×7 (doporučuje se volit liché číslo pro snazší počítání středu příznakové mapy [38]. Za každou konvoluční vrstvou je zařazena ReLU vrstva. Po aktivační funkci ReLU lze podle potřeby zařadit pooling

vrstvy (často používané velikosti pool filtrů jsou 2×2, 3×3 či 5×5 s posunem rovným jejich velikosti) či dropout vrstvy (s nízkou hodnotou dropoutu). Takto za sebou seřazené vrstvy lze libovolně sériově spojovat. Počet těchto sekvencí závisí na složitosti úlohy. Je samozřejmé, že síť s více vrstvami bude obsahovat více parametrů a proces hledání minima ztrátové funkce bude trvat déle.

Na výstupu každé sítě se nejčastěji za sebou zařazují flatten, dense a softmax vrstvy. Dense vrstva obsahuje počet neuronů roven počtu tříd, do kterých je možno klasifikovat. Každý neuron v dense vrstvě tedy reprezentuje hodnoty pro dané třídy, které softmax vrstva převede na pravděpodobnosti. Výstupem konvoluční neuronové sítě je tedy zmíněný vektor s pravděpodobnostmi příslušnosti obrazu do jednotlivých tříd.

Příkladem klasické architektury konvoluční neuronové sítě může být například navržená síť na obrázku 3.28, kterou používám při klasifikaci pokerových karet.

2.3.8 Praktické úvahy

V této sekci budou zmíněny některé metody a doporučení, které mohou sloužit jako základní obecné předpoklady při trénování konvolučních neuronových sítí.

Rozdělení dat

Dataset se často rozděluje na trénovací, validační a testovací data. Trénovací a validační data se používají při trénování sítě. Každou trénovací epochou se přesnost sítě vyhodnocuje na validačních datech. Díky tomu lze zjistit, jak přesně dokáže síť klasifikovat neznámé vstupní obrazy. Podíl trénovacích, validačních a testovacích dat závisí na architektuře sítě. Dobrým odrazovým můstkem se zdá být poměr 70 % trénovacích dat ku 20 % validačních dat ku 10 % testovacích dat [34]. Pro složitější sítě s více neurony stoupá riziko přetrénování a tudíž je vhodné použít vyšší podíl validačních dat [53].

Velikost trénovacího datasetu

Velikost trénovacího datasetu ovlivňuje přesnost a robustnost sítě. Pro velikost datasetu neexistuje univerzální předpis. Základním doporučením podle [54] je použít pro každou třídu 1000 obrazů. Toto číslo závisí na složitosti řešeného problému a na tom, jak dobře jsou třídy separabilní. Obecně se doporučuje použít stejný počet obrazů pro každou třídu [34]. Síť, která byla natrénovaná s nerovnoměrným počtem vzorů v jednotlivých třídách může při klasifikaci neznámých obrazů upřednostňovat třídy obsahující vysoký počet trénovacích vzorů. S velikostí trénovacího datasetu také úzce souvisí velikost dávek (tzv. batch size), pomocí kterých se síť trénuje. Podle [55] se optimální velikost dávky pohybuje kolem hodnoty 200 a výše. S vyšším počtem dávek na iteraci se zvyšuje i přesnost sítě. Na druhou stranu rostou i nároky na výkon [34].

Augmentace

Na digitální obraz je možné provádět tzv. augmentace. Augmentacemi mohou být například rotace, posun či změna barev v obrazu. Po aplikaci augmentací je vytvořen nový, pozměněný obraz. Díky augmentacím lze z původních dat vytvořit nové obrazy a rozšířit tak trénovací dataset. Tím je možné zvýšit přesnost klasifikace konvoluční neuronové sítě [56]. Obrázky je nutné augmentovat tak, aby byly zachovány jejich hlavní rysy. Příliš silné augmentace mohou být pro trénování sítě kontraproduktivní [56]. Příkladem použití augmentací může být například sekce 3.3.3 v praktické části.

Rozměr vstupních obrazů

Ve většině případů se rozměry trénovacích dat upravují, aby všechny obrazy měly stejné rozměry. Sítě s proměnnou velikostí vstupního obrazu se používají pouze ve specifických případech (např. segmentace obrazu) [57]. Opět není pevně dáno, jak velké mají vstupní obrazy být. S rostoucími rozměry obrázku rostou i požadavky na výkon zařízení, na kterém se síť trénuje. Pro představu uvádím velikostí vstupních obrazů benchmarkových datasetů:

Název	**Rozměry**	**Počet tříd**	**Popis tříd**
MNIST	28×28	10	Ručně psané číslice
CIFAR-10	32×32	10	Dopravní prostředky a zvířata
ImageNet	Proměnné	21841	Reálné objekty
Fashion-MNIST	28×28	10	Oděvy

Tabulka 2.5: Standardně používané datasety

Rozměr obrazů v *ImageNet* datasetu je proměnný, většinou se ale dataset používá s rozlišením 256×256. Vstupní rozměry však nemusí být čtvercové. Ve většině případů se pro přesnost sítě jeví zvolení vyššího rozlišení (až 500×500 pixelů) vstupního obrazu lépe, než zvolení nižšího rozlišení (224×224) [58].

3 Praktická část a vyhodnocení experimentů

Tato kapitola se zaobírá návrhem pokerové aplikace určené k vyhodnocení stavu hry z digitálního obrazu. Pro analýzu stavu hry musí být karty v obraze správně identifikovány. V tomto případě jsou karty klasifikovány konvoluční neuronovou sítí. Po klasifikaci karet je zjištěna informace o tom, kdo dané karty vlastní (kterému hráči karta patří a jaké karty hráči sdílí). Výpočet pravděpodobností hráčů na výhru či remízu je proveden pomocí simulací pokerových her.

Nejprve krátce popíšu algoritmus vyjmutí karet z obrazu. Algoritmus pro získání karet je vylepšenou verzí algoritmu, která byla navržena v [2]. Nepoužívám zde však některé kroky, které se ukázaly jako zbytečné či dokonce kontraproduktivní.

Dále popisuji tvorbu syntetických karet a syntetického hracího stolu. Syntetické karty budou použity při trénování neuronové sítě. Díky těmto kartám se rozšíří trénovací dataset. Na počítačem generovaném snímku hracího stolu poté ověřím správné fungování výsledné aplikace.

Další sekcí je příprava dat pro neuronovou síť. Ve předchozí práci [2] nebyl kladen patřičný důraz na přípravu trénovacího datasetu. To se projevilo na přesnosti natrénované sítě (síť dosáhla na trénovacích datech přesnosti $93,45\%$). V této práci jsem si dal na tvorbě datasetu více záležet. Zároveň popíšu fungování tensorflow modulu ImageDataGenerator, který podstatně zjednoduší proces augmentace karet.

Z připravených trénovacích dat natrénuji pět neuronových sítí různých architektur. Tato část práce je spíše experimentální. Cílem je ukázat, jak se změní přesnost sítě, jsou-li vynechány některé důležité komponenty jako například pool, dropout či dense vrstva. Nejpřesnější z těchto neuronových sítí bude sloužit ke klasifikaci reálných karet.

Pátá sekce se věnuje analýze stavu hry. Popíšu zde algoritmus přiřazení identifikované karty jednotlivým hráčům. Pravděpodobnosti na výhru jsou v této práci spočteny pomocí Monte-Carlo simulace.

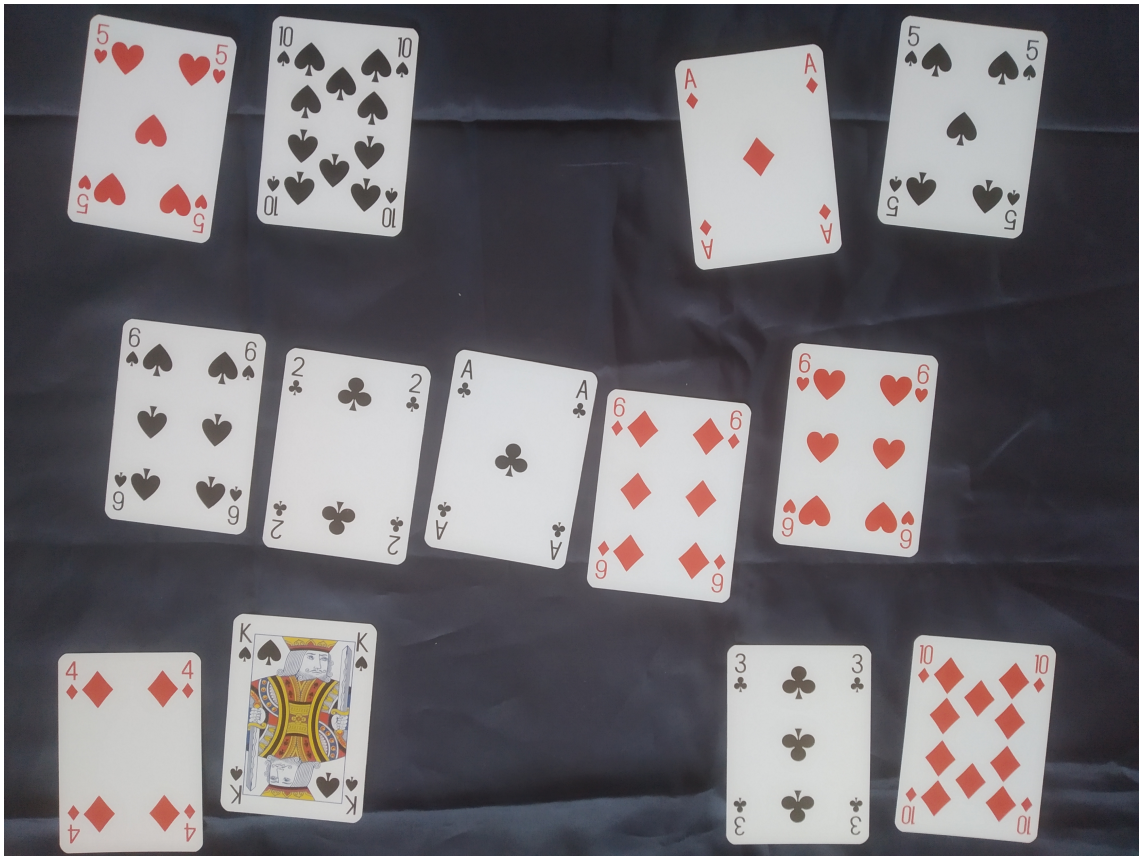
Další podkapitolou je tvorba grafického uživatelského rozhraní, které sjednotí všechny zde zmíněné procesy. V práci budu často tento výraz nahrazovat zkratkou GUI (z anglického názvu graphical user interface). GUI jsem vytvořil pomocí knihovny PyQt5. Díky aplikaci QtDesigner lze v PyQt5 vytvořit GUI bez ručního psaní kódu. Grafické komponenty se v QtDesigneru přetáhnou z lišty nástrojů, což přispívá k efektivnějšímu a rychlejšímu návrhu GUI. Kód je aplikací generován automaticky.

Nakonec aplikaci otestuji na syntetických a reálných snímcích pokerových stolů. Bude vyhodnocena přesnost natrénované neuronové sítě na testovacích datech a porovnám mnou spočtené pravděpodobnosti na výhru s online kalkulátory. Tato aplikace je spolu s návodem na instalaci dostupná v github repozitáři na adrese https://github.com/hrdlickajan/DP [59].

3.1 Získání pokerových karet z obrázku

Je nutné, aby algoritmus na vyjmutí karet ze snímku byl co nejspolehlivější a nejpřesnější. Pokud tento algoritmus selže, nebude možné provést správnou analýzu herního stolu. Modifikace tohoto algoritmu je použita také při tvorbě trénovacího datasetu pro neuronovou síť (viz. 3.3.1). Základní myšlenka pro získání karet ze snímku zůstala stejná jako v předchozí práci [2], tedy:

1. Segmentace snímku
2. Detekce regionů s vysokým obsahem pixelů a rozhodnutí o přítomnosti karty
3. Vykreslení kontur kolem karet
4. Rotace snímku tak, aby dolní hrana vybrané karty kopírovala vodorovnou osu
5. Vyjmutí karty ze snímku



Obrázek 3.1: Obrázek, ze kterého budou karty získávány

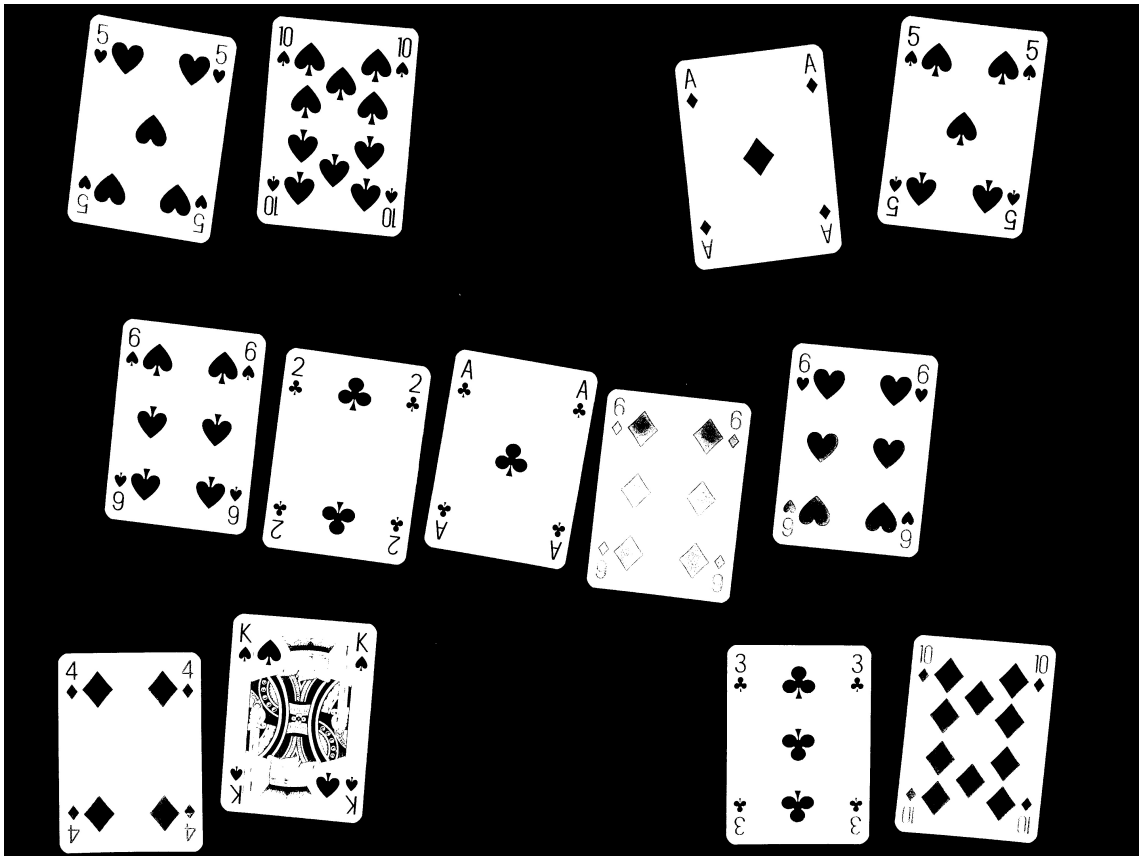
Jako předlohu, na které budu výše zmíněné metody aplikovat, jsem zvolil snímek s reálnými kartami - obrázek 3.1. Na tomto snímku jsou u stolu přítomni 4 hráči a hra je ve stavu river, tzn. všechny karty jsou odkryty.

3.1.1 Segmentace snímku

Segmentace snímku slouží k zaměření regionů s kartami. Důležité při segmentaci je, aby se dobře oddělily karty od pozadí. V předchozí práci [2] byla použita metoda adaptivní segmentace pro vyrovnání jasu ve snímku. To se ukázalo jako zbytečné. Pro klasifikaci karty neuronovou sítí je používána barevná karta, tzn. není důležité, že segmentovaný snímek obsahuje odrazy světla či jiné nežádoucí světelné jevy.

Jak už bylo naznačeno v kapitole 2.2.2, pokud se jas pozadí liší vůči jasu objektů v obraze, postačuje k segmentaci metoda prahování. Všechny snímky stolu tuto podmínku splňují, proto volím právě tuto techniku. Obrázek 3.1 je nutné před prahováním převést na šedotónový barevný model (viz. rovnice 2.3).

V práci předpokládám, že všechny snímky stolu s tmavým pozadím budou mít bimodální histogam (jsou-li v popředí bílé karty). Pro tyto případy je vhodná metoda optimálního nastavení prahu Otsu. Zde je metoda Otsu použita z knihovny skimage [60].



Obrázek 3.2: Segmentovaný snímek metodou otsu

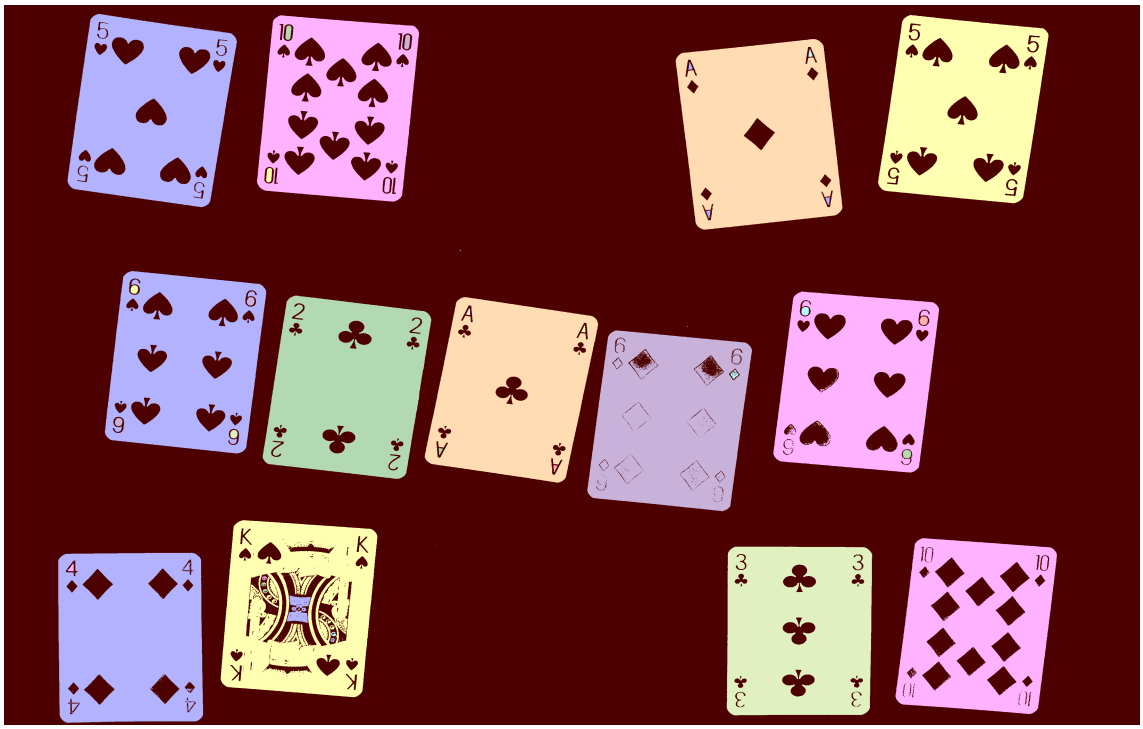
Jak je vidět na obrázku 3.2, metoda Otsu zvolila práh tak, že většina symbolů na kartách je prohlášena za pozadí. V tomto případě to není důležité. Podstatné je, že se oddělily okraje karet od pozadí a bude tak možné dobře analyzovat vlastnosti segmentovaných regionů.

3.1.2 Detekce regionů s potenciální kartou

Jak bylo zmíněno v kapitole 2.2.2, segmentace se provádí za účelem usnadnění analýzy obrazu. Aby byl v algoritmu správně definován kontext karty, musí být zjištěny následující informace:

1. **Obsah regionů**: Regiony s nízkým obsahem pixelů jsou pravděpodobně objekty, které na snímku nemají co dělat (šum, stíny, atp.). Regiony s vysokým obsahem budou pravděpodobně karty.
2. **V jakých souřadnicích se regiony nachází**: Z této informace je možné spočítat výšku a šířku karet a zároveň zjistit, pod jakým úhlem jsou karty otočené.
3. **Souřadnice středů regionů**: Toto bude důležité pro analýzu stavu hry, kdy bude nutné přiřadit karty hráčům (například: karty se středy blízko sebe budou pravděpodobně patřit stejnému hráči).

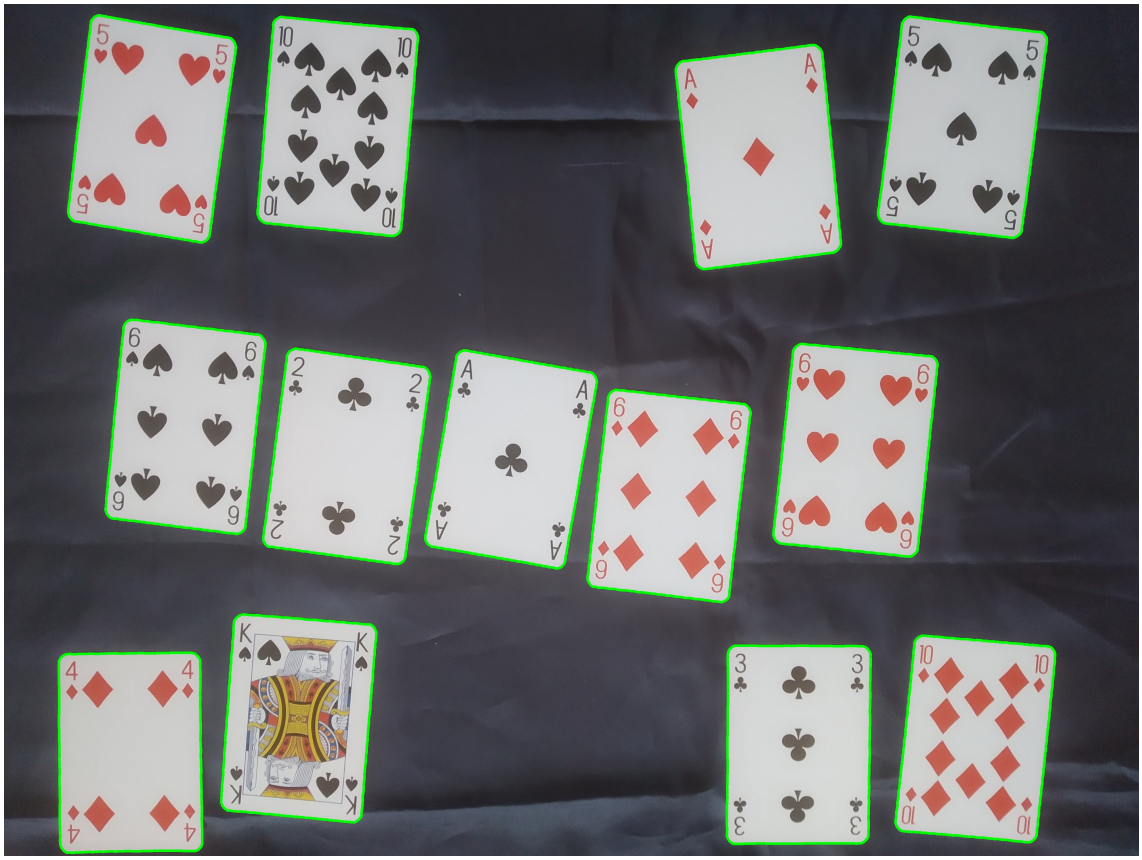
Díky těmto informacím je možné rozhodnout, zda se v segmentovaném regionu opravdu nachází karta. Nejprve jsem na segmentovaný obrázek aplikoval morfologickou operaci otevření pro rozbití tenkých hran. Díky operaci otevření lze předejít situaci, kdy souvislý region uvnitř karty je kvůli svému velkému obsahu prohlášen za kartu (poté by v jednom regionu byly nalezeny dvě karty, což není možné). Pro měření vlastností regionů jsem použil metodu *skimage.measure*. Další použitá funkce *skimage.measure.regionprops* slouží pro spočtení obsahů regionů (obsahy jsou počítány v pixelech).



Obrázek 3.3: Detekce regionů v segmentovaném snímku - tzv. barvení

3.1.3 Vykreslení kontur kolem karet

V této chvíli dokáže algoritmus v regionech segmentovaného snímku rozlišit kartu od nepotřebného regionu. Nyní se karty musí zaměřit konturami. To bude důležité pro jejich rotaci a následné vyjmutí ze snímku. Použitím metody *cv2.findContours* dojde k nalezení kontur kolem karet. Je vhodné, aby kontury kolem karet měly obdélníkový tvar s rovnými hranami, proto jsou kontury aproximovány na přímky. Poté se kontury vykreslí funkcí *cv2.drawContours* se specifikovaným parametrem *cv2.RETR_EXTERNAL*, který zaručí vykreslení kontur pouze po obvodu objektů.



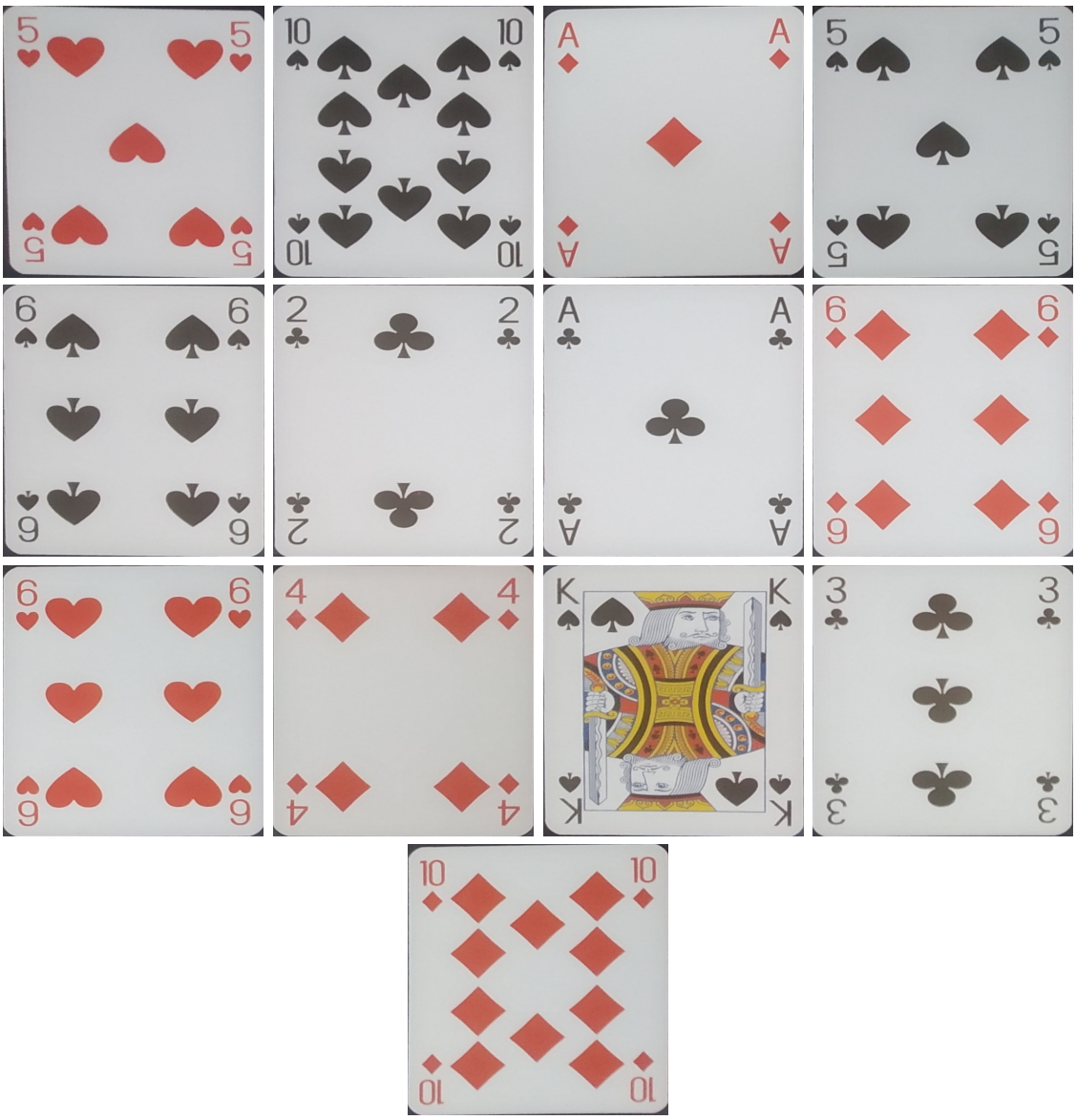
Obrázek 3.4: Kontury kolem karet

Samotná aproximace kontur kolem karet však nedokáže zajistit, aby přímky kontur tvořily opravdu pravoúhlý obdélník. Proto na aproximované kontury postupně aplikuji metody *cv2.minAreaRect* pro opětovnou aproximaci kontur nejmenším obdélníkem a *cv2.boxPoints* ke zjištění souřadnic rohů tohoto obdélníka. Případ překrývajících se karet není v této práci zohledněn hlavně z toho důvodu, že taková situace běžně příliš často nenastává (pokud by se počítalo i s tímto případem, návrh celé práce by byl mnohem složitější). Od této chvíle mohu počítat s tím, že všechny karty jsou obdélníkové, což přijde velice vhod při počítání úhlu rotace a vyjmutí karty. Může se naskytnout otázka, proč počítám nové souřadnice rohů, když tato informace už byla zjištěna v předchozí sekci 3.1.2. Je to nutné kvůli tomu, že rohy karet jsou zaoblené a tedy není snadné určit přesné souřadnice rohu karty. Přesnost

souřadnic rohů karet je rovněž hlavním předpokladem pro přesnou rotaci a vyjmutí karty.

3.1.4 Rotace a vyjmutí karty ze snímku

V tuto chvíli jsou po procesech segmentace, detekce regionů a konturování zachyceny všechny regiony s kartami. Tyto regiony jsou ohraničeny minimálním obdélníkem. Díky obdélníkové hranici je nyní jednoduché spočítat výšku a šířku hranice kolem karty. Zároveň je výhodou, že strany obdélníka jsou pravoúhlé. Pro pravoúhlé obrazce lze pro spočtení rotace použít goniometrické funkce. Po zjištění úhlu rotace karty vůči vodorovné ose lze kartu rotovat, aby byla její spodní hrana taktéž vodorovná. Posledním krokem je vyjmutí karty pomocí metody *np.argwhere*.



Obrázek 3.5: Výsledné karty po jejich rotaci a vyjmutí ze snímku

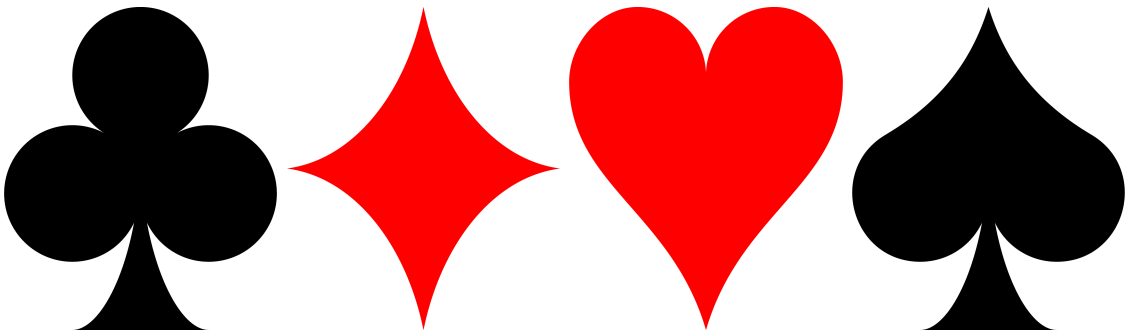
Velikosti karet na obrázku 3.5 byly pro zlepšení vizuální stránky práce normalizovány. Na obrázku lze vidět malé odchylky od rovin, které tvoří nepravidelné tmavé okraje. Tato nepřesnost nemusí být na škodu. Mírné odlišnosti karet mohou posloužit jako forma augmentace dat (trénovací dataset bude augmentován v sekci 3.3.3, jako augmentace byly použity například právě horizontální a vertikální posuny).

3.2 Generování syntetických obrazů

Výhodou počítačem generovaných dat je možnost automatického vytvoření velkého množství obrázků za krátkou dobu. Synteticky vytvořené karty použiji jako trénovací data pro neuronovou síť a zároveň je použiji pro vytvoření umělého snímku hracího stolu. V této části práce používám zejména knihovnu PIL, která obsahuje metody pro práci s digitálním obrazem. Hlavní náplní této sekce je zjistit, do jakých souřadnic vložit předem připravené obrazy (pro generování karty to jsou písmena a symboly a při generování pokerového stolu karta).

3.2.1 Generování pokerových karet

Na rozdíl od sběru reálných dat budu generovat 3 sety karet, tzn. že v této práci budou použity karty ze třech různých hracích balíků. Jeden set karet je totožný s tím reálným (pouze jeden balík karet vlastním fyzicky) a zbylé dva jsou stažené ze stránky [61]. Karty generuji vkládáním symbolů barev a písma do snímku. K tomu používám třídy ImageFont a ImageDraw z knihovny PIL.



Obrázek 3.6: Symboly barev, ze kterých se vytváří karta

Vstupními argumenty skriptu na generování karet jsou následující parametry:

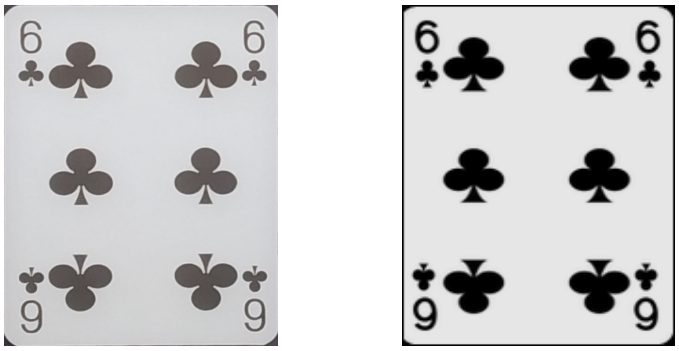
1. **Ze kterého hracího balíku karta bude**: Karty v jednotlivých hracích balících vypadají odlišně. Celkem jsou k dispozici 3 hrací balíky.
2. **Rotace karty**: Tento parametr bude důležitý při generování syntetického hracího stolu.
3. **Hodnota karty**: Podle zadané hodnoty je zvolen počet symbolů a jejich umístění na kartě.

4. **Barva karty**: Na barvě karty závisí barva písma v kartě a zvolený symbol, který bude do karty vkládán .

5. **Rozměry karty**: Pro zjednodušení je nejprve vytvořena karta s konstantní výškou a šířkou. Až poté, když je karta hotová, se změní její velikost a tudíž se nemusí podle vstupních parametrů měnit velikost písma a symbolů.

6. **Cesta, do které bude karta uložena**: Tento parametr je volitelný, tzn. standardně se karta neukládá a při specifikaci cesty se karta uloží do požadované destinace.

Výsledkem je počítačem vygenerovaná karta, která má požadovanou velikost, hodnotu, barvu a rotaci. U všech karet jsou pro jejich reálnější vzhled zároveň zaobleny rohy. Obrázek 3.7 porovnává reálnou kartu s kartou vygenerovanou skriptem. Lze vidět, že symbol, písmo a barva digitální karty se od reálné karty odlišují, to nemusí být nutně na škodu. Vygenerované karty společně s reálnými kartami se budou používat jako trénovací data v neuronové síti, pro kterou je diverzita dat vítaná.



(a) Karta vyjmutá z vyfoceného snímku

(b) Generovaná syntetická karta

Obrázek 3.7: Porovnání reálné a generované karty

3.2.2 Generování pokerového stolu

Obrázky počítačem vygenerovaných stolů budou sloužit jako testovací data pro finální aplikaci. Výhodou takto vygenerovaného obrázku je nepřítomnost šumů a odrazů světla. V předchozí sekci byl navržen algoritmus generování syntetické karty. Pro vygenerování stolu bude nutné následující:

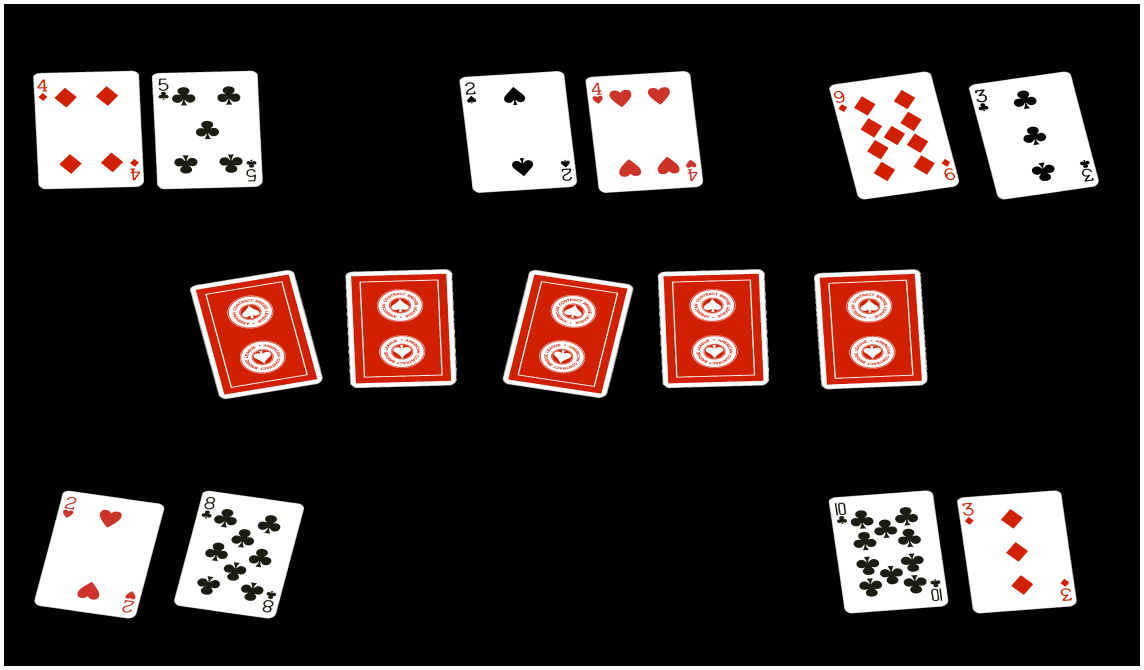
1. **Vygenerovat požadované karty**: Pro každý stůl je náhodně vybrán jeden set karet, ze kterého se karty vytvoří. Hodnoty a barvy karet jsou rovněž vybrány náhodně. Jediné, na co si je třeba dát pozor je to, aby se karta se stejnou hodnotou a barvou nevyskytovala na stole dvakrát. Navíc je nutné vygenerovat ještě pět karet pozadí, které budou reprezentovat neznámé karty na stole. Tzn. hra na vygenerovaném snímku je vždy v počátečním stavu preflop.

2. **Stanovit počet hráčů**: Každý hráč obdrží dvě náhodné karty. Rotace je pro obě tyto karty stejná. V této práci byl zvolen náhodný počet hráčů od dvou do šesti, jejichž karty jsou rotovány o náhodný malý úhel.

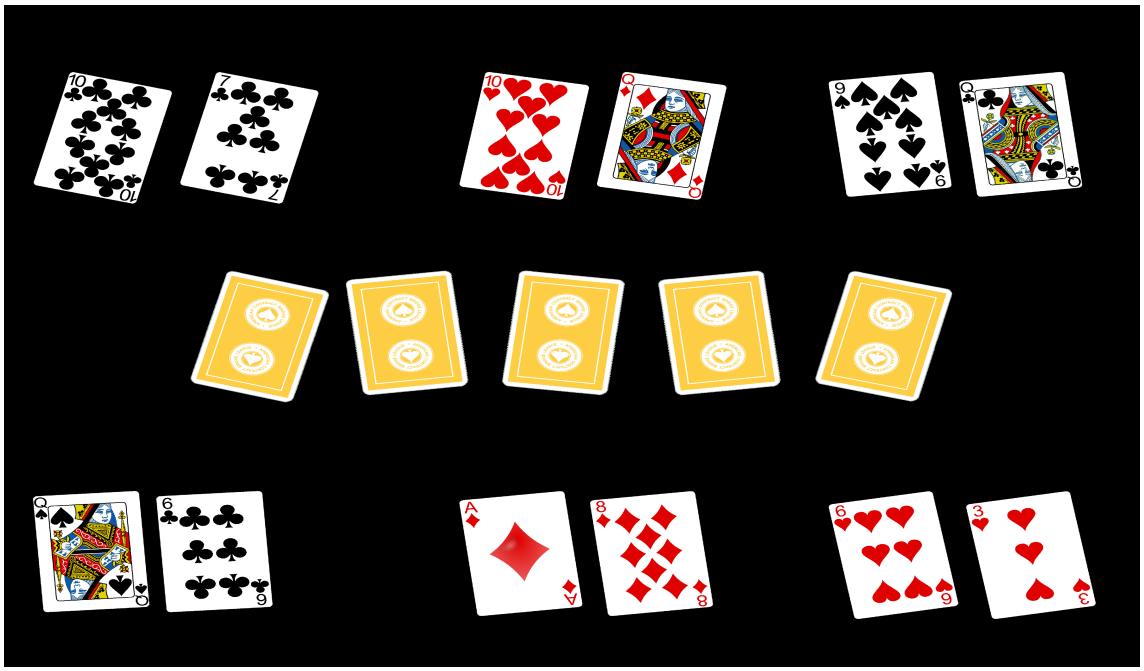
3. **Stanovit souřadnice, do kterých se karty vloží**: Každý hráč má pevně dané souřadnice, ve kterých se nacházejí jeho karty. Stejně je to s kartami na stole, kde každá může mít jinou rotaci.

4. **Vložit vygenerované karty**: Do prázdného obrázku s konstantní výškou a šířkou se postupně vloží všechny vygenerované karty .

Výsledky lze vidět níže na obrázcích 3.8 a 3.9. Na tyto obrázky je možné rovněž aplikovat metody ze sekce 3.1 a získat tak zpětně všechny karty.



Obrázek 3.8: Náhodně vygenerovaný obrázek stolu s pěti hráči



Obrázek 3.9: Náhodně vygenerovaný obrázek stolu s šesti hráči

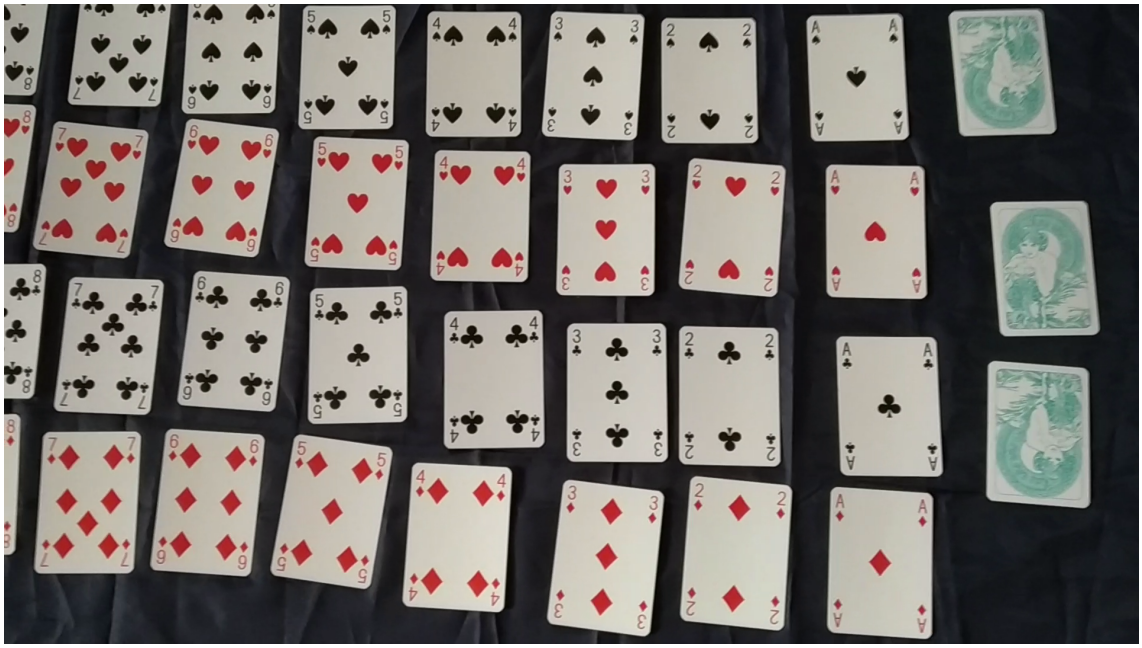
3.3 Příprava dat pro konvoluční neuronovou síť

Použití kvalitních trénovacích dat je základním předpokladem pro natrénování přesné neuronové sítě [37]. V mém případě bude trénovací dataset tvořen z reálných karet a z karet generovaných pomocí algoritmu zmíněného v sekci 3.2.1. Aby trénovací data byla dostatečně rozmanitá, budou obrazy navíc augmentovány pomocí tensorflow modulu ImageDataGenerator.

3.3.1 Sběr reálných karet pro trénovací dataset

Pořizování snímků reálných karet je časově náročné. I když jsem se snažil sběr snímků karet co nejvíce automatizovat, trvalo mi přibližně 20 hodin na vytvoření rozmanitého trénovacího datasetu. Pro zachycení všech aspektů reálné karty je trénovací dataset obsahující pouze syntetické karty nedostačující. Proto je třeba pořídit i dostatečné množství obrazů reálných karet.

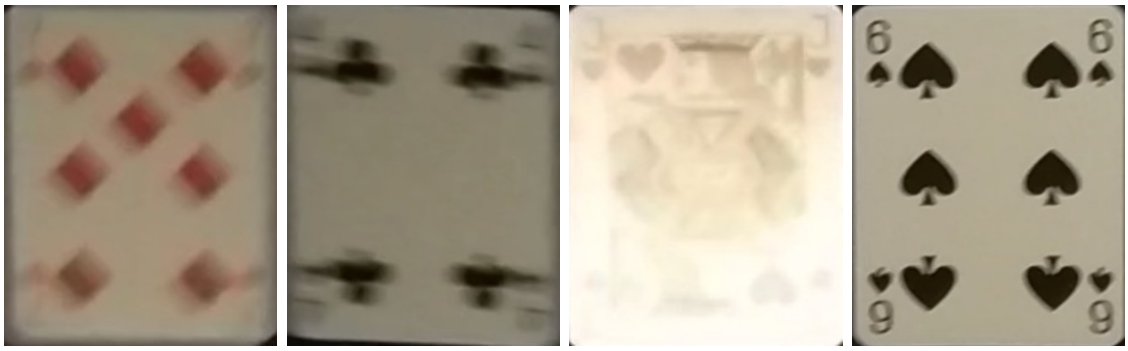
Ke zjednodušení práce používám algoritmus na vyjmutí a identifikaci karet ze snímků (viz. sekce 3.1). Místo pořizování mnoha snímků jsou natáčena videa. Tato videa jsou poté rozložena na jednotlivé snímky, ze kterých se zmíněným algoritmem získají karty. Karty jsou rozloženy na segmentovatelné pozadí tak, aby se nepřekrývaly. Bylo natočeno několik videí za různých denních dob v různě osvětlených místnostech.



Obrázek 3.10: Pořizování dat - snímek z videa

Karty automaticky zařazuji do 53 složek (všechny druhy karet a pozadí) pomocí neuronové sítě z předchozí práce [2]. Síť má přesnost 93, 5% na testovacích datech, tudíž je nutné složky jednu po druhé zkontrolovat, že obsahují pouze karty, které tam opravdu patří. Karty, které jsou špatně identifikovány a nepatří do správné složky se mohou buď smazat nebo přesunout do správných složek.

Předpokládal jsem, že špatně identifikované karty jsou z nějakého důvodu složité na identifikaci a proto jsem je ručně přesouval do správných složek. Díky tomu by se mohla zvýšit přesnost nové neuronové sítě. Zejména kvůli tomuto procesu byl sběr trénovacích dat tak časově náročný.



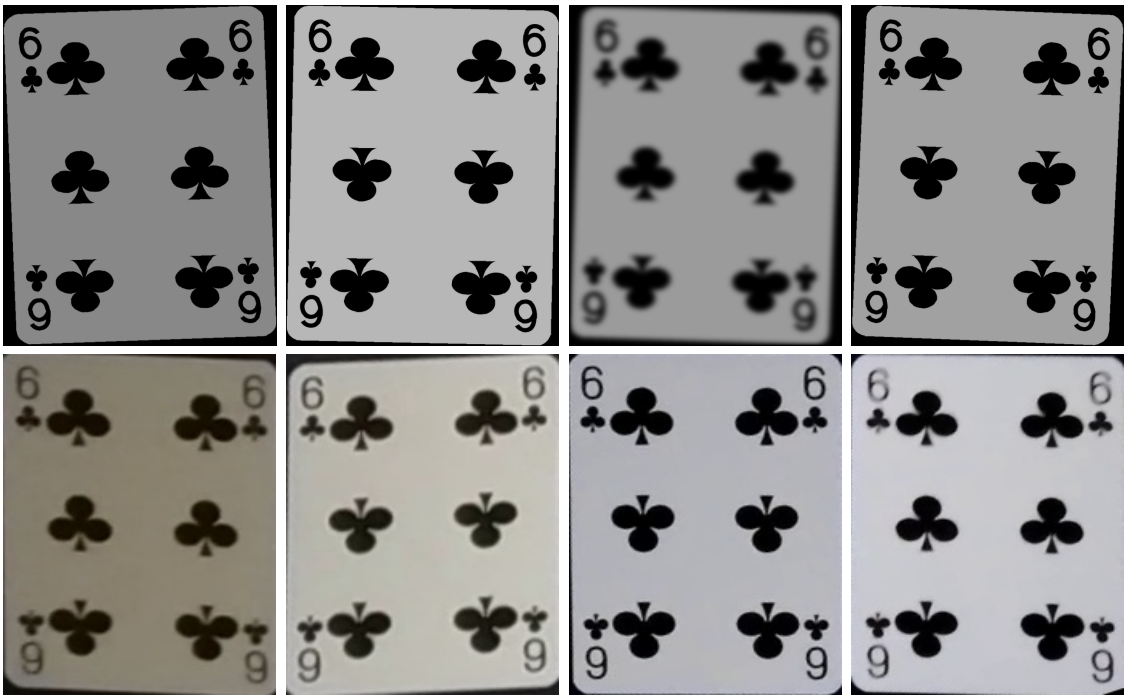
Obrázek 3.11: Špatně klasifikované karty

Nejlepším postupem by při řešení problémů ručního přesouvání byla nejspíš střední cesta. Z obrázku 3.11 bych přesunul do správné složky pouze srdcového kluka, který je znehodnocen odrazem světla a zbytek karet bych smazal. Na obrázku také lze vi-

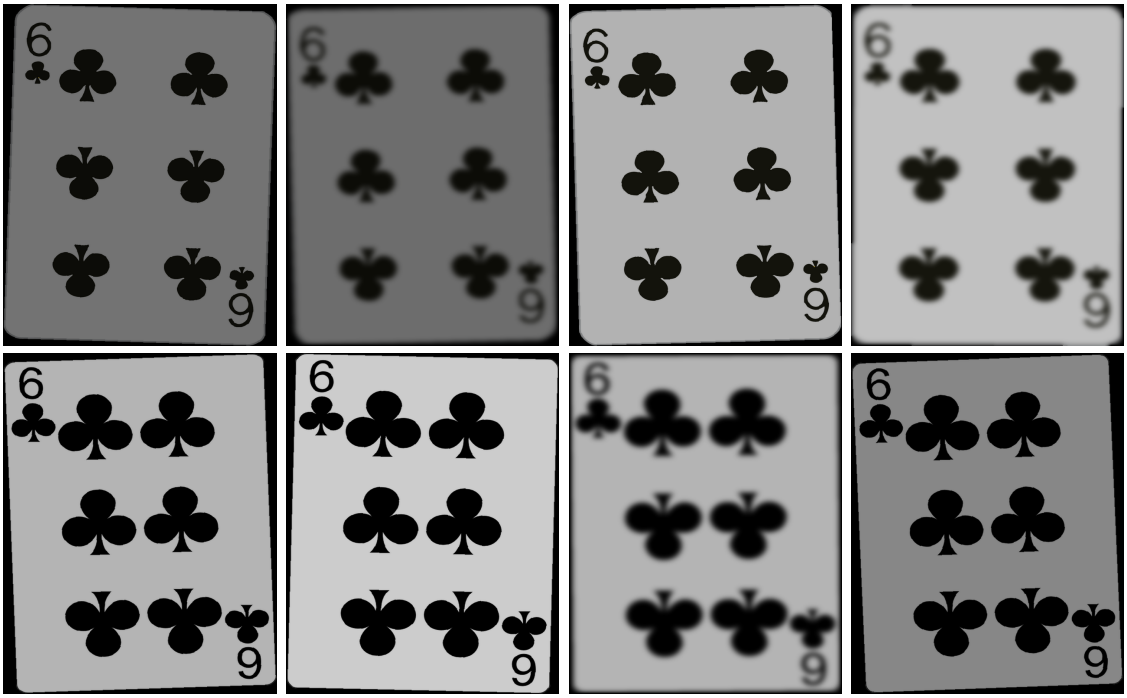
dět, že původní neuronová síť rozpoznala špatně i karty bez nějaké vady - viz piková šestka nejvíce vpravo.

3.3.2 Výroba syntetických karet pro trénovací dataset

Aby generovaná karta vypadala více reálně, jsou na ní aplikovány augmentace. Jako augmentace jsem zvolil rotaci, změnu jasu, horizontální a vertikální posun, rozostření a rotaci o 180°. V práci [2] byl použit navíc i šum. Tento jev však na reálných kartách nikdy nenastal a proto ho v této práci nepoužívám. Hlavním cílem augmentací syntetických karet je, aby se co nejvíce podobaly kartám reálným.



Obrázek 3.12: Porovnání: Nahoře syntetické karty po augmentaci, dole reálné karty



Obrázek 3.13: Syntetické karty po augmentaci, zbylé dva sety

Zde provedené augmentace slouží k tomu, aby se v datasetu neobjevovala stále stejná karta a aby se syntetické karty podobaly co nejvíce reálným kartám. Všechny karty budou augmentovány ještě jednou v další sekci.

3.3.3 Augmentace trénovacího datasetu

V předchozí sekci jsem augmentoval syntetické karty, aby se podobaly reálným kartám. Cílem augmentací v této sekci je simulovat i nechtěné scénáře, které mohou nastat při reálném použití sítě. Například:

1. **Špatné osvětlení**: Karta se leskne nebo naopak není dobře vidět.
2. **Používání kamery se špatným barevným vyvážením**: Toto nastává zejména u bílé barvy. Bílá barva je kamerami často natočena spíše jako žlutá.
3. **Karta je posunuta či rotována**: To je způsobeno nejčastěji chybou programu na vyjmutí karet.

ImageDataGenerator

Pomocí tohoto modulu jsou augmentovány všechny karty trénovacího datasetu. V definicích augmentace často stojí, že augmentace slouží k rozšíření trénovacího datasetu. Při standardním použití ImageDataGeneratoru se trénovací dataset nerozšiřuje, ale nahrazuje. Návod, jak augmentované obrázky z modulu získat a rozšířit tak trénovací dataset je zde [62].

Zajímavé je, že tento modul neobsahuje často používané metody rozostření a šumu. Konvoluční neuronové sítě jsou na rozostření a šum velice citlivé. I pro mírné augmentace těmito metodami značně klesala přesnost sítí při klasifikaci ImageNet datasetu [63]. V argumentu *preprocessing_function* lze ale specifikovat libovolnou funkci předzpracování a tudíž je možné rozostření a šum přidat.

Obrázek 3.14: Karta před augmentacemi

Síla jednotlivých augmentací je náhodná. Většinou se v argumentech při inicializaci generátoru udává maximální možná míra augmentace nebo interval, ze kterého se vybere náhodné číslo. Na karty jsou použity následující augmentace:

1. **Rotace**

   V argumentu *rotation_range* se specifikuje celým číslem úhel maximální rotace. Obrázek je poté rotován o náhodný úhel z intervalu $< -rotation\_range, +rotation\_range >$. Níže lze vidět vygenerované obrázky pro $rotation\_range = 45$:

Obrázek 3.15: Rotace o maximálně 45°

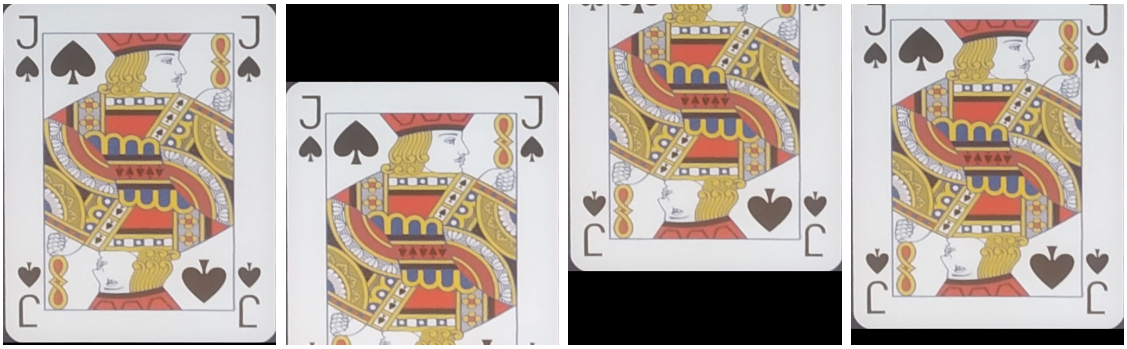
2. **Horizontální posun**

   width_shift_range se udává jako číslo od nuly do jedné. Obrázek se posune počet pixelů z intervalu
   $< -width\_shift \cdot$ *šířka obrázku v pixelech* $, +width\_shift \cdot$ *šířka obrázku v pixelech* $>$. Obrázky níže ukazují posun maximálně o 0.3 šířky obrázku:

Obrázek 3.16: Posun o maximálně 0.3 šířky obrázku

3. **Vertikální posun**

   height_shift_range se používá stejně jako horizontální posun, obrázek se pochopitelně posune vertikálně.



Obrázek 3.17: Posun o maximálně 0.3 výšky obrázku

4. **Jasové transformace**

   Do parametru *brightness_range* se zadává interval od nuly do jedné, kde nula je obrázek bez jasu a jedna je maximální jas.

Obrázek 3.18: Změna jasu od 0.2 do 0.9

5. **Posun intenzit barevných kanálů**

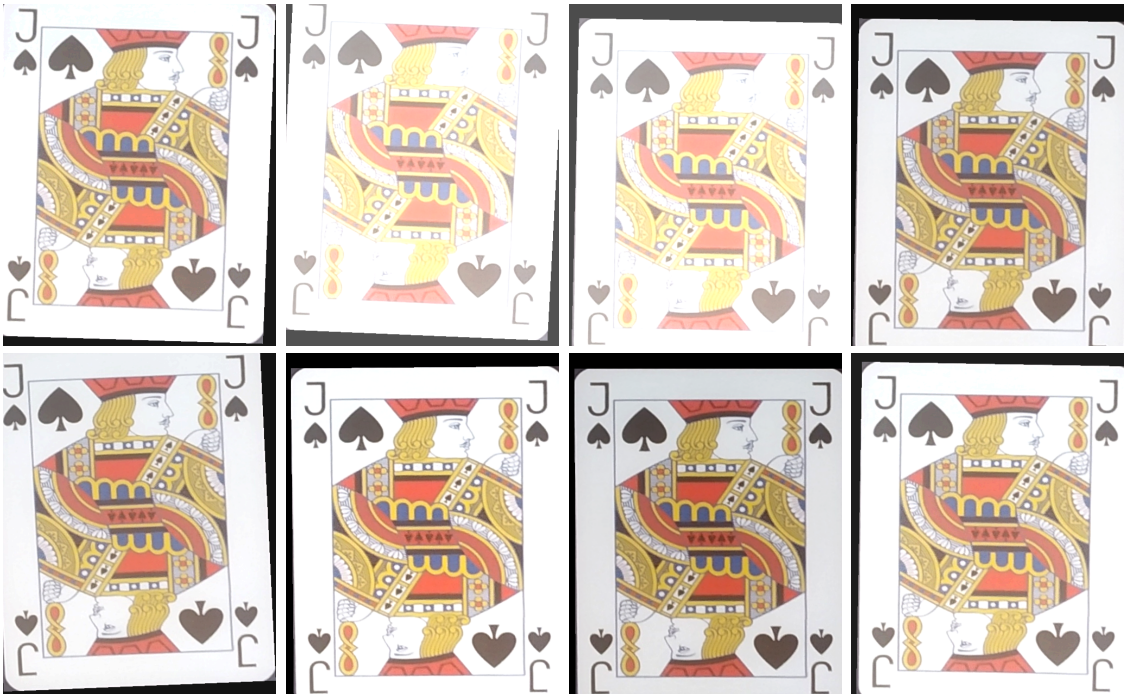
channel_ shift_ range posouvá intenzity barev v jednotlivých barevných kanálech o náhodné číslo. Požadovaný formát je celé číslo od nuly do 255, které znovu udává maximální posun intenzit barevných kanálů.

Obrázek 3.19: Posun barevných kanálů maximálně o 150

Pro augmentaci pokerových karet jsem po několika experimentech použil následující hodnoty:

1. **Rotace**: maximálně o 3°
2. **Horizontální posun**: maximálně o 5% šířky
3. **Vertikální posun**: maximálně o 5% výšky
4. **Změna jasu**: v rozmezí od 10% do 100% současného jasu
5. **Posun intenzit barevných kanálů**: maximálně o 100 jednotek intenzity

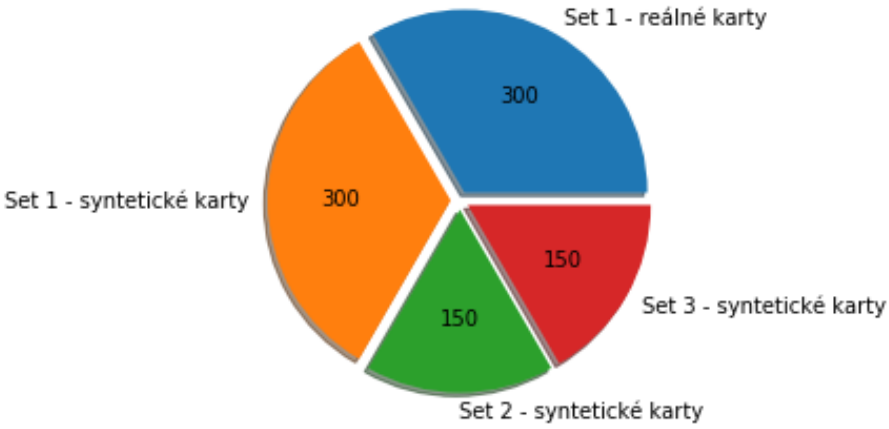
Všechny augmentace jsou na kartu aplikovány naráz s náhodnou silou. Obrázek 3.20 ukazuje, jaké karty vzniknou po augmentaci karty 3.14:



Obrázek 3.20: Náhodně provedené augmentace

3.3.4 Formát dat

Pro klasifikaci do 53 tříd používám pro každou třídu 900 karet s následující strukturou:



Obrázek 3.21: Rozdělení trénovacích obrazů v jedné třídě

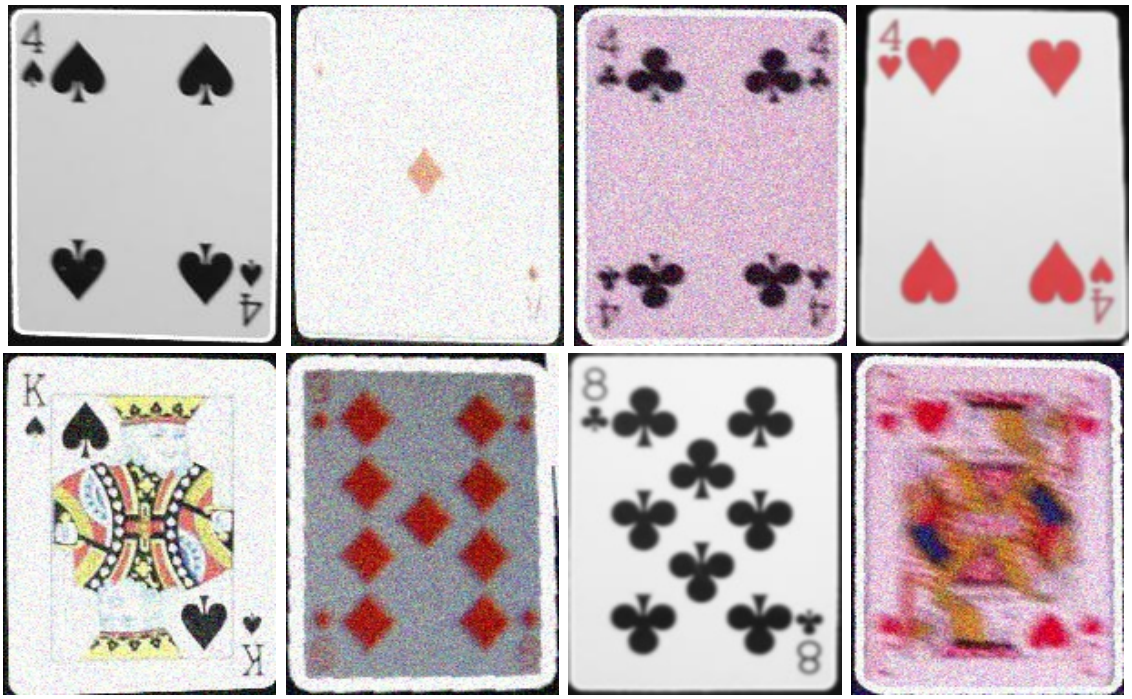
Karty ze setu 1 mám jediné k dispozici fyzicky a proto tento set upřednostňuji

vyšším počtem obrázků v trénovacím datasetu. Počítačem vytvořené karty se budou identifikovat snáz než reálné karty ze setu 1 a proto je jich v datasetu méně.

V tomto případě je dobré zachovat obdélníkový tvar karty. Třídy v pokerových kartách jsou dobře separabilní a proto mi přijde rozlišení 150×200 pixelů jako optimální kompromis mezi velikostí a kvalitou obrázku. Knihovna tensorflow navíc poskytuje možnost změnit rozměr obrázků přímo při jejich načítání ze složek.

Obrazy mohou být uloženy buď klasicky ve složkách, kde v každé složce jsou obrázky náležící jedné třídě nebo ve formátech HDF5 [64], resp. Pickle (Pickle je součástí standardní Python knihovny). Soubory v tomto formátu je těžké modifikovat v uživatelském prostředí a proto raději ukládám data do složek. Modul ImageDataGenerator datasety automaticky promíchá při načítání. Modul keras dokáže ze složek načíst obrázky a podle názvu složky jim přidělit správné labely. Složky karet jsou pojmenovány podle standardní pokerové notace (C - kříže, S - píky, D - káry, H - srdce, A - eso, J - kluk, Q - dáma, K - král). Symboly ♣, ♠, ♢, ♡ nejsou součástí nejčastěji používaného kódování UTF-8 a proto je nepoužívám. Při použití tohoto způsobu ukládání dat je dobré si v trénovacím skriptu zároveň uložit pořadí tříd, se kterým bude síť trénována. To se bude později hodit pro testování přesnosti sítě.

Abych dokázal, že síť v této práci je podstatně kvalitnější než síť prezentovaná v práci [2], používám trénovací data z práce [2] jako validační data pro stávající síť.



Obrázek 3.22: Validační dataset obsahuje karty rozdílných kvalit

Validační data jsou tvořena jen ze setu reálných karet. Hlavním cílem je ověřit fungování sítě v reálném nasazení a proto jsem syntetické karty do validačních dat nepřidával. Jako testovací data slouží karty z reálných a vygenerovaných fotek hracích stolů určené pro vyhodnocení stavu hry. Tento dataset obsahuje jak reálné,

tak syntetické karty. Jak lze vidět z obrázku 3.23, testovací karty jsou podstatně kvalitnější než karty v trénovacím a validačním datasetu.

Obrázek 3.23: Testovací data

3.4 Trénování konvolučních neuronových sítí

V této sekci natrénuji pomocí knihovny tensorflow několik neuronových sítí s různými architekturami, abych zjistil, jak architektura sítě ovlivňuje její přesnost. Ukáže se, že i s jednoduchou architekturou lze dosáhnout uspokojivých výsledků.

Všechny sítě budou nepřímo porovnávány se sítí z [2]. Jak už jsem zmiňoval v předchozí sekci, jako validační data jsem použil trénovací data pro síť ze zmíněné práce.

3.4.1 Inicializace trénovacích a validačních datasetů ImageDataGeneratoru

Zde je podle mého názoru důležité zmínit, že změna rozlišení obrazu je prováděna jako první (i před funkcí *preprocessing*). Proto je vhodné všechny augmentace testovat na tak velkém obrázku, jaký je definován při inicialiaci datasetů. Síla některých augmentací (např. ořez, šum, rozostření...) závisí právě na velikosti obrazu. Mnoha uživatelům kerasu se stalo, že jejich síť byla natrénována s vysokou přesností na trénovací data, ale jejich trénovací data byla kvůli silným augmentacím tak odlišná od reálných dat, že síť klasifikovala reálná data s nízkou přesností [65]. Zde jsou však použity takové augmentace, u kterých na rozlišení obrázku nezáleží.

Na čtvrté řádce provádím augmentace na trénovacích datech. Validační data už byla augmentována dříve a proto jsou jenom normalizována (v páté řádce kódu). Následuje načtení obrazů ze složky se specifikováním velikosti obrazu, velikosti dávky a typu klasifikace.

Parametr, který je důležitý, je velikost obrazu *target_size*. Jak je vidět, prvním číslem v definici je výška a ne šířka obrazu. Tento způsob zápisu může způsobit komplikace uživatelům, kteří předpokládají standardní zápis se šířkou na prvním místě.

Parametr *categorical* kóduje labely do binární matice. Počet řádků matice odpovídá počtu obrazů a počet sloupců odpovídá počtu tříd. V řádku je vždy právě jedna jednička, která podle sloupce indikuje, do jaké třídy obraz patří. Zbytek čísel v řádku jsou nuly. To znamená, že do ostatních tříd obraz nenáleží.

```
batch_size = 64
TRAINING_DIR = "karty/"
VALIDATION_DIR = "karty_validace/"
train_dg = ImageDataGenerator(rescale = 1./255,
                                        rotation_range=3,
                                        width_shift_range=0.05,
                                        brightness_range=(0.1, 1),
                                        height_shift_range=0.05,
                                        shear_range=0.2,
                                        channel_shift_range=90.0,
                                        fill_mode='constant')
val_dg = ImageDataGenerator(rescale = 1./255)
train_gen = train_dg.flow_from_directory(TRAINING_DIR,
                                                target_size=(200,150),
                                                batch_size=batch_size,
                                                class_mode='categorical')
val_gen = val_dg.flow_from_directory(VALIDATION_DIR,
                                          target_size=(200,150),
                                          batch_size=batch_size,
                                          class_mode='categorical')
train_steps = train_gen.samples // batch_size
val_steps = val_gen.samples // batch_size
```

Listing 3.1: Ukázka inicializace modulu ImageDataGenerator

3.4.2 Použité architektury konvolučních neuronových sítí

Celkem trénuji pět architektur sítí. Trénované sítě se liší svojí komplexností a použitými vrstvami. Cílem této sekce je ukázat, které architektury sítí mohou být vhodné ke klasifikaci jednodušších obrazů. Je vhodné si v trénovacím skriptu ukládat historii průběhu k jejich snazšímu vykreslení. Z grafů se dá lépe zjistit chování sítě. K ukládání průběhu používám modul Pickle.

V knihovně tensorflow se sítě vytváří jednoduše. Níže je vidět příklad inicializace architektury sítě z obrázku 3.24. Prvním řádkem se vytvoří prázdná konvoluční neuronová síť. V dalších řádcích přidávám komponenty. Na druhém řádku zároveň zadávám velikost vstupního obrazu. Číslo 3 v parametru *input_shape* označuje počet barevných kanálů vstupního obrazu.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(200, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dense(2048))
model.add(Activation('relu'))
model.add(Flatten())
```

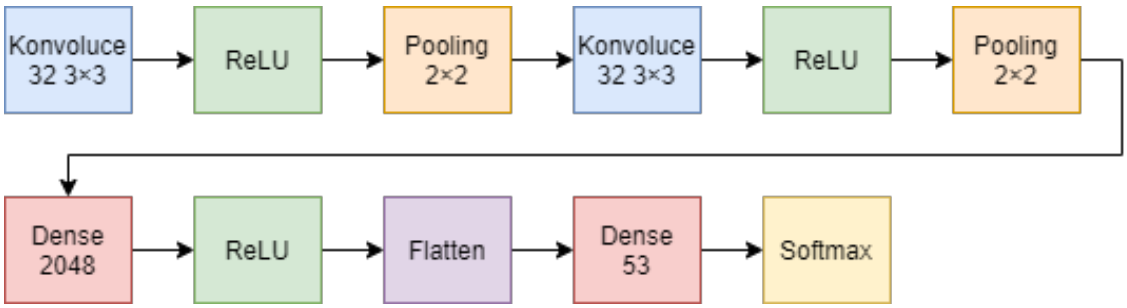
```
model.add(Dense(53))
model.add(Activation('softmax'))
```

Listing 3.2: Inicializace architektury sítě

Pro snadnější porozumění kódu přikládám vysvětlení pojmů (tyto pojmy jsou detailněji popsány v teoretické části 2.3):

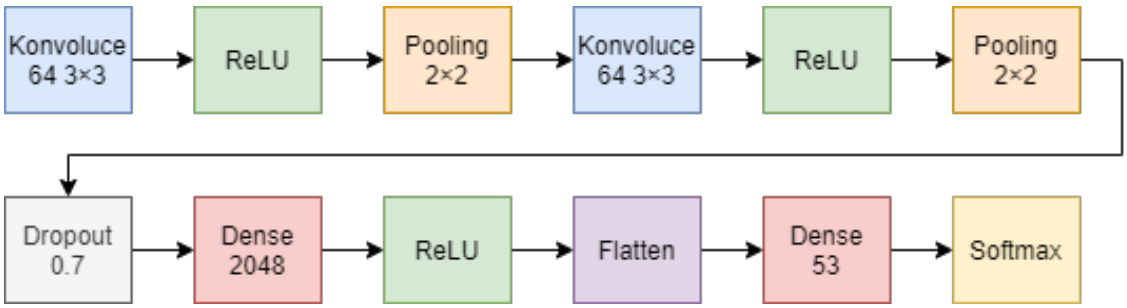
1. **Conv2D**: První číslo značí počet filtrů konvoluce. Druhým parametrem je jejich velikost.
2. **Activation('relu')**: Aktivační funkce ReLU mapující záporný příznak na nulu (viz. 2.11(b)).
3. **Flatten**: Modul převede pole ve třech dimenzích do jednodimenzionálního pole.
4. **Dropout**: Číslo od nuly do jedné udává náhodný počet odpojených vstupních příznaků.
5. **Dense**: Vrstva, která spojuje neurony v jejich blízkostí.
6. **Softmax**: Aktivační funkce normalizuje vstupní hodnoty na vektor o velikosti počtu tříd. Hodnoty vektoru jsou v rozsahu od nuly do jedné a značí pravděpodobnost příslušnosti dané třídě (viz. 2.11(d)).

Architekturu na obrázku 3.24 níže jsem převzal z práce [2]. Oproti většině ostatních sítí se v ní nachází i skrytá dense vrstva. V síti chybí dropout vrstva, což nemusí být negativem.



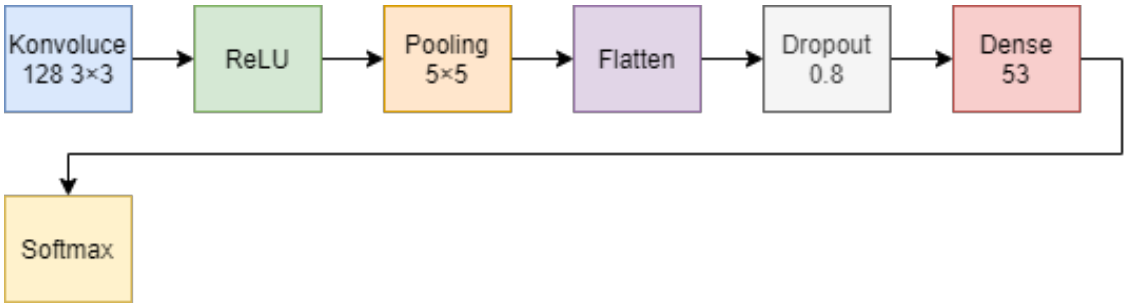
Obrázek 3.24: Konvoluční neuronová síť z práce [2]

Na obrázku 3.25 se nachází upravená verze předchozí sítě 3.24. V síti se navíc nachází dropout vrstva. Zároveň byly zdvojnásobeny počty filtrů v konvolučních vrstvách.



Obrázek 3.25: Síť z práce [2] s přidaným dropoutem a více konvolučními filtry

Síť níže obsahuje jen jednu konvoluční vrstvu s pooling vrstvou o velikosti filtru 5×5 (všechny v práci použité pooling vrstvy jsou typu max pooling) a vysokým dropoutem. Pro jednoduché úkoly by tato síť mohla být dostačující.

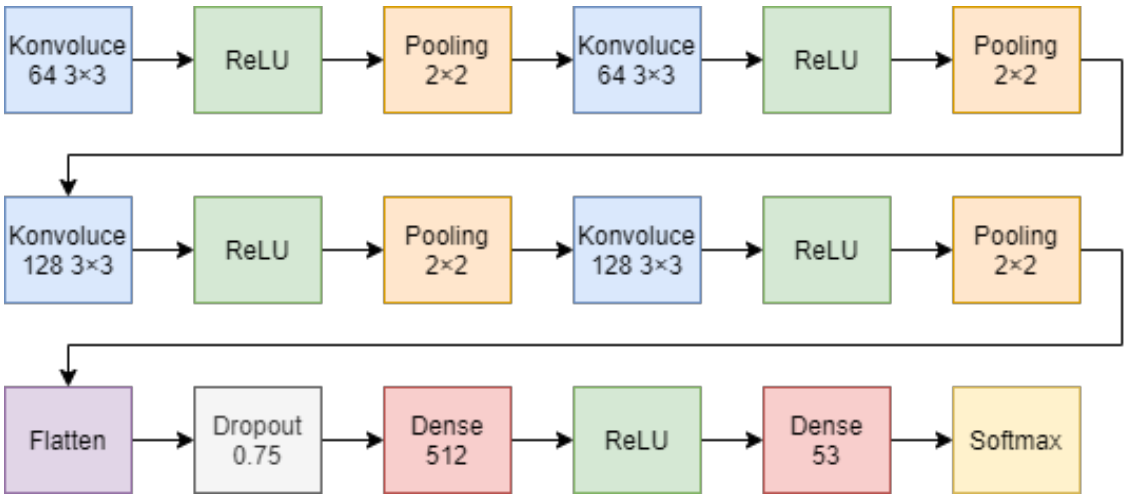


Obrázek 3.26: Síť s jednou konvoluční vrstvou, pooling vrstvou a dropoutem

Na obrázku níže je jednoduchá síť tvořená pouze jednou konvoluční vrstvou.

Obrázek 3.27: Síť tvořená pouze konvoluční a ReLU vrstvou

Síť na obrázku 3.28 obsahuje čtyři konvoluční vrstvy. Očekávám, že dosáhne nejlepších výsledků. Je ze všech sítí nejsložitější, jediná obsahuje čtyři konvoluční vrstvy. Zařazené pooling vrstvy slouží jako filtry na šum. Vysoký dropout by měl zaručit, že nedojde k přetrénování sítě.



Obrázek 3.28: Síť se čtyřmi konvolučními vrstvami a jednou skrytou dense vrstvou

Na konci architektur všech sítí se vždy vyskytují vrstvy flatten, dense a aktivační funkce softmax vyhodnocující výsledek klasifikace.

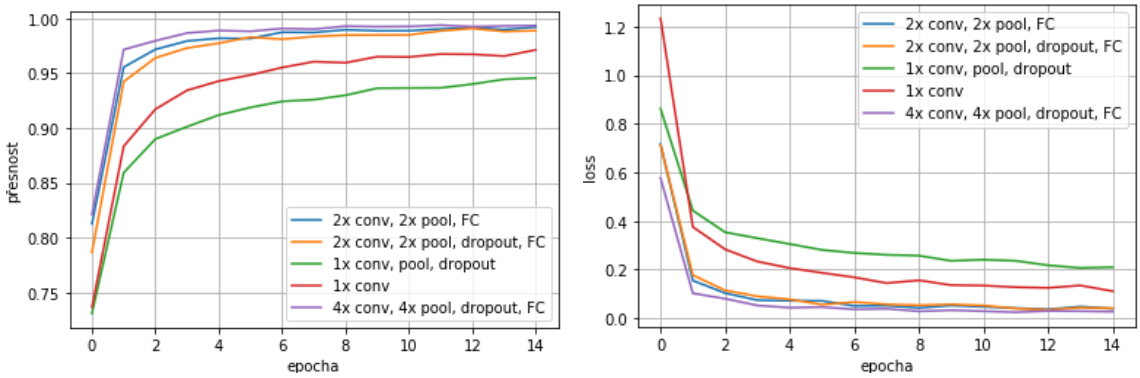
Kód níže navazuje na kód inicializace sítě. Zvolil jsem optimalizační algoritmus ADAM (viz. 2.3.6). Na posledních dvou řádcích je ukázáno ukládání historie pro jednodušší vykreslení průběhů do grafu. Průběhy trénování budou ukázány v další sekci.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

hist = model.fit_generator(train_gen,
                           steps_per_epoch = train_steps,
                           validation_data = val_gen,
                           validation_steps = val_steps,
                           epochs=15)

model.save('sit1')
with open('prubeh1', 'wb') as file_pi:
        pickle.dump(hist.history, file_pi)
```

Listing 3.3: Kód pro trénování sítí navazující na inicializaci architektury sítě 3.2

3.4.3 Výsledky trénování konvolučních neuronových sítí

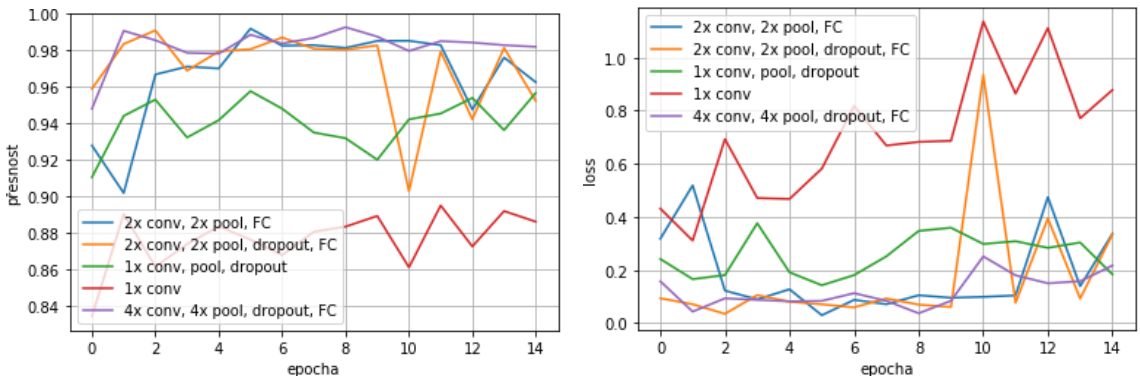
Všechny sítě jsou trénovány po dobu 15 epoch. Jak je vidět na obrázku 3.29(a), u většiny sítí se přesnost po 15 epochách zhruba ustálí. Přesnost na trénovacích datech nemusí být v reálném nasazení relevantní (obzvláště když jsou při trénování použita augmentovaná data a některé sítě obsahují vysoký dropout). Z obrázků 3.4.3 je však možné předpovídat, které sítě dosáhnou vysokých přesností na reálných datech a které ne.



(a) Přesnost klasifikace (b) Ztrátová funkce

Obrázek 3.29: Průběhy na trénovacích datech

Nejméně přesné na trénovacích datech jsou sítě s jednou konvoluční vrstvou. Přesnost sítě s jednou konvoluční vrstvou, pool vrstvou a dropout vrstvou (v obrázcích 3.4.3 a 3.4.3 zeleně) je na trénovací data méně přesná než síť bez pool a dropout vrstvy (červené průběhy). Za to může pravděpodobně přidaná dropout vrstva, která v průběhu trénování náhodně odpojuje příznaky z předchozí vrstvy. Naopak slušných přesností dosáhly všechny tři složitější sítě. Upravená síť z práce [2] dosahuje podobných přesností jako původní síť bez úprav. Zde se dropout vrstva neprojevila tolik. Je to nejspíš kvůli přítomnosti vysokého počtu neuronů v dense vrstvě. Jako ztrátovou funkci jsem zvolil funkci kategorické křížové entropie. Obrázek 3.29(b) ukazuje, že tato ztrátová funkce zrcadlí průběhy na obrázku 3.29(a).



(a) Přesnost klasifikace (b) Ztrátová funkce

Obrázek 3.30: Průběhy na validačních datech

Dalšími obrázky 3.4.3 jsou průběhy validace. Na obrázku 3.30(b) je vidět průběh ztrátové validační funkce. Tento průběh opět zrcadlí graf průběhů validačních přesností. Počáteční fluktuace ztrátové validační funkce může být způsobena tím, že proces nastavování vah při trénování sítě je náhodný a síť v tuto chvíli ještě nenašla robustní řešení. V pozdějších fázích trénování to však může být známkou přetrénování sítě. To ukazuje například průběh druhé sítě (v grafech oranžově) s dvěma

konvolučními vrstvami a dropoutem. Tato síť by měla mít právě díky dropout vrstvě vyrovnané průběhy na validačních datech bez skoků, což podle obou obrázků 3.4.3 není pravda. Je možné, že zvolená hodnota dropoutu 0.7 je již příliš vysoká.

V tabulce 3.4.3 níže lze vidět dosažené přesnosti všech sítí. Pozitivním zjištěním je, že 4 z 5 sítí dosáhly na validačních datech lepších přesností než síť, která pomocí těchto dat byla natrénovaná (jedná se o síť z práce [2]). Cílem této práce však je natrénovat konvoluční neuronovou síť s přesností alespoň 99% na testovacích datech, což podle přesností na validačních datech splňuje pouze poslední síť se čtyřmi konvolučními vrstvami (v grafech fialově).

Popis sítě	**trénovací [%]**	**validační [%]**
2x conv, 2x pool, FC	0,992	0,962
2x conv, 2x pool, dropout, FC	0,989	0,952
1x conv, pool, dropout	0,946	0,957
1x conv	0,971	0,877
4x conv, 4x pool, dropout, FC	0,995	0,998

Tabulka 3.1: Porovnání přesností sítí

Upravená síť na druhém řádku (v grafech oranžově) dosáhla nižších přesností než původní síť (v průbězích modrá). Přidané konvoluční filtry v kombinaci s dropout vrstvou by měly zajistit vyšší přesnost alespoň na validačních datech. Vysoký dropout by vzhledem k velkému počtu neuronů v dense vrstvě sítě mohl přispět k větší robustnosti sítě. Podle výsledků se však zdá, že přítomnost dropout vrstvy s vysokou hodnotou v architektuře této sítě je kontraproduktivní.

Překvapivé je, že třetí síť s jednou konvoluční vrstvou (v průbězích odlišena červenou barvou) dosahuje na validačních datech podobné přesnosti jako první dvě sítě. Nejspíše je to dáno právě silnějším pooling filtrem, díky kterému se z relativně velkého obrázku extrahují pouze důležité příznaky. U třetí sítě je zároveň vidět pozitivní vliv vysokého dropoutu na její přesnost (na rozdíl od oranžové sítě). Ačkoliv jsou trénovací data podstatně kvalitnější než validační data, dosáhla síť vyšších přesností na validačních datech.

Důležitost pool a dropout vrstev ve třetí síti (zelené průběhy v grafech) dokazují výsledky jednoduché sítě obsahující pouze jednu konvoluční vrstvu (v tabulce na čtvrtém řádku). Tato síť sice dosahuje lepších přesností na trénovacích datech, je však přetrénovaná a tedy nepřesná na validačních datech. Vzhledem k její jednoduchosti je pochopitelné, že nedosahuje přesností ostatních sítí.

Navržená síť se čtyřmi konvolučními vrstvami se zdá být nejlepší. Je to zřejmě hlavně díky její komplexnosti. Podobné přesnosti na trénovacích a validačních datech jsou známkou robustnosti sítě. Tuto síť budu používat dále ve své práci.

3.5 Analýza snímku pokerového stolu

V této chvíli je vytvořen algoritmus pro vyjmutí karty ze snímku a je natrénována neuronová síť, která kartu klasifikuje. Jsou tedy připraveny všechny potřebné nástroje k analýze snímku stolu.

3.5.1 Zjištění příslušností karet

Nyní se zjistí, kolik hráčů je ve hře, jaké mají hráči karty a jaké karty byly rozdány na stůl. Princip řešení lze rozdělit do pěti kroků:

1. **Vyjmutí všech karet ze snímku**: Karty se vyjmou algoritmem popsaným v sekci 3.1. Kromě samotných karet jsou uloženy i jejich souřadnice středů v obraze. Ty se využijí v krocích 4 a 5.

2. **Klasifikace všech karet**: Pomocí neuronové sítě se klasifikují všechny karty v obraze. Zjistí se tedy jejich hodnota a barva.

3. **Zjištění počtu hráčů**: Algoritmus předpokládá, že každý hráč vlastní dvě karty a zároveň že je na snímku vždy právě pět společných karet (libovolně otočených). Pokud se celkový počet karet označí jako K, potom se neznámý počet hráčů N vypočítá následovně:

$$N = \frac{K - 5}{2} \tag{3.1}$$

4. **Seřazení karet podle jejich středů**: Hlavní myšlenkou je rozdělit hrací stůl podle souřadnicové výšky středů karet na tři oblasti (oblast nahoře, oblast veprostřed a oblast dole).

   Souřadnice všech středů karet se nejprve seřadí sestupně podle jejich výšky. Tím vznikne pole, kde první element je karta, která má nejvýše střed a posledním elementem je karta se středem nejníže. Z tohoto pole spočítám pro všechny sousední elementy rozdíl mezi výškami jejich středů. Pokud je rozdíl výšek sousedních elementů vysoký, začíná nová oblast. Pro prostřední oblast je známo, že je tam vždy pět karet (společné karty). Tudíž, následujících pět elementů pole od vysokého rozdílu výšek jsou karty na stole. Poté lze jednoduše prohlásit elementy před vysokým rozdílem výšek za karty v oblasti nahoře. To znamená, že zbytek karet v poli se nachází v dolní oblasti. Výsledkem tohoto kroku jsou tedy tři pole, které obsahují karty s podobně vysokými souřadnicemi středů.

5. **Přiřazení karet**: Nyní seřadím podle souřadnicové šířky středů vzestupně elementy ve všech 3 polích (pole horní oblasti, pole oblasti veprostřed a pole dolní oblasti). Tedy, každé pole reprezentuje jednu oblast výšky a prvním elementem v tomto poli je karta se středem nejvíce vlevo a poslední element pole je karta se středem nejvíce vpravo. Pole oblasti veprostřed reprezentuje karty na stole, tyto karty jsou seřazeny a práce v této oblasti je dokončená. Pro

oblast nahoře se z pole vyjmou vždy první dvě karty. Tyto dvě karty patří pravděpodobně jednomu hráči. Poté se vyjmou další dvě karty a přiřadí se dalšímu hráči. Proces se opakuje dokud horní pole není prázdné. To samé se provede pro karty v dolní oblasti.

Obrázek 3.31: Analýza rozdělením snímku hracího stolu na tři části

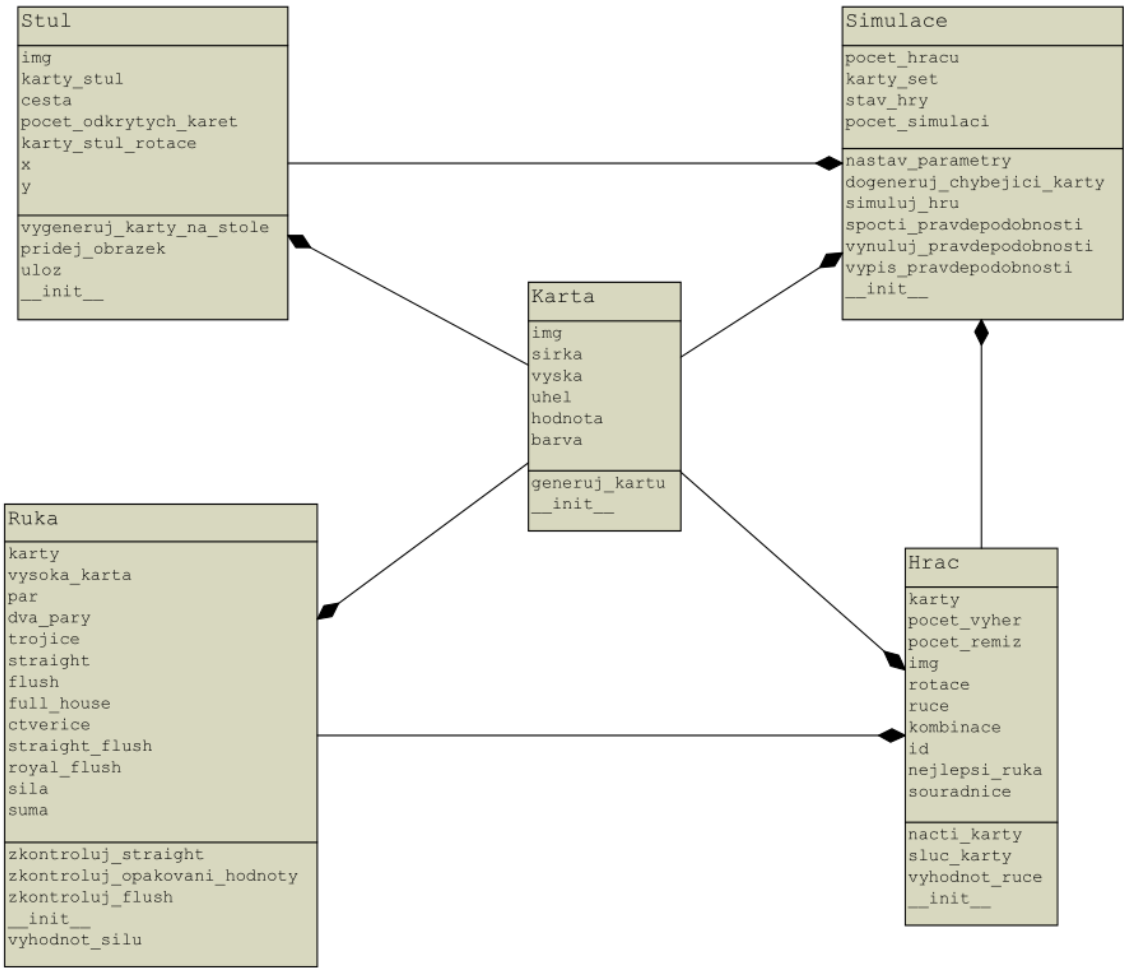
Zjištění příslušnosti karet je prováděno rozdělením hracího stolu na 3 části, přičemž je využit předpoklad, že prostřední část obsahuje 5 společných karet. Ostatní karty tedy patří hráčům. Pokud mají karty středy blízko sebe, patří pravděpodobně tomu samému hráči.

3.5.2 Výpočet pravděpodobností hráčů na výhru a remízu

Nyní jsou hodnoty a barvy karet známé. Rovněž je známo, jaký hráč drží jaké karty a jaké karty leží na stole. Podle počtu zakrytých společných karet lze zjistit, v jakém stavu je hra (preflop, flop, turn, river). Používané metody výpočtu pravděpodobností v pokeru byly zmíněny v sekci 2.1.4. Já jsem si pro výpočet pravděpodobností vybral Monte-Carlo simulace a to zejména kvůli jejich robustnosti a jednoduchosti.

Počet prováděných simulací závisí na stavu hry. Pokud je hra ve stavu river, jsou všechny karty již známé a ke spočtení pravděpodobností na výhru tedy stačí jen jedna simulace. Pro hru ve stavu turn je neznámá pouze jedna karta (tedy možných budoucích stavů hry je $52 - (2 \cdot N + 4)$, kde N je počet hráčů. Při řešení

této práce se ukázala jedna z nevýhod Monte-Carlo simulací - časová náročnost. Pro spočítání všech pravděpodobností ve hře pěti hráčů ve stavu preflop bylo potřeba přibližně 40 sekund (na počítači s procesorem i5-6200U, 2.4 GHz a 4GB RAM bylo pro výpočet pravděpodobností provedeno 20000 simulací). To je způsobeno zejména kombinací dvou skutečností. Tou první je, že tato část práce byla naprogramována v jazyce Python, který sice poskytuje vysokou úroveň abstrakce, avšak oproti kompilovaným jazykům ztrácí na rychlosti. Druhým problémem je samotná optimalizace kódu. Pokud bych tento problém řešil znovu, pravděpodobně bych Monte-Carlo simulace implementoval v jiném, rychlejším, programovacím jazyku (pravděpodobně bych zvolil jazyky C++ či Java). Doba simulací se samozřejmě dá zrychlit snížením počtu simulovaných her. To však negativně ovlivňuje přesnost výsledků. Doba trvání kalkulace pravděpodobností s rostoucím počtem hráčů narůstá lineárně (vztah mezi počtem simulací a dobou trvání je rovněž přibližně lineární).



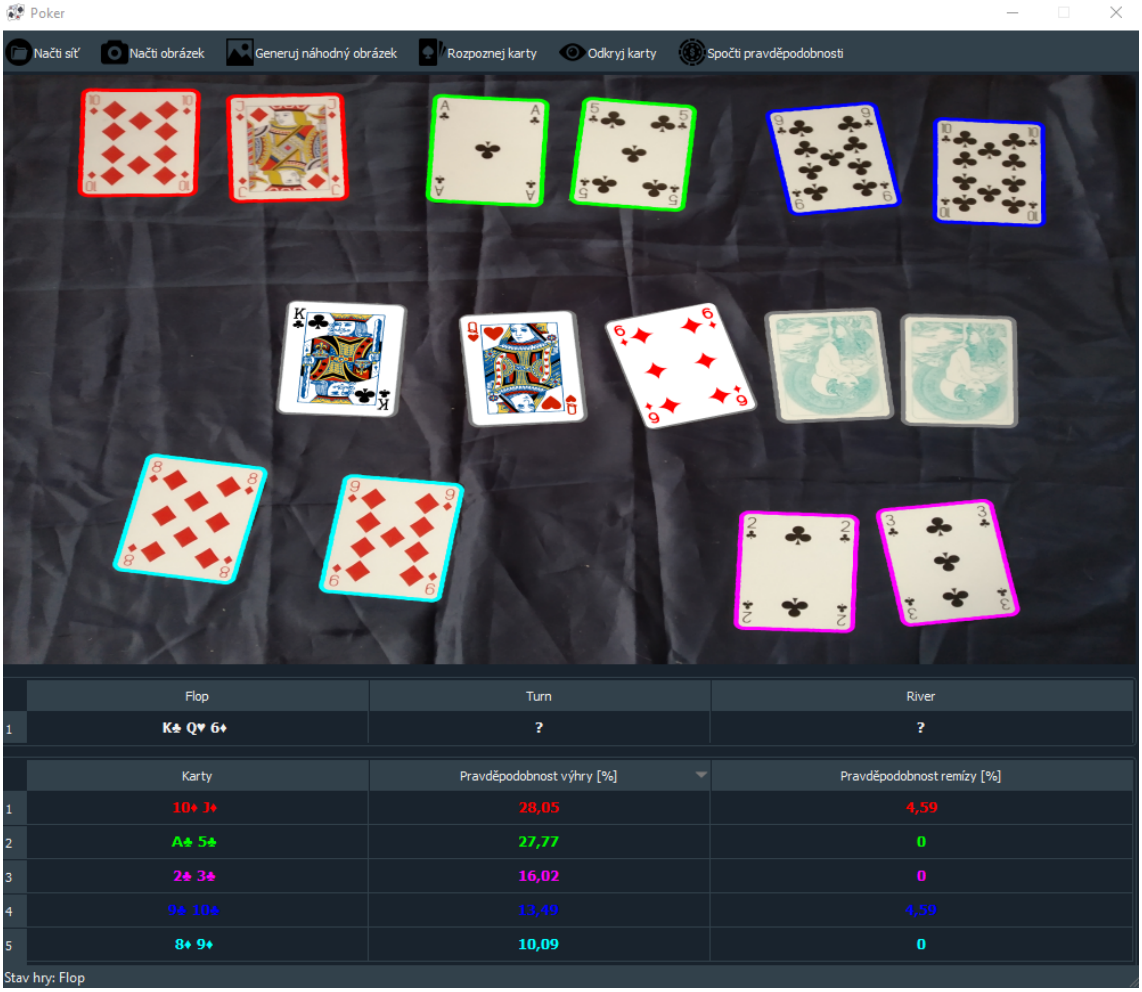
Obrázek 3.32: UML diagram simulace hry

Obrázek 3.32 může lépe nastínit průběh výpočtu pravděpodobností. Simulace probíhá tak, že jsou nejprve místo neznámých karet na stole vygenerovány karty s náhodnou barvou a hodnotou (samozřejmě tak, aby se stejná karta nevyskytovala

na stole vícekrát, počet takto vygenerovaných karet závisí na stavu hry). Následně dochází k simulacím her. Pro každého hráče je vyhodnocena nejsilnější kombinace a hráč, který vyhraje, získá bod (hráči, kteří remizují získají bod v proměnné *pocet_remiz*). Poté následuje další iterace simulace, tzn. místo zakrytých karet na stole se vygenerují další náhodné karty, vyhodnotí se síly rukou a vítěz hry získá bod. Podle rovnice 2.2 se po dokončení simulací nakonec spočítají výsledné pravděpodobnosti na výhru a remízu.

3.6 Grafické uživatelské prostředí

Pro ukázku funkce práce jsem v knihovně *PyQt5* vytvořil grafické rozhraní. Na obrázku 3.33 níže lze vidět výsledek.



Obrázek 3.33: Grafické uživatelské rozhraní

Barvy kontur kolem karet udávají informaci o příslušnosti karet. Na obrázku 3.33 patří karty ohraničené červenými konturami hráči vlevo nahoře. Tento hráč má tedy červenou barvu a jeho karty a pravděpodobnosti na výhru jsou v tabulce rovněž

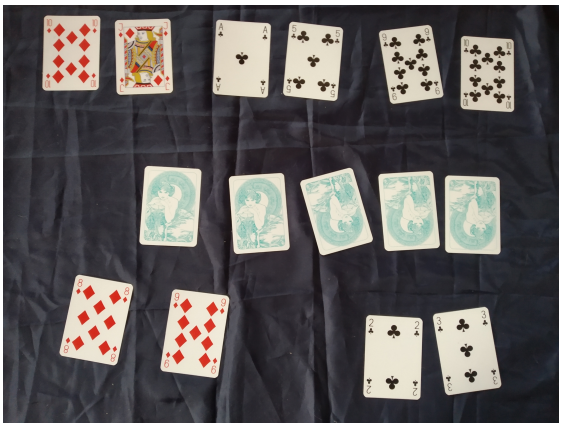
červené. Kolem sdílených karet jsou vždy šedé kontury. Dolní polovina grafického rozhraní poskytuje informace o stavu hry, tzn. které karty byly rozpoznány, komu karty patří a pravděpodobnost hráčů na výhru. Osobně se mi líbí více tmavá tématika aplikací, proto jsem místo standardního stylu *PyQt5* aplikace použil tématiku *qdarkstyle*.

GUI se skládá z následujících komponentů:

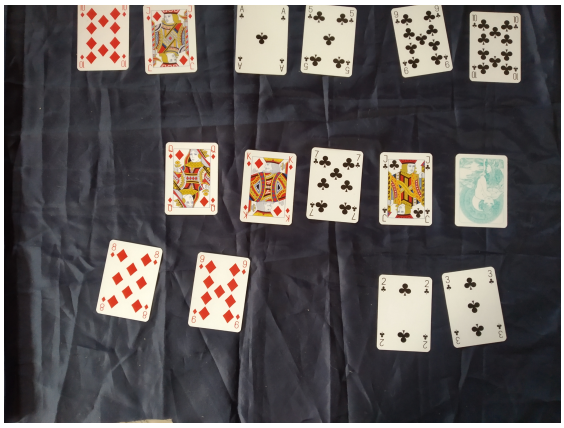
1. **Panel tlačítek**: Panel tlačítek se nachází nahoře. Tlačítky v tomto panelu se ovládá grafické prostředí. Ikony tlačítek jsou staženy z webu [66].
   (a) **"Načti síť"**: Po kliknutí se otevře dialog pro vybrání složky. Z této složky bude načtena neuronová síť na klasifikaci karet.
   (b) **"Načti obrázek"**: Tlačítko otevře dialog pro načtení libovolného obrázku, který bude analyzován.
   (c) **"Generuj náhodný obrázek"**: Vygeneruje náhodný syntetický snímek stolu, tento proces je popsán v sekci 3.2.2.
   (d) **"Rozpoznej karty"**: Toto tlačítko vyjme všechny karty z obrazu (viz. sekce 3.1), pomocí neuronové sítě zjistí jejich hodnoty a barvy a nakonec zjistí jejich příslušnost (viz. sekce 3.5.1).
   (e) **"Odkryj karty"**: Toto tlačítko slouží k simulaci dalšího stavu hry. Místo zakrytých karet se vygenerují nové karty a přepočítají se pravděpodobnosti hráčů na výhru a remízu. Na obrázku 3.33 byly dogenerovány 3 karty na stole.
2. **Plátno**: Zde je zobrazen obrázek a průběh jeho analýzy. Po kliknutí na tlačítko "Rozpoznej karty"se do obrázku zakreslí kolem karet kontury. Po použití tlačítka "Odkryj karty"se do obrázku vloží nové karty.
3. **Tabulky**: Tabulky zobrazují, které karty byly detekovány, komu patří a pravděpodobnosti hráčů na výhru či remízu. První, horní tabulka podává informace o kartách na stole pro flop, turn a river. Řádky druhé tabulky představují hráče a jejich šance na výhru. Sloupce v této tabulce lze řadit - viz. obrázek 3.33, kde jsou hráči seřazeni od nejvyšší pravděpodobnosti na výhru po nejnižší.
4. **Stavový panel**: Stavový panel se nachází ve spodní části GUI. Poskytuje informace jak o stavu hry (jako na obrázku 3.33), tak o stavu programu (např. síť nebyla správně načtena atd.).

3.7 Výsledky testování

V této části bude ověřeno, zda navržené algoritmy a konvoluční neuronová síť fungují správně. Pro testování budu používat jak vyfocené reálné snímky stolů, tak syntetické snímky stolů (viz. sekce 3.2.2). Testovány budou přesnosti správné klasifikace konvoluční neuronovou sítí a přesnosti správně vypočtených šancí hráčů na výhru a remízu. Níže lze vidět některé snímky, na kterých byla aplikace testována.

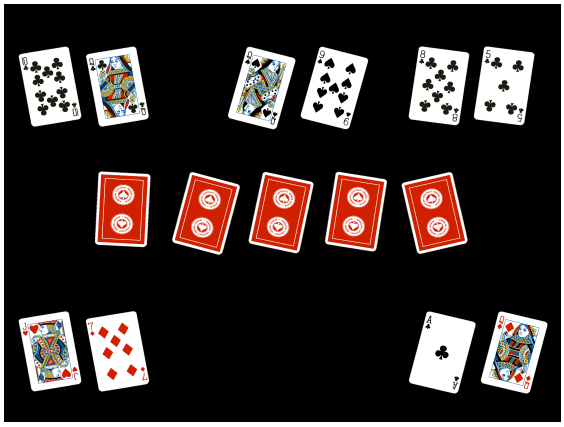


(a) Snímek číslo 1



(b) Snímek číslo 2

(c) Snímek číslo 3



(d) Snímek číslo 4

Obrázek 3.34: Testovací snímky

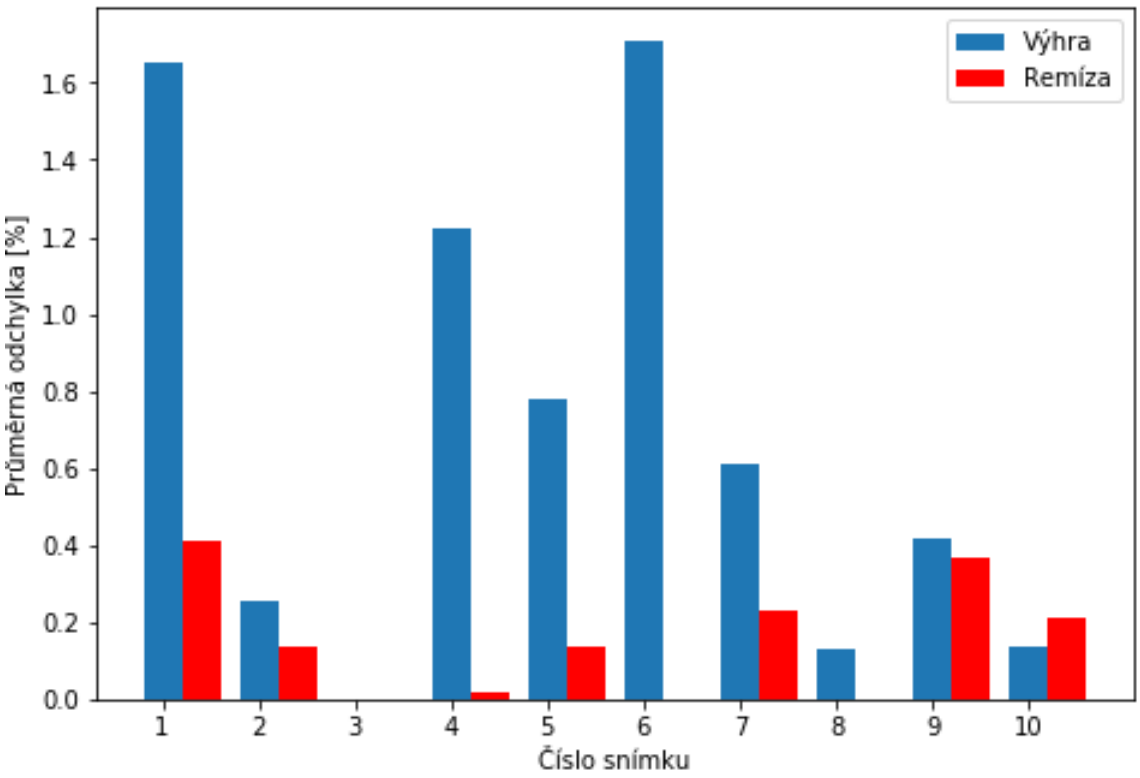
Pro testování jsem zvolil celkem 10 snímků, z toho bylo 6 snímků pořízených za různých světelných podmínek a zbývající 4 snímky byly vygenerovány počítačem. Algoritmus pro analýzu hry byl úspěšný ve všech případech. Dohromady se na snímcích nacházelo 136 karet. Konvoluční neuronová síť klasifikovala všechny karty správně.

K vyhodnocení přesnosti spočtených pravděpodobností hráčů na výhry a remízy jsem zvolil porovnání pravděpodobností z mé aplikace s online kalkulátory (kalkulátor cardschat [67] a kalkulátor cardplayer [68]). Pro každého hráče jsou nejprve spočteny průměrné pravděpodobnosti z online kalkulátorů (výsledky z online kalku-

látorů nejsou konsistentní, proto používám průměrování). Tato hodnota je porovnána s pravděpodobnostmi v navržené aplikaci. Celková odchylka pro daný snímek je poté sumou všech odchylek zprůměrovanou přes počet odchylek. Výpočet průměrné absolutní odchylky MAD_i pro snímek i může být lépe zřejmý z rovnice 3.2 níže:

$$MAD_i = \frac{\sum_{j=1}^{N_i} |x_j - \bar{x_j}|}{N_i}, \tag{3.2}$$

kde N_i je počet hráčů, kteří se na snímku i vyskytují, j je index hráče, x_j představuje vypočítanou pravděpodobnost hráče j z mé aplikace a $\bar{x_j}$ je průměrná pravděpodobnost hráče j z online kalkulátorů [67] [68]. Výsledky lze vidět na grafu 3.35:



Obrázek 3.35: Průměrné odchylky pravděpodobností pro 10 testovacích snímků

Průměrné odchylky znázorněné v grafu 3.35 se liší podle stavu hry. Například snímek číslo 3 (obrázek 3.34(c)) je ve stavu turn a na stole je všech pět karet a tudíž nebylo nutné provádět náhodné simulace. Nepřítomnost náhody má za následek nulové odchylky (pokud byla samozřejmě správně vyhodnocena hra). Naopak pro hru ve stavu preflop (ve stavu preflop je například snímek číslo 1 3.34(a) a snímek číslo 4 3.34(d)) je pro hru pěti hráčů celkem $C(42, 5) = 850668$ možných stavů hry a proto jsou pro tyto snímky odchylky vyšší.

	Flop	Turn	River
1	? ? ?	?	?

	Karty	Pravděpodobnost výhry [%]	Pravděpodobnost remízy [%]
1	A♣ Q♦	26,45	0,75
2	Q♠ 9♠	19,94	0,75
3	10♣ Q♣	19,4	0,75
4	J♥ 7♦	17,84	0,12
5	8♠ 5♣	15,62	0,12

Stav hry: Pre-Flop

(a) Výstup z aplikace pro snímek číslo 4 (3.34(d)

(b) Výstup z online kalkulátoru cardplayer [68]

(c) Výstup z online kalkulátoru cardschat [67]

Obrázek 3.36: Porovnání výsledků pro hru ve stavu preflop (snímek číslo 4 z obrázku 3.34(d))

4 Závěr

V této práci byl vylepšen algoritmus na vyjmutí pokerových karet ze snímku z práce [2] a navržen generátor syntetických karet a snímků stolů. Nasbíráním obrazů reálných karet v kombinaci se syntetickými kartami byl vytvořen dataset pro trénink konvoluční neuronové sítě. Poté byl snímek analyzován a pomocí Monte-Carlo simulací byly vypočteny pravděpodobnosti všech hráčů na výhru či remízu pro variantu Texas hold'em. Pro vizualizaci výsledků práce jsem v knihovně PyQt5 navrhl grafické prostředí.

Algoritmus vyjmutí pokerových karet ze snímku fungoval na testovacích snímcích spolehlivě a výsledné karty dosahovaly na snímcích s dobře segmentovatelným pozadím uspokojivých kvalit (viz. obrázek 3.5).

Návrh algoritmů pro tvorbu syntetických dat proběhl bez většího problému. Karty jsou vytvořeny s požadovanou rotací, hodnotou, barvou a velikostí. Náhodně vygenerované syntetické snímky stolů posloužily jako část testovacích dat při vyhodnocení výsledků práce.

Za hlavní úspěch považuji natrénování konvoluční neuronové sítě s přesností 99,998% na validačních datech, navzdory tomu, že některé karty ve validačním datasetu byly velice silně augmentovány (příklady karet ve validačním datasetu jsou na obrázku 3.22). Konvoluční neuronová síť byla natrénována na tři odlišné sety karet.

Karty byly přiřazeny rozdělením snímku na tři pomyslné oblasti. Toto řešení fungovalo na testovacích snímcích spolehlivě. Místo ke zlepšení práce vidím ve výpočtu pravděpodobností na výhru. Monte-Carlo simulace jsou robustním, avšak v mém případě časově náročným řešením. Jak bylo zmíněno v sekci 3.5.2, použitím vhodnějšího programovacího jazyku by se mohla doba výpočtu pravděpodobností snížit. Nad rámec zadání této práce jsem naprogramoval simulaci dalšího stavu hry, kde byly do snímku místo otočených karet vloženy syntetické karty.

Naprogramované grafické prostředí je funkční. Design GUI by mohlo vylepšit například elegantnější zobrazení příslušností karet. Vizualizace pomocí tabulek poskytuje možnost seřazení výher, postrádá však podle mého názoru estetiku.

Do budoucna by bylo zajímavé vytvořit umělou inteligenci na rozhodování o herní strategii a vytvořit tak autonomní pokerovou aplikaci. K tomu by však bylo nutné natrénovat konvoluční neuronovou síť na identifikaci a počítání žetonů, pravděpodobně zvolit jinou metodu výpočtu pravděpodobností, která by brala v úvahu sázky hráčů a navrhnout rozhodovací algoritmus pro volbu strategie (k tomu by nejspíše byla vhodná klasická neuronová síť či metoda CFR - viz. 3.5.2). Dalším možným budoucím použitím práce může být například analýza snímku stolu v reálném čase či rozpoznání varianty pokerové hry a její následná analýza.

Literatura

[1] N. Brown and T. Sandholm, "Superhuman ai for heads-up no-limit poker: Libratus beats top professionals," *Science*, vol. 359, no. 6374, pp. 418–424, 2018.

[2] J. Hrdlička, "Detekce a rozpoznávání pokerovỳch karet v digitálním obraze," 2017.

[3] D. Billings, *Algorithms and assessment in computer poker*. 2006.

[4] W. Li and L. Shang, "Estimating winning probability for texas hold'em poker," *International Journal of Machine Learning and Computing*, vol. 3, no. 1, p. 70, 2013.

[5] G. Hanks, "Pokeorology," 2019.

[6] J. Rubin and I. Watson, "Computer poker: A review," *Artificial intelligence*, vol. 175, no. 5-6, pp. 958–987, 2011.

[7] M. Johanson, "Measuring the size of large no-limit poker games," *arXiv preprint arXiv:1302.7008*, 2013.

[8] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 201–240, 2002.

[9] H. Quek, C. Woo, K. Tan, and A. Tay, "Evolving nash-optimal poker strategies using evolutionary computation," *Frontiers of Computer Science in China*, vol. 3, no. 1, pp. 73–91, 2009.

[10] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, "Monte carlo sampling for regret minimization in extensive games," in *Advances in neural information processing systems*, pp. 1078–1086, 2009.

[11] A. E. Nicholson, K. B. Korb, and D. Boulton, "Using bayesian decision networks to play texas holdem poker," *International Computer Games Association (ICGA) Journal (Unveröffentlichter Entwurf)*, 2006.

[12] M. Dlouhỳ and P. Fiala, *Úvod do teorie her*. Oeconomica, 2007.

[13] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Advances in neural information processing systems*, pp. 1729–1736, 2008.

[14] D. Szafron, R. Gibson, and N. Sturtevant, "A parameterized family of equilibrium profiles for three-player kuhn poker," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pp. 247–254, International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[15] C. Kanan and G. W. Cottrell, "Color-to-grayscale: does the method matter in image recognition?," *PloS one*, vol. 7, no. 1, p. e29740, 2012.

[16] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.

[17] D. Kaur and Y. Kaur, "Various image segmentation techniques: a review," *International Journal of Computer Science and Mobile Computing*, vol. 3, no. 5, pp. 809–814, 2014.

[18] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[19] W. M. Wells, W. E. L. Grimson, R. Kikinis, and F. A. Jolesz, "Adaptive segmentation of mri data," *IEEE transactions on medical imaging*, vol. 15, no. 4, pp. 429–442, 1996.

[20] F. Gasparini and R. Schettini, "Skin segmentation using multiple thresholding," in *Internet Imaging VII*, vol. 6061, p. 60610F, International Society for Optics and Photonics, 2006.

[21] S. S. Al-Amri, N. Kalyankar, and S. Khamitkar, "Image segmentation by using edge detection," *International journal on computer science and engineering*, vol. 2, no. 3, pp. 804–807, 2010.

[22] S. Angelina, L. P. Suresh, and S. K. Veni, "Image segmentation based on genetic algorithm for region growth and region merging," in *2012 international conference on computing, electronics and electrical technologies (ICCEET)*, pp. 970–974, IEEE, 2012.

[23] R. Pohle and K. D. Toennies, "Segmentation of medical images using adaptive region growing," in *Medical Imaging 2001: Image Processing*, vol. 4322, pp. 1337–1346, International Society for Optics and Photonics, 2001.

[24] R. Pohle and K. D. Toennies, "A new approach for model-based adaptive region growing in medical image analysis," in *International conference on computer analysis of images and patterns*, pp. 238–246, Springer, 2001.

[25] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, 2015.

[26] R. R. Schaller, "Moore's law: past, present and future," *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.

[27] D. Elizondo, "The linear separability problem: Some testing methods," *IEEE Transactions on neural networks*, vol. 17, no. 2, pp. 330–344, 2006.

[28] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.

[29] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2018.

[30] H. Yonaba, F. Anctil, and V. Fortin, "Comparing sigmoid transfer functions for neural network multistep ahead streamflow forecasting," *Journal of Hydrologic Engineering*, vol. 15, no. 4, pp. 275–283, 2010.

[31] F. Zang and J.-s. Zhang, "Softmax discriminant classifier," in *2011 Third International Conference on Multimedia Information Networking and Security*, pp. 16–19, IEEE, 2011.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[33] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[34] A. Karpathy, "Cs231n: Convolutional neural networks for visual recognition," *Online Course*, 2016.

[35] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of operations research*, vol. 134, no. 1, pp. 19–67, 2005.

[36] J. O. Berger, *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[38] M. Medek *et al.*, "Knihovna pro práci s hlubokými konvolučními neuronovými sítěmi v jazyce c," 2018.

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

[40] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[41] A. Daniely, R. Frostig, and Y. Singer, "Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity," in *Advances In Neural Information Processing Systems*, pp. 2253–2261, 2016.

[42] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[44] A. Karpathy and F.-f. Li, "Automated image captioning with convnets and recurrent nets," 2013.

[45] D. Britz, "Understanding convolutional neural networks for nlp," 2015.

[46] N. M. Rad, A. Bizzego, S. M. Kia, G. Jurman, P. Venuti, and C. Furlanello, "Convolutional neural network for stereotypical motor movement detection in autism," *arXiv preprint arXiv:1511.01865*, 2015.

[47] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[48] M. Stewart, "Advanced topics in deep convolutional neural networks," 2019.

[49] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in neural information processing systems*, pp. 901–909, 2016.

[50] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift. cornell university library," 2017.

[51] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[52] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," in *Asian conference on computer vision*, pp. 189–204, Springer, 2016.

[53] I. Guyon, "A scaling law for the validation-set training-set size ratio," *AT&T Bell Laboratories*, vol. 1, no. 11, 1997.

[54] P. Warden, "How many images do you need to train a neural network?," 2017.

[55] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20–24, 2017.

[56] J. Wang and L. Perez, "The effectiveness of data augmentation in image classification using deep learning," *Convolutional Neural Networks Vis. Recognit*, p. 11, 2017.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[58] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *arXiv preprint arXiv:1312.5402*, 2013.

[59] J. Hrdlička, "Poker." `https://github.com/hrdlickajan/DP`, 2020.

[60] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, "scikit-image: image processing in python," *PeerJ*, vol. 2, p. e453, 2014.

[61] A. C. B. League, "52 playing cards," 2018.

[62] S. Sarin, "Exploring data augmentation with keras and tensorflow," 2019.

[63] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[64] H. Group *et al.*, "Hierarchical data format, version 5," 2014.

[65] dmus, "Keras: resize is done before preprocessing." Github issues, 2018.

[66] Flaticon, "Flaticon, the largest database of free vector icons," 2020.

[67] CardsChat, "Why should i use a poker odds calculator?," 2020.

[68] CardPlayer, "Texas hold'em poker odds calculator," 2020.