

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 grafické prostředí, ve kterém lze buď analyzovat snímek reálného hracího stolu nebo vygenerovat a analyzovat syntetický snímek.

Klíčová slova: poker, obraz, identifikace, Python, simulace, 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 a graphical environment 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, neural network, tensorflow, probability

Obsah

1	Úvod	1
2	Texas hold'em poker	2
2.1	Pravidla Texas hold'em	2
2.2	Limitace pravidel v této práci	2
3	Praktická část a vyhodnocení experimentů	3
3.1	Získání pokerových karet z obrázku	4
3.1.1	Segmentace snímku	5
3.1.2	Detekce regionů s potenciální kartou	6
3.1.3	Vykreslení kontur kolem regionů	6
3.1.4	Rotace a vyjmutí karty ze snímku	6
3.2	Generování syntetických hracích karet a stolů	6
3.3	Příprava dat pro neuronovou síť	7
3.3.1	Sběr reálných karet pro trénovací dataset	7
3.3.2	Výroba syntetických karet pro trénovací dataset	9
3.3.3	Augmentace trénovacího datasetu	11
3.3.4	Formát dat	14
3.4	Trénování neuronových sítí	18
3.4.1	Inicializace trénovacích a validačních datasetův ImageDataGeneratoru	19
3.4.2	Použité architektury neuronových sítí	20
3.4.3	Výsledky trénování neuronových sítí	23
3.5	Analýza snímku pokerového stolu	27
3.5.1	Zjištění, které karty patří jakému hráči	27
3.5.2	Výpočet pravděpodobností na výhru a remízu	27
3.6	Grafické uživatelské prostředí	27
3.7	Výsledky testování aplikace	27

1 Úvod

Zde navazuji na svou bakalářskou práci [8]. Rozhodl jsem se v tématu poker po kračovat, protože v něm lze zábavně použít metody zpracování digitálního obrazu.

Cílem této práce je spočítat pravděpodobnosti hráčů na výhru z obrazu hracího stolu. K tomu používám metody zpracování obrazu, konvoluční neuronové sítě a jednoduchou statistiku. Ve své bakalářské práci jsem navrhl algoritmus k vyjmutí pokerových karet ze snímku a natrénoval jsem neuronovou síť na jejich identifikaci s přesností 92,67% na testovacích datech. Upravený algoritmus a vyjmutí pokerových karet používám i zde. Vývoj neuronových sítí se za tři roky, které od dokončení mé bakalářské práce uběhly, velice změnil a bylo by tedy vhodné neuronovou síť vylepšit alespoň na přesnost 99%. Kromě reálných karet a snímků hracích stolů používám i počítačem vygenerované karty a snímky stolů. Výsledky práce vizualizují v grafickém prostředí *PyQt5*.

Mnou řešená úloha bude podrobněji vysvětlena v další kapitole, kde se zároveň snažím vysvětlit základní pravidla pokeru. Tato práce nerozebírá herní strategii pokeru.

2 Texas hold'em poker

Texas hold'em poker je světově nejpopulárnější verzí pokeru. (todo něco o Libratusu nebo nějaký statistiky)

2.1 Pravidla Texas hold'em

(todo popsat základy - 4 barvy, 8 hodnot, flop river turn atd., ranky hand)

2.2 Limitace pravidel v této práci

(todo 2-6 hráčů, karty hráčů jsou vidět, flop river turn jsou vyloženy pozadím na začátku hry)

3 Praktická část a vyhodnocení experimentů

V této kapitole se zaměřím na vysvětlení použitých metod na praktických příkladech.

Nejprve krátce popíšu algoritmus na vyjmutí karet z obrazu. Algoritmus pro získání karet je převzat z mé bakalářské práce. 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 hracím stole poté ověřím správné fungování výsledné aplikace.

Ve své bakalářské práci jsem nekladl příliš velký důraz na přípravu trénovacího datasetu. To se projevilo na přesnosti natrénované sítě. V této práci jsem si dal na tvorbě datasetu více záležet. Budu zároveň popisovat *tensorflow* modul *ImageDataGenerator*, který podstatně zjednoduší proces augmentace karet.

Z připravených trénovacích dat natrénuji pět neuronových sítí různých architektur. Tato část práce je spíše experimentální. Chci ukázat, jak se změní přesnost sítě, vynechám-li některé důležité komponenty jako například pool, dropout či fully connected vrstvu. Jedna z těchto neuronových sítí bude sloužit ke klasifikaci reálných karet.

Pátá sekce se věnuje analýze hry. Popíšu zde algoritmus, jakým se přiřazují detekované karty jednotlivým hráčům. Pravděpodobnosti na výhru jsou v této práci spočteny pomocí simulací pokerových her.

Další podkapitolou je tvorba grafického uživatelského rozhraní, které sjednotí všechny zde zmíněné procesy. V práci budu často tento výraz nahrazovat zkratkou GUI (graphical user interface). Popisují zde základy nejčastěji používaných Python knihoven pro tvorbu GUI *tkinter* a *PyQt5*. V *PyQt5* lze GUI vytvořit i bez programátorských schopností díky aplikaci *QtDesigner*. *QtDesigner* v podkapitole také stručně popíšu.

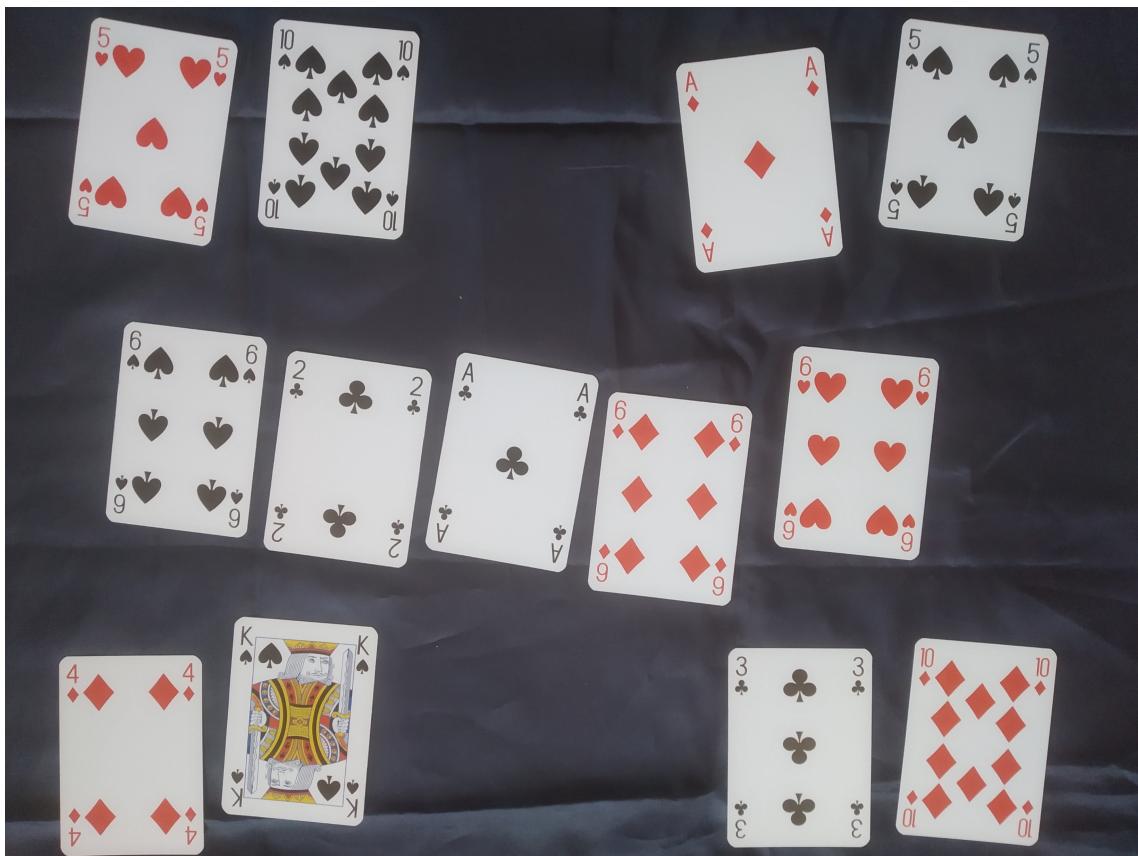
Nakonec aplikaci otestuji na vygenerovaných a reálných snímcích pokerových stolů. Vyhodnotím přesnost natrénované neuronové sítě na testovacích datech a porovnám mnou spočtené pravděpodobnosti na výhru s online kalkulátory.

3.1 Získání pokerových karet z obrázku

Prvním úkolem je ze snímku získat všechny karty. Proces získání karet je nezbytný pro správnou analýzu stavu hry. Zároveň je jeho modifikace použita při tvorbě trénovacího datasetu pro neuronovou síť. Základní myšlenka pro získání karet ze snímku zůstala stejná jako v mé bakalářské práci:

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

Kroky 4 a 5 se opakují do doby, než se ze snímku vyjmou všechny karty.



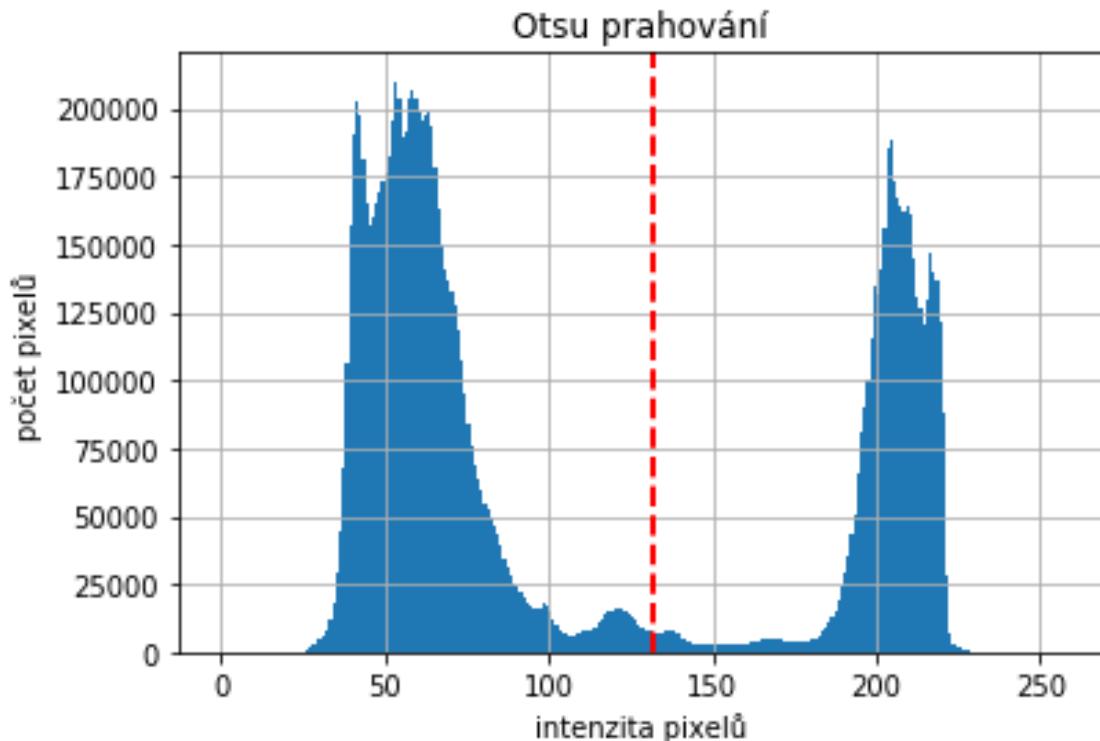
Obrázek 3.1: Obrázek, ze kterého budou karty získávány

Jako předlohu jsem zvolil snímek reálného stolu - obrázek 3.1. Na tomto snímku budu postupně ukazovat výše zmíněné metody.

3.1.1 Segmentace snímku

V bakalářské práci jsem používal metodu adaptivní segmentace pro vyrovnání jasu ve snímku. To je zbytečné. Pro klasifikaci karty je používána barevná karta a segmentovaný snímek slouží v tomto případě pouze k zaměření regionů s kartami. Důležité při segmentaci je, aby se dobře oddělily objekty od pozadí. V této práci volím metodu prahování *otsu* z knihovny *skimage* [18].

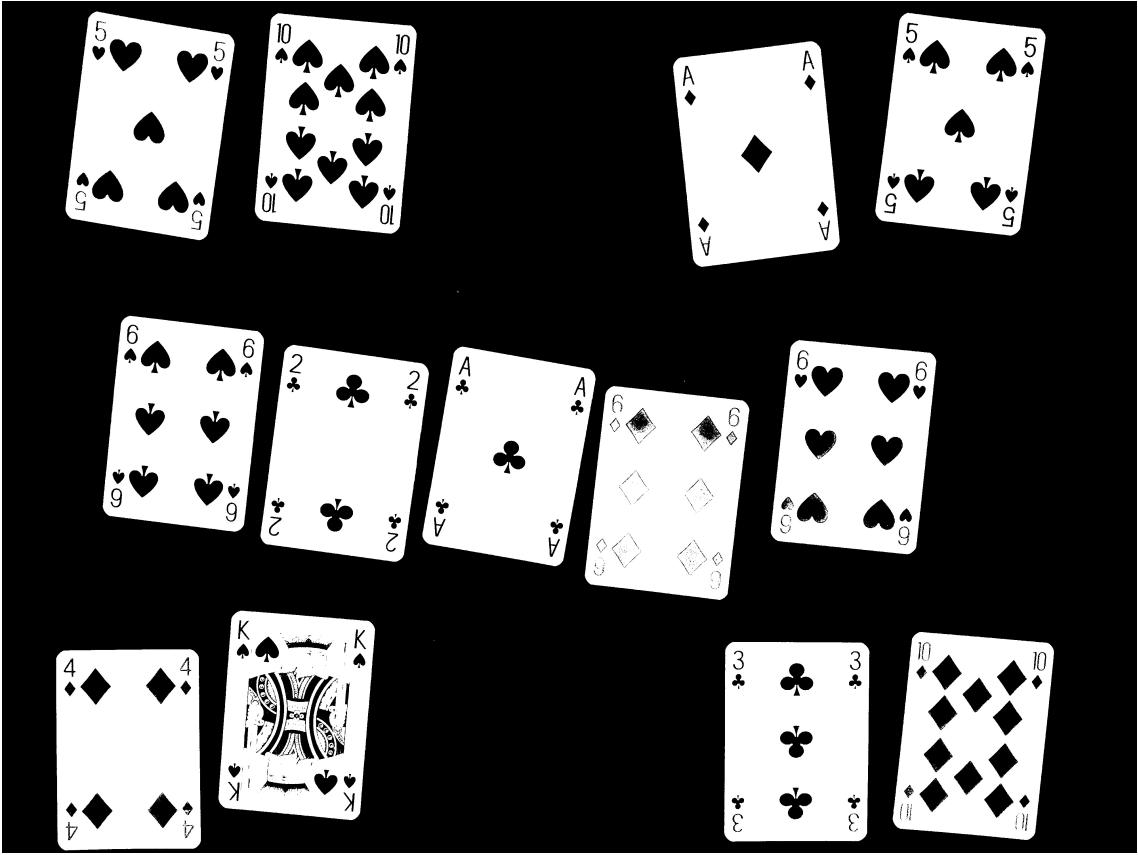
Metoda *otsu* je technika automatického hledání optimálního prahu. Algoritmus nalezne práh, který minimalizuje vážený rozptyl dvou tříd jasů - třídy pozadí a popředí.



Obrázek 3.2: Histogram šedotónového snímku stolu

Na obrázku 3.2 jsou dva "kopce". Levý kopec představuje tmavé pixely - pozadí a tmavé znaky na kartách. Pravý kopec jsou pixely světlé - karty. Metoda *otsu* nalezne práh tak, že rozdělí tyto dva kopce, aby variance obou kopců byly minimální. Nalezený práh je v obrázku znázorněn červenou přerušovanou čarou.

Jak je vidět na obrázku 3.3, *otsu* metoda zvolila práh tak, že většina symbolů na kartách jsou prohlášena za pozadí. V tomto případě to není důležité. Podstatné je, že se oddělily okraje karet od pozadí a bude tak možné dobře počítat obsah regionů popředí. To dělám v další sekci.



Obrázek 3.3: Segmentovaný snímek metodou otsu

3.1.2 Detekce regionů s potenciální kartou

3.1.3 Vykreslení kontur kolem regionů

3.1.4 Rotace a vyjmutí karty ze snímku

3.2 Generování syntetických hracích karet a stolů

3.3 Příprava dat pro neuronovou síť

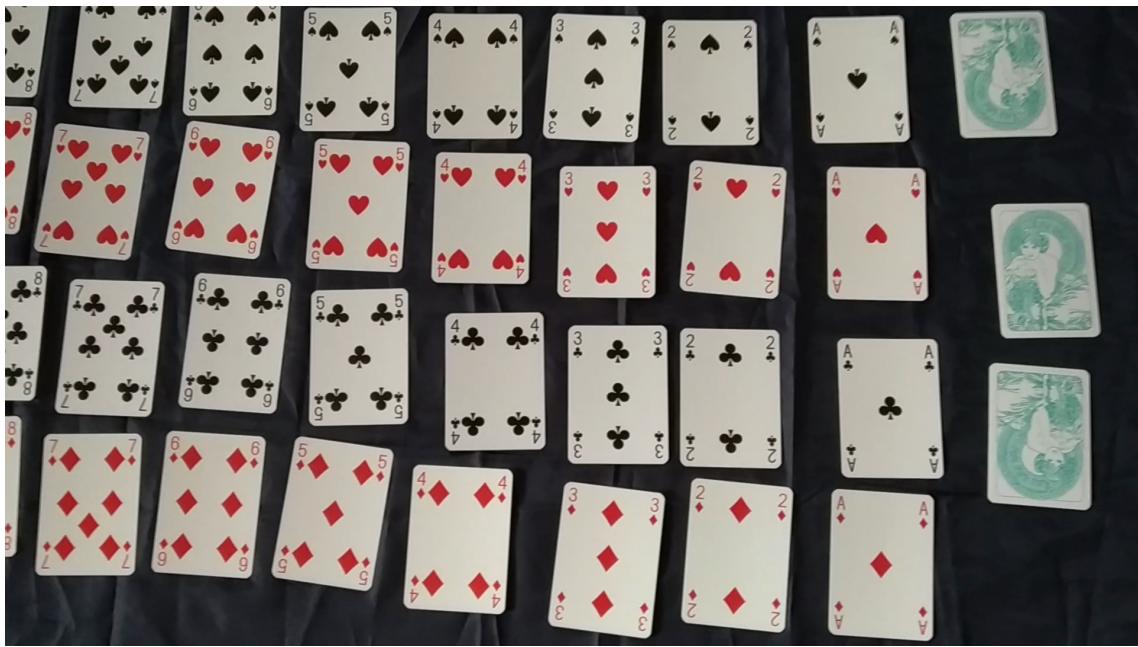
Použití kvalitních trénovacích dat je základním předpokladem pro natrénování přesné neuronové sítě [10]. V mém případě bude trénovací dataset tvořen z reálných karet a z karet generovaných pomocí programu zmíněného v kapitole (todo). Aby trénovací data byla dostatečně rozmanitá, budou navíc augmentována pomocí *tensorflow* modulu *ImageDataGenerator*.

Oproti mé bakalářské práci zde kladu větší důraz na přípravu dat než na samotnou architekturu sítě. V kapitole (todo) používám stejná data pro sítě s různými architekturami. Ukázalo se, že soustředit se na kvalitu dat je v tomto případě opravdu důležitější než architektura sítě, většina sítí totiž dosáhla lepších výsledků než síť, kterou jsem natrénoval v bakalářské práci.

3.3.1 Sběr reálných karet pro trénovací dataset

Pořizování reálných karet zabere hodně času. 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. Důkazem je experiment v podkapitole (todo).

Ke zjednodušení práce používám program na vyjmutí a identifikaci karet ze snímků. Aby bylo snímků dostatek, natáčím videa za pomoci knihovny *OpenCV*. Tato videa poté rozložím na jednotlivé snímky, ze kterých získám karty. Rozložil jsem všechny karty na segmentovatelné pozadí tak, aby se nepřekrývaly a natočil několik videí za různých denních dob v různě osvětlených místnostech.



Obrázek 3.4: Pořizování dat - snímek z videa

Karty automaticky zařazuji do 53 složek (všechny druhy karet a pozadí) pomocí neuronové sítě z mé bakalářské práce. Síť má přesnost 93,5% na testovacích datech, tudíž bylo nutné složky jednu po druhé zkонтrolovat, že obsahují pouze karty, které tam opravdu patří. Karty, které byly 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.



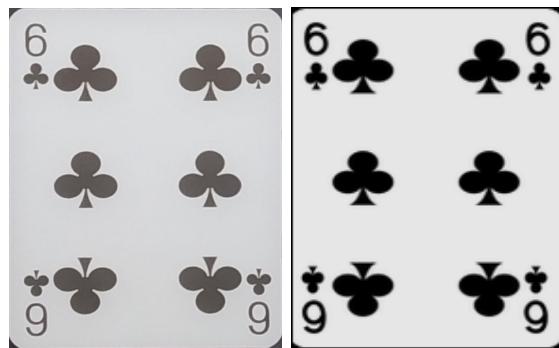
Obrázek 3.5: Příklad špatně klasifikovaných karet

Nejlepším postupem by při řešení problémů ručního přesouvání byla nejspíš střední cesta. Z obrázku 3.5 bych přesunul do správné složky pouze srdcového kluka, který je už hodně znehodnocen odrazem světla a zbytek karet bych smazal. Složky 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.

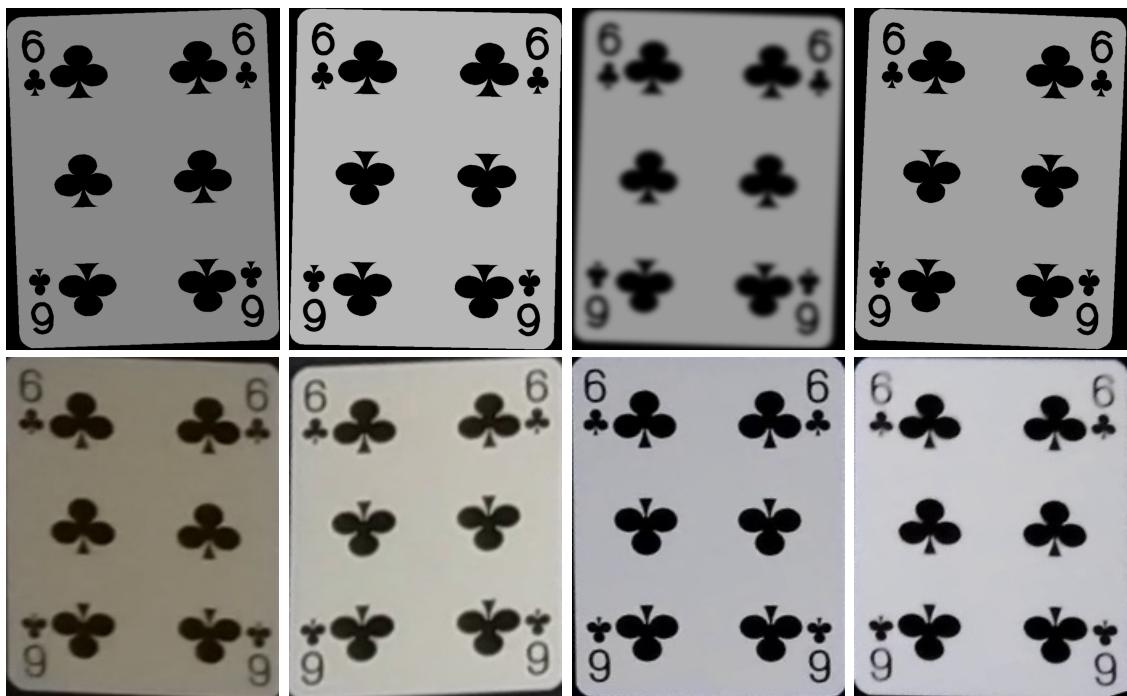
3.3.2 Výroba syntetických karet pro trénovací dataset

Výhodou počítačem generovaných dat je možnost vygenerovat velké množství obrázků za krátkou dobu a rozšířit tak trénovací dataset. Na rozdíl od sběru reálných dat budu generovat 3 sety karet. Jeden set karet je totožný s tím reálným a zbylé dva jsou stažené z internetu. Výrobu syntetických karet jsem použil i v kapitole (todo), kde generují hrací stůl. Reálný set karet generují pomocí vkládání obrazců a písma do snímku. Porovnání syntetické karty s reálnou kartou lze vidět na obrázku 3.6.

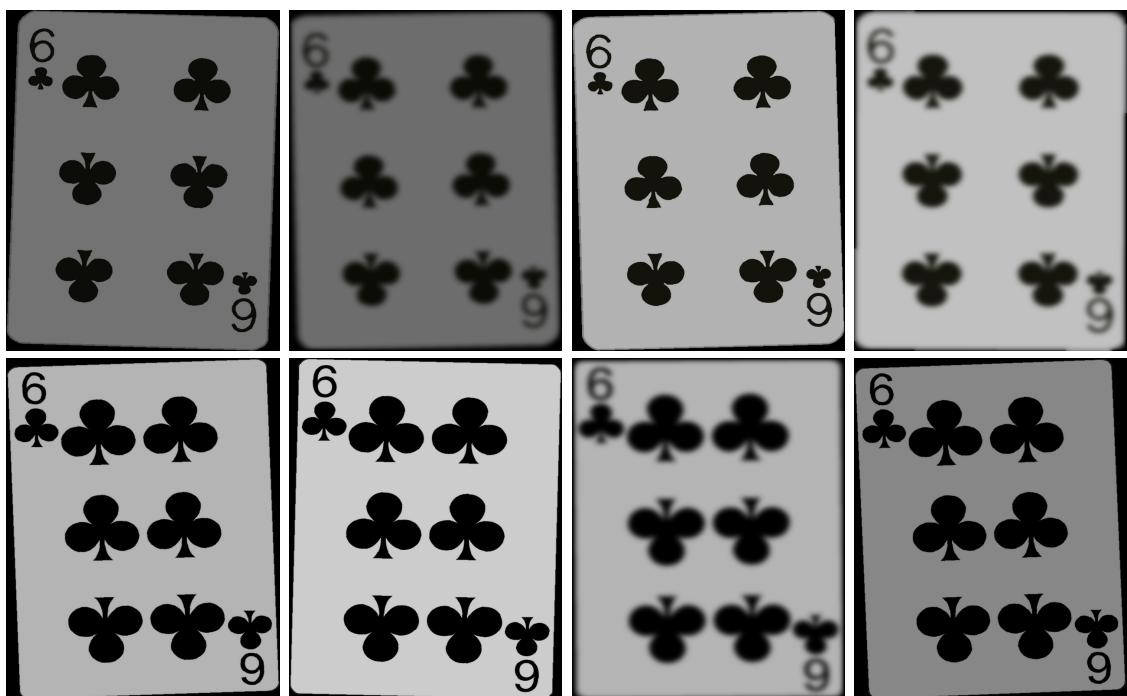


Obrázek 3.6: Porovnání: vpravo reálná karta, vlevo generovaná karta

Aby generovaná karta vypadala více reálně, provádím na ní 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. Tento jev však na reálných kartách nikdy nenastal a tak ho v této práci nepoužívám. Chci, aby se syntetické karty podobaly co nejvíce kartám reálným.



Obrázek 3.7: Porovnání: Nahoře syntetické karty po augmentaci, dole reálné karty



Obrázek 3.8: Syntetické karty po augmentaci, zbylé dva sety

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

3.3.3 Augmentace trénovacího datasetu

Díky augmentacím lze vytvořit či upravit data. Čím více rozličných dat má síť k dispozici, tím lépe by měla najít řešení problému. Obrázky je třeba augmentovat tak, aby byly zachovány jejich hlavní rysy. Příliš silné augmentace mohou být pro trénování sítě kontraproduktivní [11]. 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 nasazení. Například:

- Špatné osvětlení - karta se leskne nebo naopak není kvůli tmě dobře vidět
- Používání kamery se špatným barevným vyvážením (zejména u bílé barvy kamery často natočí žlutou)
- Karta je posunuta či rotována chybou programu na vyjmutí karet

ImageDataGenerator

Pomocí tohoto modulu augmentují 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 [16].

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 [14]. V argumentu *pre-processing_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. Já budu používat následující augmentace:

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 3.9: Karta před augmentacemi



Obrázek 3.10: Rotace o maximalně 45°

2. Horizontální posun

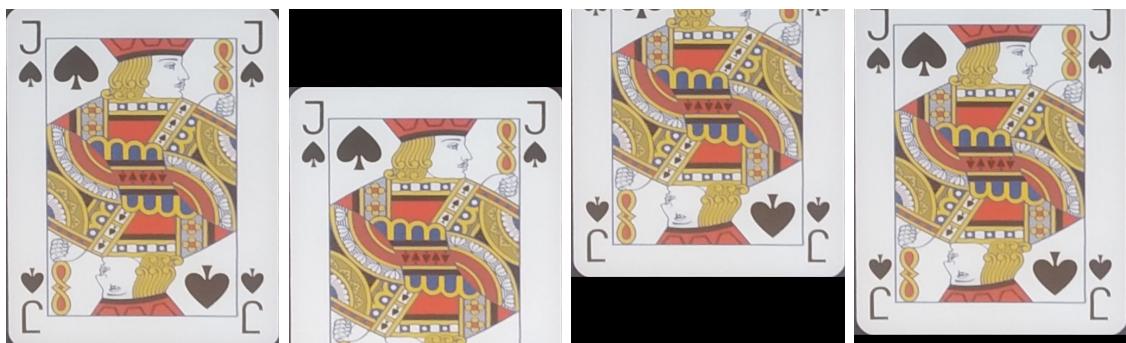
width_shift_range se udává jako číslo od nuly do jedné. Obrázek se posune počet pixelů z intervalu $< -width_shift \cdot \text{šířka obrázku v pixelech}, +width_shift \cdot \text{šířka obrázku v pixelech} >$. Obrázky níže ukazují posun maximálně o 0.3 šířky obrázku:



Obrázek 3.11: Posun o maximálně 0.3 šířky obrázku

3. Vertikální posun

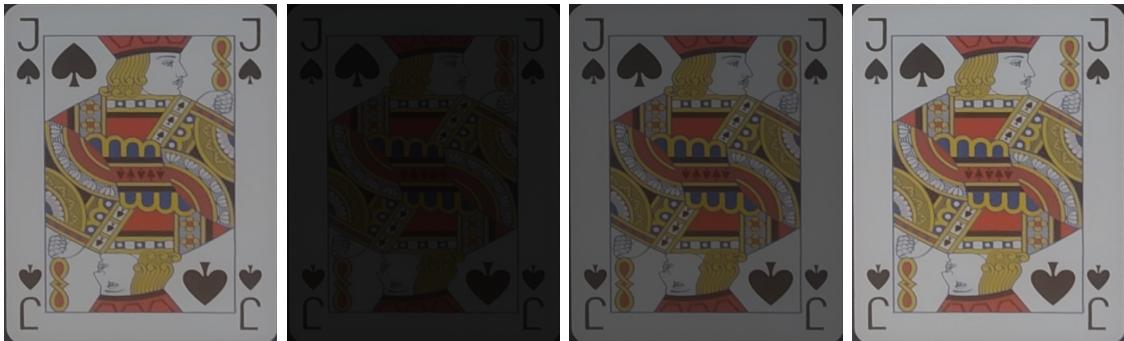
height_shift_range se používá stejně jako horizontální posun, obrázek se pochopitelně posune vertikálně.



Obrázek 3.12: 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 3.13: 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 znovu maximálním posunem.



Obrázek 3.14: Posun barevných kanálů maximálně o 150

Proč nepoužívám flip? Todo

Pro augmentaci karet jsem použil následující hodnoty:

- **rotace** maximálně o 3°
- **horizontální posun** maximálně o 5% šířky
- **vertikální posun** maximálně o 5% výšky
- **změna jasu** v rozmezí od 10 do 100% současného jasu
- **posun intenzit barevných kanálů** maximálně o 100 jednotek intenzity

Všechny augmentace se na kartu aplikují s náhodnou silou. Obrázek 3.15 ukazuje, jaké karty vzniknou kombinací těchto úprav:



Obrázek 3.15: Náhodně provedené augmentace

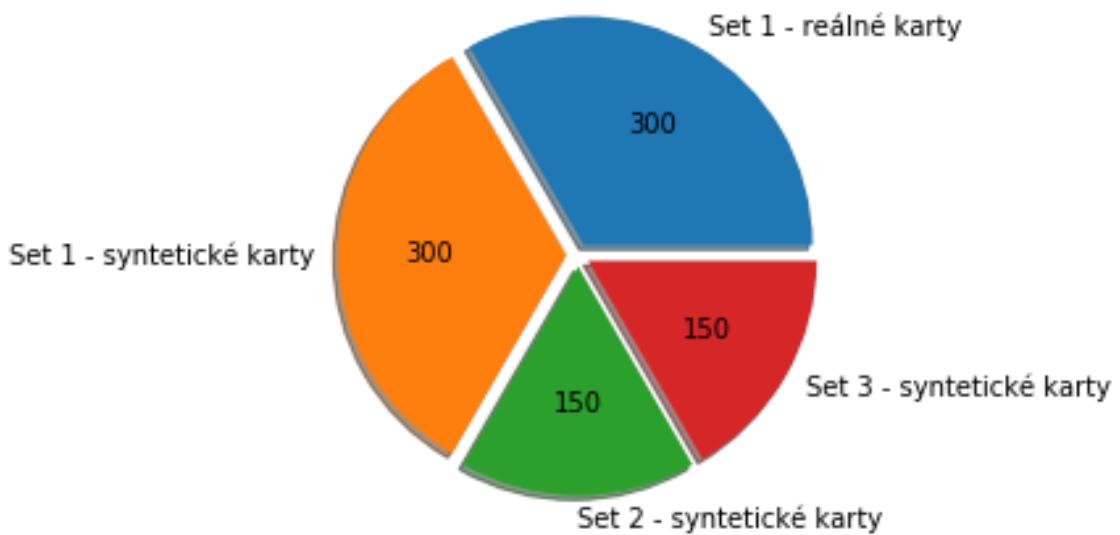
3.3.4 Formát dat

Velikost trénovacího datasetu

Velikost trénovacího datasetu ovlivňuje přesnost a robustnost sítě. Pro velikost datasetu neexistuje univerzální předpis. Základním doporučením podle [19] 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 [9]. 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, kde jsem počet vzorů v třídách nevyrovnával a síť dosáhla přesnosti na trénovacích datech pouze 93,5 %.

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



Obrázek 3.16: 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. Počítáčem vytvořené karty se budou identifikovat snáz než reálné karty ze setu 1 a proto je jich v datasetu méně.

Rozměr obrázků

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) [6]. 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 velikosti 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 3.1: Standardně používané datasety

Rozměr obrazů v *ImageNet* je proměnný, většinou se ale dataset používá s rozlišením 256×256 . Vstupní rozměry však nemusí být čtvercové.

V mém případě je dobré zachovat poměry stran karty. V reálném nasazení čtvercová karta neexistuje. Ve většině případů se pro přesnost sítě jeví zvolení vyššího rozlišení (až 500×500 pixelů) vstupního obrazu lépe, než zvolení nižšího rozlišení (224×224) [7]. Třídy v pokerových kartách jsou však dobře separabilní a proto mi přijde rozlišení 150×200 pixelů jako optimální kompromis mezi velikostí a kvalitou obrázku. V *kerasu* je možnost změnit rozměr obrázků přímo při jejich načítání ze složek.

Normalizace

Normalizace vstupu pozitivně ovlivňuje stabilitu a rychlosť konvergencie sítě [15]. Tato procedura není nutná a síť dokáže fungovať i bez ní. Použitím rovnice (3.1) níže se hodnoty intenzit pixelů v každém barevném kanálu upraví z rozsahu 0 – 255 na rozsah 0 – 1:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (3.1)$$

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.

Způsob uložení dat

Obrazy mohou být uloženy buď klasicky ve složkách, kde v každé složce jsou obrázky naležící jedné třídě nebo ve formátech *HDF5* [3], resp. *Pickle* (*Pickle* je součástí standardní Python knihovny).

Tyto formáty uchovávají obrázky ve formě polí. Standardní postup při použití těchto formátů je vytvořit dva soubory. Do jednoho souboru se uloží všechny obrazy a ve druhém souboru budou tzv. labely; popis do jakých tříd obrazy patří. Formáty *HDF5* a *Pickle* slouží zejména k přesunu velkého množství dat z jednoho skriptu do druhého. Před vytvořením datasetů tímto způsobem je důležité data náhodně promíchat, aby se po sobě neopakovala karta jedné třídy. To by mohlo způsobit problém při dávkovém trénování, kde se síti předkládá pouze zlomek celkové velikosti datasetu.

Soubory v tomto formátu je těžké modifikovat v uživatelském prostředí a proto radši 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 - 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.



Obrázek 3.17: Připravená trénovací data

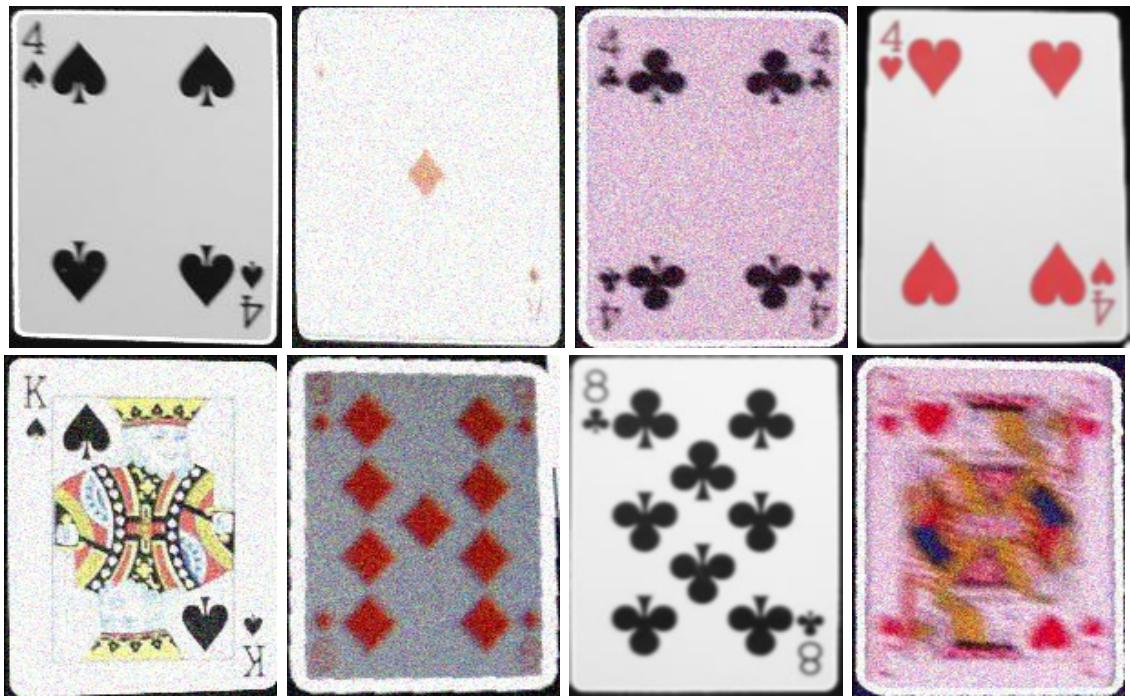
Při použití tohoto způsobu ukládání dat je dobré si v trénovacím skriptu zároveň uložit pořadí daných tříd. To se bude hodit po natrénování sítě.

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. 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" [5]. Opačný případ vyšší přesnosti na validačních datech může nastat, pokud při trénování sítě byly použity regularizační techniky jako dropout či cross validace - více v kapitole (todo). Tento jev pouze ukazuje, že je síť robustní.

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 [4].

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 3.18: 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 sety 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 3.19, testovací karty jsou podstatně kvalitnější než karty v trénovacím a validačním datasetu.



Obrázek 3.19: Testovací data

3.4 Trénování neuronových sítí

V této kapitole natrénuji několik neuronových sítí s různými architekturami abych zjistil, jak architektura sítě ovlivňuje přesnost. Ukáže se, že i s jednoduchou architekturou lze dosáhnout dobrých výsledků. Problém rozpoznávání pokerových karet je totiž relativně jednoduchý úkol.

Dále se pokusím natrénovat síť jen pomocí počítačem vytvořených karet se zachováním velikosti trénovacího datasetu.

Dalším pokusem bude natrénování sítě pouze pomocí prvního setu karet, který obsahuje kombinaci reálných a syntetických karet. Validační set obsahuje také pouze tento set. Zajímavé zjištění je, že tato síť dosahuje na validačních datech nižších přesností než síť natrénovaná na tři různě vypadající balíky karet.

Všechny sítě budou přímo 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

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 [2]. Já používám takové augmentace, že na rozlišení obrázku nezáleží.

Z kódu níže lze vidět, že jsem velikost jedné trénovací dávky zvolil jako 64. Při psaní této práce jsem však zjistil, že jsem spíše měl volit vyšší čísla. Podle [12] 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.

Na čtvrté řádce provádím 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 šírkou 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 neuronových sítí

Celkem trénuji 5 architektur sítí. Sítě se liší svojí komplexností a použitými vrstvami. Cílem této kapitoly je ukázat, které architektury sítí na můj problém fungují a které méně. 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 kerasu se sítě vytváří jednoduše. Níže je příklad inicializace architektury sítě z obrázku 3.20, který je na další stránce. Prvním řádkem se vytvoří 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'))

```

Vysvětlení pojmu:

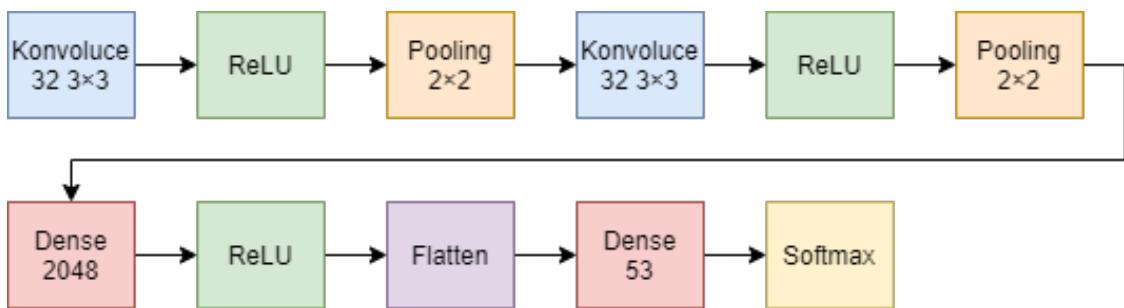
- *Conv2D* - první číslo značí počet filtrů konvoluce. Druhým parametrem je jejich velikost

- *Activation('relu')* - aktivační funkce ReLU mapující záporný příznak na nulu. Dána předpisem

$$f(x) = \max(0, x) \quad (3.2)$$

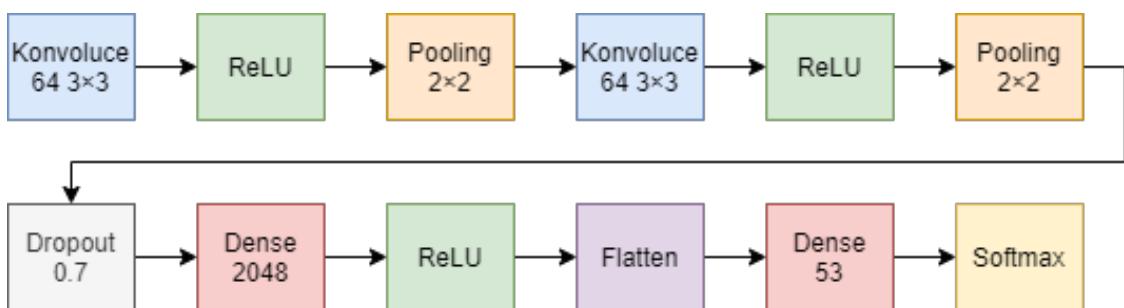
- *Flatten* - převede pole ve třech dimenzích do jednodimenzionálního pole
- *Dropout* - číslo od nuly do jedné udává náhodný počet odpojených vstupních příznaků. Doporučuje se volit hodnoty v rozmezí 0,5 - 0,8 [17]. Dropout vrstva je funkční pouze po dobu trénování.
- *Dense* - název *kerasu* pro plně propojenou vrstvu
- *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. Podle rozdělení pravděpodobnosti je suma *Softmax* vektoru rovna jedné.

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



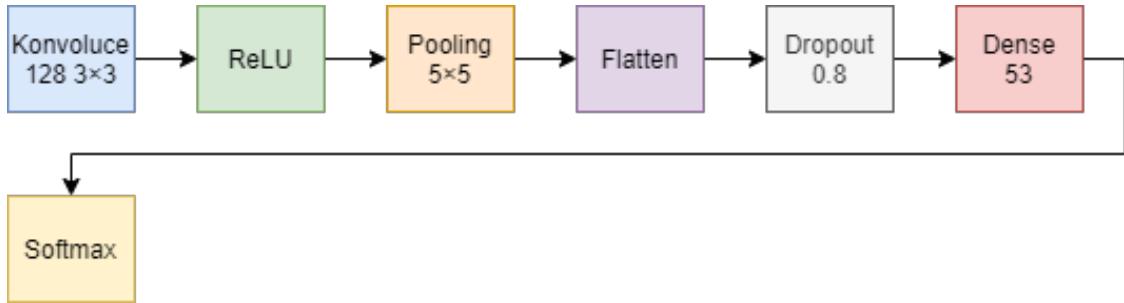
Obrázek 3.20: Síť z bakalářské práce

Na obrázku 3.21 se snažím síť z bakalářské práce vylepšit zvýšením počtu konvolučních filtrů a přidáním dropout vrstvy.



Obrázek 3.21: 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.



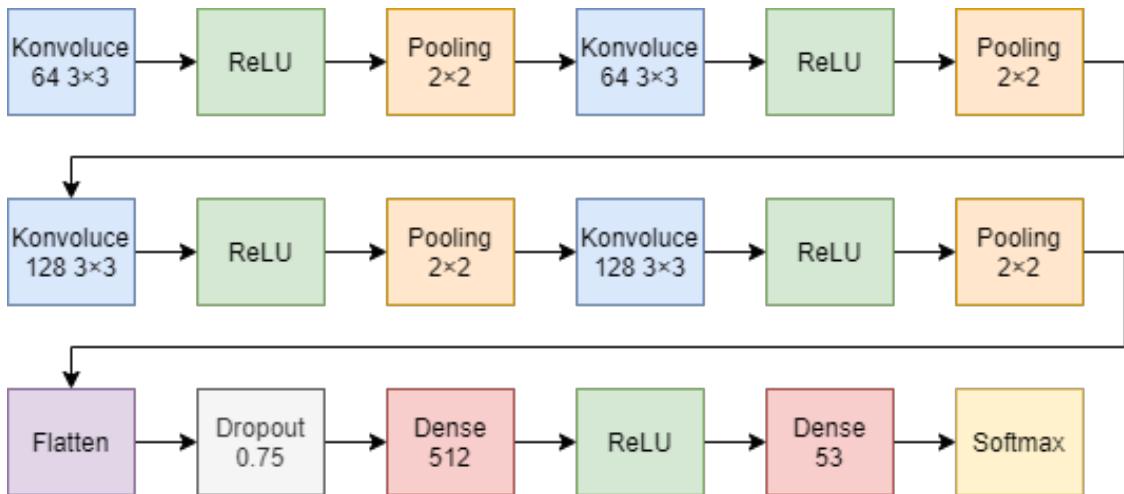
Obrázek 3.22: Síť s jednou konvoluční vrstvou a vysokým poolem

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



Obrázek 3.23: Síť tvořená pouze konvoluční a ReLU vrstvou

Síť na obrázku 3.24 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 3.24: Síť se čtyřmi konvolučními vrstvami a jednou skrytou vrstvou

Na konci všech sítí se vždy vyskytují vrstvy *Flatten* a *Dense* a aktivační funkce *Softmax* vyhodnocující výsledek klasifikace.

Kód níže navazuje na inicializaci sítě. Zvolil jsem standardní optimizační algoritmus *ADAM* [13]. Na posledních dvou řádcích je ukázáno ukládání historie pro jednodušší vykreslení průběhů do grafu. Průběhy trénování ukážu v další kapitole.

```
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)
```

3.4.3 Výsledky trénování neuronových sítí

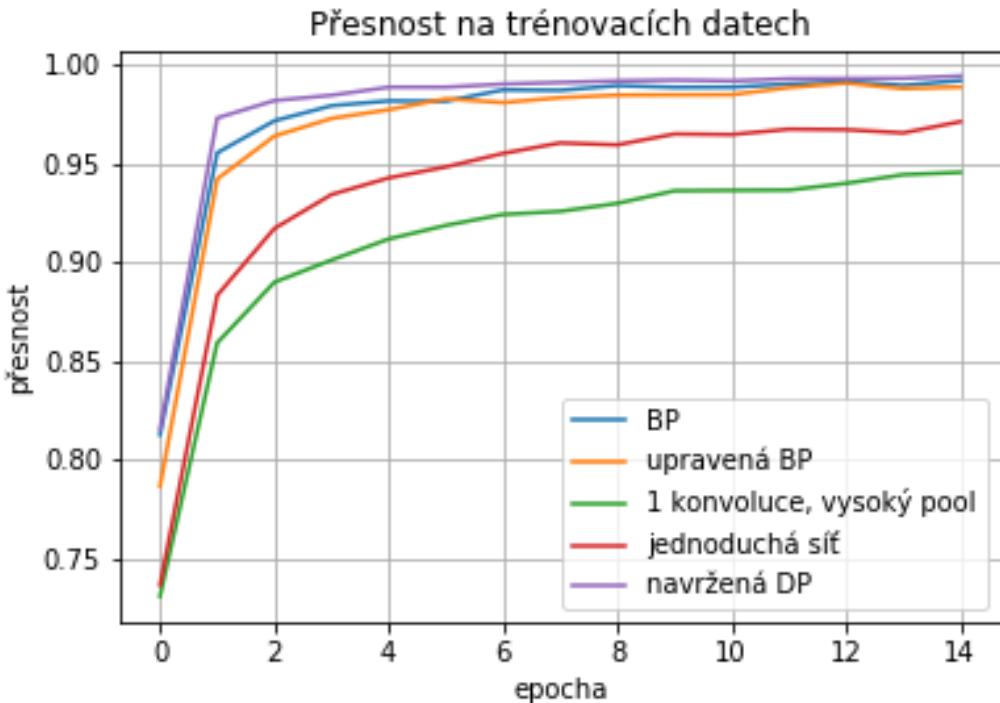
Všechny sítě jsem nechal trénovat po dobu 15 epoch. Jedna epocha znamená, že se síti předloží všechny trénovací vzory z datasetu. Síť by se měla nechat trénovat do té doby, dokud se validační ztrátová funkce snižuje. Proto počet epoch není přesně dán. Doba trénování sítí závisí na velikosti trénovacího datasetu a počtu spojení v síti. Tato závislost je přibližně lineární [1].

Přesnost (accuracy) je procentuální vyjádření správné klasifikace. Jak je vidět na obrázku 3.25, u všech sítí se přesnost po určitém počtu epoch zhruba ustálí.

Síť predikuje podle následujícího předpisu:

$$index = \arg \max_{i=1,2,\dots,53} y(i), \quad (3.3)$$

kde $index$ je pořadí třídy s nejvyšší predikovanou pravděpodobností. i představuje pořadí jedné třídy a y je výstup ze Softmaxu. Síť jednoduše vezme nejvyšší hodnotu pravděpodobnosti a podle pořadí této hodnoty klasifikuje.



Obrázek 3.25: Průběhy přesnosti na trénovacích datech

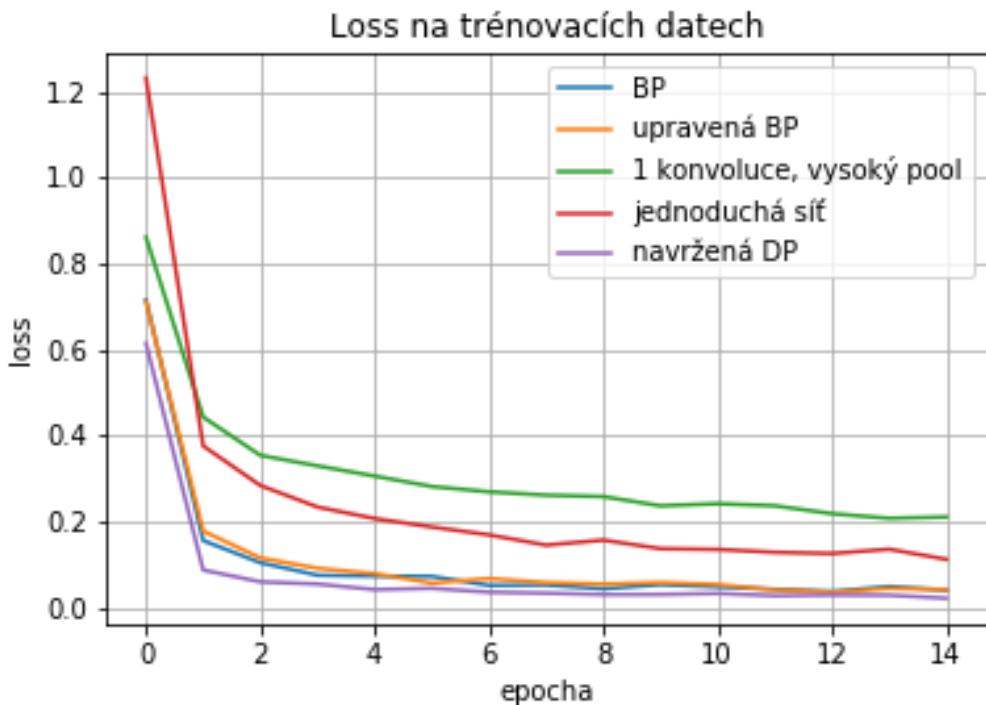
Přesnost na trénovacích datech nemusí být v reálném nasazení relevantní. Z obrázku 3.25 je však možné předpovídат, které sítě dosáhnou vysokých přesností na reálných datech a které ne. Nejhůře na tom jsou sítě s jednou konvoluční vrstvou. Slušných přesností dosáhly všechny tři složitější sítě. Upravená síť z bakalářské práce dosahuje podobných přesností jako původní síť bez úprav.

Jako ztrátovou funkci jsem zvolil funkci nazývanou *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:

$$H(p, q) = - \sum_{\forall x} p(x) \cdot \log(q(x)) \quad (3.4)$$

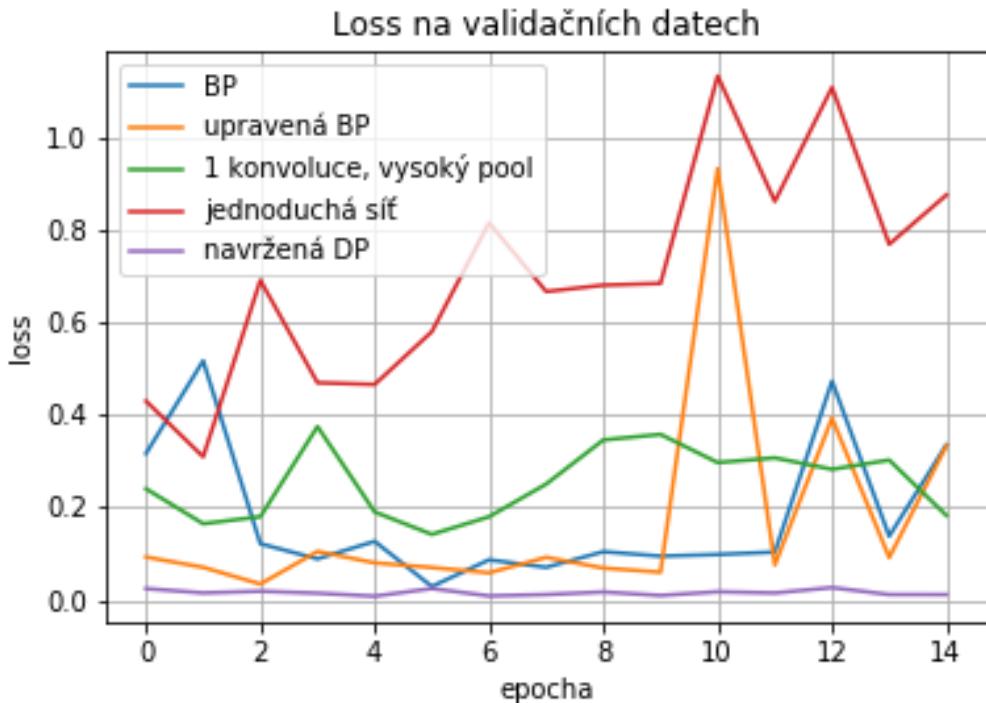
kde $p(x)$ je skutečná pravděpodobnostní funkce, $q(x)$ je predikovaná pravděpodobnostní funkce a x reprezentuje jeden konkrétní obraz. Obrázek 3.26 ukazuje, že ztrátová funkce zrcadlí průběhy na obrázku 3.25.



Obrázek 3.26: Ztráta na trénovacích datech

Dalším obrázkem 3.27 je ztrátová validační funkce. Validační přesnosti opět zrcadlí průběhy validačních ztrát. Přesnosti zhodnotíme podrobněji v další podkapitole.

Fluktuace ztrátové validační funkce je dána tím, že proces nastavování vah sítě při jejím trénování je náhodný. Pokud ztrátová funkce skáče i po delší době, kdy už se trénovací přesnost výrazně nemění, může to být známkou přetrénování sítě.



Obrázek 3.27: Průběh ztrát na validačních datech

Vyhodnocení výsledků trénování neuronových sítí

V tabulce 3.4.3 se budu zaměřovat hlavně na přesnosti na validačních datech. Podobných přesností by totiž mohly sítě dosahovat i v reálném nasazení.

Ukázalo se, že jsem síť z bakalářské práce příliš nevylepšil. Přidaná *Dropout* vrstva by se pravděpodobně projevila až po více epochách. Překvapivé je, že třetí síť s jednou konvolucí dosahuje na validačních datech podobné přesnosti jako první dvě sítě. Nejspíše je to dáné právě větším *pooling* filtrem, díky kterému se z relativně velkého obrázku extrahují pouze důležité příznaky. Důležitost *Pooling* a *Dropout* vrstev v této síti dokazují výsledky jednoduché sítě na čtvrtém řádku. Tato síť sice dosahuje lepších přesností na trénovacích datech, je však přetrénovaná a tedy nepřesná na validační data. 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.

Popis sítě	trénovací [%]	validační [%]
síť z BP	0,992	0,962
upravená síť z BP	0,989	0,952
1 konvoluce, vysoký pool	0,946	0,957
jednoduchá síť	0,971	0,877
navržená síť na DP	0,995	0,998

Tabulka 3.2: Porovnání přesnosti sítí

3.5 Analýza snímku pokerového stolu

- 3.5.1 Zjištění, které karty patří jakému hráči
- 3.5.2 Výpočet pravděpodobností na výhru a remízu

3.6 Grafické uživatelské prostředí

3.7 Výsledky testování aplikace

4 Závěr

Literatura

- [1] Jason Brownlee. What is the difference between a batch and an epoch in a neural network? *Machine Learning Mastery*, 2018.
- [2] dmus. Keras: resize is done before preprocessing. Github issues, 2018.
- [3] HDF Group et al. Hierarchical data format, version 5. 2014.
- [4] Isabelle Guyon. A scaling law for the validation-set training-set size ratio. *AT&T Bell Laboratories*, 1(11), 1997.
- [5] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.
- [6] 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.
- [7] Andrew G Howard. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*, 2013.
- [8] Jan Hrdlička. Detekce a rozpoznávání pokerových karet v digitálním obrazu. 2017.
- [9] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. *Online Course*, 2016.
- [10] 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.
- [11] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.
- [12] 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.
- [13] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
- [14] 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.

- [15] 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.
- [16] Sumit Sarin. Exploring data augmentation with keras and tensorflow, 09 2019.
- [17] 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.
- [18] 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.
- [19] P Warden. How many images do you need to train a neural network?, 2017.