Středoškolská technika 2017

Setkání a prezentace prací středoškolských studentů na ČVUT

OPTICKÉ ROZPOZNÁVÁNÍ RUČNĚ PSANÉHO TEXTU

Břetislav Hájek

Gymnázium Český Brod
Vítězná 616, Český Brod

Anotace

Tato práce se zabývá tvorbou programu pro optické rozpoznávání českého ručně psaného textu, kde jsou písmena navzájem spojena. Ručně psaný text má stále své místo v každodenním životě a přesto, že zde již existují programy pro rozpoznání některých stylů rukopisů, většina neumí rozpoznat vázané písmo. V práci je rozbíráno jedno z možných řešení tohoto problému využívající techniky strojového učení a počítačového vidění.

Klíčová slova

OCR (Optické rozpoznávání znaků); strojové učení; počítačové vidění; ručně psaný text

Obsah

1 Úvod

Nápad na tento projekt vznikl ve chvíli, kdy jsem byl nucen přepsat ručně psanou slohovou práci do digitální podoby a nemohl jsem nalézt jiný způsob než manuální přepsání celého textu. Ručně psaný text je stále neodmyslitelným nástrojem každého z nás a schopnost převést ho do digitální podoby by mohla ušetřit velké množství času nejen ve škole, ale i na úřadech, v archivech a mnohých dalších místech, protože na rozdíl od ručně psaného textu, je práce s digitálním textem značně automatizovaná a můžeme jej snadno upravovat, kopírovat nebo v něm vyhledávat.

Tato práce se tedy zabývá tvorbou OCR (Optical Character Recognition) softwaru pro ručně psaný text. Vzhledem k rozsáhlosti tohoto problému (různým jazykům, rukopisům a stylům písma) je nemožné snažit se optimalizovat program pro všechny druhy textů, a proto je tento projekt zaměřen na převod česky psaného vázaného písma, především mého rukopisu, psaného na bílém papíře. Program tedy na vstup dostane fotografii stránky s ručně psaným textem a vrátí text v digitální podobě. I přes to, že tato omezení snižují praktické uplatnění tohoto programu, cílem je především nalézt spolehlivou metodu, kterou by bylo možné dále přizpůsobit pro konkrétní případy.

Pro řešení tohoto problému byly využity algoritmy strojového učení (anglicky machine learning) a počítačového vidění (anglicky computer vision), které umožňují zpracování dat a obrazu. Klíčovou technikou pro tento projekt je strojové učení, neboť nám umožňuje rozdělovat vstupní data na základě již známých příkladů.

2 POUŽITÉ TECHNOLOGIE

Pro vývoj programu byl zvolen jazyk Python 3, neboť se jedná o velmi rozšířený programovací jazyk, pro který existuje velké množství kvalitních knihoven implementujících funkce strojového učení a počítačového vidění. Tyto knihovny mají vysoce optimalizovaný kód, který umožňuje efektivní využití výkonu hardwaru. Knihovny byly vybrány především na základě poskytovaných funkcí a vzájemné kompatibilitě, zároveň byla snaha jejich počet minimalizovat.

Pro vývoj softwaru byla využita webová aplikace Jupyter Notebook (dříve zvaná Ipython Notebook), která umožňuje rozdělit kód do bloků, které můžeme upravovat, spouštět a zároveň zobrazovat jejich výstup. Aplikace dále umožňuje zobrazovat grafy a obrázky a pomocí interaktivních prvků (widgetů) je upravovat. V aplikaci je také možné přidávat textová pole pro přehlednější členění a prezentaci celého kódu. Vzhledem k těmto funkcím, se jedná o populární nástroj pro analýzu dat a návrh algoritmů strojového učení.

2.1 Použité knihovny

2.1.1 Numpy

Numpy poskytuje efektivní datové struktury pro práci s vektory a maticemi, spolu se základními operacemi. Knihovna je také využívána knihovnou OpenCV. Ve zdrojovém kódu se používá pod zkratkou *np*.

2.1.2 Matplotlib

Matplotlib poskytuje funkce pro vytváření grafů, které můžeme zobrazovat přímo v interaktivním prostředí aplikace Jupyter notebook.

2.1.3 OpenCV

OpenCV (Open Source Computer Vision) je knihovna zaměřená především na počítačové vidění a jeho využití v reálném čase. Ve zdrojovém kódu se používá pod zkratkou *cv2*.

2.1.4 Tensorflow

Tensorflow je knihovna určená především pro strojové učení. Tensorflow umožňuje vytvářet grafy reprezentující matematické operace a pohyb dat mezi nimi. Zároveň poskytuje i komplexní funkce strojového učení. Ve zdrojovém kódu se používá pod zkratkou *tf*.

3 POČÍTAČOVÉ VIDĚNÍ

Počítačové vidění je odvětví výpočetní techniky, které se zabývá zpracováním a získáváním informací z obrazových dat (obrázků nebo videa) [1]. Digitální obrázky rozlišujeme na vektorové (definované pomocí základních geometrických útvarů) a rastrové (definované pomocí hodnot jednotlivých barevných pixelů). Počítačové vidění se zaměřuje především na zpracování rastrového obrazu, který počítač chápe pouze jako pole hodnot udávajících barvu jednotlivých bodů.

I přes to, že počítačové vidění je rozsáhlý obor, tato práce využívá pouze techniky zpracování statického obrazu poskytované knihovnou OpenCV. Knihovna OpenCV obsahuje jak funkce pro základní úpravy obrázků, tak i pokročilé metody počítačového vidění.

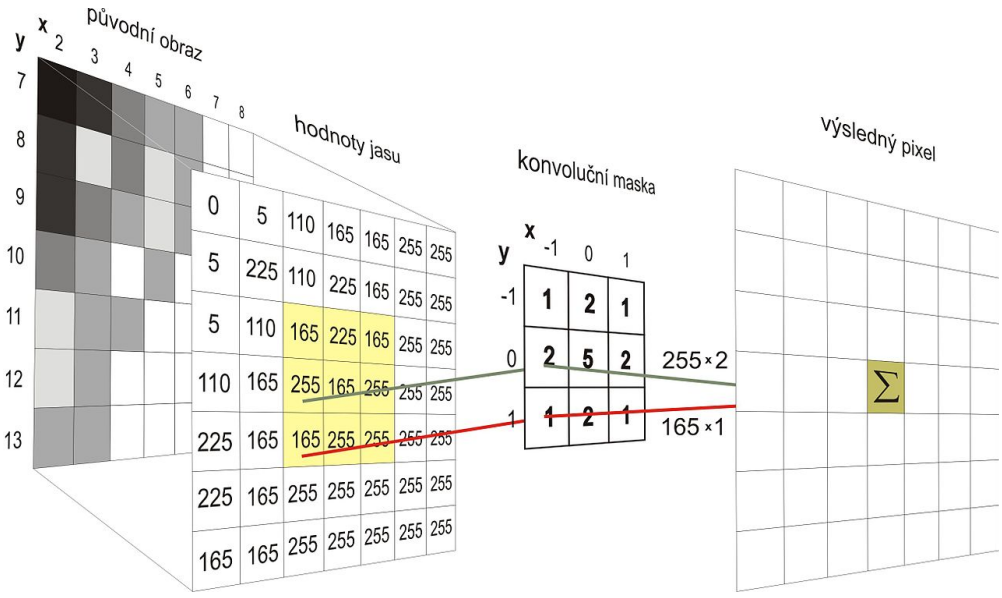
3.1 Prahování

Prahování (anglicky thresholding) umožňuje změnit hodnotu všech pixelů přesahující danou hranici (prah) [2]. Tato jednoduchá metoda je často využívána k odstranění nepotřebných hodnot či vytvoření binárního obrázku. V OpenCV a v této práci se můžeme setkat se třemi základními typy prahování, podle určení hraniční hodnoty. První druh využívá ručně definovanou hranici, což je nevýhodné v případě velkých odlišností mezi vstupními obrázky. Druhý druh automaticky využívá střední hodnotu z histogramu obrázku. Poslední druh tzv. adaptivní prahování počítá a využívá odlišnou hranici pro jednotlivé regiony, to je výhodné například při nerovnoměrném nasvícení obrazu [3]. Dále je uveden příklad těchto funkcí aplikovaných na obrázek uložený v proměnné *img*.

```
# Definovaná hranice na 127
ret,thresh1 = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
# Adaptivní prahování
th2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                            cv2.THRESH_BINARY, 11, 2)
# Střední hodnotu z histogramu obrázku (Otsu)
ret2,th2 = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

3.2 Filtry

Filtry slouží ke zpracování rastrového obrazu (např.: rozmazání nebo zdůraznění kontur) pomocí tzv. konvoluce. Tato technika využívá konvoluční jádro (matici hodnot), někdy také nazýváno konvoluční maska, které je postupně aplikováno (hodnoty pixelů jsou vynásobeny hodnotami z matice) na každý pixel a pixely v dosahu jádra a výsledný součet je zapsán na místo zpracovávaného pixelu (viz obr. 1). Jednoduchým příkladem může být průměrový filtr, který hodnotu pixelu nahradí průměrem okolních pixelů a tím rozostří celý obraz [4]. Filtry mají rozsáhlé uplatnění a tato práce využívá celou řadu různých filtrů a jejich kombinací.



Obr. 1: Princip konvoluce – Autor obrázku: Grt. Dostupné z: https://commons.wikimedia.org/wiki/File:Konvoluce_2rozm_diskretni.jpg

3.3 Detekce hran

Detekce hran je jedním ze základních problémů počítačového vidění, neboť přesná detekce hran nám umožňuje rozlišovat jednotlivé objekty. Pro detekci hran je využito předpokladu, že v místě hrany dochází k velké změně jasu. I přes to, že existuje více druhů detekce hran, v této práci je využit pouze Cannyho hranový detektor a samostatný Sobelův operátor.

Sobelův operátor se skládá ze dvou filtrů (funkce *cv2.Sobel*), které rozpoznávají změnu jasu v daném směru, jeden ve vertikálním a druhý v horizontální směru. Pro každý pixel tak dostaneme dvě hodnoty, ze kterých můžeme vypočítat intenzitu a směr přechodu (vztah (1) a (2)). Vztah (1) je využit v naší implementaci Sobelova operátoru, protože OpenCV poskytuje pouze odpovídající filtr, nikoliv však kompletní implementaci [5].

$$I = \sqrt{F_x^2 + F_y^2} \tag{1}$$

$$\theta = \arctan\left(\frac{F_y}{F_x}\right) \tag{2}$$

Cannyho hranový detektor (funkce *cv2.Canny*) vychází z Sobelova operátoru, který rozšiřuje o další kroky, a vrací tak přesnou polohu hran. Po provedení Sobelova operátoru následuje fáze ztenčení, během níž jsou ponechána pouze lokální maxima ve směru nalezeného přechodu. Závěrečná fáze prahování využívá dvou prahů (maximálního a minimálního) a tím rozdělí nalezené hrany do tří skupin. Nejvýraznější hrany (nad maximálním prahem) jsou ponechány, zatímco slabé hrany (pod minimálním prahem) jsou odstraněny. Zbylé hrany jsou ponechány pouze v případě, že jsou napojeny na některou z výrazných hran [6].

4 STROJOVÉ UČENÍ

Strojové učení můžeme rozdělit do několika kategorií: strojové učení s učitelem, učení bez učitele a učení posilováním (anglicky supervised learning, unsupervised learning a reinforcement learning) [7]. Učení s učitelem využívá známých dat pro vytvoření modelu, který dokáže predikovat cílovou veličinu (kategorii, hodnotu, atd.), zatímco učení bez učitele nevyužívá známých dat a slouží k hledání struktury dat (závislostí, skupin a anomálií). Pro řešení problému této práce bylo využito pouze učení s učitelem.

4.1 Učení s učitelem

Úkolem učení s učitelem je pro vstupní data predikovat správnou hodnotu nebo kategorii, my ho budeme využívat pro rozdělování obrázků do kategorií (klasifikaci). Základní algoritmy klasifikace využívají vztahu (3), kde x značí j vstupních veličin a W, b značí hledané parametry (anglicky weights a bias). Funkce f značí aktivační funkci, která vrací hodnoty ve známém rozsahu. Příkladem aktivační funkce může být sigmoida (logistická funkce ze vztahu (4)), která se pohybuje v rozsahu od 0 do 1, což nám umožňuje rozdělovat data do dvou kategorií.

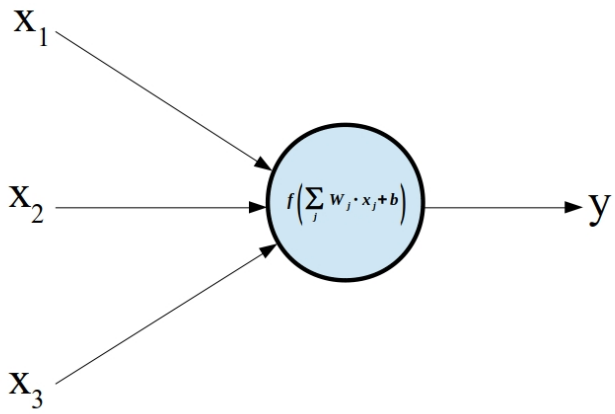
$$y = f\left(\sum_j W_j \cdot x_j + b\right) \tag{3}$$

$$f(z) = \frac{1}{1 - e^{-z}} \tag{4}$$

Během procesu učení (trénování) jsou hledány takové parametry W a b, aby odchylka od očekávaného výsledku byla pokud možno co nejmenší, proto jsou potřeba již roztříděná data. Jedním ze základních způsobů hledání optimálních hodnot je tzv. klesání podle gradientu (anglicky gradient descent), které využívá derivace funkce hodnotící chybu predikce (anglicky cost function) na trénovacích datech a parametry upravuje tak, aby se chyba minimalizovala. Opakováním této metody se program dostane do lokálního minima a získáme tak predikční model. V praxi se pro zvýšení rychlosti používají komplexnější metody využívající dalších vlastností. Tensoflow poskytuje hned několik takových funkcí a tato práce využívá funkce *tf.train.AdamOptimizer*.

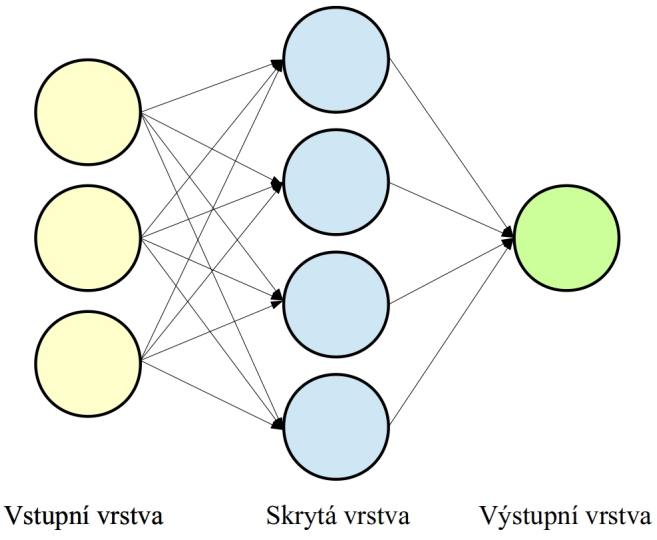
Fáze učení může být výkonnostně velmi náročná, a to především pro komplexní modely s velkým množstvím dat. V případě velké datové sady (především z důvodu omezené operační paměti) se obvykle v každém kroku (určení chyby a úpravení parametrů, dále jen krok) určuje chyba pouze na malé skupině náhodně vybraných dat. Ačkoli tento postup vyžaduje větší počet kroků k dosáhnutí minima, je efektivnější než načítání celé datové sady. Jednou z výhod využití specializovaných knihoven (např. Tensorflow) je paralelizace učení, což umožňuje využití všech dostupných procesorů a výrazně tak zvýšit výkon.

4.1.1 Neuronová síť



Obr. 2: Grafické znázornění neuronu, který na vstup aplikuje aktivační funkci a dál předává její výsledek.

Protože samotná aktivační funkce nemůže reprezentovat komplexní modely, byl vytvořen koncept neuronové sítě (inspirován biologickou strukturou mozku). Neuronová síť se skládá z mnoha navzájem propojených neuronů (obr. 2), které dohromady vytváří komplexní model (obr. 3). Neuronová síť se obvykle skládá z několika vrstev, první vrstvu nazýváme vstupní a poslední výstupní, vrstvy mezi nimi označujeme jako skryté. Neurony po sobě jdoucích vrstev jsou nejčastěji propojeny stylem každý s každým. Pro trénování neuronové sítě se využívá algoritmus zpětného šíření, který je založen na podobném principu jako klesání podle gradientu.



Obr. 3: Znázornění jednoduché neuronové sítě

Při použití knihovny Tensorflow, vždy nejprve definujeme tzv. graf, který značí pohyb dat mezi jednotlivými vrstvami a následně ho využijeme pro trénování parametrů, čímž vytvoříme predikční model.

4.1.2 Konvoluční neuronová síť

Konvoluční neuronové sítě vycházejí ze struktury neuronových síti a nejčastěji se využívají pro klasifikaci obrazu (popřípadě vícerozměrných dat). Konvoluční neuronové sítě se skládají z několika vrstev: konvoluční, škálovací a neuronové.

Základem je konvoluční vrstva, která se skládá z několika trénovatelných filtrů, které jsou aplikovány na obrázek (vstupní data) a výsledek je předán další vrstvě. Konvoluční vrstva se definuje počtem filtrů, rozměrem a velikostí posunu filtru a aktivační funkcí. Rozšířenou aktivační funkcí pro konvoluční vrstvu je tzv. rectified linear unit (ReLU), doslovně přeloženo jako usměrněná lineární jednotka, vztah (5).

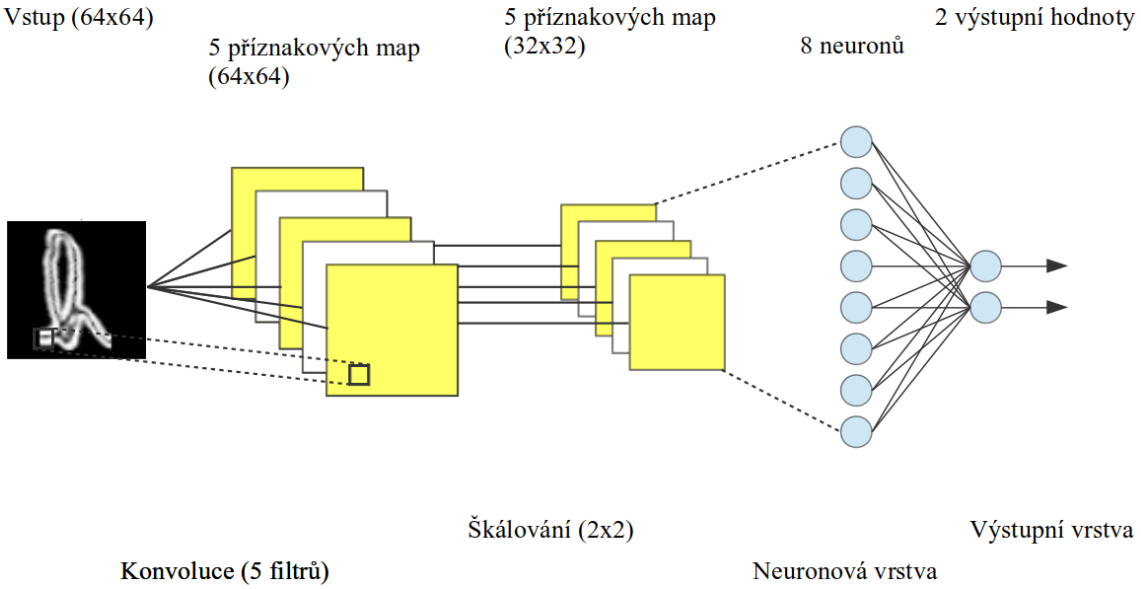
$$F(z) = max(0, z) \tag{5}$$

Tato jednoduchá aktivační funkce umožňuje efektivnější učení modelu [8], a proto v tomto projektu je ReLU využita ve všech konvolučních vrstvách. Definování konvoluční vrstvy pomocí Tensorflow:

```
# Parametry W a b pro konvoluční vrstvu - 20 filtrů velikosti 5x5
W = tf.Variable(tf.truncated_normal([5, 5, 1, 20], stddev=0.1), name="W")
b = tf.Variable(tf.constant(0.1, shape=[20]), name="b")
# Definování konvoluční vrstvy (pro vstup image)
conv = tf.nn.relu(tf.nn.conv2d(image, W, strides=[1, 1, 1, 1],
                               padding='SAME') + b)
```

Během fáze učení modelu se filtry z konvoluční vrstvy učí detekovat různé vizuální vlastnosti obrázku. To umožňuje efektivnější zpracování obrazových dat s menším počtem parametrů, než v případě samostatné neuronové sítě [9].

Vrstva škálování (anglicky pooling layer) slouží ke zmenšení obrázku (předávaných dat) a tím zvýšení výkonu aplikace. Pro zmenšování se využívají různé metody, příkladem je tzv. max pooling (Tensorflow ji poskytuje jako *tf.nn.max_pool*), filtr nejčastěji velikosti 2x2, který se vždy posouvá o 2 pixely a z každého regionu vybere nejvýraznější pixel. V důsledku aplikace filtru na každý druhý pixel se výška a šířka zmenší na polovinu a dochází tak k 75% redukci dat. Protože v této vrstvě dochází ke ztrátě dat na úkor výkonu, v některých moderních aplikacích je tato vrstva nahrazována odpovídající konvoluční vrstvou, která se naučí vhodný škálovací filtr [10].



Obr. 4: Příklad konvoluční neuronové sítě

Poslední vrstvou je vrstva klasické neuronové sítě využívající propojení každý s každým. Tato vrstva je obsažena téměř v každé architektuře a využívá se zejména na konci celé struktury pro získání finálního výstupu. Neuronová vrstva v Tensorflow:

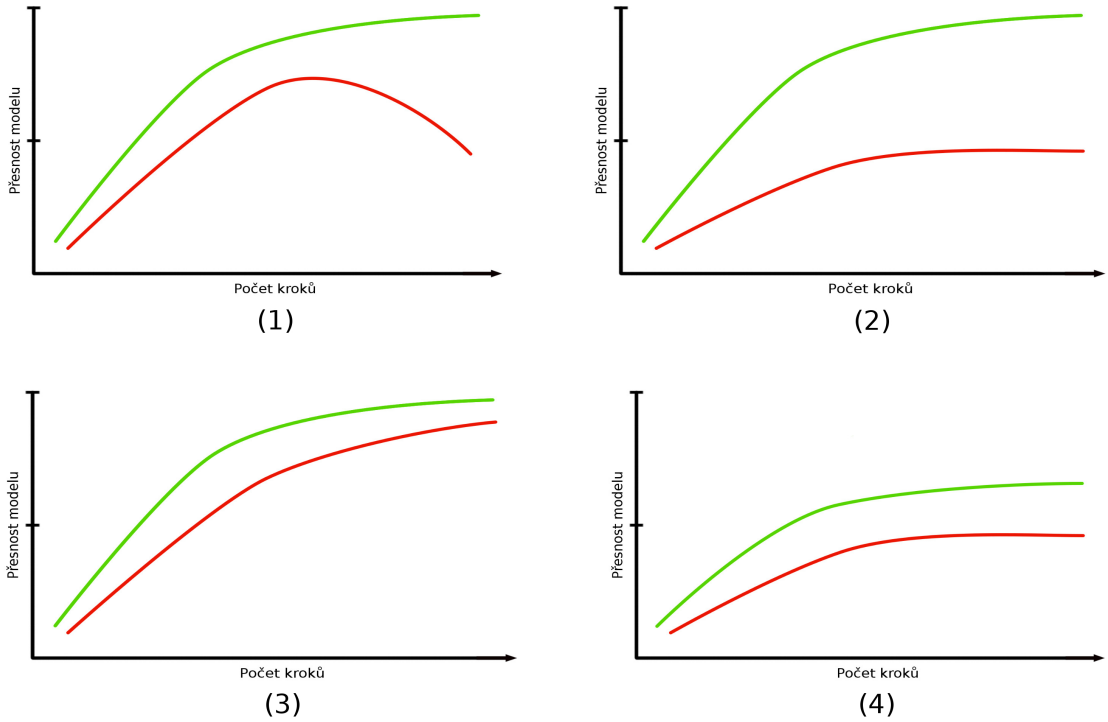
```
# Parametry W a b pro neuronovou vrstvu
# Pro 100 vstupních hodnot vytvoří 50 neuronů
W = tf.Variable(tf.truncated_normal([100, 50], stddev=0.1), name="W")
b = tf.Variable(tf.constant(0.1, shape=[50]), name="b")
# Neuronová vrstva
vystup = tf.nn.relu(tf.matmul(vstup, W) + b)
```

4.1.3 Přeučení

Se schopností vytvářet komplexní modely se objevuje problém přeučení. Přeučení nastává v momentě, kdy je model příliš optimalizovaný pro trénovací data a není schopný generalizace pro zatím neznámá data [11]. Pro detekci tohoto problém se cvičná data rozdělují na dvě resp. tři skupiny, kdy první slouží k učení a další slouží k testování. Přeučení se projeví jako výrazný rozdíl mezi přesností predikce na trénovacích a testovacích datech. Přeučení lze předcházet využitím regularizace, jednoduššího modelu nebo větší a rozmanitější kolekce dat.

V této práci je pro detekci přeučení využíváno grafu, ve kterém je na ose x počet trénovacích kroků a na ose y je přesnost predikce na trénovacích datech (zelená křivka) a testovacích datech (červená křivka). Obr. 5 značí několik typů grafů, které se mohou objevit při učení. Při dostatečném množství kroků predikční model obvykle dosahuje téměř stoprocentní úspěšnosti na trénovacích datech. V případě, že tomu tak není (typ grafu 4), model nemá dostatek

parametrů pro vytvoření komplexního modelu, a proto je nutné tento počet zvýšit např. přidáním neuronů. Pokud je počet parametrů naopak příliš velký, dochází k přeučení a graf značí velký rozdíl mezi trénovací a testovací přesností (typy 1 a 2), v extrémních případech dochází k poklesu testovací přesnosti. Ideální stav nastává, když obě křivky rostou podobným tempem a blíží se k hodnotě 100 %.



Obr. 5: Typy průběhu učení modelu (1, 2 – přeučení; 3 – ideální stav; 4 – nedostatečně komplexní model)

5 STRUKTURA PROGRAMU

Celý program je rozdělen do několika modulů podle funkce. Tyto moduly jsou vzájemně nezávislé a sdílí mezi sebou pouze přesně stanovená data. Hlavní kostru programu tvoří pět modulů (detekce stránky, detekce slov, normalizace dat, oddělení znaků, rozpoznání znaků). Moduly využívající strojového učení jsou provázány s podpůrnými moduly, které slouží k vytváření trénovacích datových sad a trénování modelu. Výhodou této struktury je vysoká flexibilita a možnost moduly nezávisle upravovat. Zároveň umožňuje měřit úspěšnost jednotlivých modulů a optimalizovat nejslabší z nich. Tato struktura ovšem ztrácí svou flexibilitu v případě změny formátu předávaných dat. Taková změna může vyžadovat změnu všech následujících modulů.

Tento program neřeší problém finálního uložení převedeného textu do odpovídajícího formátu. Pouze vytváří jednoduché rozhraní pro procházení detekovaných slov a zobrazování rozpoznaného slova. Vhodné postup pro zobrazování a ukládání není rozebírán, neboť závisí na typu textu a přesnosti modelu.

5.1 Detekce stránky

První část programu slouží k načtení fotografie listu papíru s textem a odstranění nepotřebného pozadí. Pro odstranění pozadí je využito techniky počítačového vidění, pomocí které nalezneme okraje stránky a ořízneme okolní pozadí. Toto řešení je velmi triviální a v mnohých případech nedostačující (postup vyžaduje dostatečný kontrast mezi pozadím a stránkou). V praxi se očekává, že tento modul bude nahrazen metodou optimalizovanou pro daný formát vstupního obrazu. Odstranění přebytečného pozadí je ovšem nezbytné pro zvýšení přesnosti detekce slov a výkon programu.

5.2 Detekce slov

Tento modul na vstupu dostane obraz již zbavený přebytečného pozadí a vyhledá v něm jednotlivá slova, která se dále zpracovávají. I přes to, že existují komplexní metody pro řešení podobných problémů, které mohou například využívat strojového učení (tzv. rekurentní neuronové sítě), v našem řešení bylo využito pouze metod počítačového vidění. Slova jsou vyhledávána na základě předpokladu, že v ideálním případě jsou od sebe slova oddělena mezerou a splňují velikostní kritéria. Podobně jako u detekce stránky, lze v praxi využít specifických vlastností stránky (např.: linkování) pro zlepšení tohoto postupu.

5.3 Normalizace dat

Fáze normalizace má za cíl eliminovat nežádoucí odchylky v datech a je nezbytná pro správnou činnost modulů využívajících strojového učení. Mezi takové odchylky můžeme

zařadit odlišnou barvu, velikost či sklon písma, a proto jsou v tomto modulu jednotlivé obrazy slov zmenšeny na danou velikost a je provedena korekce barev a sklonu písma. Největším problémem tohoto modulu je právě korekce sklonu, pro kterou neexistuje spolehlivé řešení a v některých případech je zcela nemožná.

5.4 Oddělení znaků

Protože se zatím nepodařilo nalézt žádný spolehlivý způsob na oddělování vázaných znaků pomocí technik počítačového vidění, muselo být využito strojového učení. Program nejprve na základě dříve klasifikovaných dat vytvoří model, pomocí kterého bude predikovat, zda je na obrázku písmeno nebo mezera mezi písmeny. Poté budeme jednotlivá slova procházet po malých výřezech a určovat zda se jedné o mezeru či nikoliv. Postupně tak rozdělíme celé slovo na jednotlivá písmena, která jsou dále zpracovávána. Tento modul je rozšířen o dva přídavné moduly, jeden pro vytvářené kolekce roztříděných dat a druhý pro trénování modelu. Nevýhodou tohoto řešení je především vysoká závislost na velikosti a kvalitě datové kolekce.

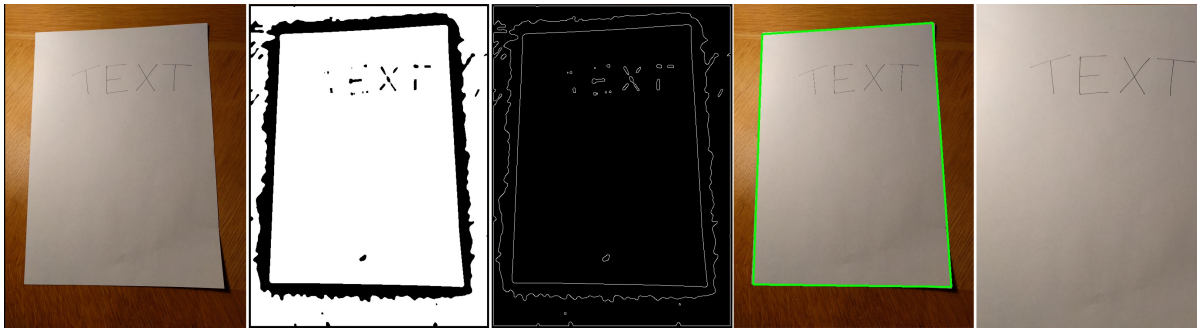
5.5 Rozpoznání znaků

Tento modul také využívá strojového učení. Nejprve jsou obrázky jednotlivých písmen převedeny na danou velikost a následně jsou klasifikovány předem natrénovaným modelem. Tento modul spolu s dvěma přídavnými moduly je velmi podobný struktuře modulu pro oddělování znaků. Na rozdíl od předchozího modulu, kde se obrázky rozdělují na dva druhy, v tomto modulu rozdělujeme obrázky na 83 druhů (znaků), popřípadě 53 znaků bez interpunkce. Z toho vyplývá, že pro vytvoření kvalitního modelu je zapotřebí několika násobně více cvičných dat. Vzhledem ke komplexnosti tohoto problému ovšem jiné řešení zatím nepřipadá v úvahu.

6 TECHNICKÉ ZPRACOVÁNÍ

6.1 Detekce stránky

Metoda pro detekci stránky vychází z článku publikovaném na pyimagesearch.com [12]. Podstatou této metody je využití jedné z funkcí knihovny OpenCV (*cv2.warpPerspective*), která nám umožňuje vyříznout obrázek na základě čtyř bodů (přesněji přesunout čtyři body na jinou čtveřici bodů). Cílem tohoto modulu je nalezení rohových bodů stránky a aplikování této metody. Předpokladem pro úspěšné fungování této metody je fotka obsahující čtyřúhelníkovou stránku zabírající většinu fotky a dostatečný kontrast mezi pozadím a stránkou.



Obr. 6: Postup detekce stránky. 1 - originální fotka; 2 – filtrace, prahování a přidání rámečku; 3 – detekované hrany; 4 – nalezená kontura; 5 – vyříznutá stránka

Pro hledání rohů stránky je využito Cannyho detekce hran a následné aplikování funkce pro hledání kontur (funkce *cv2.findContours*), kdy konturou se myslí nepřerušená uzavřená hrana. Ještě před provedením detekce hran je nutné provést odstranění šumu a rozostření obrazu, čímž se předejde detekci nevýznamných hran. Za tímto účelem je využit bilaterální filtr, který umožňuje rozostření a zároveň zachování hran [13], v kombinaci s adaptivním prahováním a přidáním rámečku pro případ, kdy se stránka dotýká okraje (funkce hledání kontur nenalezna kontury dotýkající se okraje obrazu). Po nalezení všech kontur, vybereme největší konvexní konturu, která má při jisté míře aproximace čtyři rohy (využitím funkcí OpenCV pro práci s konturami). V momentě získání kontury stránky, můžeme aplikovat na úvod zmíněnou funkci. Jednotlivé fáze jsou zachyceny na obr. 6.

Tato metoda má vysokou úspěšnost v případě, kdy fotka stránky splňuje všechny předpoklady (dostatečný kontrast s jednotvárným pozadím a dostatečná velikost stránky). V opačném případě stránka obvykle není rozpoznána a to má za následek zhoršení přesnosti následujících částí programu.

6.2 Detekce slov

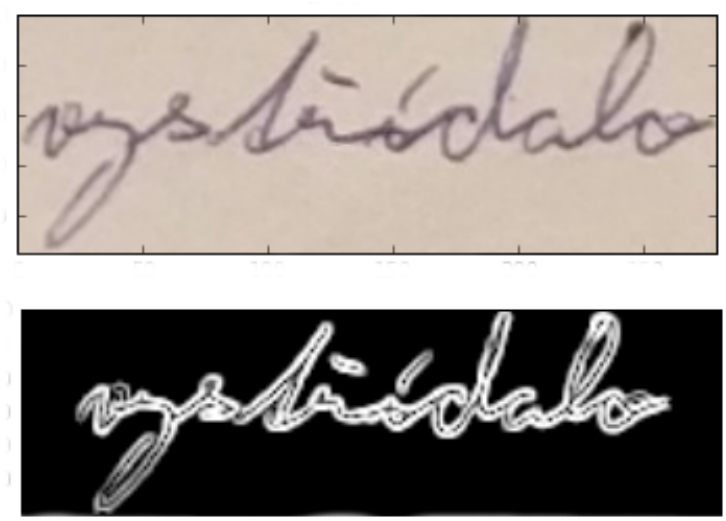
Pro detekci slov je využito experimentálně získané metody vycházející z hledání kontur a čtyřúhelníků, které je ohraničují. Pro detekci textu je využita detekce hran (pouze Sobelův operátor) v kombinaci s prahováním, které předchází rozostření pomocí Gaussianova filtru. Následuje morfologická operace uzavření, která zacelí mezery mezi bílými pixely, pokud je jejich vzdálenost menší než je definováno. Poté obraz zmenšíme a vyhledáme kontury. Při hledání kontur je využito parametru hierarchie udávající vzájemnou polohu kontur. Námi využitá hierarchie (označená *cv2.RETR_CCOMP*) rozděluje kontury na ohraničující kontury a vnitřní kontury. Dále projdeme pouze ohraničující kontury a v případě, že splňují velikostní omezení, je označíme za slova a ponecháme pro další zpracování.

Nedostatkem tohoto řešení je neschopnost oddělit slova, která se vzájemně dotýkají. Pro řešení tohoto problému byl také testován algoritmus nazvaný Watershed [14]. I přes správné oddělení některých slov, tento algoritmus nebyl využit, protože v mnohých případech výsledné ohraničení produkované tímto algoritmem nezabíralo celé slovo. Budoucím řešením tohoto problému by mohla být kombinace obou postupů.

6.3 Normalizace dat

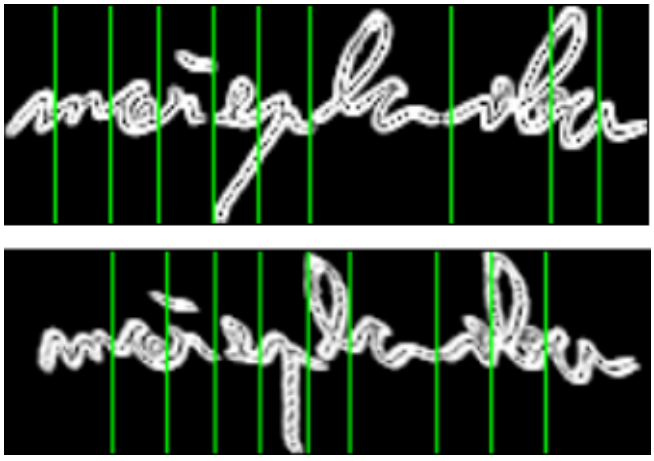
Normalizace můžeme rozdělit na normalizaci před rozdělováním slov na jednotlivé znaky a na normalizaci jednotlivých znaků. Tento proces je důležitý pro správnou funkci algoritmů strojového učení, které vyžadují standardizovaná data přesných rozměrů.

Během normalizace obrázku slova, je nejprve provedeno zmenšení na standardizovanou výšku (60 pixelů), následuje rozostření bilaterálním filtrem v kombinaci s detekcí hran Sobelovým operátorem a prahováním. Výsledkem těchto kroků je černobílý obrázek pouze s hranami jednotlivých písmen (obr. 7). Cílem tohoto postupu je odstranění závislosti na barvě písma a podkladu.



Obr. 7: Normalizace barvy slova (původní a normalizovaný obraz)

Následuje korekce náklonu písma, která je důležitá především proto, že jednotlivá písmena jsou oddělována svislou hranicí a bez korekce sklonu nelze písmena jednoznačně oddělit. Úspěšná korekce náklonu usnadňuje vytvoření datové kolekce a také zvyšuje přesnost predikčního modelu. Pro příklad je uvedeno oddělení písmen ve slově s korekcí náklonu a bez korekce (obr. 8). Náklon slova je detekován pomocí Houghovy transformace pro hledání přímek, která najde všechny přímky protínající zvolený minimální počet bílých bodů. Z těchto přímek vybereme pouze přímky s maximálním sklonem 40 úhlových stupňů od vertikální osy. Sklony přímek zprůměrujeme, a pokud se úhel nachází v daných mezích, obrázek o daný úhel nakloníme.



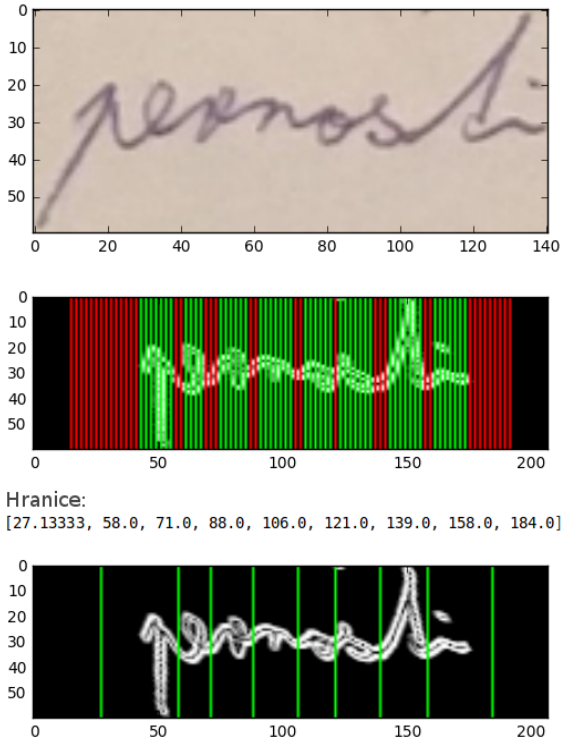
Obr. 8: Oddělení písmen ve slově bez korekce (horní) a s korekcí (spodní)

Tato korekce funguje pouze při menších úhlech náklonu a pro přesnější korekci by bylo nutné využít odlišné metody detekce náklonu. Problém této funkce nastává v případě, kdy náklon písmen není jednotný, v takovém případě by zřejmě bylo nutné rozdělit slovo na menší úseky a provést pouze lokální korekci.

Jelikož velká část normalizace proběhne během normalizace celého slova, tak pro normalizaci před rozpoznáváním jednotlivých písmen stačí pouze oříznout přebytečné volné místo a písmeno zmenšit nebo zvětšit na daný rozměr (64x64 pixelů) tak, aby byl zachován poměr stran (menší strana je rozšířena o černé pixely) a byl ponechán okraj 4 pixely. Tento postup normalizace byl převzat za světoznámé datové kolekce MNIST [15].

6.4 Oddělení znaků

Během oddělování znaků jsou jednotlivé obrazy slov procházeny po výřezech 30x60 pixelů (výška je stejná jako výška slova) a výřez se vždy posouvá o dva pixely doprava. Klasifikační model rozděluje výřezy na písmena a mezery mezi písmeny (podle umístění středu výřezu). V případě, že je detekováno více po sobě jdoucích mezer, je jako hranice využita prostřední z nich. Výsledkem této operace jsou hranice jednotlivých písmen, které umožní vyříznutí a další zpracování písmen.



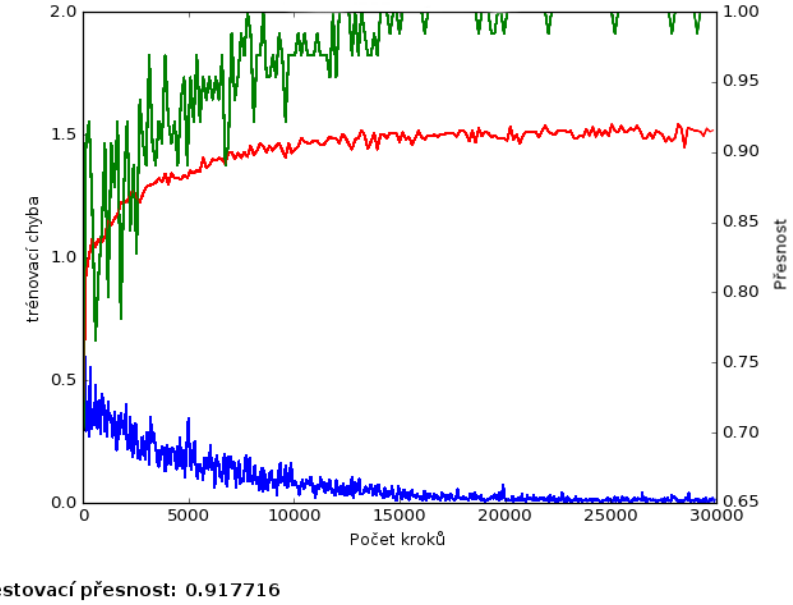
Obr. 9: Postup při oddělování písmen. 1 – originální obraz, 2 – rozdělení jednotlivých úseků (zelená=písmeno, červená=mezera), 3 – určené mezery

Správné oddělení znaků závisí na úspěšném rozpoznání písmen a mezer. Za tímto účelem byla využita konvoluční neuronová síť naprogramovaná s využitím knihovny Tensorflow. Tensorflow dále poskytuje funkce pro výpočet nepřesnosti modelu, učení modelu a jeho ukládání. To nám umožňuje vytvoření modelu na několika málo řádcích a následně model jednoduše načíst a aplikovat. Struktura této sítě je založena na modelu zvaném LeNet-5 [16], který byl vytvořen pro klasifikaci datové kolekce MNIST. Níže je uveden kód vytvářející odpovídající graf v Tensorflow.

```
# Pomocné funcke pro vytváření základních vrstev (konvoluce a škálování)
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
                          padding='SAME')

# Placeholder pro data (x) and třídy (y_)
x = tf.placeholder(tf.float32, [None, 1800], name="x")
y_ = tf.placeholder(tf.float32, [None, 2])

# Převedení dat na dvoj rozměrné pole hodnot (obrázek)
x_image = tf.reshape(x, [-1, 30, 60, 1])

# K1 - Konvoluce, 20 filtrů 5x5
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 20], stddev=0.1),
                                          name="W_conv1")
b_conv1 = tf.Variable(tf.constant(0.1, shape=[20]), name="b_conv1")
conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

# S2 - Škálování pomocí maxima
pool1 = max_pool_2x2(h_conv1)

# K3 - Konvoluce, 50 filtrů 5x5
W_conv2 = tf.Variable(tf.truncated_normal([5, 5, 20, 50], stddev=0.1),
                                          name="W_conv2")
b_conv2 = tf.Variable(tf.constant(0.1, shape=[50]), name="b_conv2")
conv2 = tf.nn.relu(conv2d(pool1, W_conv2) + b_conv2)

# S4 - Škálování pomocí maxima
pool2 = max_pool_2x2(conv2)

# N5 - Neuronová vrstva o 500 neuronech (anglicky fully connected layer)
W_fc1 = tf.Variable(tf.truncated_normal([8*15*50, 500], stddev=0.1),
                                        name="W_fc1")
b_fc1 = tf.Variable(tf.constant(0.1, shape=[500]), name="b_fc1")
# Je nutné převést příznakové mapy na jednorozměrný seznam hodnot
conv2_flat = tf.reshape(pool2, [-1, 8*15*50])
fc1 = tf.nn.relu(tf.matmul(conv2_flat, W_fc1) + b_fc1)

# 6. Dropout - regularizace vynecháváním neuronů
# pravděpodobnost vynechání uložena v keep_prob
keep_prob = tf.placeholder(tf.float32)
fc1_drop = tf.nn.dropout(fc1, keep_prob)
```

```
# 7. Výstup o 2 neuronech (Určený pro trénování)
W_fc2 = tf.Variable(tf.truncated_normal([500, 2],stddev=0.1), name="W_fc2")
b_fc2 = tf.Variable(tf.constant(0.1, shape=[2]), name="b_fc2")
y_conv = tf.matmul(fc1_drop, W_fc2) + b_fc2

# Aktivační funkce, přidání kolekce pro snadný import a export
# Určená pro reálné využití modelu
activation = tf.argmax(tf.matmul(fc1, W_fc2) + b_fc2, 1)
tf.add_to_collection("activation", activation)

# Funkce počítající chybu + regularizace
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=y_conv,
                                                        labels=y_)

beta = 0.001 # Kontrola intenzity regularizace
regularizers = tf.nn.l2_loss(W_fc2) + tf.nn.l2_loss(W_fc1)
cost = tf.reduce_mean(cross_entropy + beta * regularizers)

# Tréninkový krok, definování komplexního optimalizačního algoritmu
# a funkce, kterou minimalizuje
train_step = tf.train.AdamOptimizer(1e-4).minimize(cost)
```

Struktura nejprve provede konvoluční vrstvu (K1) obsahující 20 filtrů 5x5. Následuje vrstva škálování (S2) velikosti 2x2 využívající metody max pooling. Tato posloupnost se znovu opakuje s konvoluční vrstvou (K3) o 50 filtrech 5x5 a stejné vrstvy škálování (S4). Tyto čtyři vrstvy produkují 50 tzv. příznakových map (anglicky feature map) velikosti 8x15 pixelů. Všechny tyto hodnoty jsou dále propojeny s neuronovou vrstvou (N5) o 500 neuronech, která je napojena na výstupní vrstvu (N6) o 2 neuronech. Důvodem využití dvou výstupních neuronů je aktivační funkce softmax, která je zobecněním funkce sigmoid. Softmax vrací pro každý výstup hodnotu mezi 0 a 1 tak, že součet všech hodnot se rovná jedné a zvolená třída odpovídá výstupu s největší hodnotou. Pro regularizaci byla využito metody zvané dropout [17], která s určitou pravděpodobností při učení vynechává náhodné neurony, a regularizace L2, která k funkci počítající chybu přičítá korekci vynásobenou kontrolním parametrem *beta*.



Obr. 10: Graf úspěšnosti na tréninkových datech (zeleně) a na testovacích datech (červeně) a trénovací chyba (modře) v závislosti na počtu iterací

Datová sada v době trénování obsahovala 22 200 obrázků, z toho 20 000 bylo určeno k trénování a 2 200 k testování. Tato struktura se ukázala jako nejúspěšnější a opakovaně dosáhla úspěšnosti okolo 91 %. Na grafu je vidět poslední učení sítě, kdy bylo dosaženo úspěšnosti 91,77 %. Rozdíl mezi testovací a trénovací úspěšností si lze vysvětlit tak, že v některých případech nelze jednoznačně určit začátek a konec písmena a datová sada není dostatečně rozsáhlá a rozmanitá pro pokrytí všech možných případů. Situaci lze označit za jistou míru přeučení a jako jediné řešení se ukazuje rozšíření datové sady, protože všechna ostatní opatření (vyšší regularizace, jednoduší model) vedla ke zhoršení úspěšnosti na testovacích datech.

6.5 Rozpoznání znaků

Ještě před aplikací modelu pro predikci písmen jsou detekovaná písmena převedena na společnou velikost 64x64. Následně je z jednotlivých predikovaných písmen složeno celé slovo. Za účelem testování byli vytvořeny dvě datové sady jedna pouze pro anglickou abecedu (26 znaků, 52 včetně velkých písmen) a druhá pro českou abecedu bez CH včetně háčků a čárek (41 znaků, 82 včetně velkých písmen). Pro případ falešné detekce (oblasti neobsahující žádné písmeno) byla přidána jedna speciální třída ke každé sadě.

Pro klasifikaci je opět využita konvoluční neuronová síť, která se svou strukturou podobá struktuře pro oddělení znaků, ale je rozšířena o jednu konvoluční vrstvu s vrstvou škálování. Struktura se tedy skládá z vrstvy K1 (konvoluční vrstva) z 10 filtrů 5x5 následované S2 (max pooling) 2x2. Dále je vrstva K3 obsahující 20 filtrů o velikosti 5x5 a S4 velikosti 2x2. Nová vrstva K5 se skládá z 40 filtrů 3x3 a škálováním 2x2 (S6). Po provedení těchto vrstev získáme

40 příznakových map velikosti 8x8, které jsou napojeny na vrstvu N7 skládající se z 80 neuronů napojených na 83 (popřípadě 53 pro anglickou abecedu) výstupních neuronů. Regularizaci provádí funkce dropout a L2.

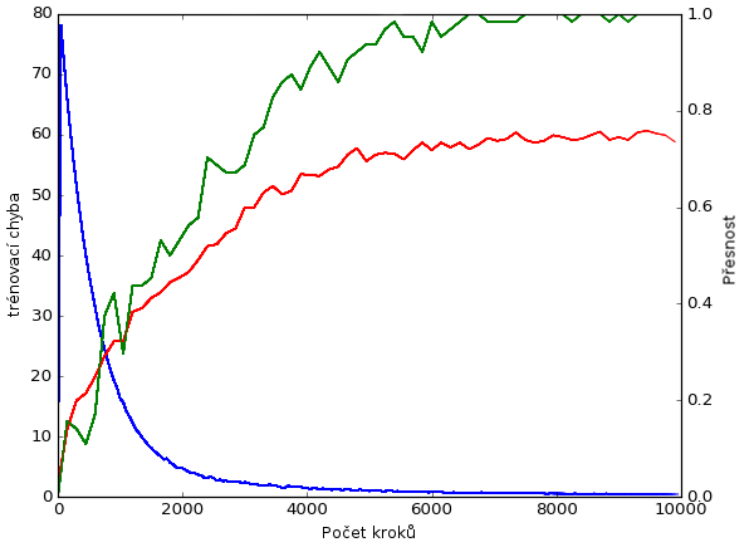
Síť byla trénována na datové sadě obsahující 2 600 příkladů, z toho 2 000 určených pro učení a 600 pro testování. Vzhledem k malému počtu dat, se často objevoval problém přeučení, a proto musely být testovány různé struktury a velikosti korekce L2. V závěru byla vybrána výše uvedená struktura spolu s korekcí L2 vynásobenou parametrem *beta* o velikosti 0.1 (obvykle se využívá hodnota okolo 0.001). Struktura k-NN (anglicky k-nearest neighbors) uvedená v tabulce 1 predikuje třídu neznámého písmene pouze na základě matematické podobnosti se známými daty (KNS označuje konvoluční neuronovou síť). Úspěšnost k-NN byla brána jako hraniční a od konvolučních sítí byla očekávána vyšší přesnost predikce.

Tab. 1: Úspěšnost predikce odlišných struktur

Struktura (abeceda)	Úspěšnost predikce
k-NN (CZ)	65,58 %
KNS, beta = 0.001 (CZ)	69,24 %
KNS, beta = 0.1 (CZ)	72,66 %
k-NN (EN)	67,05 %
KNS, beta = 0.001 (EN)	71,84 %
KNS, beta = 0.1 (EN)	75,28 %

Z tabulky je patrné, že největší úspěšnosti byly dosaženy využitím vyšší regularizace. Dále je patrné, že přesnost detekce je vždy vyšší pro anglickou abecedu než pro českou, především proto, že anglická abeceda obsahuje méně tříd pouze 53 na rozdíl od 83. Úspěšnost by jsme mohli zvýšit také tím, že bychom znali pozici písmena ve slově (zda se jedná o první písmeno nebo ne) a tím by jsme mohli zredukovat počet detekovaných tříd o velká písmena, která se nemohou nacházet uprostřed slova. Možností, jak zvýšit přesnost na českých znacích, by také bylo kombinované rozpoznání anglických znaků a následně českých, kdy by bylo prvně detekované písmeno změněno na české pouze v případě, že český znak je ekvivalentem anglického znaku rozšířeného o háček či čárku.

Maximální dosažená úspěšnost byla 78,02 % na anglické abecedě, běžné hodnoty se ovšem pohybují okolo 76 %, a 73 % na české abecedě. Z níže uvedeného grafu je patrné, že dochází k výraznému přeučení, především z důvodu velkého počtu tříd a nedostatečné kolekce dat. Proto jedním z jednoduchých způsobů, jak zvýšit přesnost predikce, je rozšíření datové sady. Další možností by bylo využití programu pro automatické generování nečistot a malých změn do již známých obrázků a rozšířit tak datovou sadu.



Obr. 11: Graf úspěšnosti na anglických tréninkových datech (zeleně) a na testovacích datech (červeně) a trénovací chyba (modře) v závislosti na počtu kroků

7 ZÁVĚR

I přes nedostatky v datové sadě, byl vytvořen fungující postup, který dokázal úspěšně rozpoznat 65 ± 10 % písmen z testovacího obrázku, který nikdy dříve neviděl (měřeno na 5 odlišných stránkách s textem o 200 znacích). Tento model byl trénován především na mém rukopisu, a proto je nepoužitelný pro odlišné rukopisy. Nicméně vytvoření programu pro rozpoznávání ručně psaného textu je možné a závisí především na dostatku kvalitních dat. Vytvoření takové datové sady rozhodně není nemožné, neboť již existuje datová sada zvaná NIST Special Database 19 [18], která byla vytvořena pomocí dotazníku a obsahuje přes 800 000 roztříděných znaků od 3 600 odlišných lidí. Tato datová sada se bohužel zaměřuje na odlišný styl písma.

Dalším krokem by proto mělo být vytvoření dotazníku, který by byl automaticky zpracováván a umožňoval tak rychlé vytvoření rozsáhlé datové sady. Dále by bylo možné využít již roztříděných znaků a jejich kombinováním vytvářet data pro oddělování znaků a vyhnout se tak komplikovanému vytváření dvou odlišných datových sad.

Kromě rozšiřování datové sady je možné zvyšovat přesnost i odlišnými způsoby jako například využíváním slovníků pro hledání překlepů, zlepšením metod normalizace či experimentováním s komplexnějšími predikčními modely, které pro své trénování vyžadují výkonné servery. Nabízí se také otázka, zda je tento postup optimální, protože pokud by jsme měli velmi přesnou metodu pro rozpoznávaní znaků, mohli bychom se ji pokusit aplikovat přímo na výřezy z nalezeného slova a vyhnout se tak oddělování písmen a vytváření dvou modelů a datových sad. Tato metoda nebyla rozebírána, neboť vyžaduje velmi přesné rozpoznávání znaků, kterého se nepodařilo dosáhnout.

8 POUŽITÁ LITERATURA

[1] Počítačové vidění. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-28]. Dostupné z: https://cs.wikipedia.org/wiki/Po%C4%8D%C3%Adta%C4%8Dov%C3%A9_vid%C4%9Bn%C3%AD

[2] Prahování. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-28]. Dostupné z: https://cs.wikipedia.org/wiki/Prahov%C3%A1n%C3%AD

[3] Image Thresholding. *OpenCV: Open Source Computer Vision* [online]. 2017 [cit. 2017-03-28]. Dostupné z: http://docs.opencv.org/trunk/d7/d4d/tutorial_py_thresholding.html

[4] PIKORA, Jan. *Implementace grafických filtrů pro zpracování rastrového obrazu* [online]. Brno, 2008 [cit. 2017-03-28]. Dostupné z: http://is.muni.cz/th/60754/fi_b/. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Tomáš Staudek.

[5] Sobel Derivatives. *OpenCV documentation: Open Source Computer Vision* [online]. 2017 [cit. 2017-03-28]. Dostupné z: http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html

[6] Canny Edge Detection. *OpenCV documentation: Open Source Computer Vision* [online]. 2015 [cit. 2017-03-28]. Dostupné z: http://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

[7] Strojové učení. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-28]. Dostupné z: https://cs.wikipedia.org/wiki/Strojov%C3%A9_u%C4%8Den%C3%AD

[8] Rectifier (neural networks). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-04-05]. Dostupné z: https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[9] Convolutional Neural Networks. *CS231n: Convolutional Neural Networks for Visual Recognition* [online]. 2016 [cit. 2017-03-28]. Dostupné z: http://cs231n.github.io/convolutional-networks/

[10] DOSOVITSKIY, Alexey. *Striving for Simplicity: The All Convolutional Net* [online] 2015 [cit. 2017-03-28]. Dostupné z: https://arxiv.org/abs/1412.6806

[11] Učení s učitelem. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2017-03-28]. Dostupné z: https://cs.wikipedia.org/wiki/U%C4%8Den%C3%AD_s_u%C4%8Ditelem

[12] How to Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes. *Pyimagesearch* [online]. 2014 [cit. 2017-03-28]. Dostupné z: http://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/

[13] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271), Bombay, 1998,

pp. 839-846. Dostupné z:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=710815&isnumber=15374

[14] Image Segmentation with Watershed Algorithm. *OpenCV: Open Source Computer Vision* [online]. 2016 [cit. 2017-03-28]. Dostupné z:
http://docs.opencv.org/3.2.0/d3/db4/tutorial_py_watershed.html

[15] THE MNIST DATABASE of handwritten digits. *THE MNIST DATABASE* [online]. 2012 [cit. 2017-03-28]. Dostupné z:
http://yann.lecun.com/exdb/mnist/

[16] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: *Gradient-Based Learning Applied to Document Recognition* [online]. Proceedings of the IEEE, 86(11):2278-2324, Listopad 1998 [cit. 2017-03-28]. Dostupné z:
http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[17] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* [online]. Toronto, 2014 [cit. 2017-03-28]. Dostupné z:
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

[18] NIST Special Database 19. *NIST* [online]. 2010 [cit. 2017-03-28]. Dostupné z:
https://www.nist.gov/srd/nist-special-database-19

9 SEZNAM OBRÁZKŮ A TABULEK

10 PŘÍLOHA 1: ZDROJOVÝ KÓD

Zdrojový kód lze nalézt na uložišti GitHub: https://github.com/Breta01/handwriting-ocr. Pro prohlížení doporučuji využit webové aplikace Jupyter nbviewer: http://nbviewer.jupyter.org/github/Breta01/handwriting-ocr/tree/master/.
Upozornění, kód je z praktických důvodů komentovaný anglicky.

Pro spuštění souborů je nutný program Jupyter Notebook spolu s Pythonem 3 a odpovídajícími knihovnami (Numpy, Tensorflow 1.0, OpenCV 3.2, ...) Hlavním souborem kombinujícím všechny částí je soubor **OCR.ipynb**.

Struktura zdrojového kódu:

```
1. Detekce stránky  - PageDetection.ipynb
2. Detekce slov     - WordDetection.ipynb
3. Normalizace      - ocr/normalization.py (nemá vlastní notebook)
4. Oddělení znaků   - GapClassification.ipynb
    model:          GapClassifier.ipynb
    tvorba dat:     GapClassification.ipynb
    převod na CSV:  Gap-DataCSV.ipynb
5. Rozpoznání znaků - OCR.ipynb
    model:          CharClassifier.ipynb
    tvorba dat:     CharClassificationDM.ipynb
    převod na CSV:  Char-DataCSV.ipynb

SLOŽKY:
data/   - datové sady (dostupná z:
https://drive.google.com/file/d/0Bw95a8U_pp2aakE0emZraHpHczA/view?
usp=sharing)
models/ - naučené modely
ocr/    - notebooky převedené do čistého pythonu + pomocné funkce
test/   - testovací obrázky
```