

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 Ing. Miroslavu Hlaváčovi, Ph.D. za jeho cenné rady 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 karetní hře poker ve variantě Texas hold’em. Cílem práce je vyhodnotit stav hry ze snímku stolu: kolik hráje hráčů, kdo má jaké karty a spočítat pravděpodobnosti hráčů na výhru. 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 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	1
2	Použité metody a současný stav	2
2.1	Texas hold’em poker	2
2.1.1	Hrací karty	2
2.1.2	Cíl hry	3
2.1.3	Karetní kombinace	4
2.1.4	Počítání pravděpodobností na výhru	5
2.2	Digitální obraz a jeho zpracování	6
2.2.1	Barevné modely obrazu	6
2.2.2	Segmentace obrazu	7
2.3	Konvoluční neuronové sítě	13
2.3.1	Vrstvy konvolučních neuronových sítí	13
2.3.2	Architektury konvolučních neuronových sítí	13
2.3.3	Praktické úvahy	14
3	Praktická část a vyhodnocení experimentů	16
3.1	Získání pokerových karet z obrázku	17
3.1.1	Segmentace snímku	18
3.1.2	Detekce regionů s potenciální kartou	19
3.1.3	Vykreslení kontur kolem karet	20
3.1.4	Rotace a vyjmutí karty ze snímku	21
3.2	Generování syntetických hracích karet a stolů	22
3.2.1	Generování pokerových karet	22
3.2.2	Generování pokerového stolu	23
3.3	Příprava dat pro konvoluční neuronovou síť	25
3.3.1	Sběr reálných karet pro trénovací dataset	25
3.3.2	Výroba syntetických karet pro trénovací dataset	26

3.3.3	Augmentace trénovacího datasetu	28
3.3.4	Formát dat	32
3.4	Trénování konvolučních neuronových sítí	34
3.4.1	Inicializace trénovacích a validačních datasetů ImageDataGeneratoru	34
3.4.2	Použité architektury konvolučních neuronových sítí	35
3.4.3	Výsledky trénování konvolučních neuronových sítí	38
3.5	Analýza snímku pokerového stolu	40
3.5.1	Zjištění příslušností karet	40
3.5.2	Výpočet pravděpodobnosti hráčů na výhru a remízu	42
3.6	Grafické uživatelské prostředí	44
3.7	Výsledky testování	46
4	Závěr	47

1 Úvod

Tato práce navazuje na mou bakalářskou práci [18]. 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 [7].

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, který je vytvořena v knihovně *tensorflow*. Kromě reálných karet a snímků hracích stolů používám i počítačem vygenerované pokerové karty a syntetické snímky herních stolů. Výsledky práce jsou vizualizovány v grafickém prostředí *PyQt5*.

Praktická část této práce může být použita jako pomocný nástroj při hraní Texas hold'em pokeru, jako automatický rozpoznávač stavu pokerové hry či jako jednoduchý simulátor hry. Jelikož se v této práci zabývám pokerem z pohledu počítače, není zde vysvětlena strategie hry ani herní psychologie.

2 Použité metody a současný stav

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

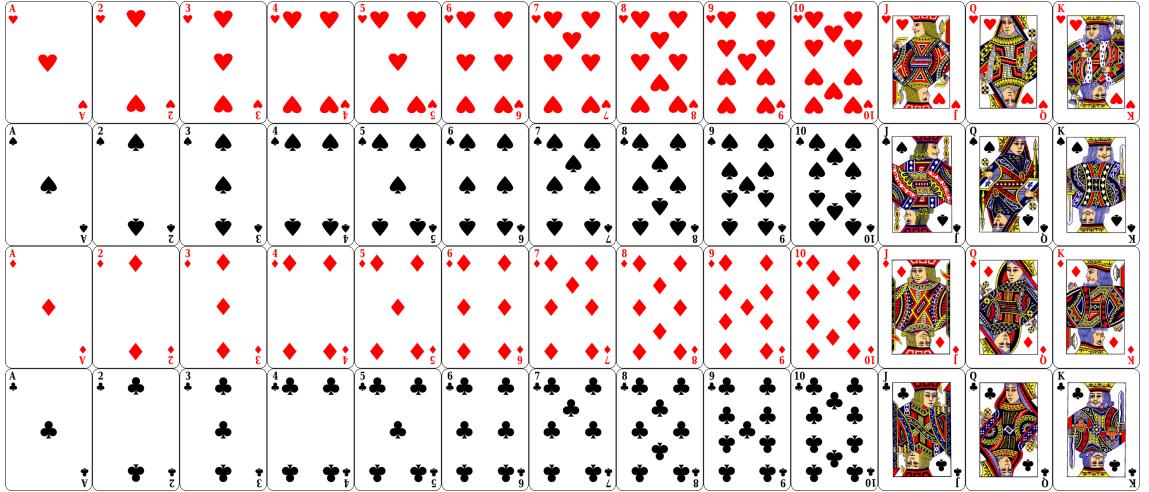
2.1 Texas hold'em poker

Poker se hraje v mnoha, často velice odlišných, variantách. Zdaleka nejrozšířenější z těchto variant je právě Texas hold'em. 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. Není zde popsán systém vsázení, počet žetonů či herní strategie. Z hlediska výzkumu pokeru jako hry jsou stěžejní tyto vlastnosti [5]:

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 2 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 ♠. Barva karty neovlivňuje její sílu. Sílu karty ovlivňuje pouze 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. Dopravostřed stolu se vyloží 5 náhodných karet otočených pozadím vzhůru, aby hráči na začátku hry nevěděli, jaké karty na stole jsou. Tyto karty hráči sdílí. Karty na stole se odkrývají v průběhu hry, který je následující:

1. **Preflop:** všechny karty jsou zakryté, otočeny pozadím vzhůru
2. **Flop:** první tři karty se otočí pozadím dolů a odkryjí se
3. **Turn:** čtvrtá karta se odkryje
4. **River:** poslední pátá karta se odkryje

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 banku (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. Vítězem celé hry 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 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

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 5 karet na stole. Níže je vypočten počet kombinací pěti karet, které lze ze sedmi celkových karet vytvořit:

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

Čím nižší je pravděpodobnost, že daná kombinace nastane, tím je kombinace hodnotnější a silnější. V tabulce 2.1.3 níže uvádí 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
Čtverice	7 ♣, 7 ♥, 7 ♦, 7 ♠, 3 ♣	0,024
Trojice a dvojice	K ♣, K ♥, K ♦, 8 ♠, 8 ♣	0,144
Barva	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 - například postupka či barva. 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 uvádí 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 ♠	Vysoká karta
J ♥, 6 ♣	2 ♠, 2 ♦, A ♦, 3 ♠, 9 ♣	J ♥, 6 ♣, 2 ♠, A ♦, 3 ♠, 9 ♣	Pár

Tabulka 2.2: Příklady tvorby karetních kombinací

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 fungovat i jako hodnota 1, tedy lze z něj složit i takovouto postupku: $A \heartsuit, 2 \diamondsuit, 3 \clubsuit, 4 \spadesuit, 5 \spadesuit$.

2.1.4 Počítání pravděpodobností na výhru

Přesné vyjádření pravděpodobností na výhru zatím neexistuje ani pro hru dvou hráčů (tzv. Heads-up). Kvůli obrovskému stavovému prostoru této hry se používají abstrakce a obecné poučky typu: "pravděpodobnost, že vyhraji, pokud na začátku hry dostanu pár a protivník má pár s nižší hodnotou, je přibližně 80%" [14]. Pokeroví hráči nejčastěji 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 [35]. (todo ins outs)

V tabulce 2.1.4 je znázorněn počet možných kombinací pro hru dvou hráčů. Lze vidět, že i po použití bucketingu je stavový prostor v pozdějších stavech hry velice rozsáhlý. 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: Počet kombinací karet pro hru dvou hráčů v Texas hold'em, převzato z [20]

Z výše zmíněných poznatků je zřejmé, že se pravděpodobnosti na výhru musí odhadovat. Pro odhad pravděpodobnosti v pokeru se používá široké spektrum přístupů [26]. 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 [6].
- **Neuronové sítě a evoluční algoritmy:** Použití umělé inteligence pro zjištění dalšího průběhu hry s využitím samoučících se algoritmů [33].
- **Monte-Carlo simulace:** Simulace náhodných her, které mohou pro současný stav v budoucnu nastat. Časová náročnost je oproti ostatním přístupům vyšší. Monte-Carlo simulace používá drtivá většina online poker kalkulátorů [25].

- **Bayesovské sítě:** Vytvoření pravděpodobnostního modelu, který využívá grafou reprezentaci pro zobrazení pravděpodobnostních vztahů mezi jednotlivými jevy [27].
- **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) [9]. Optimální strategie se hledá pomocí algoritmu, který hraje sám se sebou (algoritmus je často označován jako CFR či Counterfactual regret minimization) [46]. Metodu CFR+ používá například v úvodu zmíněný superpočítač Libratus [7]. 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" [42].

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 používám v praktické části za účelem vyjmutí hrací karty ze snímku.

2.2.1 Barevné modely obrazu

Digitální obraz je tvořen pixely. Pixely jsou typicky organizovány ve dvourozměrném pravoúhlém poli. Šířka a výška obrazu jsou určují velikost tohoto 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 odzadola nahoru.

Každý pixel v obraze má určitou intenzitu. Intenzita rozhoduje o barvě pixelu. 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). 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 [21] následujícím předpisem:

$$Y = 0,21R + 0,72G + 0,07B \quad (2.2)$$

kde Y je výsledný šedotónový pixel a R, G, B jsou intenzity barevných kanálů pixelu. Binární obraz lze z šedotónového obrazu vytvořit tzv. prahováním, které je zmíněno v další sekci věnované segmentaci obrazu.



Obrázek 2.2: 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. 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* [40].

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 [23].

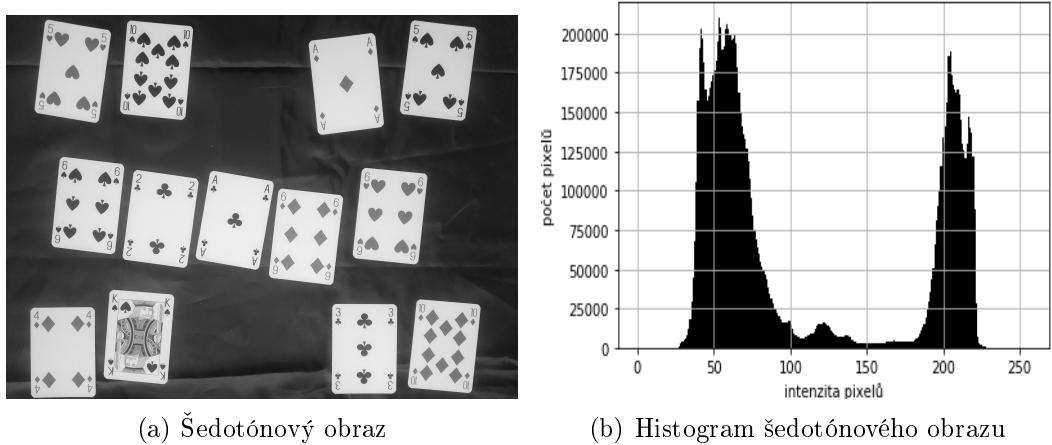
Segmentace obrazu může být obecně kategorizována dvěma přístupy:

- **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.
- **Detekce podobnosti** (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 narušením oblastí.

Segmentace prahováním

Prahování je nejjednodušší metodou segmentace obrazu. Princip prahování spočívá v analýze intenzit pixelů. Intenzity pixelů v obrazu se porovnávají s konstantní 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 [23]. 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 obrazu mají tmavé pixely. Výsledkem segmentace takového obrazu bude snímek s inverzními hodnotami k snímkům s tmavým pozadím a světlými objekty, tzn. pro úspěšnou segmentaci závisí pouze na tom, aby byly intenzity pixelů objektů 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 obrazu vyskytuje. Histogram, kde se pravděpodobně nachází třída pozadí a popředí, se nazývá bimodální a je tvořen dvěma shluky intenzit, podobně jako na obrázku 2.3(b). Levý shluk představuje tmavé pixely, v tomto případě obsahuje tmavé pixely pozadí a znaky na kartách. Pravý shluk jsou světlé pixely karet.



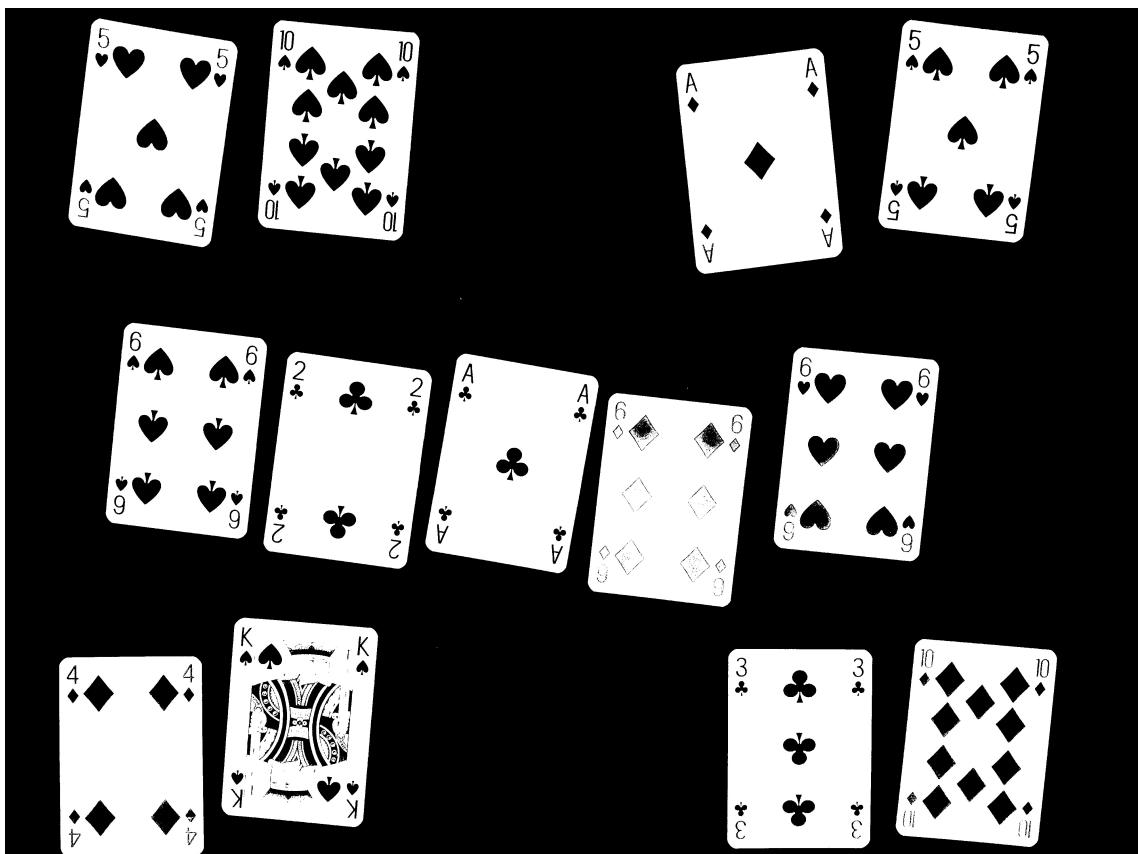
Obrázek 2.3: Šedotónový obraz a jeho histogram

Prahování lze rozdělit podle způsobu nastavování prahu:

1. **Globální prahování:** Nastavení konstantního prahu pro celý snímek. Jako nejlepší se pro obrazy s bimodálním histogramem jeví metoda *Otsu* (také známa jako metoda optimálního prahování) [28]. Algoritmus nalezne automaticky práh, který z histogramu minimalizuje vážený rozptyl dvou tříd jasů. Na obrázku 2.3(b) by nastavila metoda *Otsu* 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, pokud se kvůli světlým podmírkám mění intenzity pixelů v obrazu [45].

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 [11]. Takový histogram obsahuje 3 a více shluků, které obsahují vysoký počet pixelů.

Hlavní výhodou metody prahování je její jednoduchost a rychlosť. Nevhodou 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 2.4: 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 [1]. Pro hranovou detekci je důležitá definice gradientního operátoru.

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

$$\nabla f = \left[\begin{array}{c} G_x \\ G_y \end{array} \right] = \left[\begin{array}{c} \delta f / \delta x \\ \delta f / \delta y \end{array} \right]. \quad (2.3)$$

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

$$|\nabla f| = \sqrt{G_x^2 + G_y^2}, \quad (2.4)$$

nebo často její approximaci:

$$|\nabla f| = |G_x| + |G_y|. \quad (2.5)$$

Pro nalezení hran v digitálním obrazu se používají tzv. masky, pomocí kterých se derivace approximují diferencemi (je to dáno tím, že obraz v počítači je reprezentován diskrétně a ne spojité). 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.

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

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

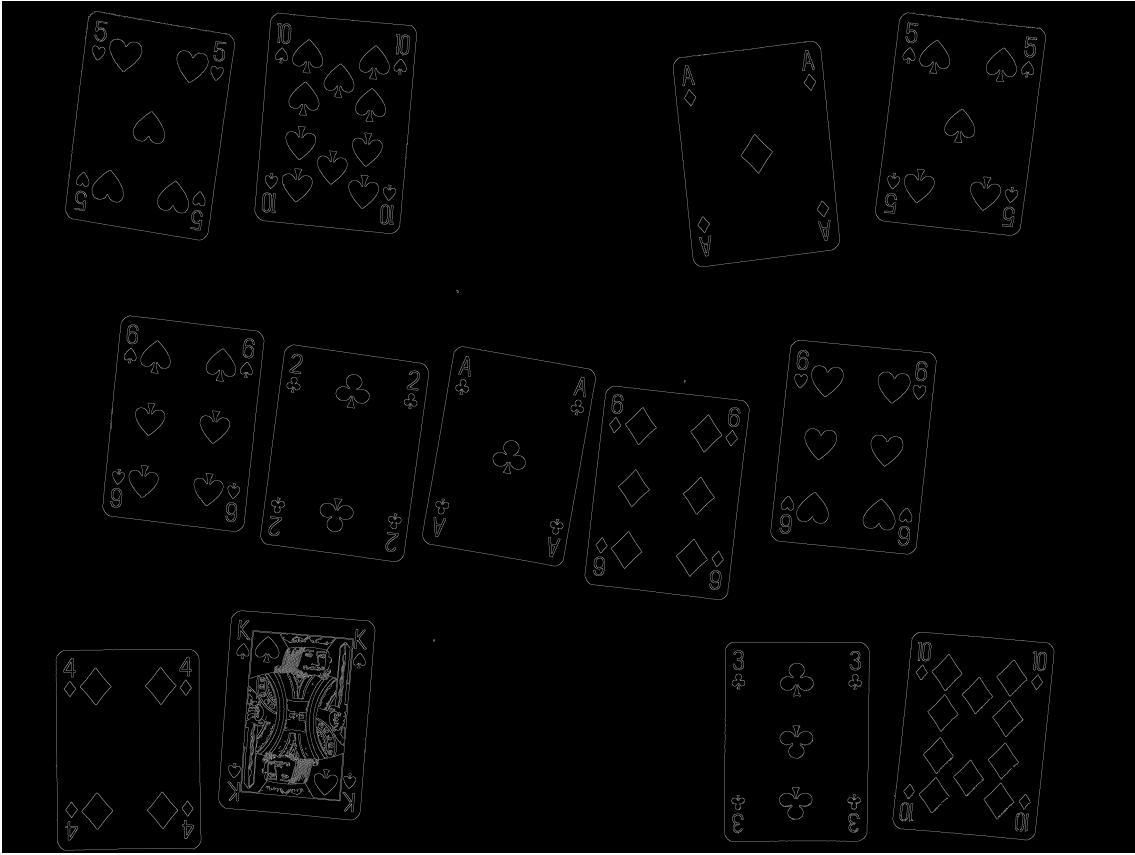
$$R = -x_1 + x_3 - 2x_4 + 2x_6 - x_7 + x_9,$$

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 obrazu. 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 [1].

Pro detekci hran v zašuměném obrazu se používá Canny hranový detektor [1], 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 konvoluje 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 [1].



Obrázek 2.5: Canny hranová detekce

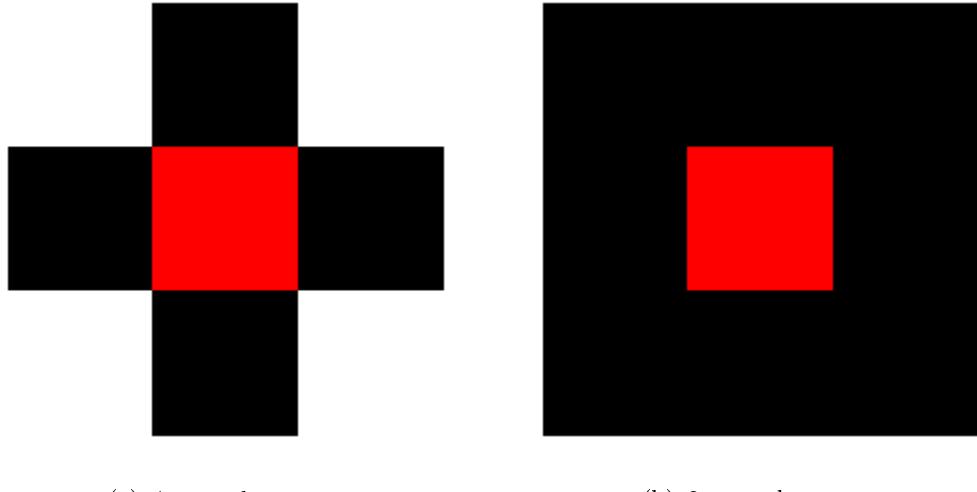
Hranové detektory jsou vhodné pro obrazy, na kterých je mezi objekty vysoký kontrast. Na rozdíl od metody prahování není pro detekci hran důležitý tvar histogramu. Nevýhodou hranových detektorů je vyšší citlivost na šum, což lze vidět i z obrázku 2.5.

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 oblasti. Pro každý pixel v této oblasti se poté znova kontrolují vlastnosti sousedních pixelů. Výsledkem narůstání oblasti je homogenní oblast s maximálním možným obsahem [3]. Určení, jak spolu pixely sousedí se nazývá sousednost, která v tomto algoritmu určuje, jakým způsobem budou regiony růst. Nejčastěji se používá

tzv. 8-sousednost, tzn. zkoumaný pixel (na obrázcích 2.6 červeně) sousedí s 8 pixely.



Obrázek 2.6: Sousednosti pixelů

Červená značka na obrázku 2.7(b) označuje místo počátečního pixelu, ze kterého oblast narůstá. 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.



Obrázek 2.7: 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 [32] [31] [29]. Pro obrazy s vyšším rozlišením může být tento způsob segmentace zdlouhavý a paměťově náročný [1].

Segmentace pomocí neuronových sítí

Speciálním odvětvím segmentace obrazů je segmentace pomocí konvolučních neuronových sítí. Narozený od ostatních zmíněných přístupů nachází síť algoritmus řešení sama od sebe tím, že je trénována. Umělá inteligence dokáže lépe generalizovat řešený problém než pevně naprogramované algoritmy a tudíž je segmentace těmito metodami vhodná téměř pro každý složitější úkol [4].

Jedná se o vcelku nový přístup, který začíná velice rychle nahrazovat ostatní metody v širokém spektru oborů [2]. Téma neuronových sítí detailněji popisují v nadcházející kapitole.

2.3 Konvoluční neuronové sítě

Bude doplněno - všechn obsah v této kapitole je provizorní, zatím to nemá cenu číst

Neuronové sítě

Podle [34] se optimální batch size pohybuje kolem 200 a výše. S vyšším batch size se zvyšuje i přesnost sítě. Na druhou stranu rostou i nároky na výkon.

$$f(x) = \max(0, x) \quad (2.6)$$

2.3.1 Vrstvy konvolučních neuronových sítí

Doporučuje se volit hodnoty v rozmezí 0,5 - 0,8 [41]. Dropout vrstva je funkční pouze po dobu trénování.

2.3.2 Architektury konvolučních neuronových sítí

Pokud ztrátová funkce skáče i po delší době, kdy už se trénovací přesnost výrazně nemění, může to být známkou přetrvávání sítě. (categorical crossentropy) Tato ztrátová funkce porovnává skutečné a predikované pravděpodobnostní funkce pro jeden obraz. Pravděpodobnostní funkce jsou v tomto případě diskrétní. Skutečná pravděpodobnostní funkce je dána jako 1 pro třídu, do které obraz náleží a 0 v

ostatních třídách. To platí i pro predikovanou pravděpodobnostní funkci, která však může mít 1 na nesprávném místě.

Ztrátová funkce se dá vyjádřit matematicky takto:

$$H(p, q) = - \sum_{\forall x} p(x) \cdot \log(q(x)) \quad (2.7)$$

kde $p(x)$ je skutečná pravděpodobnostní funkce, $q(x)$ je predikovaná pravděpodobnostní funkce a x reprezentuje jeden konkrétní obraz.

2.3.3 Praktické úvahy

Rozdělení datasetu

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 si síť dokáže poradit s neznámými vzory. Podíl trénovacích, validačních a testovacích dat závisí na architektuře sítě. Dobrým odrazovým můstkom se zdá být poměr 70 % trénovacích dat ku 20 % validačních dat ku 10 % testovacích dat. Pro složitější sítě s více neuronů stoupá riziko předtrénování a tudíž je vhodné použít vyšší podíl validačních dat [13].

Velikost trénovacího datasetu

Velikost trénovacího datasetu ovlivňuje přesnost a robustnost sítě. Pro velikost datasetu neexistuje univerzální předpis. Základním doporučením podle [44] je použít pro každou třídu 1000 obrazů. Toto číslo závisí na složitosti řešeného problému. Naučit síť rozlišovat černou a bílou barvu půjde pomocí méně trénovacích vzorů než pokud by se síť měla naučit rozlišovat mezi kočkou a psem. Záleží na tom, jak dobře jsou od sebe třídy odlišitelné. S přibývajícím počtem tříd bude potřeba více vzorů.

Obecně se doporučuje použít stejný počet obrazů pro každou třídu [22]. Síť, která byla natrénovaná s nerovnoměrným počtem vzorů v jednotlivých třídách může při klasifikaci neznámých obrazů upřednostňovat třídy obsahující vysoký počet trénovacích vzorů. V mé případě se toto potvrdilo při zpracování mé bakalářské práce [18], kde jsem počet vzorů v třídách nevyrovnával a síť dosáhla přesnosti na trénovacích datech pouze 93,5 %.

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) [16]. 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 uvádíme 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 2.4: 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) [17].

Normalizace

Normalizace vstupu pozitivně ovlivňuje stabilitu a rychlosť konvergencie sítě [38]. Tato procedura není nutná a síť dokáže fungovat i bez ní. Použitím rovnice (2.8) níže se hodnoty intenzit pixelů v každém barevném kanálu upraví z rozsahu 0 – 255 na rozsah 0 – 1:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (2.8)$$

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 snáz inicializovat váhy sítě.

Optimalizační algoritmy

[36]

Klasifikátory

Analýza průběhů

Pokud síť dosahuje vysokých přesností pouze na trénovacích datech, došlo k jejímu přetrénování. Přetrénování nastane, pokud se síť trénuje příliš dlouho a nastaví tak své parametry "až příliš přesně" [15]. Opačný případ vyšší přesnosti na validačních datech může nastat, pokud při trénování sítě byly použity regularizační techniky jako dropout či cross validace. Tento jev pouze ukazuje, že je síť robustní. Jedna epocha znamená, že se síti předloží všechny trénovací vzory z datasetu. Síť by se měla nechat trénovat do té doby, dokud se validační ztrátová funkce snižuje. Proto počet epoch není přesně dán. Doba trénování sítě závisí na velikosti trénovacího datasetu a počtu spojení v síti. Tato závislost je přibližně lineární [8].

3 Praktická část a vyhodnocení experimentů

V této kapitole se zaměřím na aplikaci teoretických znalostí v praxi.

Nejprve krátce popíšu algoritmus vyjmutí karet z obrazu. Algoritmus pro získání karet je převzat z mé bakalářské práce [18]. Nepoužívám zde však některé kroky, které se ukázaly jako zbytečné či dokonce kontraproduktivní.

Dále popisují tvorbu syntetických karet a syntetického hracího stolu. Syntetické karty poté 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 své bakalářské práci jsem nekladl příliš velký důraz na přípravu trénovacího datasetu. To se projevilo na přesnosti na-trénované sítě. 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 fully connected vrstva. Nejpřesnější z těchto neuronových sítí bude sloužit ke klasifikaci reálných karet.

Pátá sekce se věnuje analýze stavu hry. Popíšu zde algoritmus přiřazení detekované 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*. V *PyQt5* lze GUI vytvořit i bez programátorských schopností díky aplikaci *QtDesigner*, ve které lze grafické komponenty přetáhnout z lišty nástrojů. Kód se poté vygeneruje automaticky.

Nakonec aplikaci otestuji na vygenerovaných a reálných snímcích pokerových stolů. Bude vyhodnocena přesnost natrénuvané neuronové sítě na testovacích datech a porovnán mnou spočtené pravděpodobnosti na výhru s online kalkulátory. Tato aplikace je spolu s návodem na instalaci dostupná v *github* repozitáři [19].

3.1 Získání pokerových karet z obrázku

Karty na snímcích budou později v této práci klasifikovány neuronovou sítí a analyzovány, proto je nutné, aby algoritmus na vyjmutí karet ze snímku byl co nejspolehlivější a nejpřesnější. Modifikace tohoto algoritmu je použita také při tvorbě trénovacího datasetu pro neuronovou síť. Základní myšlenka pro získání karet ze snímku zůstala stejná jako v mé bakalářské práci [18], 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

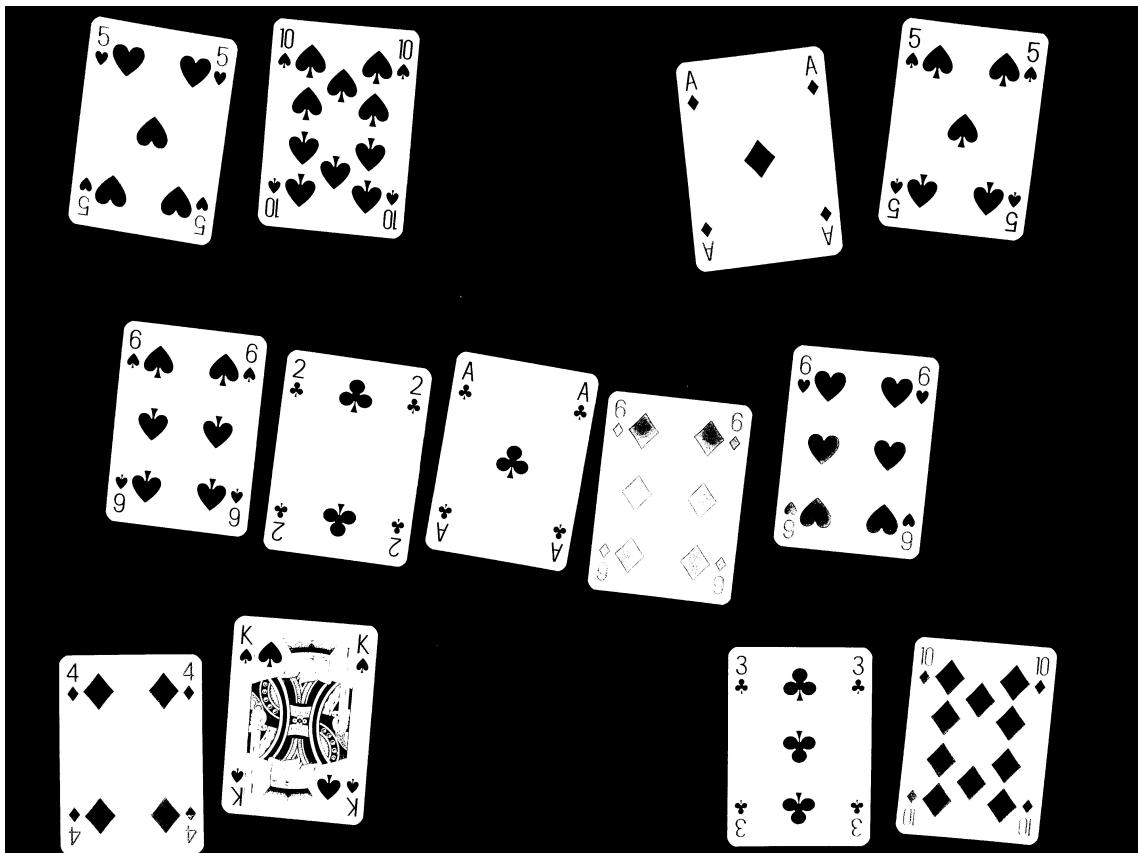
Jako předlohu, na které budu výše zmíněné metody demonstrovat, jsem zvolil snímek reálného stolu - obrázek 3.1, kde jsou u stolu přítomni 4 hráči hra je ve stavu River.

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 bakalářské práci [18] jsem používal metodu adaptivní segmentace pro vyrovnání jasu ve snímku. To je 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 podmínu splňují. Obrázek 3.1 je nutné před prahováním převést na šedotónový barevný model.

Všechny snímky stolu, kde je pro karty použito tmavé pozadí, budou mít bimodální histogam. Důkazem může být i obrázek z teoretické části 2.3(b). Pro tyto případy je vhodná metoda optimálního nastavení prahu *Otsu*. Zde je metoda *Otsu* použita z knihovny *skimage* [43].



Obrázek 3.2: Segmentovaný snímek metodou otsu

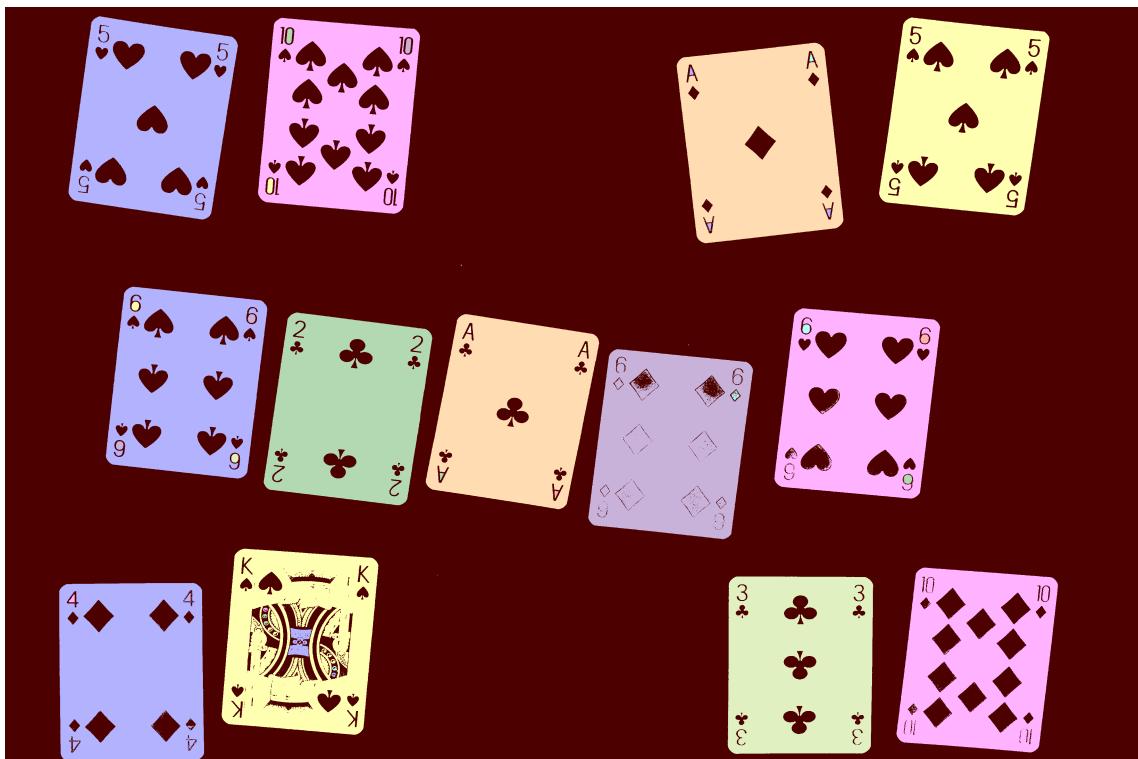
Jak je vidět na obrázku 3.2, metoda *otsu* zvolila práh tak, že většina symbolů na kartách je prohlášena za pozadí. V tomto případě to není důležité. Podstatné je, že se oddělily okraje karet od pozadí a bude tak možné dobře analyzovat vlastnosti segmentovaných regionů.

3.1.2 Detekce regionů s potenciální kartou

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

- **Obsah regionů:** Regiony s nízkým obsahem jsou pravděpodobně objekty, které na snímku nemají co dělat (šum, stíny, atp.)
- **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
- **Souřadnice středu regionů:** Toto bude důležité pro analýzu stavu hry, kdy bude nutné přiřadit karty hráčům (například: karty blízko sebe budou pravděpodobně patřit stejnému hráči)

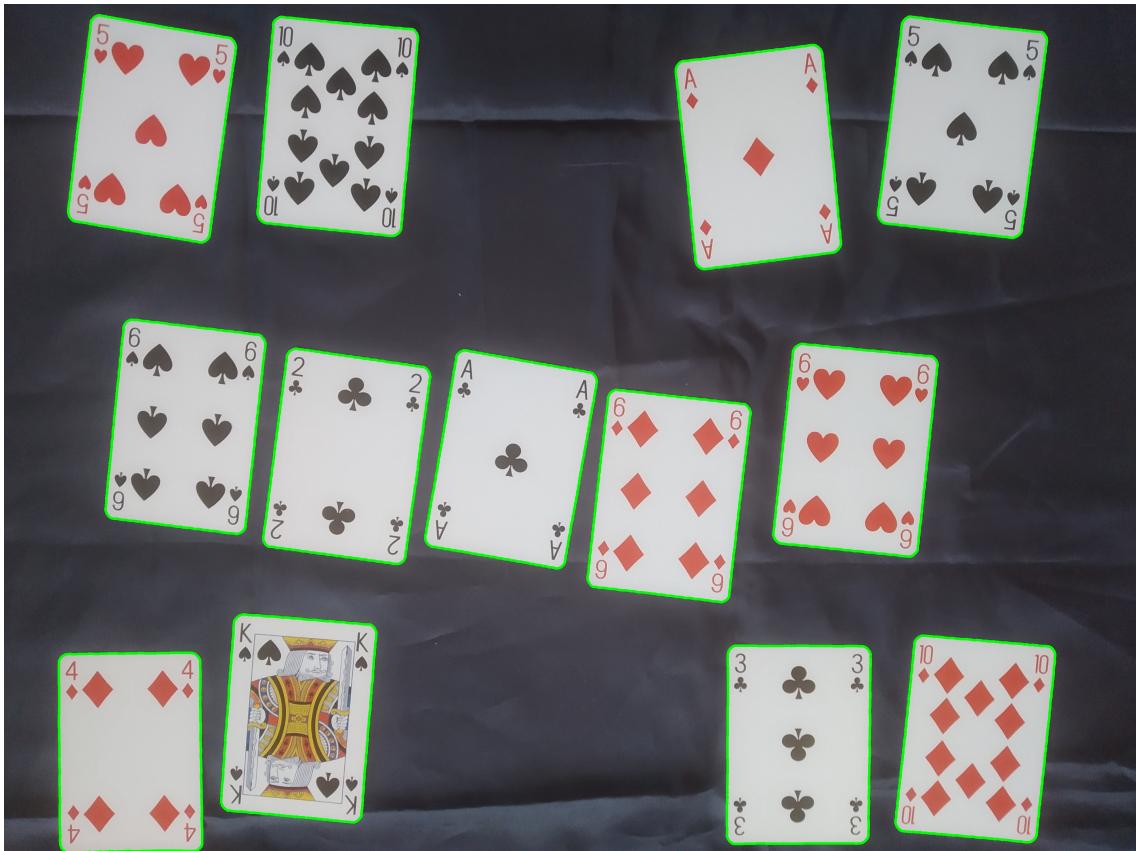
Díky těmto informacím je možné rozhodnout o tom, zda segmentovaný region je opravdu karta. Nejprve jsem na segmentovaný obrázek aplikoval morfologickou operaci zavření pro rozbití hran. To je důležité pro prevenci situace, kdy souvislý region uvnitř karty je kvůli svému velkému obsahu prohlášen za kartu. Pro měření vlastností regionů jsem použil metodu *skimage.measure*. Další použitá funkce *skimage.measure.regionprops* slouží pro spočtení obsahů regionů.



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

3.1.3 Vykreslení kontur kolem karet

V této chvíli dokáže algoritmus ze 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 approximovány na přímky. Poté se kontury vykreslí funkcí `cv2.drawContours` s parametrem `cv2.RETR_EXTERNAL`, který zaručí vykreslení kontur pouze po obvodu objektů.

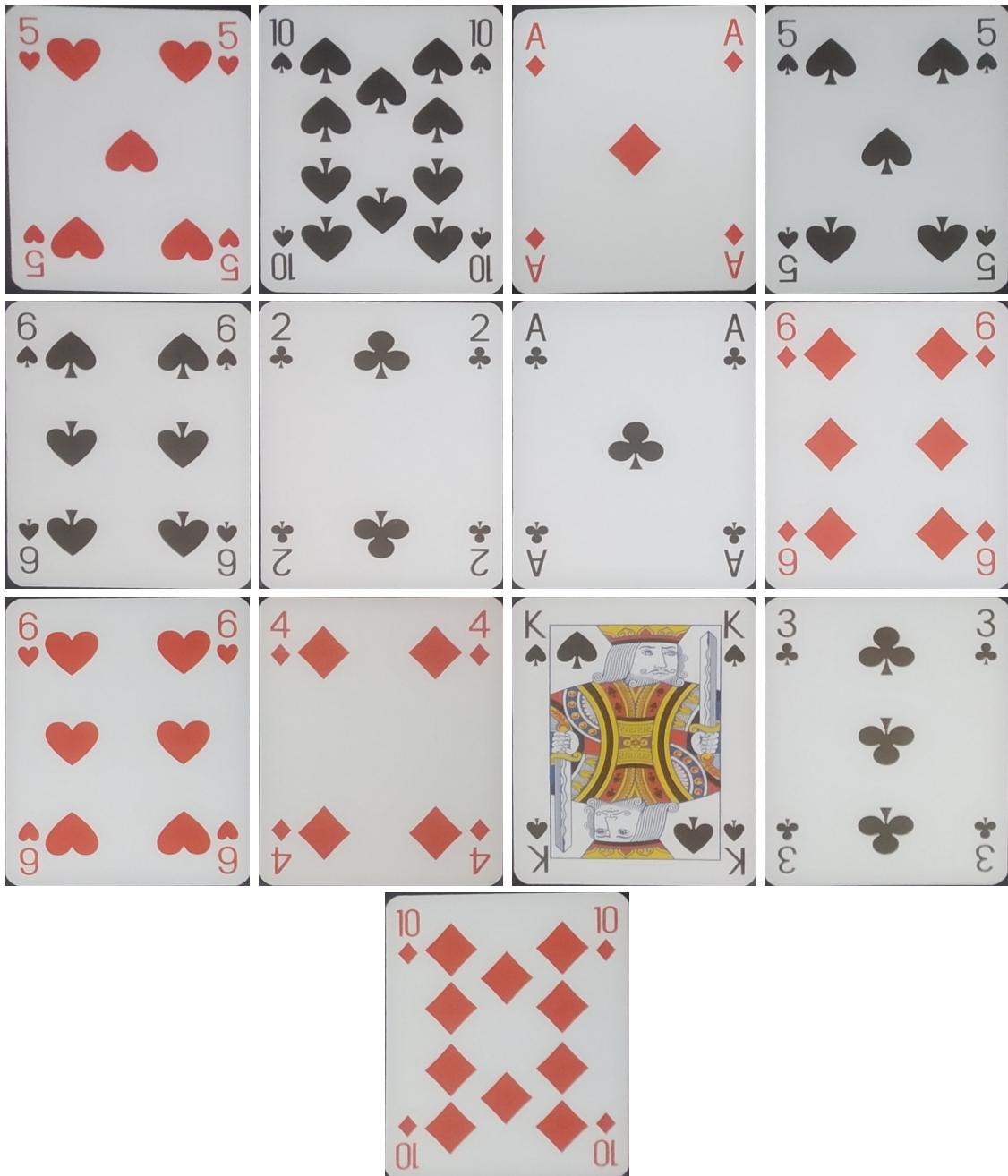


Obrázek 3.4: Kontury kolem karet

Samotná approximace kontur kolem karet však nedokáže zajistit, aby přímky kontur tvořily opravdu pravoúhlý obdélník. Proto na approximované kontury postupně aplikují metody `cv2.minAreaRect` pro opětovnou approximaci kontur nejmenším obdélníkem a `cv2.boxPoints` ke zjištění souřadnic rohů tohoto obdélníka. Od této chvíle mohu počítat s tím, že všechny karty jsou obdélníkové, což bude velice důležité při počítání úhlu rotace a vyjmutí karty. Může se naskyttnout 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 zjistit přesné souřadnice rohu karty. Přesnost souřadnic rohů karet je rovněž důležitým 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 také vodorovná. Posledním krokem je vyjmutí karty pomocí metody *np.argwhere*.



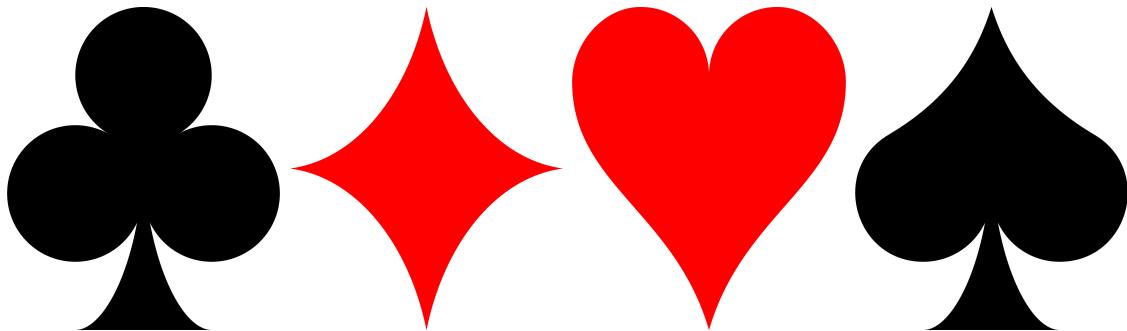
Obrázek 3.5: Výsledné karty po rotaci a vyjmutí ze snímku, velikosti karet byly pro vylepšení vizuální stránky práce normalizovány

3.2 Generování syntetických hracích karet a stolů

Výhodou počítačem generovaných dat je možnost automatického vytvoření velkého množství obrázků za krátkou dobu. Synteticky vytvořené karty použiji jako trénovací data pro neuronovou síť. 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 písmeno, symbol či kartu.

3.2.1 Generování pokerových karet

Na rozdíl od sběru reálných dat budu generovat 3 sety karet, tzn. v této práci budou použity karty ze třech různých hracích balíků. Jeden set karet je totožný s tím reálným a zbylé dva jsou stažené ze stránky *acbl.org*. Karty generují 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

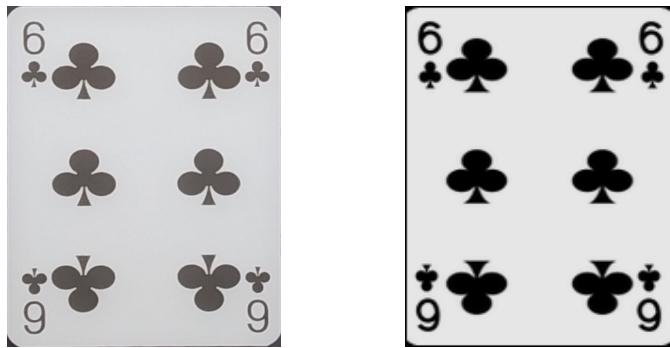
Vstupními argumenty skriptu na generování karet jsou následující parametry:

1. **Ze kterého 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. **Výška karty:** Pro zjednodušení je nejprve vytvořena karta s konstantní výškou a šírkou. Až poté, co je karta hotová se změní její velikost a tudíž se nemusí podle vstupních parametrů měnit velikost písma a symbolů.

6. Šířka karty

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

Výsledkem je počítacem 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 potřebná.



(a) Karta vyjmutá z vyceného snímku

(b) Generovaná syntetická karta

Obrázek 3.7: Porovnání reálné a generované karty

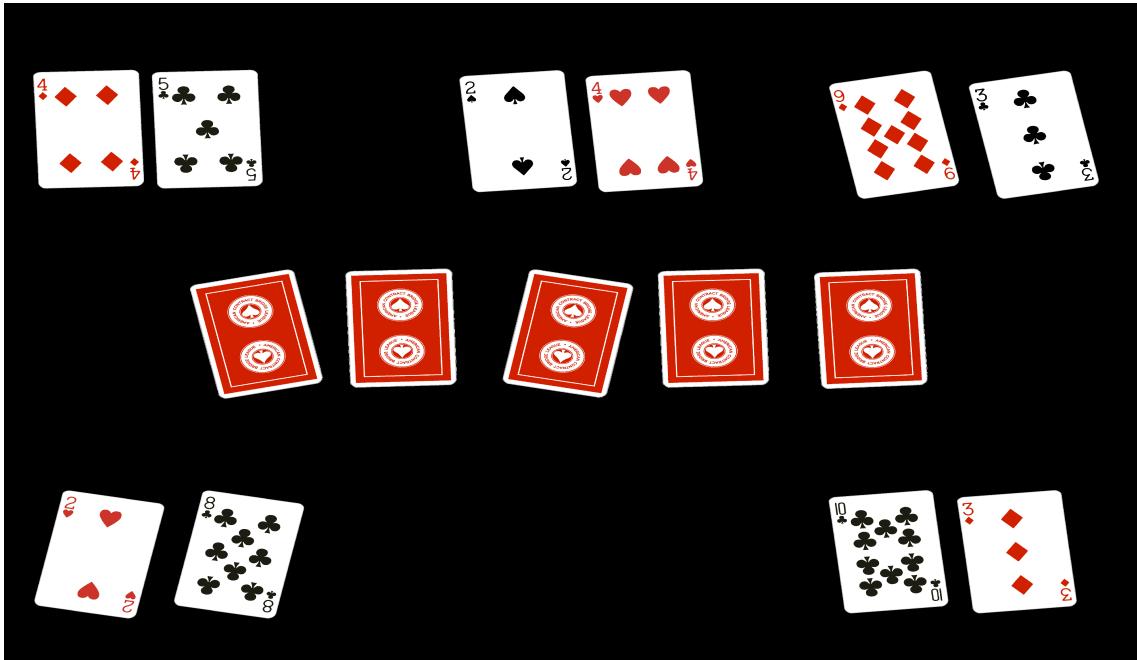
3.2.2 Generování pokerového stolu

Obrázky počítacem 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:

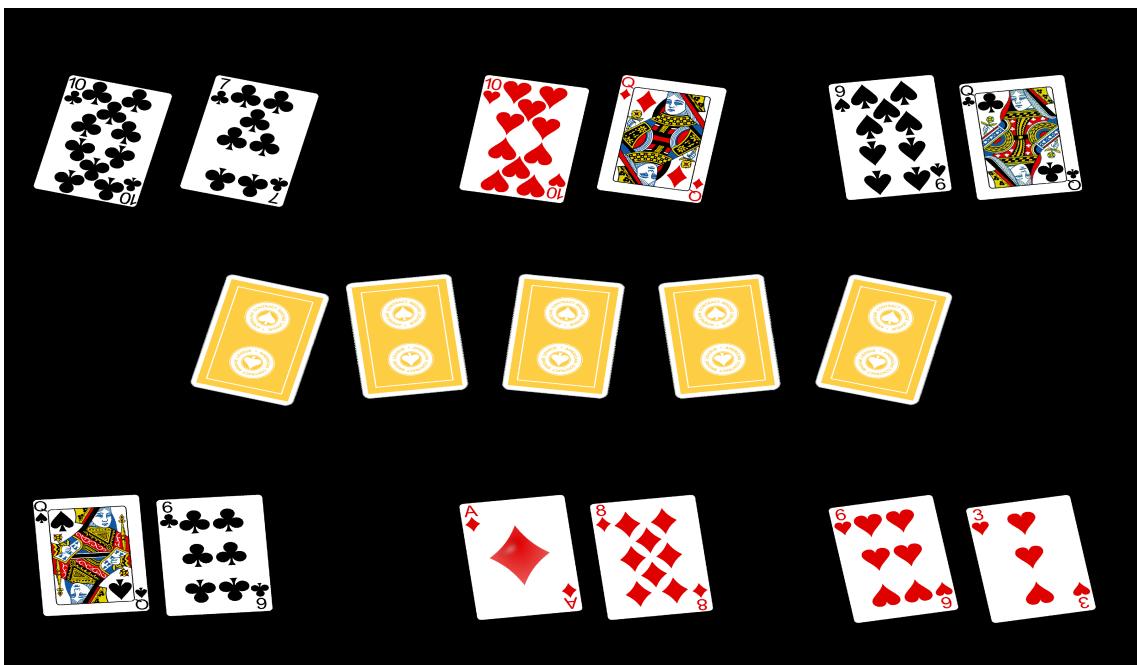
1. **Vygenerovat požadované karty:** Pro každý stůl je náhodně vybrán jeden set karet, ze kterého se karty vytvoří. Hodnoty a barvy karet jsou také náhodně vybrány. Jediné, na co si je třeba dát pozor je to, aby se karta se stejnou hodnotou a barvou nevyskytovala na stole dvakrát. Navíc je nutné vygenerovat ještě 5 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 2 náhodné karty. Rotace je pro obě tyto karty stejná. V této práci jsem zvolil náhodný počet hráčů od 2 do 6.
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



Obrázek 3.9: Náhodně vygenerovaný obrázek stolu

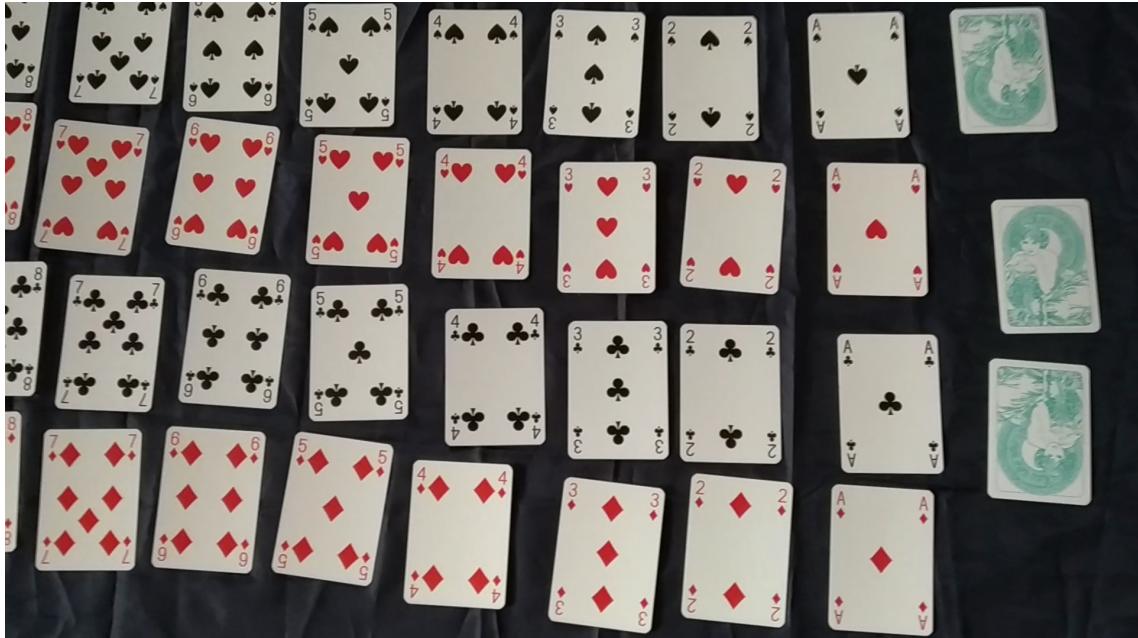
3.3 Příprava dat pro konvoluční neuronovou síť

Použití kvalitních trénovacích dat je základním předpokladem pro natrénování přesné neuronové sítě [24]. V mém případě bude trénovací dataset tvořen z reálných karet a z karet generovaných pomocí programu zmíněného v sekci 3.2.1. Aby trénovací data byla dostatečně rozmanitá, budou navíc augmentována pomocí *tensorflow* modulu *ImageDataGenerator*.

3.3.1 Sběr reálných karet pro trénovací dataset

Pořizování reálných karet je časově náročné. I když jsem se snažil sběr karet co nejvíce automatizovat, trvalo mi přibližně 20 hodin, abych vytvořil rozmanitý trénovací dataset. Pro zachycení všech aspektů reálné karty je trénovací dataset obsahující pouze syntetické karty nedostačující, proto je třeba použít i reálné karty.

Ke zdědění 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 získají karty. Karty jsou rozloženy na segmentovatelné pozadí tak, aby se neprekryvaly. Bylo natočeno několik videí za různých denních dob v různě osvětlených místnostech.



Obrázek 3.10: Pořizování dat - snímek z videa

Karty automaticky zařazují do 53 složek (tzn. všechny druhy karet a pozadí) pomocí neuronové sítě z mé bakalářské práce [18]. Síť má přesnost 93,5% na testovacích datech, tudíž je nutné složky jednu po druhé zkonto rovat, 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.

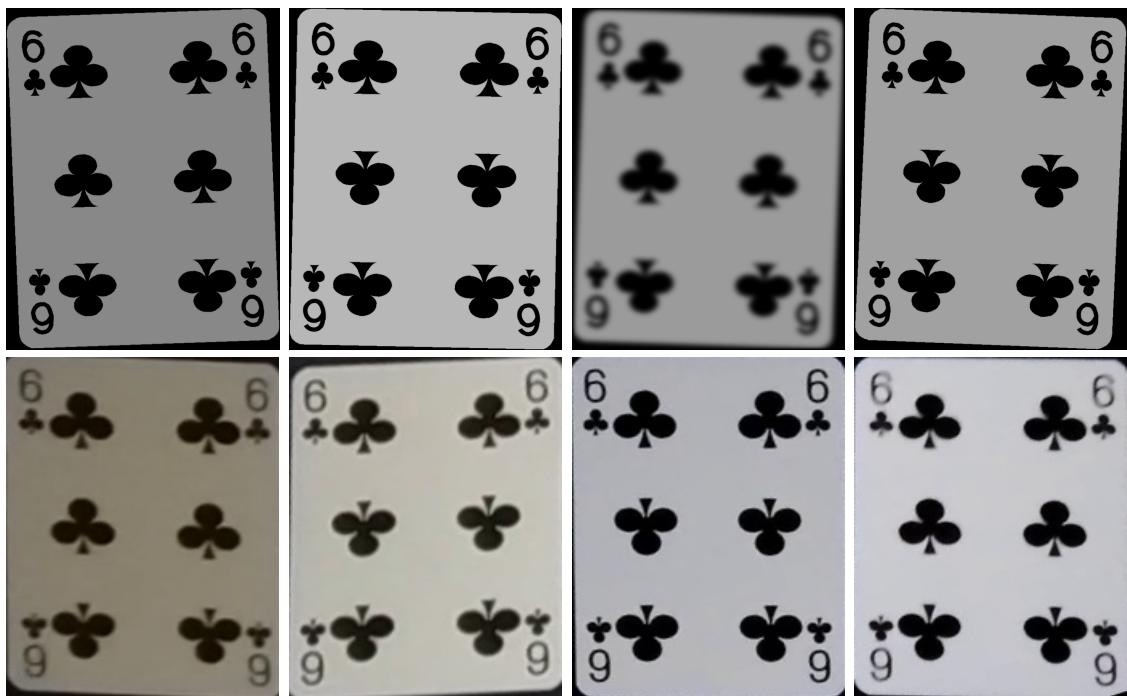


Obrázek 3.11: Špatně klasifikované karty

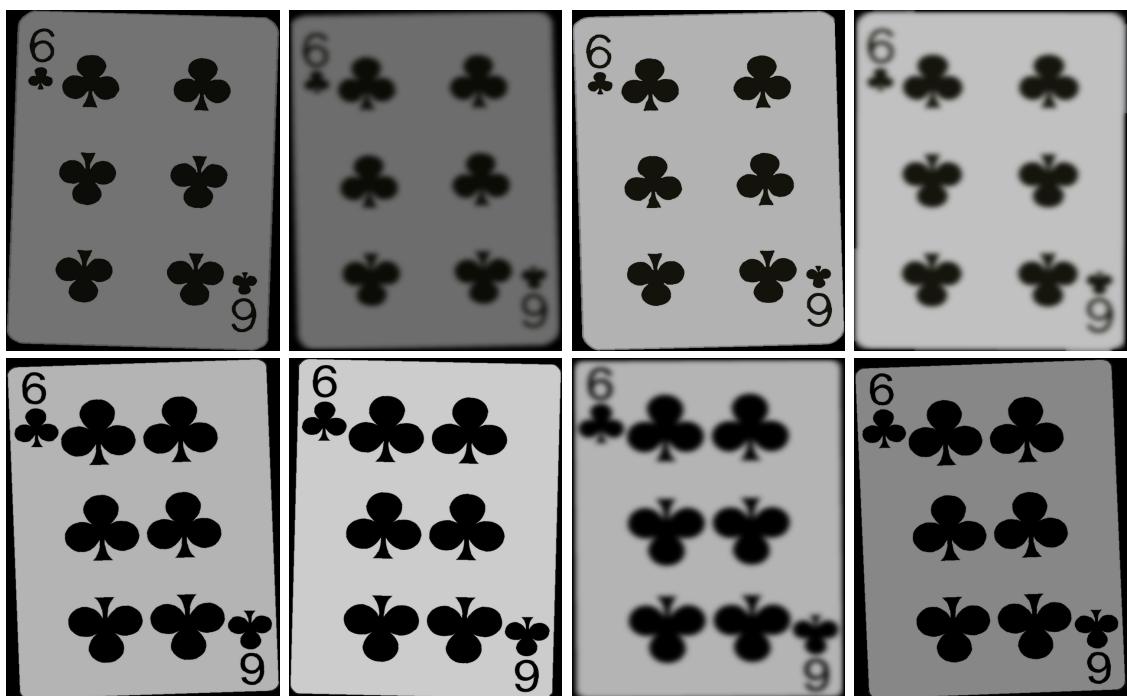
Nejlepším postupem by při řešení problémů ručního přesouvání byla nejspíš střední cesta. Z obrázku 3.11 bych přesunul do správné složky pouze srdcového kluka, který je znehodnocen odrazem světla a zbytek karet bych smazal. Složky totiž obsahují už mnoho rozostřených karet a není tedy nezbytně nutné přidávat další. Na obrázku také lze vidět, že stará 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 bakalářské práci jsem používal navíc i šum [18]. 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 kapitole.

3.3.3 Augmentace trénovacího datasetu

Díky augmentacím lze z původních dat vytvořit nová a rozšířit tak trénovací dataset. Čím více rozličných dat má síť k dispozici, tím přesněji by měla karty klasifikovat. Obrázky je nutné augmentovat tak, aby byly zachovány jejich hlavní rysy. Příliš silné augmentace mohou být pro trénování sítě kontraproduktivní [30]. V předchozí kapitole jsem augmentoval syntetické karty, aby se podobaly reálným kartám. Cílem augmentací v této kapitole 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 se stává zejména u bílé barvy. Bílá barva je kamerami často natočena jako žlutá.
3. **Karta je posunuta či rotována:** To je způsobeno nejčastěji chybou programu na vyjmutí karet.

ImageDataGenerator

Pomocí tohoto modulu jsou augmentovány připravené karty. 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 [39].

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 [37]. V argumentu *preprocessing_function* lze ale specifikovat libovolnou funkci předzpracování a tudíž je možné rozostření a šum přidat.

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



Obrázek 3.14: Karta před augmentacemi

1. Rotace

V argumentu *rotation_range* se specifikuje celým číslem úhel maximální rotace. Obrázek je poté rotován o náhodný úhel z intervalu $< -\text{rotation_range}, +\text{rotation_range} >$. 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

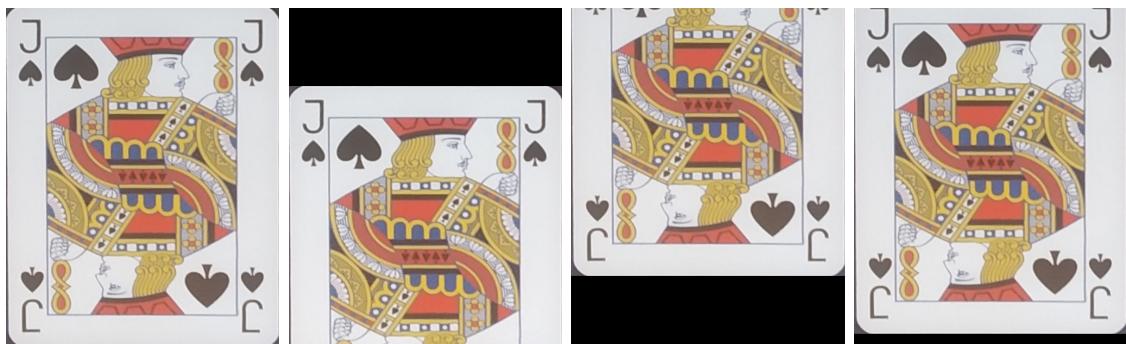
$< -width_shift \cdot \text{šířka obrázku v pixelech}, +width_shift \cdot \text{šířka obrázku v pixelech} >$. Obrázky níže ukazují posun maximálně o 0.3 šířky obrázku:



Obrázek 3.16: Posun o maximálně 0.3 šířky obrázku

3. Vertikální posun

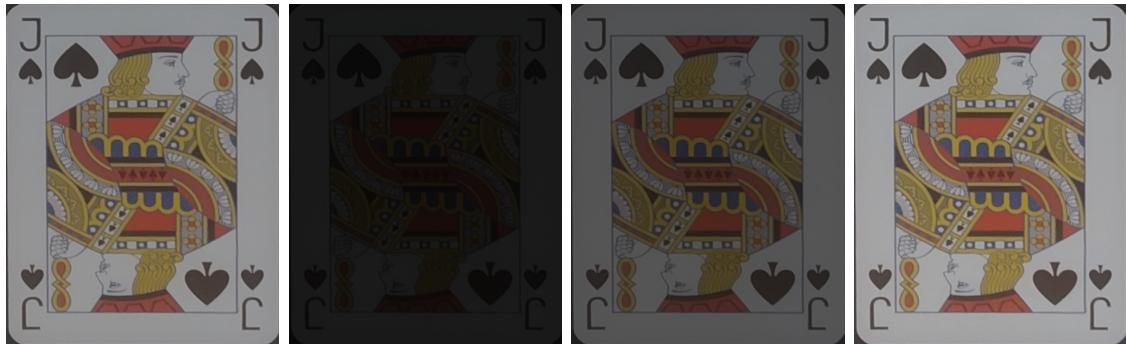
`height_shift_range` se používá stejně jako horizontální posun, obrázek se pochopitelně posune vertikálně.



Obrázek 3.17: Posun o maximálně 0.3 výšky obrázku

4. Jasové transformace

Do parametru *brightness_range* se dává 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 je znova maximálním posunem.



Obrázek 3.19: Posun barevných kanálů maximálně o 150

Pro augmentaci karet jsem 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

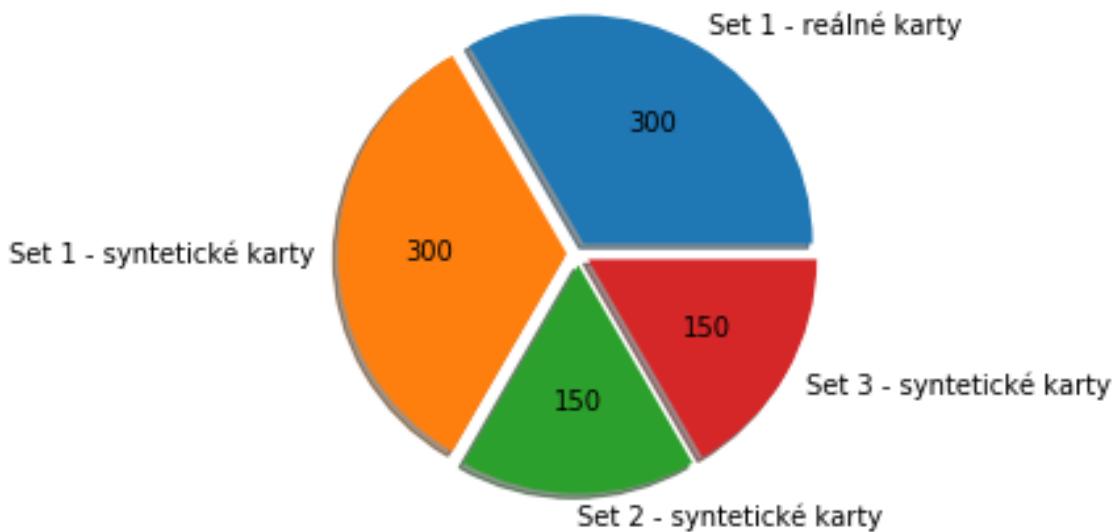
Všechny augmentace jsou na kartu aplikovány naráz s náhodnou silou. Obrázek 3.20 ukazuje, jaké karty vzniknou kombinací těchto úprav:



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

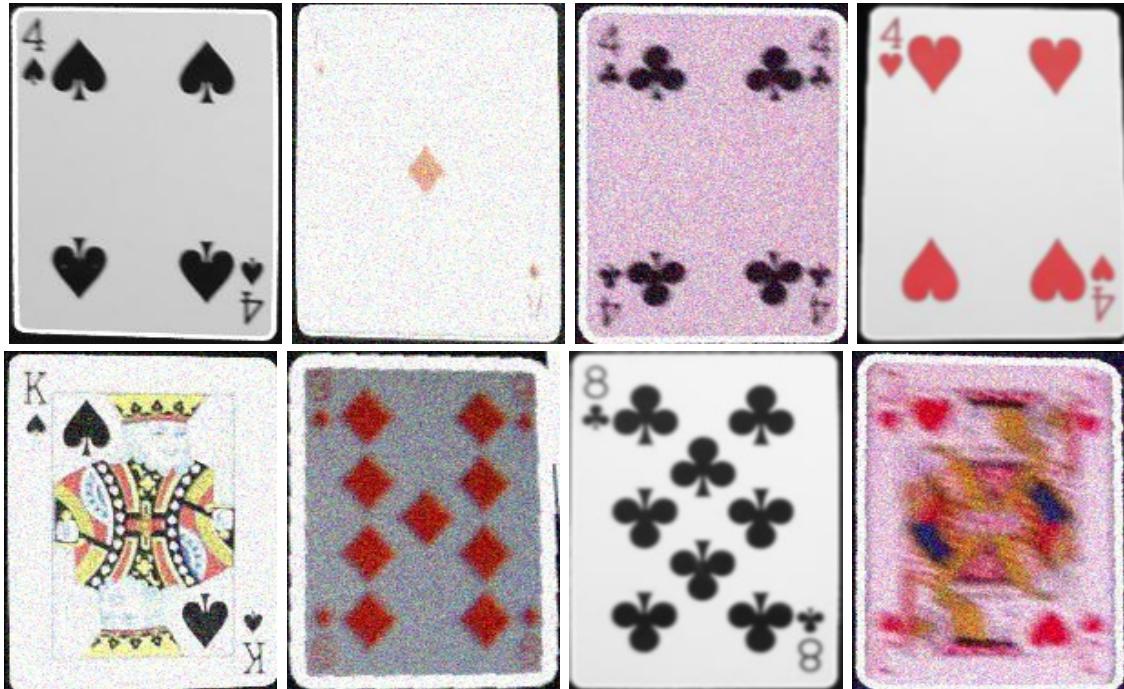
Karty ze setu 1 mám jediné k dispozici fyzicky a proto tento set upřednostňuji vyšším počtem obrázků v trénovacím datasetu. Počítačem vytvořené karty se budou identifikovat snáz než reálné karty ze setu 1 a proto je jich v datasetu méně.

V mém případě je dobré zachovat obdélníkový tvar karty. Třídy v pokerových kartách jsou však 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* [12], 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* datasets 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 - páky, 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 daných

tříd. To se bude později hodit pro testování přesnosti sítě.

Abych dokázal, že síť v této práci je podstatně kvalitnější než ta, kterou jsem vytvořil v bakalářské práci, používám trénovací data z bakalářské práce jako validační data pro stávající síť.



Obrázek 3.22: Validační dataset obsahuje karty rozdílných kvalit

Validační data jsou tvořena jen ze setu reálných karet. Prioritou 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 bakalářské práce. Jak už jsem zmiňoval, jako validační data jsem použil trénovací data pro sít z bakalářské práce.

3.4.1 Inicializace trénovacích a validačních datasetů ImageDataGeneratoru

((mám tady tuhle část o inicializaci nechat? přijde mi, že je to tu navíc)))

Zde je podle mého názoru důležité zmínit, že změna rozlišení obrazu je prováděna první (i před funkcí *preprocessing*). Proto je vhodné všechny augmentace testovat na tak velkém obrázku, jako je definován při inicializaci 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 [10]. Zde jsou však použity takové augmentace, u kterých na rozlišení obrázku nezáleží.

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

Parametr, který může být 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

```

3.4.2 Použité architektury konvolučních neuronových sítí

Celkem trénuji 5 architektur sítí. Trénované sítě se liší svou 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 tomu používám modul *pickle*.

V knihovně *tensorflow* se sítě vytváří jednoduše. Níže je vidět příklad inicializace architektury sítě z obrázku 3.24, který je na další stránce. Prvním řádkem se vytvoří prázdná sekvenční konvoluční neuronová síť. V dalších řádcích přidávám komponenty. Na druhém řádku zároveň zadávám velikost vstupního obrazu. Číslo 3 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'))

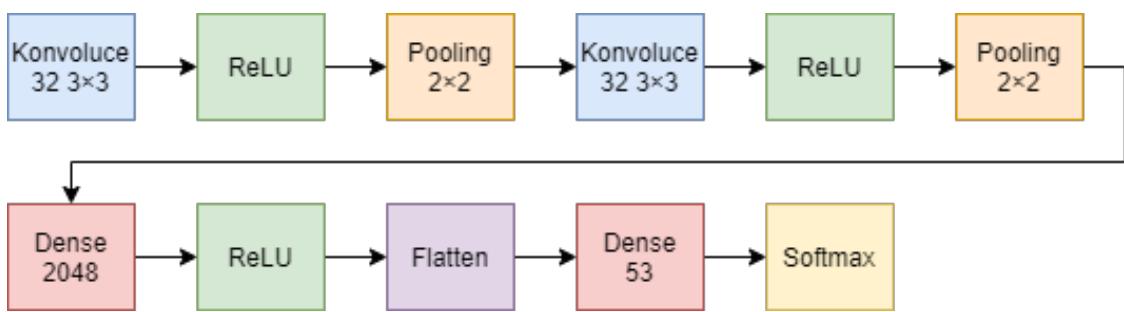
```

Pro snadnější porozumění kódu přikládám vysvětlení pojmu:

1. **Conv2D**: První číslo značí počet filtrů konvoluce. Druhým parametrem je jejich velikost.
2. **Activation('relu')**: Aktivační funkce ReLU mapující záporný příznak na nulu.

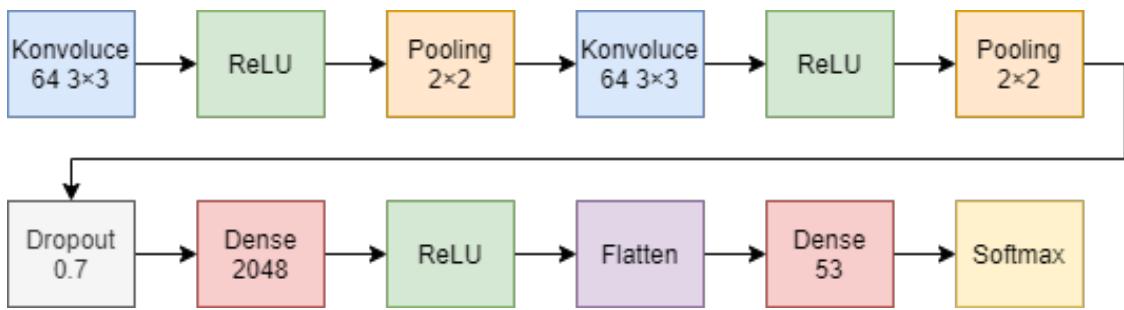
3. **Flatten**: Modul převede pole ve třech dimenzích do jednodimenzioálního pole.
4. **Dropout**: Číslo od nuly do jedné udává náhodný počet odpojených vstupních příznaků.
5. **Dense**: Plně propojená vrstva.
6. **Softmax**: Aktivační funkce normalizuje vstupní hodnoty na vektor o velikosti počtu tříd. Hodnoty vektoru jsou v rozsahu od nuly do jedné a značí pravděpodobnost příslušnosti dané třídě.

Architekturu na obrázku 3.24 níže jsem používal ve své bakalářské práci. Oproti většině ostatních sítí se v ní nachází i skrytá plně propojená vrstva. V síti chybí dropout vrstva, což nemusí být negativem.



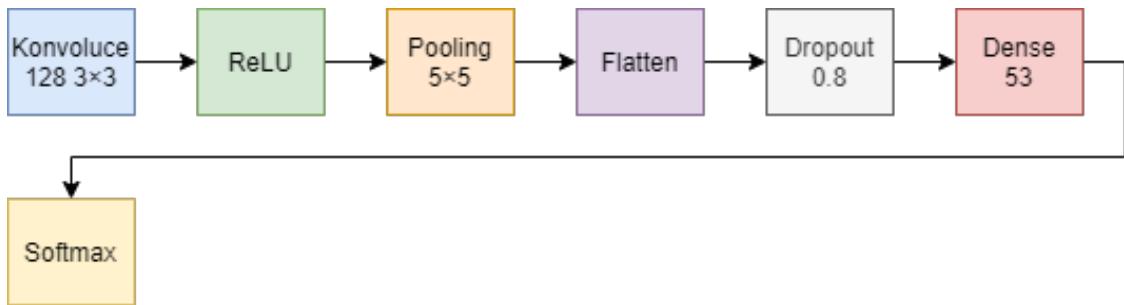
Obrázek 3.24: Síť z bakalářské práce

Na obrázku 3.25 se snažím síť z bakalářské práce vylepšit zvýšením počtu konvolučních filtrů a přidáním dropout vrstvy.



Obrázek 3.25: Síť z bakalářské práce s přidaným dropoutem

Síť níže obsahuje jen jednu konvoluční vrstvu s větším pooling filtrem a vysokým dropoutem. Pro jednoduché úkoly by tato síť mohla být dostačující.



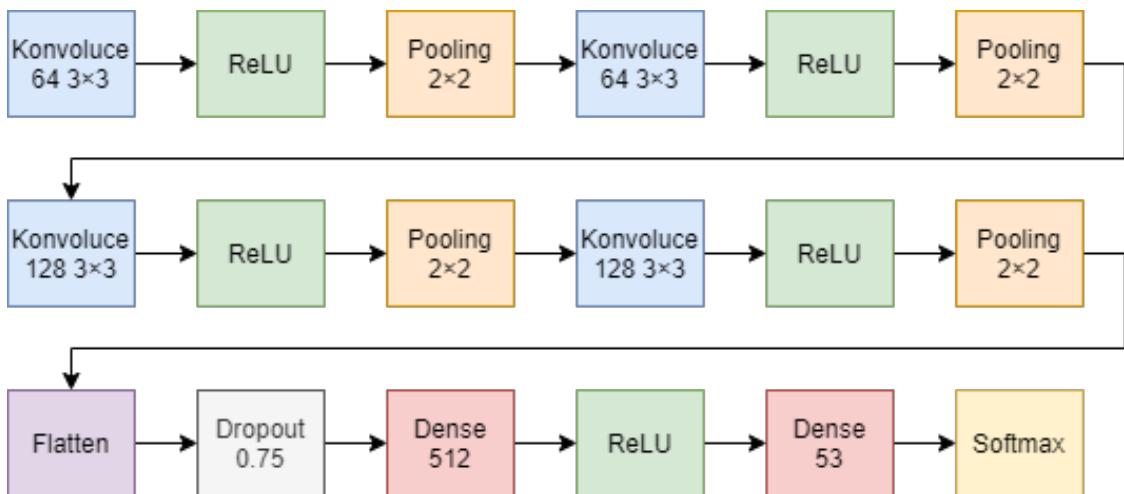
Obrázek 3.26: Síť s jednou konvoluční vrstvou a vysokým poolem

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 dobrou přesnost na validačních datech.



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

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

Kód níže navazuje na kód inicializace sítě. Zvolil jsem standardní optimizační algoritmus *ADAM*. Na posledních dvou řádcích je ukázáno ukládání historie pro

jednodušší vykreslení průběhů do grafu. Průběhy trénování budou ukázány v další sekci.

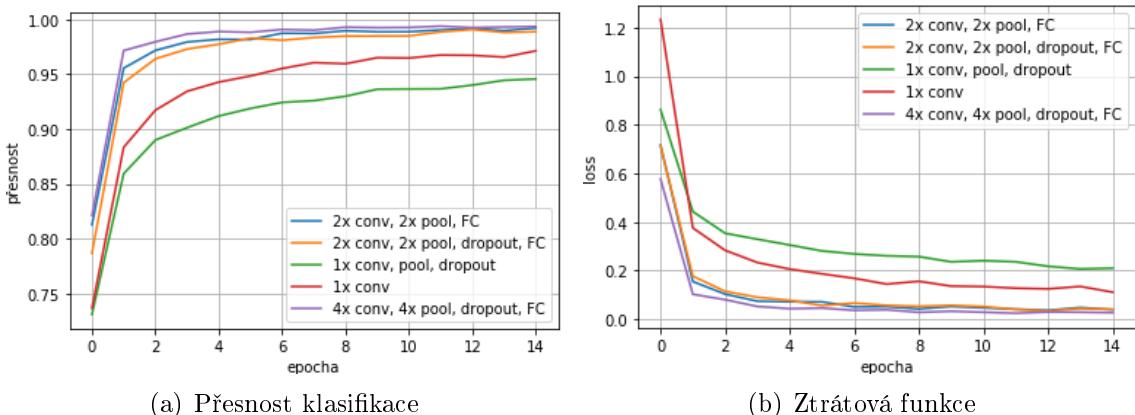
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

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

model.save('sít1')
with open('prubeh1', 'wb') as file_pi:
    pickle.dump(hist.history, file_pi)
```

3.4.3 Výsledky trénování konvolučních neuronových sítí

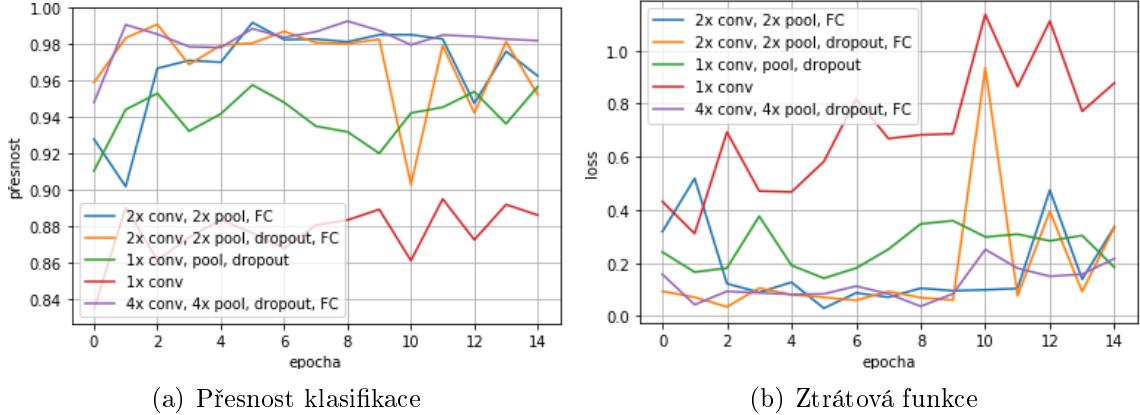
Všechny sítě jsou trénovány po dobu 15 epoch. Jak je vidět na obrázku 3.29(a), u většiny sítí se přesnost po 15 epochách zhruba ustálí. Přesnost na trénovacích datech nemusí být v reálném nasazení relevantní (obzvláště když jsou při trénování použita augmentovaná data). Z obrázků 3.4.3 je však možné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



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 je na trénovací data nepřesnější, než síť bez pool a dropout vrstvy. Za to může pravděpodobně dropout vrstva, která v průběhu trénování náhodně odpojuje příznaky z předchozí vrstvy. Naopak slušných přesností dosáhly všechny tři složitější sítě. Upravená síť z bakalářské práce dosahuje podobných přesností jako původní síť bez úprav. Zde

se dropout vrstva neprojevila tolík nejspíše kvůli přítomnosti vysokého počtu neuronů v plně propojené vrstvě. Jako ztrátovou funkci jsem zvolil funkci *categorical crossentropy*. Obrázek 3.29(b) ukazuje, že tato ztrátová funkce zrcadlí průběhy na obrázku 3.29(a).



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 přetrenování sítě. Průběh druhé sítě s dvěma konvolučními vrstvami a dropoutem je pro mě nepochopitelný. Tato síť by měla mít právě díky dropout vrstvě průběhy na validačních datech bez vysokých změn, což tak podle obou obrázků 3.4.3 není.

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 téměř daty byla natrénovaná v bakalářské práci [18]. Cílem této práce však je natrénovat konvoluční neuronovou síť s přesností alespoň 99% na testovacích datech, což podle přesnosti na validačních datech splňuje pravděpodobně pouze poslední síť se čtyřmi konvolučními vrstvami.

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 dosáhla horších přesností než původní síť. Přidané konvoluční filtry v kombinaci s dropout vrstvou by měly zajistit vyšší přesnost alespoň na validačních datech. Je možné, že síť ještě nedosáhly svých maximálních

možných přesností (to může být velkým počtem neuronů v jejich plně propojených vrstvách). Tzn. tyto dvě sítě se pravděpodobně měly nechat trénovat déle, potom by se mohla projevit přidaná dropout vrstva. To je však pouze domněnka a je možné, že přítomnost dropout vrstvy v architektuře této sítě je kontraproduktivní.

Překvapivé je, že třetí síť s jednou konvoluční vrstvou dosahuje na validačních datech podobné přesnosti jako první dvě sítě. Nejspíše je to dáné právě silnějším pooling filtrem, díky kterému se z relativně velkého obrázku extrahuje pouze důležité příznaky. U třetí sítě je zároveň vidět vliv vysokého dropoutu na její přesnost. Ačkoliv jsou trénovací data podstatně kvalitnější než validační data, dosáhla síť vyšších přesností na validačních datech.

Důležitost pool a dropout vrstev ve třetí síti 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 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 natréno-vána neuronová síť, která kartu klasifikuje. Jsou tedy připraveny všechny potřebné nástroje k analýze snímku stolu.

3.5.1 Zjištění příslušnosti 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 5 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 2 karty a zároveň že je na snímku vždy právě 5 společných karet. Pokud se celkový počet karet označí jako K , potom se počet hráčů N vypočítá následovně:

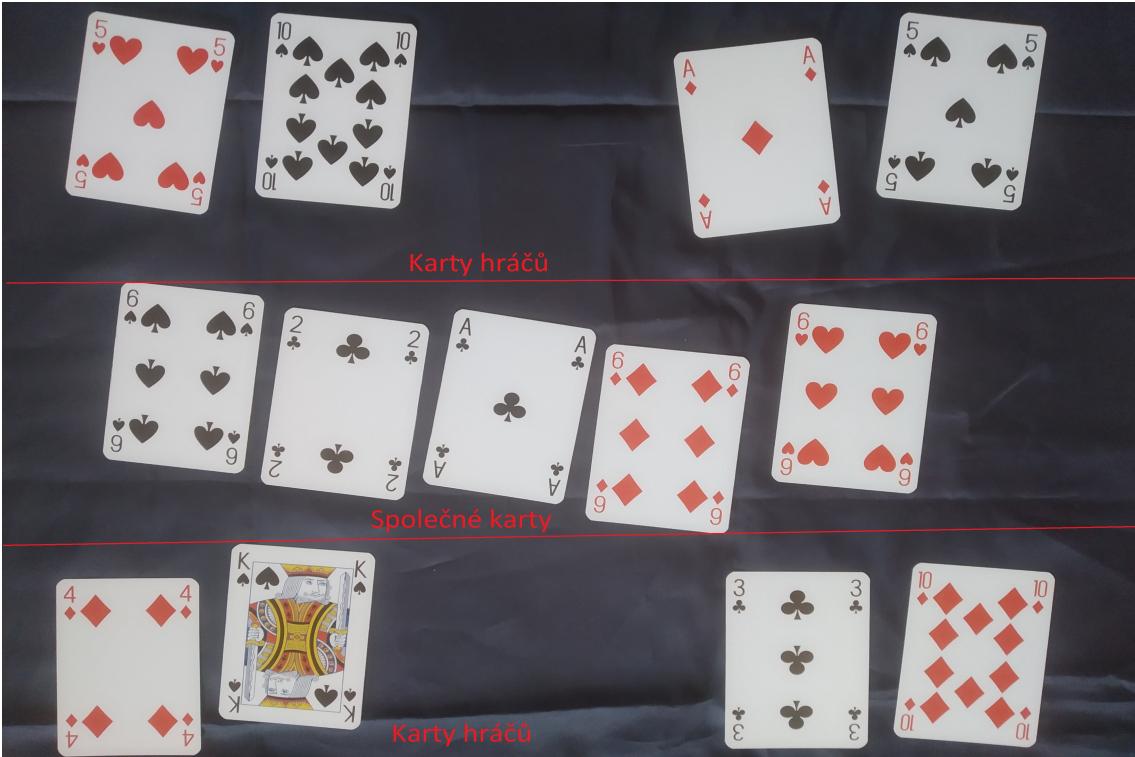
$$N = \frac{K - 5}{2} \quad (3.1)$$

4. **Seřazení karet podle jejich středů:** Hlavní myšlenkou je rozdělit hrací stůl podle souřadnicové výšky středů karet na 3 oblasti (oblast nahoře, oblast ve prostředí 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 nejnižše. 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 5 karet (společné karty). Tudíž, následujících 5 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 3 pole, které obsahují karty s podobně vysokými středy.

5. **Přiřazení karet:** Nyní seřadím podle souřadnicové šířky středů vzestupně elementy ve všech 3 polích (pole horní oblasti, pole oblasti ve prostředí 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 ve prostředí reprezentuje karty na stole, tyto karty jsou seřazeny a práce v této oblasti je dokončena. Pro oblast nahoře se z pole vyjmou vždy první 2 karty. Tyto 2 karty patří pravděpodobně jednomu hráči. Poté se vyjmou další 2 karty, dokud horní pole není prázdné. To samé se provede pro karty v dolní oblasti.

Zjištění příslušnosti karet je prováděno rozdelením hracího stolu na 3 části, přičemž prostřední část obsahuje 5 společných karet. Ostatní karty tedy patří hráčům. Pokud jsou dvě karty u sebe blízko, patří pravděpodobně tomu samému hráči.



Obrázek 3.31: Analýza rozdělením snímku hracího stolu na tři části

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 jsou všem hráčům společné. Podle počtu zakrytých společných karet lze zjistit, v jakém stavu je hra (preflop, flop, turn, river). Používané metody výpočtu pravděpodobností při pokeru byly zmíněny v sekci 2.1.4. Já jsem si pro výpočet pravděpodobností vybral Monte-Carlo simulace zejména kvůli jejich jednoduchosti.

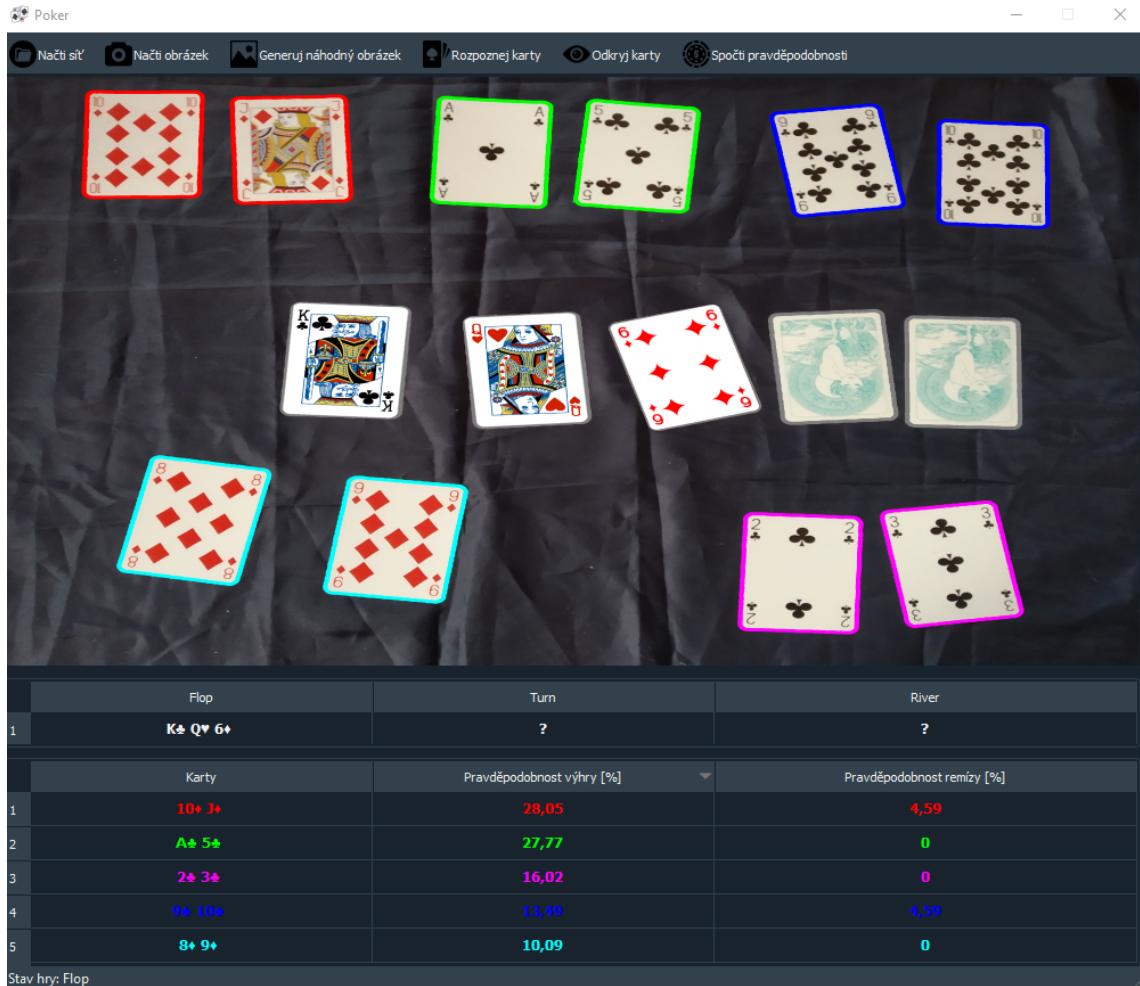
Počet prováděných simulací závisí na stavu hry. Pokud je hra ve stavu river, jsou všechny karty již známé a ke spočtení pravděpodobností na výhru tedy stačí jen jedna simulace. Pro hru ve stavu turn je neznámá pouze jedna karta (tedy možných budoucích stavů hry je $52 - (2 \cdot N + 4)$, kde N je počet hráčů). Při řešení této práce se ukázala jedna z nevýhod Monte-Carlo simulací - časová náročnost. Pro spočítání všech pravděpodobností ve hře 5 hráčů ve stavu preflop bylo potřeba přibližně 40 sekund (na počítači s procesorem i5-6200U, 2.4 GHz a 4GB RAM). 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 komplikovaným jazykům ztrácí na rychlosti. Druhým problémem je samotná optimalizace kódu, která by mohla být lepší. Pokud bych tento problém řešil znovu, pravděpodobně bych Monte-Carlo simulace implementoval v jiném, rychlejším, programovacím jazyce (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.

Co sem ještě přidat? UML diagram toho algoritmu nebo něco jinýho?

3.6 Grafické uživatelské prostředí

Pro ukázkou funkce práce jsem v knihovně *PyQt5* vytvořil grafické rozhraní. Na obrázku 3.32 níže lze vidět výsledek.



Obrázek 3.32: Grafické uživatelské rozhraní

Barvy kontur kolem karet udávají informaci o příslušnosti karet. Na obrázku 3.32 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 karet na stole 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 defaultního stylu *PyQt5* aplikace použil tématiku *qdarkstyle*.

GUI se skládá z následujících komponentů:

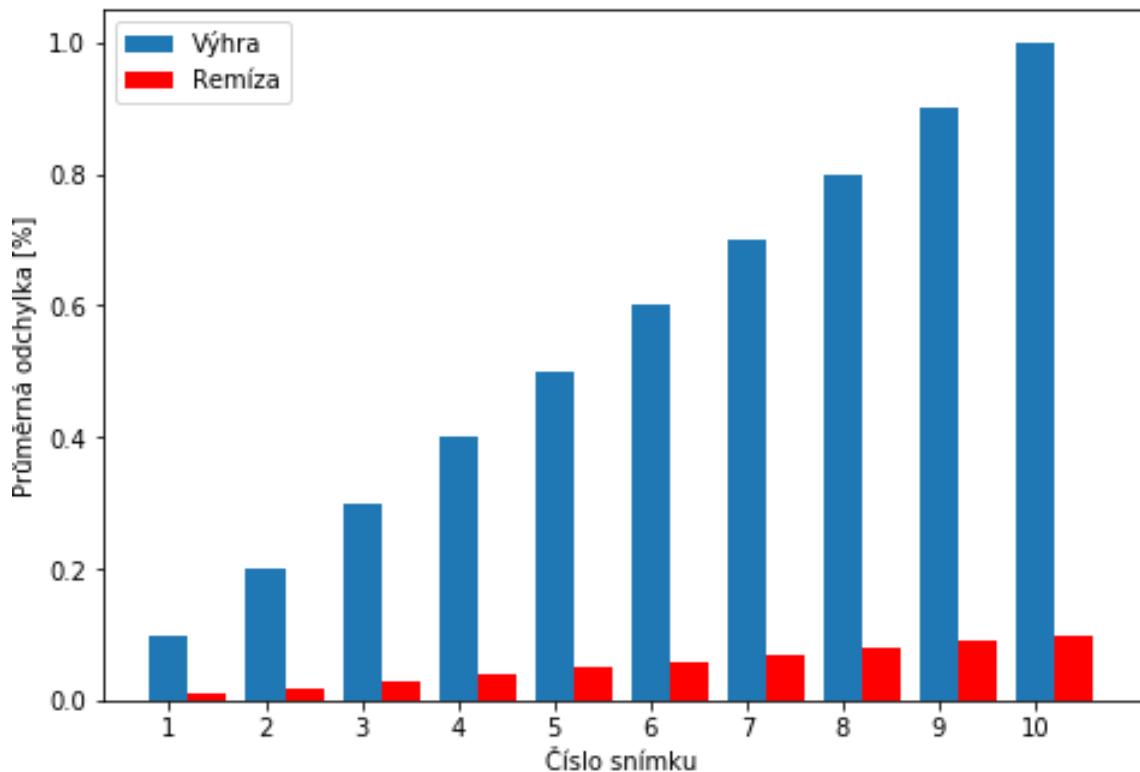
- 1. Panel tlačítek:** Panel tlačítek se nachází nahoře. Tlačítka v tomto panelu se ovládá grafické prostředí. Ikony tlačítek jsou staženy z webu *flaticon*.

- (a) "**Načti síť**": Po kliknutí se otevře dialog pro vybrání složky. Z této složky bude načtena neuronová síť na klasifikaci karet.
 - (b) "**Načti obrázek - (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 3.32 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 3.32, kde jsou hráči seřazeni od nejvyšší pravděpodobnosti na výhru po nejnižší.
4. **Stavový panel:** Stavový panel se nachází vespod GUI. Poskytuje informace jak o stavu hry (jako na obrázku 3.32), tak o stavu programu (např. síť nebyla správně načtena atd.).

3.7 Výsledky testování

V této části bude ověřeno, zda navržené algoritmy a 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 vypočtených šancí hráčů na výhru a remízu.

bude doplněno reálnýma datama, pravděpodobnosti z práce bych chtěl porovnat s online kalkulátorama, ale nevím, jestli je to dobrý nápad, protože ty kalkulátory to spočítají pokaždý jinak, liší se to třeba o 0,2%



Obrázek 3.33: Průměrné odchylky pravděpodobností pro 10 testovacích snímků

4 Závěr

V této práci byl naprogramován algoritmus na vyjmutí pokerových karet ze snímku 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 nakonec navrhl grafické prostředí.

Algoritmus vyjmutí pokerových karet ze snímku fungoval na testovacích snímcích spolehlivě a výsledné karty dosahovaly na snímcích s dobré 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 0,998% 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 různé balíky 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 robustní, avšak v mém případě časově náročným řešením. Jak bylo zmíněno v sekci 3.5.2, použitím rychlejšího 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 estetičnost.

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 navrhnut rozehodovací algoritmus pro volbu strategie (k tomu by nejspíše byla vhodná neuronová síť). 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] Salem Saleh Al-Amri, NV Kalyankar, and SD Khamitkar. Image segmentation by using edge detection. *International journal on computer science and engineering*, 2(3):804–807, 2010.
- [2] Catalin Amza. A review on neural network-based image segmentation techniques. *De Montfort University, Mechanical and Manufacturing Engg., The Gateway Leicester, LE1 9BH, United Kingdom*, pages 1–23, 2012.
- [3] S Angelina, L Padma Suresh, and SH Krishna Veni. Image segmentation based on genetic algorithm for region growth and region merging. In *2012 international conference on computing, electronics and electrical technologies (ICCEET)*, pages 970–974. IEEE, 2012.
- [4] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.
- [5] Darse Billings. *Algorithms and assessment in computer poker*. Citeseer, 2006.
- [6] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.
- [7] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.
- [8] Jason Brownlee. What is the difference between a batch and an epoch in a neural network? *Machine Learning Mastery*, 2018.
- [9] Martin Dlouhý and Petr Fiala. *Úvod do teorie her*. Oeconomica, 2007.
- [10] dmus. Keras: resize is done before preprocessing. Github issues, 2018.
- [11] Francesca Gasparini and Raimondo Schettini. Skin segmentation using multiple thresholding. In *Internet Imaging VII*, volume 6061, page 60610F. International Society for Optics and Photonics, 2006.
- [12] HDF Group et al. Hierarchical data format, version 5. 2014.
- [13] Isabelle Guyon. A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11), 1997.
- [14] Gerald Hanks, 2019.
- [15] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [17] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.
- [18] Jan Hrdlička. Detekce a rozpoznávání pokerových karet v digitálním obrazu. 2017.
- [19] Jan Hrdlička. Poker. <https://github.com/hrdlickajan/DP>, March 2020.
- [20] Michael Johanson. Measuring the size of large no-limit poker games. *arXiv preprint arXiv:1302.7008*, 2013.
- [21] Christopher Kanan and Garrison W Cottrell. Color-to-grayscale: does the method matter in image recognition? *PLoS one*, 7(1):e29740, 2012.
- [22] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. *Online Course*, 2016.
- [23] Dilpreet Kaur and Yadwinder Kaur. Various image segmentation techniques: a review. *International Journal of Computer Science and Mobile Computing*, 3(5):809–814, 2014.
- [24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [25] Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in neural information processing systems*, pages 1078–1086, 2009.
- [26] Wenkai Li and Lin Shang. Estimating winning probability for texas hold’em poker. *International Journal of Machine Learning and Computing*, 3(1):70, 2013.
- [27] Ann E Nicholson, Kevin B Korb, and Darren Boulton. Using bayesian decision networks to play texas holdem poker. *International Computer Games Association (ICGA) Journal (Unveröffentlichter Entwurf)*, 2006.
- [28] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- [29] Zhigeng Pan and Jianfeng Lu. A bayes-based region-growing algorithm for medical image segmentation. *Computing in science & Engineering*, 9(4):32–38, 2007.
- [30] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

- [31] Regina Pohle and Klaus 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*, pages 238–246. Springer, 2001.
- [32] Regina Pohle and Klaus D Toennies. Segmentation of medical images using adaptive region growing. In *Medical Imaging 2001: Image Processing*, volume 4322, pages 1337–1346. International Society for Optics and Photonics, 2001.
- [33] Hanyang Quek, Chunhoong Woo, Kaychen Tan, and Arthur Tay. Evolving nash-optimal poker strategies using evolutionary computation. *Frontiers of Computer Science in China*, 3(1):73–91, 2009.
- [34] Pavlo M Radiuk. Impact of training set batch size on the performance of convolutional neural networks for diverse datasets. *Information Technology and Management Science*, 20(1):20–24, 2017.
- [35] Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial intelligence*, 175(5-6):958–987, 2011.
- [36] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [37] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.
- [38] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.
- [39] Sumit Sarin. Exploring data augmentation with keras and tensorflow, 09 2019.
- [40] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image processing, analysis, and machine vision*. Cengage Learning, 2014.
- [41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [42] Duane Szafron, Richard Gibson, and Nathan 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*, pages 247–254. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
- [43] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.
- [44] P Warden. How many images do you need to train a neural network?, 2017.

- [45] William M Wells, W Eric L Grimson, Ron Kikinis, and Ferenc A Jolesz. Adaptive segmentation of mri data. *IEEE transactions on medical imaging*, 15(4):429–442, 1996.
- [46] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *Advances in neural information processing systems*, pages 1729–1736, 2008.