Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Semestrální práce

Automatická detekce chyb na základních deskách

Plzeň 2017 Michal Medek

Obsah

1 Úvod **2**

2 Konvoluční neuronová síť **3**
2.1 Proces učení 3

3 Popis implementace **5**
3.1 ROI Selector 6
3.1.1 Vstupy a výstupy 7
3.1.2 Popis tříd 7
3.2 CNNs 8
3.2.1 Vstupy a výstupy 8
3.2.2 Načtení trénovacích a testovacích dat 8
3.2.3 Realizace neuronové sítě 9
3.2.4 Zjištění úspěšnosti na trénovací množině 9
3.3 Recognizer 10
3.3.1 Vstupy a výstupy 10
3.3.2 Popis jednotlivých skriptů 10

4 Závěr **12**

1 Úvod

Cílem práce je vytvořit program, který za pomoci konvoluční neuronové sítě, bude s co možná největší úspěšností rozhodovat zda jsou základní desky televizorů v pořádku nebo ne.

Neuronové sítě jsou v posledních letech více a více používány pro automatizování výroby. Jejich hlavní předností je schopnost učit se. Síť je schopná se v čase zdokonalovat a dosahovat tak vyšší úspěšnosti na základě předchozích špatných rozhodnutí.

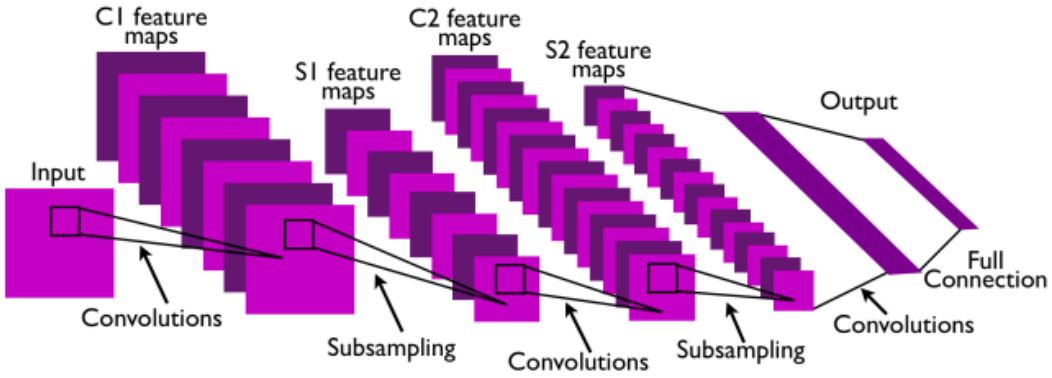
Výsledný program by v případě vysoké úspěšnosti mohl být reálně nasazen a působit tak jako takzvaný druhý čtenář pro dodatečnou kontrolu desek. Může se totiž stát, že se pracovník kvality přehlédne a nebo si chyby nevšimne. Další možné využití programu je ještě před první kontrolou zúžit množinu potenciálně vadných desek a upozornit tak pouze na tyto desky. Dostatečně zdokonalený program by v budoucnu pracovníka kvality mohl nahradit úplně.

2 Konvoluční neuronová síť

Konvoluční neuronová síť je speciální typ vícevrstvé neuronové sítě, kdy na vstupu očekáváme obrázek, který je postupně předzpracován až v samotné síti. Tato varianta neuronových sítí se používá především pro identifikaci objektů.

První tzv. konvoluční vrstva, na kterou je přiveden obrázek, slouží pro extrakci příznaků pomocí konvolučního filtru, který nazýváme recepční pole. Výstupem konvolučního filtru jsou tzv. příznakové mapy, které jsou výsledkem konvoluce určité části obrázku s recepčním polem. Druhá vrstva pak většinou slouží ke snížení počtu příznaků pomocí vzorkování. Obvykle pro vzorkování používáme metody jako max-pooling (hledání maxima) a nebo průměrování. V následujících vrstvách se pak obvykle tyto zmíněné vrstvy opakují. Z toho důvodu abychom byli schopni zachytit i složitější obrazce než jen základní hrany. Po této extrakci příznaků následuje neuronová síť tak jak ji známe z předchozí kapitoly.

Obrázek 2.1: Konvoluční neuronová síť [5]



2.1 Proces učení

Z předchozího popisu konvoluční neuronové sítě můžeme předpokládat, že za každou konvoluční vrstvou l následuje vrstva se vzorkováním $l + 1$. Proto musíme každou příznakovou mapu, před počítáním chyby zvětšit do velikosti použitého konvolučního filtru. Zvětšení probíhá tak, že pixel, který reprezentuje výsledek filtru `20x20` jednou hodnotou bude nyní představovat

matici 20x20 stejných hodnot původního výsledku (funkce g rovnice (2.1)). Následně vynásobíme zvětšenou příznakovou mapu derivací aktivační funkce konvoluční vrstvy:

$$\theta_j^l = \beta_j^{l+1}(\partial f(u_j^l) \cdot g(\theta_j^{l+1})) \tag{2.1}$$

kde β_j^{l+1} představuje váhy z navzorkované vrstvy *l+1* a funkce $f(u_j^l)$ derivaci aktivační funkce neuronu u_j^l. Tuto akci opakujeme pro každou příznakovou mapu. Poté můžeme vypočítat gradient prahu jako součet jednotlivých chyb θ_j^l:

$$\frac{\partial E}{\partial b_j} = \sum_{u,v}(\theta_j^l)_{u,v} \tag{2.2}$$

kde u, v jsou souřadnice právě počítaného elementu.

Gradient vah budeme počítat velmi podobně pomocí vztahu:

$$\frac{\partial E}{\partial k_{i,j}^l} = \sum_{u,v}(\theta_j^l)_{u,v}(p_i^{l-1})_{l-1} \tag{2.3}$$

kde (p_i^{l-1}) představuje bod v příznakové mapě.

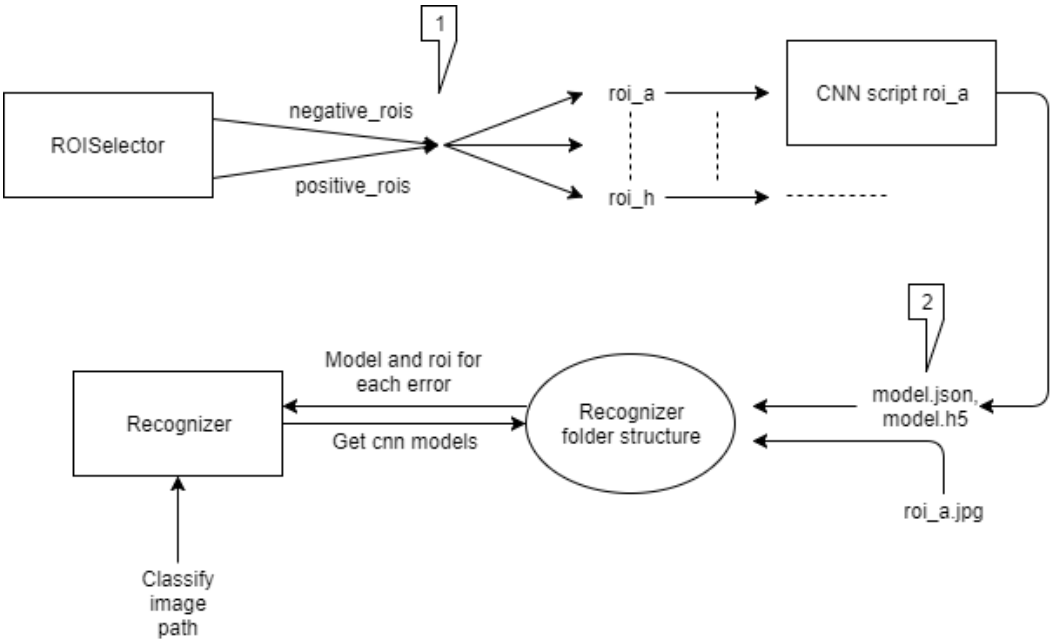
3 Popis implementace

Celá práce se skládá ze tří nezávislých programů:

- `ROI Selector` – program pro zajištění extrakce trénovacích a testovacích dat z obdržených obrázků,
- `CNNs` – jednotlivé skripty, které slouží pro natrénování a uložení konvolučních neuronových sítí,
- `Recognizer` – program, který využívá natrénované sítě a pomocí nich rozpoznává chyby na základních deskách.

Programy pak se zásahem uživatele spolupracují jak je naznačeno v následující diagramu:

Obrázek 3.1: Diagram systému



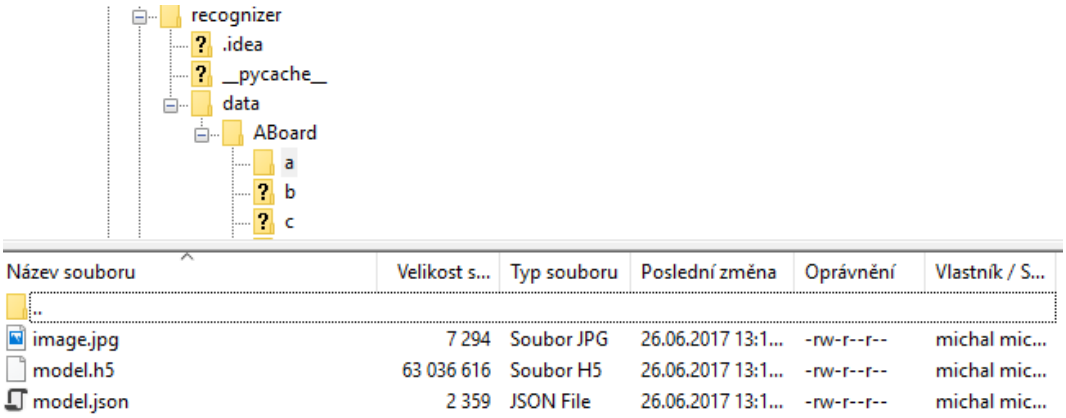
1. V prvním kroku spustíme program `ROISelector`, který vygeneruje pro každou chybu jednu složku s pozitivními ROI ve složce roi_positives

a negativními ROI ve složce roi_negatives. Tyto složky poté musí uživatel spojit a vytvořit tak pro každou chybu složku s pozitivními a negativními ROI (na Obrázku 3.1 je tato část označená číslem `1`). Taková složka pak bude vstupem pro konkrétní skript pro trénování konvoluční neuronové sítě.

2. Poté je nutné spustit jednotlivé skripty pro vygenerování jednotlivých modelů konvolučních sítí, které je pak zapotřebí překopírovat (na Obrázku 3.1 je tato část označená číslem `2`) do správné adresářové struktury (viz Obrázek 3.2).

3. Pokud máme vše takto připravené stačí spustit program `recognizer` se vstupním parametrem, kterým je jméno snímku který chceme zpracovat.

Obrázek 3.2: Adresářová struktura programu Recognizer



3.1 ROI Selector

Program je napsán v jazyce C++ za pomoci knihovny OPENCV. Slouží ke generování ROI snímků na základě svg snímků, které jsou nejprve naparsovány a následně je díky informaci o poloze chyb vybrán jeden vzor pro každou chybu. Tento vzor slouží k získání všech ostatní pozitivních a negativních snímků. Tento přístup byl použit, aby všechny ROI byly v ideálním případě stejně vycentrované.

V pozdější fázi projektu se ukázalo, že máme velmi málo pozitivních snímků a byla proto implementována základní metoda rozšíření datového

korpusu pomocí geometrických transformací. V programu používáme dva druhy geometrických transformací, kterými je shearing (zešikmení, koeficient je v intervalu od -0.1 do 0.1) a rotace (od -1 stupně do +1 stupně k původnímu natočení obrázku). Pro každé pozitivní ROI tak provedeme 24 různých pootočení (a zešikmení) obrázku + původní obrázek (s postfixem r_2_s2).

3.1.1 Vstupy a výstupy

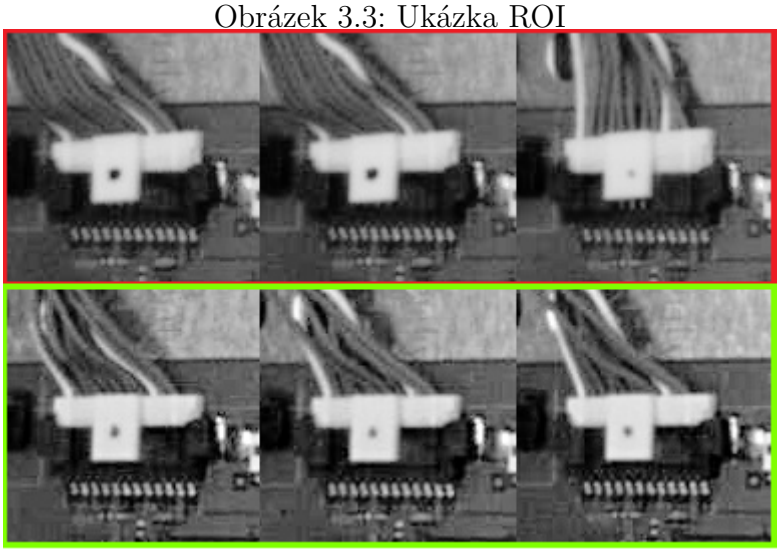
Vstupem jsou složky v adresáři `data`, který se nachází v kořenovém adresáři se zdrojovými kódy, jedná se o složku `ABoard_n` s negativními obrázky a `ABoard_p`, která obsahuje obrázky s chybou. Důležité je zmínit, že každý snímek ve formátu `jpeg` je doplněn o obrázek ve formátu `svg`, kde jsou zvýrazněné případné chyby. Tyto vstupy jsou nastaveny napevno v hlavičkovém souboru `ROISelector.h`. Výstup programu je generován do složek, které jsou (a nebo budou vytvořeny v adresáři `data`). Jedná se o složku `segmented_images`, která obsahuje vyřezané pozitivní ROI, které nejsou normalizovány pomocí metody přiřazení vzoru. Pro generování vycentrovaných ROI se poté použije vzor z této složky a jsou postupně vyřezány pozitivní ROI do složky `positive_rois` a negativní ROI do složky `negative_rois`.

3.1.2 Popis tříd

Program tvoří několik tříd, které si nyní v krátkosti přiblížíme:

- `main.cpp` – hlavní třída, která obsahuje metodu `preparePostiveROIs` a `prepareNegativeROIs`, kde se volají metody nad instancí objektu třídy `ROISelector`.
- `ROISelector.cpp` – nejdůležitější třída, která má na starosti řízení toku programu. Využívá se zde volání instance třídy `XMLParser.cpp` pro práci s svg snímky a následné vybrání ROI je realizováno pomocí knihovny `OPENCV`.
- `XMLParser.cpp` – třída pro rekurzivní naparsování `SVG` obrázku.
- `ImageRecord.cpp a Line.cpp` – slouží pouze jako přepravky, image record pro uchování informací o snímku a line pro informace o objektu čáry v `SVG` snímku

Obrázek 3.3: Ukázka ROI



3.2 CNNs

Jádro práce tvoří konvoluční neuronové sítě, které jsou implementovány pomocí knihovny Keras a Theano v programovacím jazyce Python 2.7. Vzhledem k tomu, že má výběr sítě zásadní vliv na funkčnost celého systému bude mu zde věnována největší pozornost.

3.2.1 Vstupy a výstupy

Vstupem skriptu je vždy napevno nastavená složka obsahující pozitivní a negativní ROI pro konkrétní chybu. Výstupem je pak struktura konvoluční sítě (`model.json`) a natrénované váhy (`model.h5`), které jsou vygenerovány v kořenovém adresáři skriptu.

3.2.2 Načtení trénovacích a testovacích dat

První část programu tvoří načítání obrázků do velké matice, z níž každý řádek představuje vektor reprezentující jeden ROI. Za sebou tak jsou v matici seřazené nejdříve pozitivní ROI a pak negativní. Na závěr se k matici přidá poslední sloupec, který značí do jaké skupiny trénovacích dat obrázek spadá (1 - pokud se jedná o chybu, 0 - pokud je správně). Poté se data rozdělí na trénovací a testovací množinu podle našich požadavků, prvky matice se převedou na datový typ float a matice se normalizuje vydělením 255.

3.2.3 Realizace neuronové sítě

Druhá část obsahuje vytvoření neuronové sítě, která je tvořená konvoluční vrstvou následovanou `ReLU` aktivační funkcí, poté se opakuje stejná kombinace konvoluční vrstvy a `ReLU` aktivace. Další vrstva je `max-pooling` následovaný `dropout` vrstvou, která zapříčiní náhodné prořezávání sítě, tak aby síť měla větší schopnost generalizace, proces je velmi podobný jako zanášení umělého šumu. Takto je sestavená část pro extrakci příznaku. Klasifikaci provádí plně propojená neuronová síť s `ReLU` aktivační funkcí. Poslední vrstvu tvoří dva neurony reprezentující klasifikované třídy a jako aktivační funkce je použit `softmax`, který se pro výstupní vrstvu obvykle používá.

```
model = Sequential()

model.add(Convolution2D(nb_filters, nb_conv, nb_conv,
                        border_mode='valid',
                        input_shape=(1, img_rows, img_cols)))
convout1 = Activation('relu')
model.add(convout1)
model.add(Convolution2D(nb_filters, nb_conv, nb_conv))
convout2 = Activation('relu')
model.add(convout2)
model.add(MaxPooling2D(pool_size=(nb_pool, nb_pool)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))
model.compile(loss='binary_crossentropy', optimizer='adadelta',
    metrics=["accuracy"])
```

3.2.4 Zjištění úspěšnosti na trénovací množině

Poslední část kódu pouze spustí test nad testovací množinou a vrátí celkovou úspěšnost. Následuje uložení sítě a vah.

```
score = model.evaluate(X_test, Y_test, verbose=1)
print('Test lost:', score[0])
print('Test accuracy:', score[1])
```

```
# serialize model to JSON
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model.h5")
print("Saved model to disk")
```

3.3 Recognizer

Recognizer je program pro rozpoznávání chyb na základních deskách. Na vstupu programu je obrázek ve formátu `.jpeg` tento obrázek se nejprve zařadí na základě jeho jména – které je standardizováno a můžeme tak desku zařadit dle typu. Díky tomu jsme schopni otestovat části desky pro, které byly natrénovány konvoluční sítě, protože se tam často nachází chyby. Výstupem programu je pak obrázek s potenciálně označenými chybami.

3.3.1 Vstupy a výstupy

Vstupem zadávaným z příkazové řádky je cesta ke snímku, který chceme klasifikovat. Na základě parsování jména snímku je poté zvolena správná deska a proiterován celý adresář s natrénovanými sítěmi pro jednotlivé chyby (viz Obrázek 3.2). Za předpokladu, že není nalezena žádná chyba je tato skutečnost pouze vypsána do příkazové řádky. Jinak je uživatel informován o tom, že byly nalezeny chyby a tyto chyby jsou zvýrazněny ve výstupním obrázku s názvem `error.jpeg`.

3.3.2 Popis jednotlivých skriptů

Celý program je tvořen několika skripty (jedná se o třídy):

- `run.py` – spustitelný skript, který řídí tok programu,
- `record.py` – třída sloužící jako přepravka pro záznam obrázku (obsahuje umístění obrázku, jméno komponenty, desky, apod.),
- `data_loader.py` – třída pro načítání a parsování obrázků a CNNs, které se ukládají jako instance třídy `evaluation_data.py`,

- `cnn_runner.py` – hlavní třída pro načtení neuronových sítí a zahájení evaluace, při nalezení chyby je chyba uložena jako instance třídy ze skriptu `error.py` .

4 Závěr

Do budoucna je vhodné refactorovat skripty pro vytváření konvolučních sítí, protože se jednalo o mou první zkušenost jak s knihovnou Keras tak s jazykem Python. Kód tedy není tak přehledný jak by měl být. Zároveň jsem po migraci knihovny Keras na starší verzi zjistil, že sítě konvergují mnohem pomaleji a je, zde tak prostor pro ověření skutečné funkčnosti aplikace.

Literatura

[1] Mária Krajčovičová *Konvoluční neuronová síť pro zpracování obrazu*, Diplomová práce VUTBR, 2016.

[2] Adit Deshpande *A Beginner's Guide To Understanding Convolutional Neural Networks*, 2016, https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

[3] Andrej Karpathy *Convolutional Neural Networks (CNNs / ConvNets)*, http://cs231n.github.io/convolutional-networks/ .

[4] Kamil Ekštein *Umělé neuronové sítě*, 2016

[5] Nicholas Leonard *Torch 7: Applied Deep Learning for Vision and Natural Language*, 2015

[6] The Pennsylvania State University *Neurons*, 2012

[7] Nando de Freitas *Deep Learning Lecture 10: Convolutional Neural Networks*, University of Oxford, 2015