FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Využití strojového učení a AI pro extrakci informací z webu
Student:	Bc. Martin Macho
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2020/2021

Pokyny pro vypracování

Práce je zaměřena na oblast identifikace a následné extrakce specifických informací z webových stránek, např. za účelem sledování trhu. Jedním ze směrů je strojové učení a AI využívající jako vstup grafickou podobu obsahu stránky.

- Nastudujte postupy a metody pro získávání obsahu a extrakci informací z webových stránek (zaměřte se minimálně na produktové stránky).
- Proveďte rešerši postupů a metod pro strojové zpracování obrázků, konkrétně detekce objektů a klasifikace.
- Navrhněte způsob získání obrazové podoby webové stránky a informací o poloze prvků.
- Navrhněte model, pomocí kterého budete extrahovat informace z dat získaných výše. Při návrhu modelu využijte strojové učení nebo AI.
- Nalezněte, případně vytvořte vhodný dataset.
- Proveďte experimenty pro vyhodnocení navrženého přístupu.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 12. února 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Využití strojového učení a AI pro extrakci informací z webu

bc. Martin Macho

Katedra znalostního inženýrství
Vedoucí práce: Ing. Jaroslav Kuchař, Ph.D.

27. června 2021

Poděkování

Děkuji Ing. Jaroslavu Kuchařovi, Ph.D. za vedení mé diplomové práce, cenné rady a trpělivost. Dále děkuji společnosti Simplia, s. r. o. za poskytnutí dat pro tvorbu datasetu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 27. června 2021 . . . . . .

Odkaz na tuto práci

Macho, Martin. *Využití strojového učení a AI pro extrakci informací z webu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Tato diplomová práce se zabývá tématem extrakce strukturovaných dat z webových stránek. Jejím cílem je vytvoření modelu pro automatizovanou extrakci informací z produktových stránek internetových obchodů za pomoci strojového učení a datasetu vytvořeného v rámci této práce. Teoretická část práce se věnuje úvodu do problematiky, popisuje web mining, nástroje pro extrakci informací z webu a metody zpracování obrázků pomocí neuronových sítí. Implementační část popisuje získávání obrázků webových stránek, tvorbu datasetu a jeho využití k natrénování modelu, který identifikuje klíčové prvky na produktové stránce a extrahuje je.

Klíčová slova Extrakce informací, detekce a klasifikace objektů, Faster R-CNN, konvoluční neuronové sítě, počítačové vidění, detectron2

Abstract

This thesis deals with the topic of structured data extraction from websites. The aim of the thesis is to create a model for automated information extraction from the product pages of e-shops using machine learning and dataset created within this thesis. The theoretical part of the thesis deals with an introduction to the topic of automated data extraction, describes web mining, tools for information extraction from the website and methods of image processing using neural networks. The practical part focuses on the creation of the images of websites, the creation of a dataset and its use to train a model that identifies key elements on the product page and extracts them.

Keywords Information extraction, image detection and classification, Faster R-CNN, convolutional neural networks, computer vision, detectron2

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Internet je nekonečným zdrojem informací, který každým dnem roste a spolu s tím roste i potřeba tento zdroj informací strojově zpracovávat. Pro strojové zpracování je ale vhodnější převést data z webových stránek do formátu, se kterým se lépe pracuje. Touto problematikou se zabývá extrakce strukturovaných dat, která z textu selektuje důležité informace. Ty pak můžeme využít mnoha způsoby, například srovnávat nabídky z několika různých internetových obchodů.

Cílem této práce je vytvoření modelu, který bude schopný automatizovaně extrahovat informace z produktových stránek. Využíváme k tomu metod strojového učení a datasetu, který v rámci práce vytvoříme.

V prvních třech kapitolách se věnujeme teoretickému úvodu do problematiky. Popíšeme si, co je web mining a jaké úlohy řeší. V druhé kapitole se seznámíme se způsoby, jakými můžeme webové stránky stahovat a podíváme se na existující nástroje pro extrakci informací z webu. Ve třetí kapitole rozebíráme metody pro zpracování obrázků pomocí neuronových sítí. Zaměřujeme se především na současná řešení problému detekce a klasifikace objektů, které využijeme pro návrh našeho modelu.

Čtvrtá kapitola se věnuje implementační části. Popisujeme, jaké nástrahy jsme museli řešit při tvorbě obrázků webových stránek a jakým způsobem získat seznam prvků zobrazených na stránce. Dále se v této kapitole zabýváme tvorbou datasetu, včetně možností automatického vygenerování datasetu s využitím seznamu prvků, který jsme získali v předchozím kroku. Tento dataset poté využíváme k natrénování modelu, který pomocí detekce a klasifikace objektů identifikuje klíčové prvky na stránce, a následně navrhneme algoritmus pro propojení detekovanách prvků s HTML elementy. V poslední kapitole se věnujeme analýze výsledků námi navrhnutého modelu.

Kapitola 1

Dolování dat z webu

Internet je největší databáze obsahu na světě. Můžeme zde nalézt informace prakticky o jakémkoliv tématu, navíc v mnoha formách. Existují jednoduché webové stránky, které slouží jen pro konzumaci obsahu, ale můžeme narazit i na komplexní webové aplikace, jako jsou online obchody nebo sociální sítě. Zároveň se obsah na internetu neustále mění a narůstá na objemu. Díky tomu vznikají obory, které se zabývají zpracováváním informací dostupných na internetu. Jedním z nich je web mining (překládáno jako *dolování dat z webu*), kterému se v této práci budeme věnovat.

V této kapitole si vysvětlíme, co pojem web mining znamená. Podíváme se na jeho možná využití a představíme si přístupy, které můžeme využít k extrakci informací z webových stránek.

1.1 Data mining

Data mining (česky *dolování dat*), nebo také *knowledge discovery in data* (KDD), můžeme definovat jako proces analýzy dat, kdy hledáme vztahy a závislosti mezi daty, které nejsou na první pohled patrné.[14] Cílem této analýzy je přinést vlastníkovi dat užitečný a srozumitelný pohled na vědomosti (anglicky *knowledge*) uložené v datech. Jako příklad vědomostí získaných z dat můžeme uvést periodu opakování (v případě, že analyzujeme časové řady), nebo nalezení shluků. My se v této práci zaměřujeme na web mining, který má oproti dolování dat některá specifika.

1.1.1 Web mining

Web mining je jednou z podoblastí dolování dat. Oproti data miningu se web mining liší ve formě vstupních dat. V případě data miningu jsou většinou vstupní data uložená v relační databázi[15], kdežto v případě web miningu jsou vstupními daty webové stránky, jejichž zpracování má svá úskalí, která musíme mít na paměti, například:

- Objem obsahu na webu je obrovský a neustále roste.
- Struktura webových stránek je velmi různorodá. Stejné informace mohou mít ve zdrojových kódech úplně odlišnou strukturu.
- Podoba stránek se může kdykoliv změnit. Nástroje a modely, které fungují dnes, mohou zítra přestat fungovat, pokud se autor rozhodne cokoliv na webu upravit.
- Informace na webu obsahují spoustu šumu. Typicky se jedná o různé části stránky, které pomáhají s orientací na dané stránce (nám při web miningu mohou naopak práci komplikovat), ale může jít například i o všudypřítomné reklamy.
- Stránky jsou mezi sebou provázány hypertextovými odkazy, čehož můžeme využít ke zkoumání toho, jak jsou spolu webové stránky propojeny.

Konkrétní problémy, které musíme vyřešit, se liší podle toho, jakou úlohu web miningu řešíme. Tyto úlohy můžeme rozdělit do tří hlavních oblastí: Web Structure Mining, Web Content Mining a Web Usage Mining.

1.1.1.1 Web Structure Mining

Web structure mining (česky *dolování struktury weby*) hledá užitečné informace pomocí hypertextových odkazů, které reprezentují web. Z odkazů můžeme vytvořit graf, na kterém lze následně zkoumat různé vlastnosti. To je užitečné v případě, kdy zkoumáme relevanci stránek. Weby, na které vede mnoho odkazů můžeme považovat za důvěryhodnější, což je klíčové pro internetové vyhledávače.

1.1.1.2 Web Usage Mining

Web usage mining (česky *dolování užívání webu*) hledá vzorce chování uživatelů. Tento proces pracuje se záznamem událostí, které uživatel na webu udělal. Do záznamu událostí můžeme ukládat informace o tom, kam uživatel klikl, ale také akce, které provedl (například vložení produktu do košíku nebo ohodnocení filmu ve filmové databázi). Nasbírané události pak můžeme využít pro predikci budoucích kroků nebo například k doporučování dalšího obsahu. Další z možných využití je hledání uživatelů, kteří sdílejí stejné zájmy.

1.1.1.3 Web Content Mining

Web content mining (česky *dolování obsahu webu*) se zabývá hledáním užitečných informací v obsahu webových stránek. Můžeme sem zařadit všechny úlohy, které pracují s obsahem stránek, jako například klasifikace a shlukování stránek podle tématu, segmentace stránek a hledání nerelevantních částí

(různé navigační prvky, reklamy, atd.), nebo analýza sentimentu u recenzí. Dalším tématem, kterým se web content mining zabývá, je extrakce strukturovaných informací (dat). Extrahovat lze například zprávy ze zpravodajských stránek (můžeme získávat nadpisy zpráv, perexy, nebo celé texty článků včetně obrázků a videí). Dalším dobrým příkladem je i extrakce produktových dat z internetových obchodů, kde můžeme získávat mimo jiné názvy a popisky produktů, ceny, ale i skladové zásoby nebo obrázky. Právě tématu extrakce produktových dat ze stránek internetových obchodů se budeme v této práci věnovat.

1.2 Extrakce strukturovaných dat z webových stránek

Strukturovaná data jsou taková data, která ke každé své hodnotě mají přiřazený klíč, jenž nějakým způsobem popisuje, co tyto hodnoty znázorňují nebo jak se používají (v problematice strojového učení se používá termín *feature*). Typickým příkladem strukturovaných dat jsou tabulky v relačních databázích, kde máme sloupce, které představují naše klíče (každý sloupec má svůj název a datový typ).

Oproti tomu **nestrukturovaná data** jsou data, kde přiřazení klíčů k hodnotám nemáme k dispozici. Tato data jsou většinou textově orientovaná, kde text obsahuje další informace, které je možné dále zpracovávat, jako například telefonní čísla, emailové adresy nebo poštovní směrovací číslo.

Extrakci strukturovaných dat pak můžeme definovat jako automatizovaný proces, kde na vstupu máme nestrukturovaná nebo semi-strukturovaná data a výstupem jsou data strukturovaná.[16]

1.2.1 Motivace k zabývání se extrakcí

Podoba webových stránek a technologie k jejich vytváření se velmi rychle vyvíjí. Před několika lety byla většina webových stránek převážně statický HTML obsah. Dnes se často setkáváme s interaktivními webovými aplikacemi, které svůj kód vytvářejí až u klienta v prohlížeči. Současně s vývojem webových technologií bychom se měli věnovat i rozvoji nástrojů, které webový obsah zpracovávají.

Další z důvodů pro zkoumání extrakce strukturovaných dat je využitelnost získaných dat. Příkladem může být *režim pro čtení*, který umožňují některé prohlížeče. Tato funkce zajistí, že se uživateli ze stránky zobrazí pouze důležitý obsah a zároveň upraví formát tak, aby se lépe četl. Jako další vhodný příklad pro využití extrahovaných dat můžeme uvést projekt Hlídačshopů.cz (`https://www.hlidacshopu.cz/`), který zaznamenává cenu produktů z e-shopů a hlídá, jestli obchodníci nepodvádějí s uváděnou výší slevy například tím, že týden před inzerovanou slevou produkt zdraží z obvyklé

ceny a tím slevu uměle navýší. Dalšího způsobu využití strukturovaných si můžeme všimnout, pokud budeme vyhledávat v internetových vyhledávačích. Díky strukturovaným datům mohou vyhledávače zobrazit zajímavé informace, aniž by uživatel musel web navštívit a tento údaj na stránce hledat.

Kapitola 2

Proces extrakce strukturovaných dat z webu

V této kapitole popíšeme, jaké existují možnosti pro stahování zdrojových kódů stránek, které následně použijeme k extrakci dat. Vysvětlíme si, jaké přístupy jsou vhodné v situacích, kdy potřebujeme vytvořit kopii celého webu, nebo v případě, že potřebujeme stáhnout stránky, které ke svému vykreslení využívají javascript. Následně si popíšeme přístupy pro extrakci strukturovaných dat, kde se zaměříme na možnosti využití jednoúčelových programů. Podíváme se, jakými způsoby můžeme využít metadata dostupná na stránkách a jaké existují možnosti, když stránka metadata neposkytuje.

2.1 Získání zdrojových dat z webu

Prvním krokem při řešení extrakce strukturovaných dat je získání zdrojových dat, na kterých následně budeme extrakci provádět. Pro stahování webových stránek se používají programy zvané crawlery (některá literatura je označuje anglicky také jako *spiders* nebo *robots*). Můžeme si napsat vlastní crawler, nebo využít již existujících programů (například nástroje *HTTrack*[17] nebo *wget*[18]).

Před samotným stahováním je dobré se zamyslet, jaké stránky budeme zpracovávat a podle toho zvolit vhodný nástroj. Pokud chceme vytvářet kopie celých stránek a není potřeba mít lepší kontrolu nad tím, co a jak bude program stahovat, je doporučeno použití existujících programů. V případě, kdy bychom chtěli zpracovávat interaktivní stránky, které vykreslují své části pomocí javascriptu, musíme pro stažení zdrojového kódu použít prohlížeč a nástroje pro jeho automatizaci (tuto situaci rozebíráme v části 2.1.3). V ostatních případech si vystačíme s knihovnami pro posílání HTTP požadavků a pro zpracování HTML kódu.

2.1.1 Stažení webu pomocí programu *wget*

GNU Wget je nástroj pro stahování souborů skrze protokoly HTTP, HTTPS a FTP. Mimo stahování jednotlivých souborů obsahuje i funkce pro stahování celých webových portálů. Program se ovládá pouze pomocí vstupních parametrů (neobsahuje grafické a ani interaktivní textové rozhraní), a proto je vhodný pro automatizované činnosti.[18]

Wget je dostupný v naprosté většině linuxových distribucí, případně ho můžeme nainstalovat pomocí balíčkovacího systému dané distribuce. Příkaz pro stažení celého webu FIT ČVUT je pak velmi jednoduchý, příklad uvádíme v kódu 1. Co jednotlivé volby znamenají můžeme nalézt v manuálové stránce programu[19]:

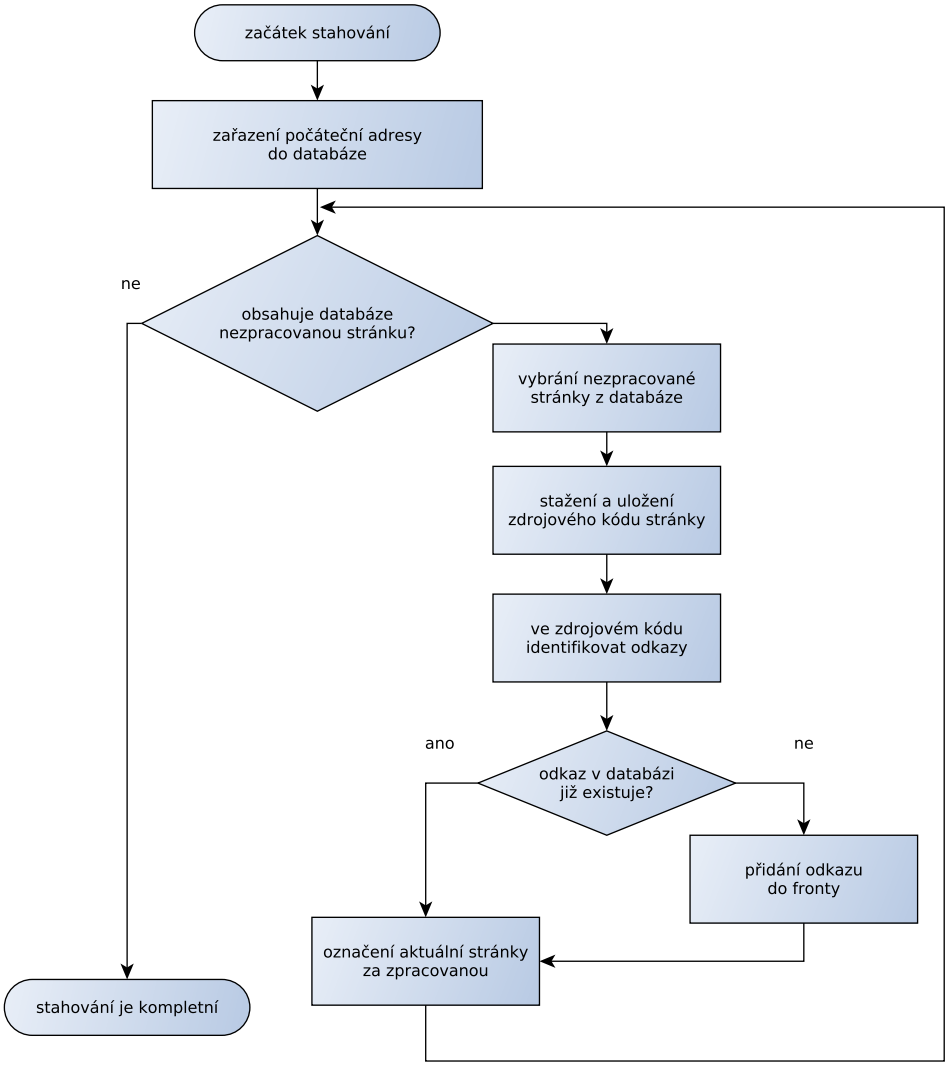
- `--mirror` zapne přepínače pro zrcadlení webových stránek – povolí rekurzivní zpracování.
- `--convert-links` překonvertuje odkazy tak, aby fungovalo lokální prohlížení (z absolutních odkazů udělá relativní).
- `--adjust-extension` přidá staženým souborům koncovku `.html`, pokud ji název souboru neobsahuje (v případě, že by url adresa končila například `.php`, bylo by obtížnější prohlížet soubory lokálně, protože by se otevíraly v editoru kódu místo webového prohlížeče).
- `--page-requisites` současně se zdrojovými kódy stránek stáhne i obsah, který je potřebný k zobrazení původní podoby stránky. Typicky jde o obrázky a soubory s kaskádovými styly.
- `--no-parent` omezí procházení stránek tak, abychom se nedostali nad úroveň, na které jsme začali. To je nezbytné, pokud chceme stáhnout jen část webu, která se nalézá pod určitou hierarchií.

```
wget --mirror --convert-links --adjust-extension \
--page-requisites --no-parent https://fit.cvut.cz
```

Ukázka kódu 1: Příkaz pro stažení celého webu pomocí příkazu *Wget*

2.1.2 Stažení webu pomocí crawleru

Pokud potřebujeme mít nad stahováním stránek lepší kontrolu, můžeme si napsat vlastní program. Pro vytváření crawlerů existuje nepřeberné množství knihoven a nástrojů. Často využívaný programovací jazyk v oblasti crawlování stránek je Python, ale poslouží nám jakýkoliv jazyk, pro který existují nástroje na posílání HTTP požadavků a zpracování HTML kódu.



Obrázek 2.1: Proces stažení celého webu, včetně odkazovaných stránek

Na obrázku 2.1 je zobrazen proces, jak crawler operuje. Na začátku máme úvodní URL adresy, které crawler postupně stáhne a hledá v nich odkazy na další stránky. Důležité je, abychom si ukládali již zpracované URL adresy, aby se crawler nezacyklil. Také je vhodné omezit rozsah zpracovávaných URL adres, například podle domény, aby se crawler nedostal na nerelevantní stránky. Tím zabráníme například tomu, aby se crawler nepokoušel stahovat kompletní obsah sociálních sítí. Pro představu, jak může realizace crawleru vypadat, uvádíme jednu z možných implementací v příloze C.

2.1.3 Stahování pomocí nástrojů na automatizaci webového prohlížeče

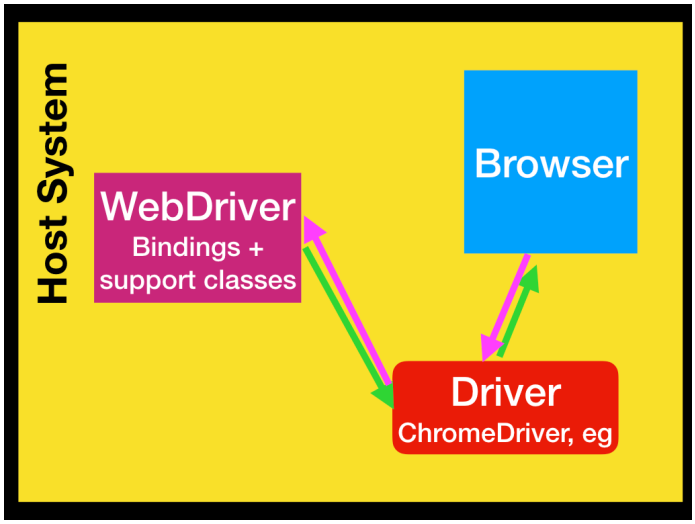
Pro stažení dat z webových stránek můžeme využít i webové prohlížeče a nástroje pro jejich automatizaci. Tento příštup má specifické využití, a to především u stránek, které využívají vykreslování pomocí javascriptu. Příkladem můžou být části stránky, které se vykreslují pomocí technologie AJAX, ale můžeme potkat i aplikace, které jsou kompletně vykreslovány pomocí javascriptu (tzv. *Single page application*). Kdybychom se tyto stránky pokoušeli stahovat pomocí přístupů zmíněných výše, nebyla by stažená data kompletní (části vykreslované javascriptem by chyběly). Ke stažení zdrojových kódů těchto stránek můžeme využít webové prohlížeče, které umí takové stránky interpretovat a správně vykreslit. Dalším vhodným příkladem k využití webového prohlížeče je situace, kdy potřebujeme pracovat s vizuální podobou stránky. Generování obrázků ze stránek lze řešit i pomocí jiných nástrojů, ale využití prohlížeče je výhodné, protože obrázky pak odpovídají tomu, co vidí uživatel. Nevýhodou tohoto přístupu je jeho rychlost – stahování stránek je oproti výše zmíněným přístupům velice pomalé, protože prohlížeč potřebuje stáhnout a zpracovat velké množství souborů, aby mohl stránku správně zobrazit.

Pro automatizované ovládání prohlížečů můžeme využít nástroje, které implementují protokol *WebDriver*[20]. Nástrojů, které implementují tento protokol existuje několik. My si zde představíme nástroj *Selenium*[21], podle kterého byl *WebDriver* protokol navržen. V první řadě musíme mít nainstalovaný webový prohlížeč, který podporuje ovládání pomocí *WebDriver* protokolu. Kompletní seznam podporovaných prohlížečů nalezneme v dokumentaci[22], ale můžeme uvést, že všechny prohlížeče s majoritním zastoupením na trhu tuto funkci plně podporují. Dále budeme potřebovat ovladač (anglicky *driver*), což je program, který zprostředkovává komunikaci mezi *Seleniem* a webovým prohlížečem. Odkaz na stažení ovladače nalezneme ve výše zmíněné dokumentaci. Poslední komponentou je *Selenium*, které komunikuje s ovladačem tak, jak je uvedeno na obrázku 2.2. V závislosti na tom, jaký programovací jazyk budeme používat, si vybereme konkrétní implementaci *Selenia* (k dispozici jsou knihovny pro programovací jazyky Javascript, Java, Python, C# a další).

Na ukázce kódu 2 uvádíme, jak může vypadat jednoduchá implementace crawleru v Node.js, která pouze stáhne zdrojový kód a vytvoří obrazovou podobu stránky. S pomocí *Selenia* ale můžeme provádět i složitější operace, jako je vyhledávání elementů na stránce, nebo spouštění vlastního javascriptového kódu. To je užitečné, pokud bychom chtěli stránku upravit, než ji budeme stahovat. Detailně se tímto přístupem zabýváme v kapitole 4.

```
const {Builder, By} = require('selenium-webdriver');
const firefox = require('selenium-webdriver/firefox');

async function download(name, url) {
  let driver = await new Builder().forBrowser('firefox')
    .setFirefoxOptions(
      new firefox.Options().setPageLoadStrategy('eager')
    ).build();

  try {
    await driver.get(url);

    // Počkáme, až se stránka načte
    await driver.wait(async function() {
      // Na stránce můžeme spoustět vlastní javascriptový kód
      let readyState = await driver.executeScript(
        'return document.readyState'
      );
      return readyState === 'complete';
    }, 20000);

    // Můžeme také získat aktuální podobu HTML kódu
    // a obrázek toho, jak je stránka vykreslená
    let element = await driver.findElement(By.css('body'));
    let image = await element.takeScreenshot();
    let html = await driver.getPageSource();

    let buff = new Buffer.from(image, 'base64');
    fs.writeFileSync(name + '.png', buff);
    fs.writeFileSync(name + '.html', html);
  } finally {
    await driver.quit();
  }
}
```

Ukázka kódu 2: Stažení zdrojového kódu stránky pomocí nástroje *Selenium*



Obrázek 2.2: Komunikace nástroje *Selenium*.[1]

2.2 Extrakce strukturovaných dat

2.2.1 Manuální extrakce

Manuální extrakce je jedna z nejjednodušších metod pro extrakci strukturovaných dat. Její princip spočívá ve vytvoření programu, který ze stránky vybere požadované informace. Prvním krokem manuální extrakce je analýza stránky, kdy na stránce hledáme informace, které budeme extrahovat. V druhém kroku napíšeme samotný program, který takto identifikované informace nalezne a extrahuje. Pro identifikaci prvků na stránce se nejčastěji využívá dotazovací jazyk *XPath* nebo *CSS* selektory.

Výsledkem je program, který lze velmi jednoduše vytvořit s pomocí nástrojů pro vývojáře, které obsahuje každý prohlížeč – na stránce nám stačí kliknout na prvek, který chceme extrahovat, a z nástroje pro vývojáře zkopírovat *CSS* selektor. Nevýhodou této metody je, že vyžaduje ruční analýzu prvků na stránce, což může být při velkém objemu zpracovávaných stránek náročné na provedení. Další nevýhodou pak je, že napsaný program je přímo závislý na struktuře HTML stránky a jakákoliv změna HTML struktury může program rozbít.

V ukázce kódu 3 uvádíme příklad možné implementace pro extrakci názvu, ceny a krátkého popisku z e-shopu alza.cz.

```python
from bs4 import BeautifulSoup
import requests

r = requests.get(
  'https://www.alza.cz/respirator-ffp2-nr-baleni-5ks-d6321852.htm',
  verify=True,
  timeout=5.000
)
soup = BeautifulSoup(r.text, 'html.parser')

exported = {
  'price': soup.select('.bigPrice.price_withVat')[0].text.strip(),
  'name': soup.select('h1')[0].text.strip(),
  'description': soup.select('.nameextc')[0].text.strip(),
}
```

```
{
  "price": "99,-",
  "name": "Respirátor FFP2 NR - 5ks",
  "description": "Respirátor s univerzální velikostí..."
}
```

Ukázka kódu 3: Příklad implementace manuální extrakce v jazyce Python s využitím knihovny *BeautifulSoup* [23].

2.2.1.1 Readability

Readability je označení programů, které mají za cíl extrahovat ze stránky relevantní textový obsah (typicky články) pomocí algoritmu *Arc90 readability*[24]. Ten funguje na principu ohodnocování HTML elementů, u kterých pak rozhoduje, jestli je daný element relevantní. Ohodnocování elementů probíhá na základě toho, jaký HTML tag byl pro daný element použit, jaký je obsah jeho atributů `id` a `class`, případně jaká je délka jeho obsahu. Tyto informace se porovnávají s hodnotami definovanými v programu, podle kterých se následně přičítá nebo odebírá ohodnocení. Na základě výsledku ohodnocení se pak rozhodne, jestli se má prvek skrýt.

Existuje několik implementací pro různé programovací jazyky, které se mohou lišit ve fungování i v definovaných hodnotách. Za pozornost stojí například implementace v javascriptu *Readability.js*[25] od společnosti Mozilla, který využívá *Readability* ve svém prohlížeči Firefox pro zobrazení v režimu pro čtení[1].

[1] V tomto režimu prohlížeč skryje různé navigační a rozptylující prvky na stránce a upraví vzhled písma tak, aby pro uživatele bylo čtení maximálně pohodlné.

Nástrojů, které se zabývají extrakcí textových částí stránek existuje několik (například *dragnet*[26] nebo *python-goose*[27]). My se v této práci zaměřujeme na extrakci produktových dat, proto se těmito nástroji nebudeme dále zabývat.

2.2.2 Extrakce s využitím metadat

Metadata jsou strojově zpracovatelné informace, které může autor umístít do svých stránek, aby usnadnil jejich zpracování roboty. Pokud webová stránka taková metadata obsahuje, můžeme jejich zpracováním získat velmi přesný nástroj pro extrakci dat. Většina stránek v dnešní době metadata v nějaké podobě již obsahuje – velký podíl na tom mají internetové vyhledávače, které díky tomu mohou svým uživatelům nabídnout lepší službu a autorům na oplátku poskytnou lepší pozice ve vyhledávání. V této části práce si popíšeme nejpoužívanější techniky, které se používají pro přidání metadat na webové stránky.[28]

2.2.2.1 Schema.org

Techniky, se kterými se seznámíme, využívají schémata, která standardizují atributy pro různé typy objektů. Schémata se často udržují v kolekcích, které se označují jako *slovníky* (anglicky *vocabulary*). Nejpoužívanějším slovníkem je projekt Schema.org[29], který je výsledkem spolupráce velkých internetových vyhledávačů (Google, Microsoft, Yahoo a Yandex). Schema.org spravuje schémata pro různé typy objektů, které můžeme nalézt na internetu (obsahuje například schémata pro knihy, události, nebo produktové informace).

2.2.2.2 Mikrodata

Mikrodata (anglicky *microdata*) jsou jedním ze způsobů, jakým může webová stránka poskytovat strukturované informace. Mikrodata se zapisují přímo k HTML elementům, kde pomocí atributů (mikrodata jsou součástí specifikace HTML5) přidávají sémantický význam:

- `itemscope` – označuje rámec, ke kterému se mikrodata vztahují. Často se užívá v kombinaci s atributem `itemtype`.
- `itemtype` – označuje typ položky ze slovníku (anglicky *item*). Zadává se celá URL adresa typu (například `https://schema.org/Product`).
- `itemprop` – označuje vlastnost, kterou daný element definuje.
- `itemid` – definuje unikátní identifikátor položky.

Na ukázce kódu 4 uvádíme příklad, jak může vypadat implementace mikrodat do webové stránky. Atributy můžeme zapsat buďto k existujícím HTML

elementům, nebo můžeme v kódu vytvořit nové elementy. Tag `meta` na řádku 15 definuje měnu nabídky (prohlížeč tento element nevykresluje). Tag `link` na řádku 18 odkazuje na konkrétní hodnotu ze slovníku.

Pro zpracování mikrodat můžeme využít existující knihovny, které převedou HTML kód do strukturovaných dat (pro Node.js můžeme využít například knihovnu microdata-node[30]).

```
1  <div>
2    <h1>Název produktu</h1>
3
4    <span>340,- Kč</span>
5    <span>Skladem</span>
6    <p>Krátký popisek produktu</p>
7  </div>
8
9  <!-- S doplněnými mikrodaty: -->
10
11 <div itemscope itemtype="https://schema.org/Product">
12   <h1 itemprop="name">Název produktu</h1>
13
14   <div itemprop="offers" itemscope itemtype="https://schema.org/Offer">
15     <meta itemprop="priceCurrency" content="CZK" />
16     <span itemprop="price" content="340.00">340,- Kč</span>
17     <span>Skladem</span>
18     <link itemprop="availability" href="https://schema.org/InStock" />
19   </div>
20   <p itemprop="description">Krátký popisek produktu</p>
21 </div>
```

Ukázka kódu 4: Ukázka použití mikrodat.

2.2.2.3 RDFa

Dalším ze způsobů publikace strukturovaných dat na webu je RDFa[31] (zkratka pro *Resource Description Framework in attributes*). Podobně jako mikrodata, i RDFa využívá k zápisu atributy. V této práci se omezíme na variantu *RDFa Lite*[32], která je minimální podmnožinou RDFa (definuje méně atributů, ale pro většinu využití je to dostatečné). *RDFa Lite* definuje následující atributy:

- `vocab` – definuje použitý slovník.
- `typeof` – označuje typ položky ze slovníku. Na rozdíl od mikrodat se v RDFa udává jen název typu, nikoliv celá URL adresa.
- `property` – označuje vlastnost, kterou daný element definuje.

- `resource` – definuje unikátní identifikátor položky.
- `prefix` – umožňuje definovat další slovníky, pokud v původním slovníku potřebné atributy chybí.

Na ukázce kódu 5 uvádíme příklad, jak může vypadat implementace RDFa Lite do webové stránky. Jak si můžeme všimnout, zápis je podobný jako v případě využití mikrodat (prakticky se liší jen v použitých atributech).

```
<div vocab="https://schema.org/" typeof="Product">
  <h1 property="name">Název produktu</h1>

  <div property="offers" typeof="Offer">
    <meta property="priceCurrency" content="CZK" />
    <span property="price" content="340.00">340,- Kč</span>
    <span>Skladem</span>
    <link property="availability" href="https://schema.org/InStock" />
  </div>
  <p property="description">Krátký popisek produktu</p>
</div>
```

Ukázka kódu 5: Ukázka použití RDFa Lite.

2.2.2.4 JSON-LD

JSON-LD[33] je zkratka pro *JavaScript Object Notation for Linked Data* a jde o další technologii, která slouží pro přidání strojově zpracovatelných informací k webové stránce. Stejně jako mikrodata využívá definované slovníky projektu Schema.org[29], ale liší se v tom, jak se do stránky zapisuje. Zatímco mikrodata se zapisují přímo k elementům v existujícím HTML kódu, JSON-LD se vkládá do hlavičky v tagu `<script>`, který obsahuje všechny informace ve formátu JSON. Výhodou této technologie je, že zpracování dat je velice jednoduché – v HTML kódu stačí nalézt odpovídající blok a zpracovat ho pomocí nástrojů pro práci s daty ve formátu JSON.

Příklad JSON-LD ve stránce uvádíme na ukázce kódu 6. Důležité jsou klíče, které začínají znakem `@`:

- `@context` uvádí, jaký slovník definic se má použít.
- `@type` označuje, jaký typ ze slovníku se má použít pro aktuální položku.

```
<script type="application/ld+json">
  {
    "@context": "http://schema.org",
    "@type": "Product",
    "name": "DJI Osmo Action",
    "description": "Outdoorová kamera senzor CMOS Sony ...",
    "image": "https://cdn.alza.cz/ImgW.ashx?fd=f5&cd=CVI549a8",
    "aggregateRating": {
      "@type": "AggregateRating",
      "ratingValue": "4.5",
      "ratingCount": "30"
    },
    "offers": {
      "@type": "Offer",
      "priceCurrency": "CZK",
      "price": "7258",
      "url": "https://www.alza.cz/dji-osmo-action-d5607533.htm",
      "itemCondition": "http://schema.org/NewCondition",
      "availability": "http://schema.org/InStock"
    },
    "brand": "DJI",
    "sku": "5607533",
    "mpn": "DJI0630",
    "gtin13": "CVI549a8"
  }
</script>
```

Ukázka kódu 6: JSON-LD data na e-shopu Alza.cz.

2.2.3 Extrakce s využitím umělé inteligence a strojového učení

Přístupy, které jsme zmínili výše, mají několik nedostatků. Manuální extrakce se nehodí na zpracování velkého množství různých stránek, nástroje na principu *readability* se zaměřují pouze na textové stránky a pokud chceme zpracovávat metadata, musí je webová stránka obsahovat. Abychom tyto nedostatky vyřešili, můžeme se pokusit data extrahovat pomocí přístupů, které využívají strojového učení. Jednou z možných oblastí strojového učení, kterou můžeme použít pro extrakci strukturovaných dat, je s pomocí strojového zpracování obrázků. Tímto přístupem se zabýváme v kapitole 3.

Kapitola 3

Strojové zpracování obrázků: detekce a klasifikace objektů

V této kapitole se seznámíme s přístupy, kterými můžeme řešit strojové zpracování obrázků. Zaměříme se na neuronové sítě, které za poslední roky dosahují velice dobrých výsledků v různých úlohách počítačového vidění, a na to, jak můžeme neuronové sítě využít k detekci a klasifikaci objektů.

3.1 Umělé neuronové sítě

3.1.1 Neuron

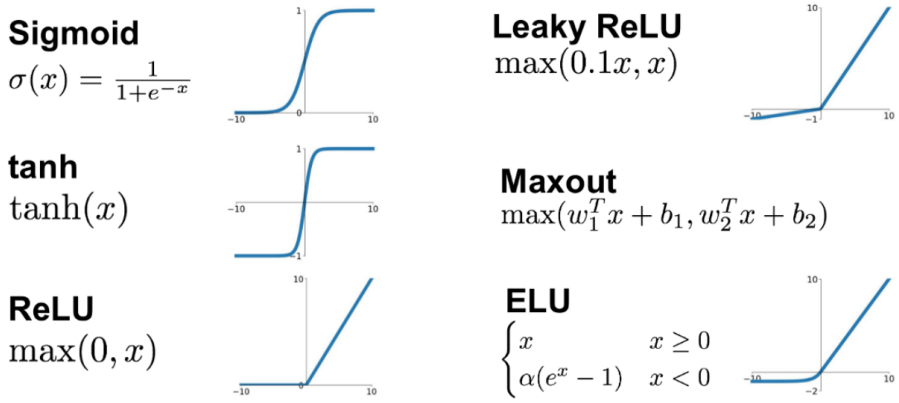
Základní komponentou umělých neuronových sítí je umělý neuron, který svojí funkčností napodobuje biologický neuron. Funkci umělého neuronu můžeme vyjádřit následujícím matematickým zápisem:

$$h(\mathbf{x}) = g\left(\sum_i \mathbf{w}_i \mathbf{x}_i + b\right) = g(\mathbf{w} \cdot \mathbf{x} + b) \tag{3.1}$$

kde $\mathbf{x}$ je vstupní vektor, $\mathbf{w}$ je vektor vah, b je bias a g je aktivační funkce neuronu.

3.1.2 Aktivační funkce

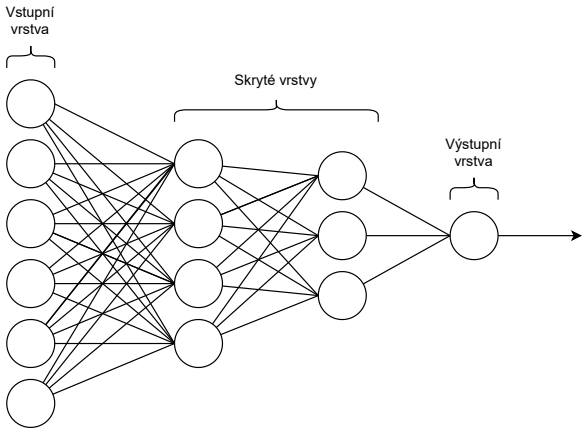
Aktivační funkce slouží k rozhodování, jestli máme neuron aktivovat a předat jeho výstupní hodnotu další vrstvě v neuronové síti. Přináší nám také možnost nelineárních transformací, což umožňuje učení komplexnějších vzorů. Nejpoužívanější aktivační funkce uvádíme na obrázku 3.1. Doporučenou aktivační funkcí pro většinu neuronových sítí je funkce ReLU, protože lze jednoduše optimalizovat pomocí metod gradientního sestupu.[34]



Obrázek 3.1: Nejpoužívanější aktivační funkce.[2]

3.1.3 Neuronová síť

Umělé neurony se nejčastěji používají tak, že se propojí několik neuronů, čímž vznikne umělá neuronová síť. Tyto propojené neurony následně můžeme uspořádat do vrstev, jak je ukázáno na obrázku 3.2. Neuronové síti, která má alespoň jednu vnitřní vrstvu, se říká vícevrstvý perceptron (MLP, z anglického *multi layer perceptron*). Pomocí neuronových sítí můžeme řešit mnoho úloh strojového učení, jako jsou klasifikace nebo regrese. Obecně platí, že čím je složitější vnitřní struktura sítě, tím složitější vzory ve vstupních datech dokáže neuronová síť rozpoznávat. Zároveň s rostoucí komplexitou struktury sítě ale roste počet parametrů, které musíme optimalizovat, což je problém například při zpracování obrázků.



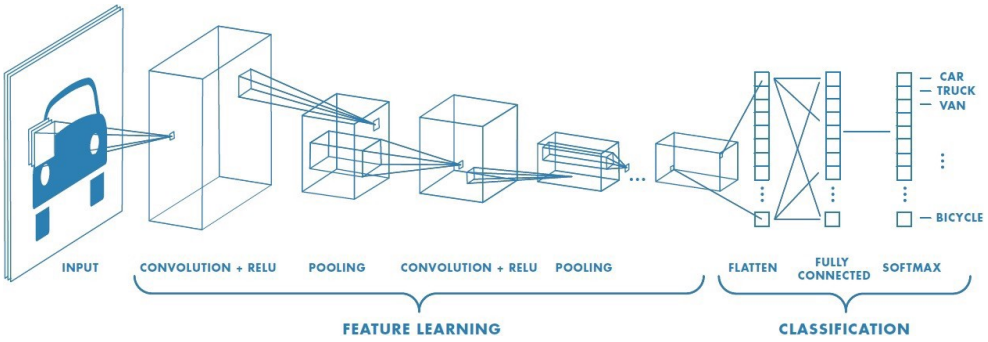
Obrázek 3.2: Vícevrstvý perceptron.

3.1.4 Konvoluční neuronové sítě

Při zpracovávání obrázků pomocí neuronových sítí je hlavní problém s množstvím dat, které by musela síť zpracovat. Pokud uvážíme obrázek o rozměrech 300 × 300 pixelů, dostaneme 270 000 vstupních parametrů (pokud uvažujeme barevný obrázek), což je příliš veliké množství. Proto se pro zpracování obrázků využívají konvoluční neuronové sítě (CNN).

Podobně jako MLP sítě, CNN jsou také tvořeny neurony, které přijímají vstupní vektor a na základě vah, biasu a aktivační funkce počítají výstupní hodnotu. Na rozdíl od MLP sítě ale konvoluční sítě předpokládají, že vstupními daty jsou obrázky (resp. data, která mají podobu mřížky), což nám umožňuje zakódovat některé vlastnosti do architektury sítě a zredukovat tím počet parametrů, čímž docílíme větší efektivity při výpočtech.[34]

Architekturu konvoluční sítě znázorňujeme na obrázku 3.3. Zpravidla se skládá ze vstupní vrstvy, několika konvolučních a pooling vrstev a nakonec plně propojenou sítí, která slouží ke klasifikaci jednotlivých tříd.

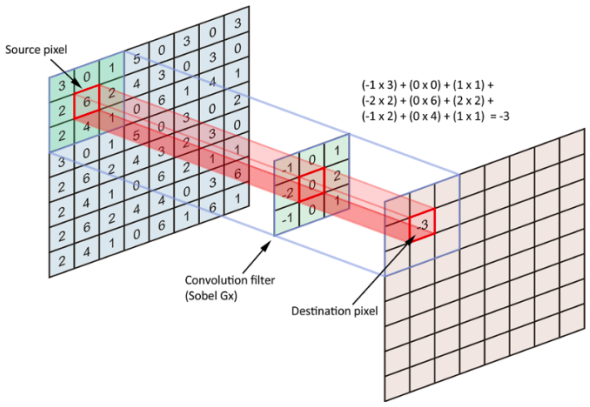


Obrázek 3.3: Architektura konvoluční neuronové sítě.[3]

3.1.4.1 Konvoluční vrstva

Konvoluční vrstvy jsou základním stavebním kamenem konvolučních neuronových sítí. Jak název napovídá, konvoluční vrstva vykonává operaci konvoluce, kterou určují tři parametry. Prvním parametrem je konvoluční jádro (anglicky *kernel*). Dalším parametrem je orámování (anglicky *padding*), které určuje, jestli a jakým způsobem ohraničíme vstupní obrázek (používá se proto, abychom byli schopní správně aplikovat konvoluci i na krajní body obrázku). Posledním parametrem je velikost kroku (anglicky *stride*), který určuje, o kolik bodů se máme po aplikaci operace konvoluce posunout.[34] Tuto konvoluční operaci aplikujeme na všechny možné pozice ve vstupním obrázku (podle parametrů popsaných výše), čímž získáme výstup konvoluční vrstvy, kterému se říká aktivační mapa (anglicky *feature map* nebo *activation map*). Výho-

dou je, že hodnoty konvolučních jáder vznikají v průběhu trénování, takže je nemusíme dopředu definovat.

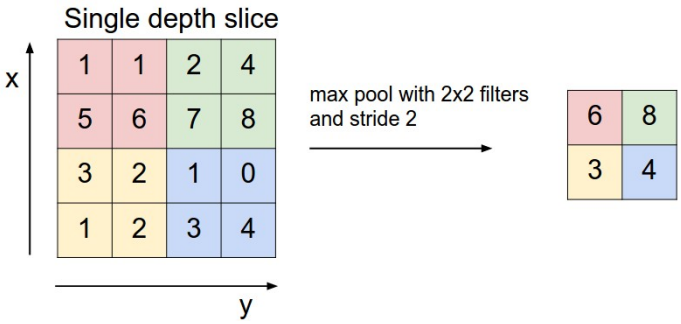


Obrázek 3.4: Ilustrace operace konvoluce.[4]

3.1.4.2 Pooling vrstva

Úkolem *pooling* vrstvy (někdy také označována jako *sub-sampling* vrstva) je postupné zmenšování prostorové reprezentace, aby se snížilo množství parametrů a výpočtů v síti. Běžně se *pooling* vrstva zapojuje za konvoluční vrstvou, jak je vidět na obrázku 3.3.[5]

Na obrázku 3.5 uvádíme příklad nejčastěji využívané formy *pooling* vrstvy v podobě *max-poolingu*, která se aplikuje podobně jako konvoluční vrstva na všechny části obrázku a vrací z nich maximum.



Obrázek 3.5: Ilustrace max poolingu.[5]

3.1.5 Ztrátová funkce

Ztrátová funkce (anglicky *loss function* nebo *cost function*) je funkce, která nám říká, jaká je vzdálenost predikce od referenčního řešení (čím větší číslo, tím je predikce horší). Ztrátové funkce můžeme rozdělit do dvou skupin podle

úloh, které řešíme. První skupina je pro klasifikační problémy (pro diskrétní hodnoty) a druhá skupina je pro regresní problémy (pro spojité hodnoty).[34]

Mezi nejčastěji používané ztrátové funkce patří například:

L2 loss pro regresi:

$$L(Y, \hat{\boldsymbol{p}}(\boldsymbol{x})) = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i(x_i) \right)^2 \tag{3.2}$$

categorical cross-entropy loss pro klasifikaci:

$$L(Y, \hat{\boldsymbol{p}}(\boldsymbol{x})) = -\sum_{j=1}^{c} \mathbb{1}_{Y=j} \log \hat{p}_j(\boldsymbol{x}) \tag{3.3}$$

kde $\hat{\boldsymbol{p}}(\boldsymbol{x}) = (\hat{p}_1(\boldsymbol{x}), \ldots, \hat{p}_c(\boldsymbol{x}))^T$ a $\mathbb{1}_{Y=j} = \begin{cases} 1, & \text{pokud } Y = j \\ 0, & \text{jinak} \end{cases}$

3.1.6 Trénování umělých neuronových sítí

Pro trénování neuronových sítí existuje několik přístupů, které se liší podle úlohy, kterou chceme řešit. Pro úlohy klasifikace a regrese se využívá přístup, který se jmenuje učení s učitelem, ke kterému potřebujeme mít data, na kterých model natrénujeme. Proces učení umělých neuronových sítí s učitelem můžeme rozdělit do několika kroků:

1. Náhodná inicializace vah sítě.
2. Klasifikace trénovacího vzorku neuronovou sítí.
3. Vyhodnocení kvality klasifikace pomocí ztrátové funkce.
4. Úprava vah sítě pomocí algoritmu zpětného šíření chyby.
5. Návrat do bodu 2, dokud nedosáhneme ukončovací podmínky (typicky konvergence nebo počet trénovacích iterací).

3.2 Detekce a klasifikace objektů pomocí umělých neuronových sítí

Detekce a klasifikace objektů je úloha, při níž se snažíme na vstupním obrázku nalézt oblasti, které obsahují hledaný objekt a následně se snažíme klasifikovat, jaký objekt oblast obsahuje. Jedná se o velice náročnou úlohu, mimo jiné protože dopředu nevíme, kolik objektů na obrázku bude zobrazených (a naším cílem je správná detekce co nejvíce těchto objektů).

Naivní přístup pro řešení problému detekce a klasifikace objektů si můžeme představit s využitím posuvného okénka (anglicky *sliding window*), kdy

toto posuvné okénko budeme posouvat a měnit mu velikost tak, aby postupně pokrylo každou možnou pozici na obrázku. Tyto pozice můžeme následně klasifikovat pomocí nástrojů pro klasifikaci obrázků. Už jen z této představy si můžeme rozmyslet, že realizace toho přístupu by byla extrémně náročná na výpočetní prostředky, protože existuje enormní množství okének, které můžeme přiřadit do obrázku.

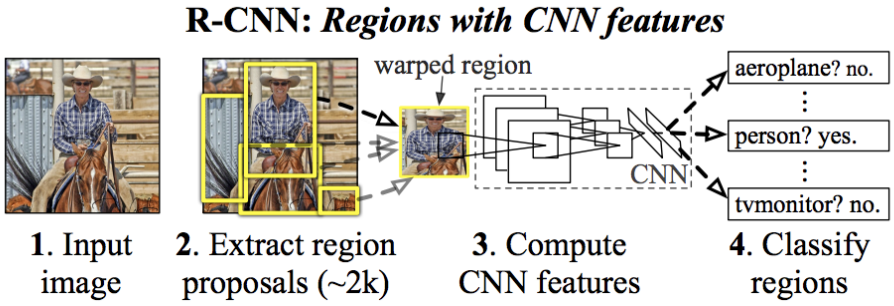
Na problém s enormním množstvím oblastí ke klasifikaci existují dvě skupiny přístupů, které se jej snaží řešit. První z nich jsou přístupy, které využívají návrhy oblastí (anglicky *region proposals*), na které aplikují neuronovou síť (síť v tomto případě nevidí celý obrázek, jen vybrané části). Tento princip využívají architektury *R-CNN*, *Fast R-CNN* a *Faster R-CNN*, které si popíšeme níže. Druhá skupina přístupů využívá pouze jednu aplikaci neuronové sítě, která zároveň navrhuje a klasifikuje oblasti. Do této rodiny patří architektury *YOLO* a *RetinaNet*.

3.2.1 Architektury neuronových sítí pro detekci a klasifikaci objektů

3.2.1.1 R-CNN

Problém s mnoha oblastmi ke klasifikaci se pokusili vyřešit autoři Ross Girshick a spol v článku [35]. Autoři navrhují systém, který obsahuje tři moduly: Generátor návrhů oblastí (anglicky *regions of interests*, *ROI*), který je nezávislý na objektech, které hledáme, konvoluční síť pro extrakci vektoru příznaků a SVM, který slouží ke klasifikaci navrhovaných oblastí (princip fungování uvádíme na obrázku 3.6). *R-CNN* byl přesnější než jiné systémy ve své době[35], ale měl řadu nevýhod[36]:

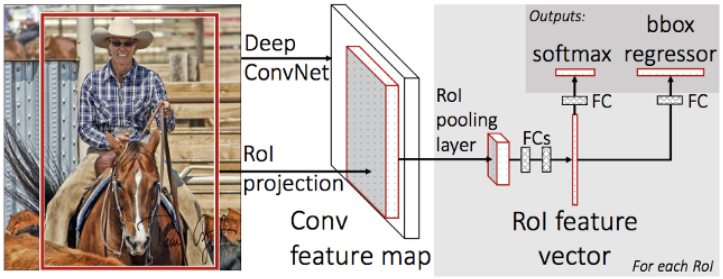
- **Vícefázové učení systému** – systém obsahuje několik modulů, které je potřeba trénovat samostatně.
- **Trénování systému je časově a prostorově náročné** – systém si ukládá mezivýstupy z konvoluční neuronové sítě na disk, což v případě hlubokých sítí může dosahovat stovky gigabajtů.
- **Detekce objektů pomocí *R-CNN* je pomalá**, protože se všechny navrhnuté oblasti zpracovávají konvoluční síťí. [36] uvádí, že detekce objektů na jednom vzorku trvala 47 sekund.



Obrázek 3.6: Princip fungování R-CNN.[6]

3.2.1.2 Fast R-CNN

Fast R-CNN je vylepšená verze systému *R-CNN*. Systém *R-CNN* nejprve navrhuje oblasti, které následně nechá zpracovat konvoluční neuronovou sítí. *Fast R-CNN* nejprve spočítá aktivační mapy pro celý vstupní obrázek, a na základě těchto aktivačních map navrhuje oblasti. Tyto oblasti pak vstupují do *ROI pooling* vrstvy, která změní jejich velikost. Nakonec jsou návrhy aplikovány do plně propojených sítí, které řeší klasifikaci objektu a finální predikci pozice oblasti. Princip fungování *Fast R-CNN* uvádíme na obázku 3.7.

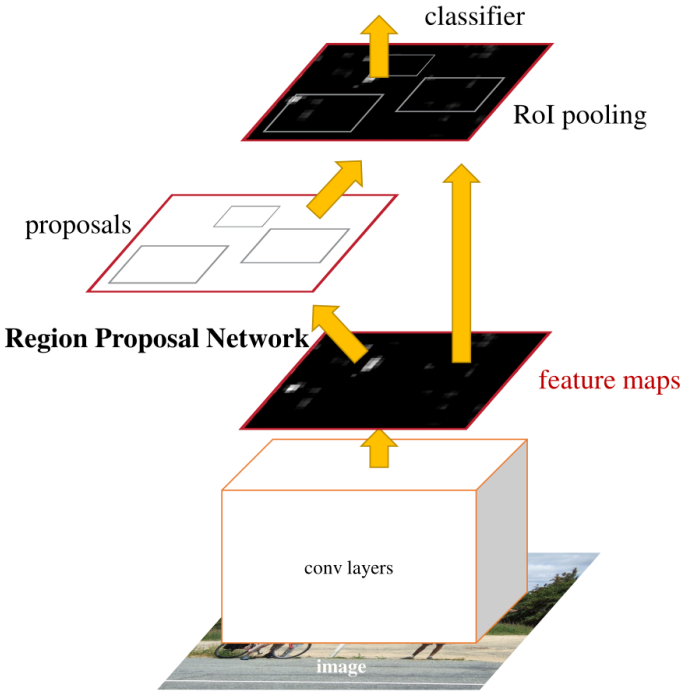


Obrázek 3.7: Princip fungování *Fast R-CNN*.[6]

3.2.1.3 Faster R-CNN

R-CNN i *Fast R-CNN* navrhují oblasti pomocí algoritmu *selective search*[37] vykonávaným na procesoru počítače, který nedosahuje takového výpočetního výkonu jako grafická karta. Znatelné je to v případě systému *Fast R-CNN*, kde je tento algoritmus úzkým hrdlem celé detekce objektů.[7] *Faster R-CNN* tento problém adresuje pomocí sítě pro návrh oblastí (anglicky *region proposal network*, RPN). Nejdříve se pro vstupní obrázek spočítají aktivační mapy

pomocí konvoluční sítě, které následně zpracovává síť pro návrh oblastí (tím se šetří čas, protože pro obrázek se počítají aktivační mapy pouze jednou). Princip fungování můžeme vidět na obrázku 3.8. RPN tyto aktivační mapy zpracovává metodou posuvného okénka, kdy pro každou pozici navrhne k různých oblastí, které se nazývají *anchor*. Pro každý *anchor* RPN predikuje skóre, které označuje pravděpodobnost, jestli daná oblast obsahuje objekt. Objektům v této oblasti je následně klasifikována třída, do které spadají.



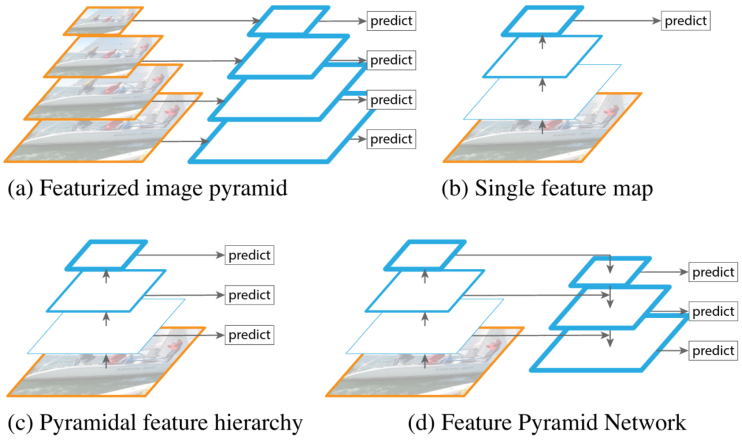
Obrázek 3.8: Princip fungování *Faster R-CNN*.[7]

3.2.1.4 Feature pyramid network

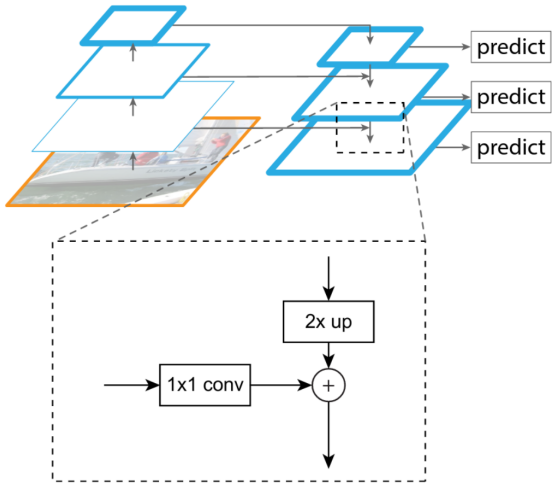
Feature pyramid network[8] (FPN) je model, který se používá k výpočtu aktivačních map (anglicky *feature extractor*). Tento model neslouží pro detekci objektů, ale využívá se ke zlepšení výsledků již existujících modelů (například ve spojení s *Faster R-CNN*).

Pro zpracování obrázků se často využívají modely, které tvoří pyramidovou architekturu, kterou můžeme vytvořit postupným převzorkováním obrázku (na obrázku 3.9 uvádíme příklady některých pyramidových architektur). Výpočet pomocí těchto pyramidových modelů můžeme rozdělit na dva průchody: Průchod zdola nahoru (anglicky *bottom-up*), kde se počítají aktivační mapy použité konvoluční sítě, a průchod shora dolů (anglicky *top-down*), kde se převzorkováním halucinují hrubší, ale sémanticky silnější aktivační mapy

z vyšších vrstev pyramidy. Nevýhodou tohoto přístupu je nepřesná poloha zobrazených objektů v poslední vrstvě pyramidy, která vznikla četným převzorkováváním vstupního obrázku. FPN proto zavádí tzv. laterální spojení, které spojují aktivační mapy z průběhu zdola nahoru s průběhem shora dolů (ilustraci laterálních spojení uvádíme na obrázku 3.10), čímž se sníží sémantika aktivačních map průchodu zdola nahoru, ale aktivace jsou mnohem přesněji lokalizovány, protože byly méněkrát převzorkovány.[8]



Obrázek 3.9: Pyramidové architektury pro zpracování obrázků.[8]

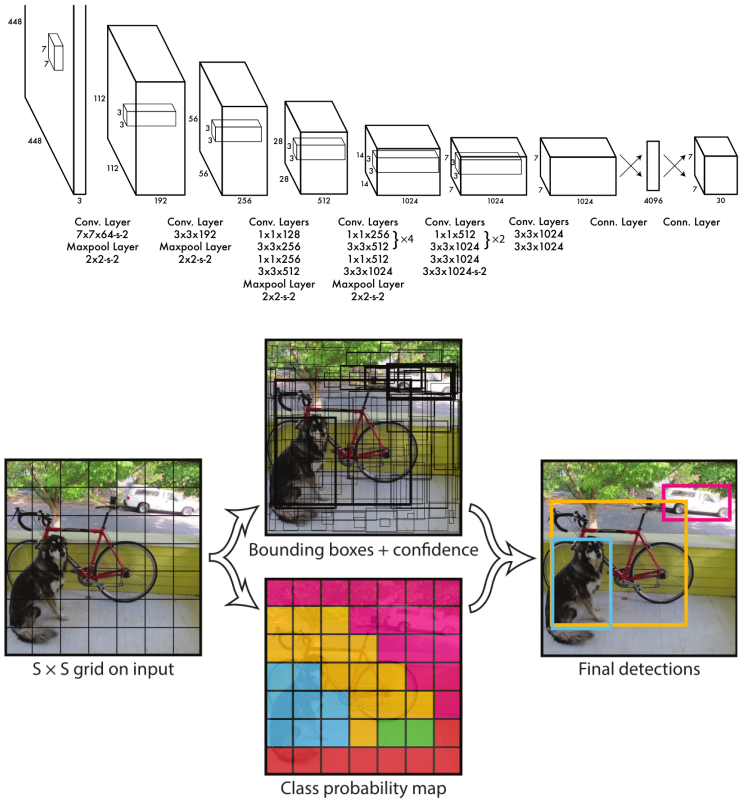


Obrázek 3.10: Laterální spojení FPN, které slučují aktivační mapy z průchodů zdola nahoru a shora dolů.[8]

3.2.1.5 YOLO

You Only Look Once[9] (YOLO) používá jiný přístup k detekci objektů, než metody, se kterými jsme se seznámili výše. Zatímco systémy *R-CNN*, *Fast R-CNN* a *Faster R-CNN* nejdříve generují návrhy oblastí, které následně dále zpracovávají, YOLO provádí detekci objektů pouze jedním zpracováním vstupního obrázku.

YOLO nejprve vstupní obrázek rozdělí do mřížky o rozměrech $S \times S$, kde každá buňka této mřížky predikuje K oblastí, ve kterých se může vyskytovat objekt. Následně se pro všechny tyto buňky predikuje, jakou třídu objektu buňka obsahuje. Dalším krokem je *non-max* suprese, která pomocí metriky *Intersection over Union* zajistí eliminaci nepotřebných oblastí. Jinými slovy algoritmus na základě IoU výpočtu odstraní z překrývajících se oblastí části s nižší pravděpodobností výskytu objektu. Architekturu a princip fungování *YOLO* uvádíme na obrázku 3.11.



Obrázek 3.11: Architektura a princip fungování YOLO.[9]

3.2.1.6 RetinaNet

RetinaNet[10] je model pro detekci objektů, který podobně jako YOLO provádí detekci pouze jedním zpracováním vstupního obrázku. Autoři *RetinaNet* identifikovali, že hlavním problémem jednoprůchodových modelů pro detekci je nerovnoměrné rozložení tříd oblastí, kde žádný objekt není (tyto oblasti jsou označované jako pozadí) a oblastí s objekty. Zároveň navrhují řešení v podobě vylepšení ztrátové funkce *cross-entropy*, zvané *Focal Loss* (FL).[10]

Pro pohodlnost zápisu nejdříve zadefinujeme proměnnou p_t, kterou použijeme k definici *Focal Loss*:

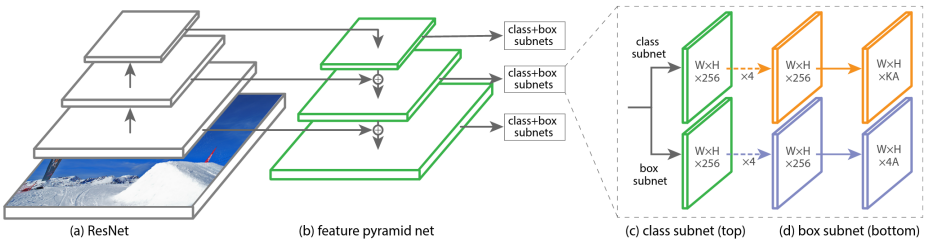
$$p_t = \begin{cases} p, & \text{pokud } y = 1 \\ 1 - p, & \text{jinak} \end{cases} \tag{3.4}$$

Definice *Focal Loss* pak vypadá následovně:

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} \log(p_t) \tag{3.5}$$

kde γ je hyperparametr modelu (v experimentech nejlépe dopadla hodnota parametru $\gamma = 2$). Pokud je vzorek chybně klasifikován a p_t nabývá malé hodnoty, výsledek ztrátové funkce se příliš nemění. Čím blíže se ale hodnota p_t blíží k 1, tím menší je faktor a výsledné hodnotě ztrátové funkce se přikládá menší váha.[10]

Architektura *RetinaNet* se skládá ze 4 komponent, které uvádíme na obrázku 3.12: Páteřní síť pro výpočet aktivačních map, FPN, neuronová síť pro klasifikaci objektů (predikuje, jestli v nalezené oblasti je nějaký objekt a jaká je jeho třída) a neuronová síť pro regresi rozměrů a polohy nalezené oblasti.



Obrázek 3.12: Architektura *RetinaNet*.[10]

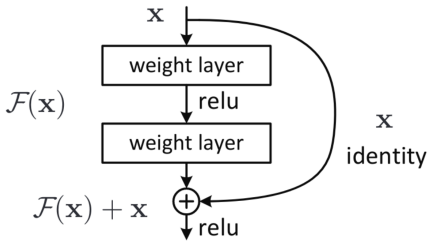
3.2.2 Architektury páteřních neuronových sítí pro modely detekce a klasifikace objektů

V předchozí části jsme popsali modely pro detekci a klasifikaci objektů. Většina z těchto modelů využívá ke svému fungování páteřní neuronovou síť, pomocí kterých vypočítá aktivační mapy a detekce objektů následně probíhá na těchto aktivačních mapách. My si popíšeme některé z architektur, které se používají ke klasifikaci obrázků, ale také se často využívají jako páteřní neuronové sítě v modelech pro detekci objektů. Zaměříme se především na architektury *ResNet* a *ResNeXt*, ale existují i další architektury, například *AlexNet*[38] nebo *DenseNet*[39].

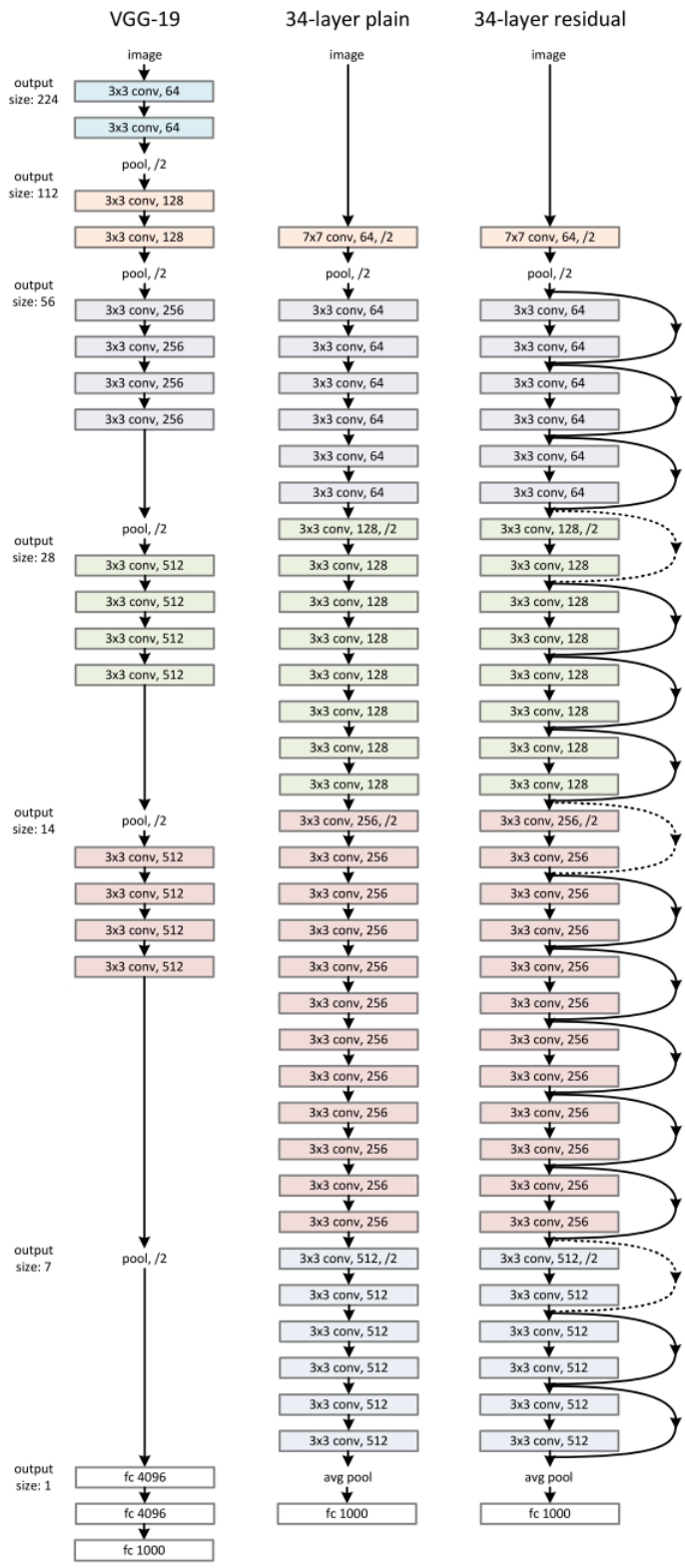
3.2.2.1 ResNet

Při konstrukci hluboké neuronové sítě vzniká problém, který se anglicky označuje *vanishing gradient problem*. Můžeme jej přeložit jako problém mizejícího gradientu: Čím více vrstev do modelu přidáme, tím blíže se gradient ztrátové funkce přibližuje nule, což ztěžuje trénování takové sítě.[11]

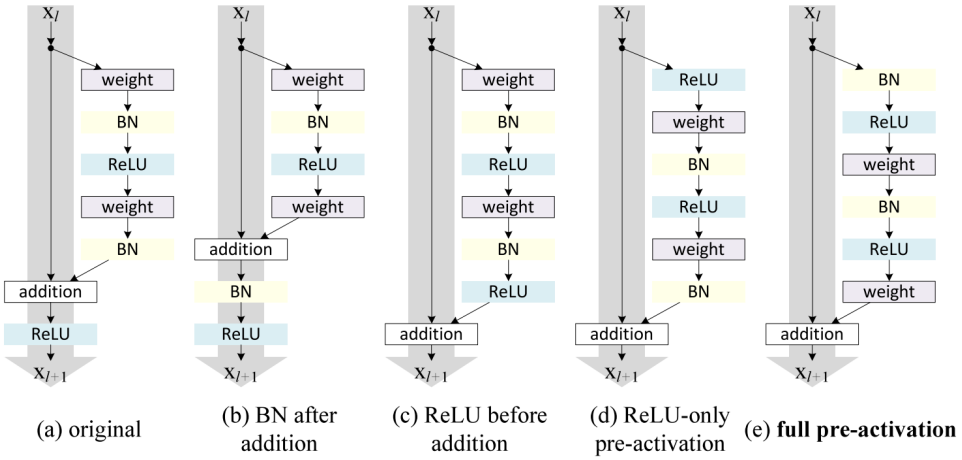
Tento problém se snaží řešit architektura s názvem *Residual network*, zkráceně *ResNet*[11]. Ta využívá tzv. *identity shortcut connection*, což je spojení, které umožňuje přeskakovat jednu nebo i více vrstev tak, jak je ukázáno na obrázcích 3.13 a 3.14. Způsobů, jak můžeme *identity shortcut connection* realizovat, existuje několik (příklady uvádíme na obrázku 3.15). Důkladně se tím zabývá článek [12].



Obrázek 3.13: *Identity shortcut connection* v síti *ResNet*.[11]



Obrázek 3.14: Architektura sítě *ResNet*.[11]

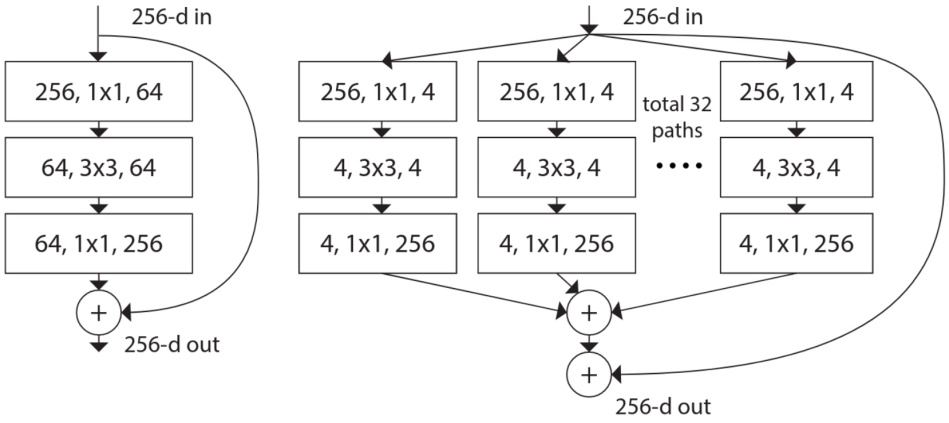


Obrázek 3.15: Varianty *identity shortcut connection* v síti *ResNet*.[12]

3.2.2.2 ResNeXt

Další architekturou, která se využívá v modelech pro zpracování obrázků je síť s názvem *ResNeXt*[13], která vychází z achitektury *ResNet*. *ResNeXt* provádí sadu transformací na částech vstupu o malých rozměrech, které následně agreguje zpět do celku o větších rozměrech (označováno jako *split-transform-merge* paradigma). Počet těchto transformací definuje parametr zvaný kardinalita (anglicky *cardinality*). Dalším parametrem, který tato architektura definuje, je tzv. šířka úzkého hrdla (anglicky *width of bottleneck*), označovaný písmenem d. Tento parametr udává, jaká bude výstupní dimenze ve vnitřních transformacích architektury.

Článek [13] uvádí, že zvyšováním kardinality lze efektivněji dosáhnout lepší přesnosti, než prohlubováním nebo rozšiřováním sítě (v porovnání s *ResNet* dosáhl *ResNeXt* lepší přesnosti, přičemž byl zhruba o polovinu jednodušší[13]). Princip *ResNeXt* architektury uvádíme na obrázku 3.16.



Obrázek 3.16: Princip *ResNeXt* architektury [13]

3.3 Vyhodnocování modelů pro detekci a klasifikaci objektů

Abychom mohli otestovat kvalitu našeho modelu, musíme si zadefinovat metriky, které se používají k vyhodnocování modelů pro detekci objektů.[40]

3.3.1 Intersection over Union

Intersection over Union (IoU), nebo také Jaccardův index, je důležitou metrikou pro vyhodnocování úloh počítačového vidění, jako je segmentace, detekce objektů nebo sledování objektů.[41]

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|I|}{|U|} \tag{3.6}$$

Kde A je predikce a B je referenční hodnota (anglicky *ground truth*, GT).

3.3.2 Chyby prvního a druhého druhu

- **Správná detekce (*true positive*, TP)** – detekce s hodnotou IOU $\geq$ hodnota prahu.
- **Špatná detekce (*false positive*, FP)** – detekce s hodnotou IOU $<$ hodnota prahu.
- **Falešne negativní (*false negative*, FN)** – referenční hodnota nebyla detekována.

- **Pravdivě negativní detekce (*true negative*, TN)** se při vyhodnocování detekce objektů nepoužívá.

Hodnota prahu (anglicky *threshold*) se obvykle nastavuje na 50 %, 75 % nebo 95 %.

3.3.3 Průměrná přesnost modelu

Přesnost modelu $Precission_{iou}$ zadefinujeme následovně (iou je hodnota prahu):

$$Precission_{iou} = \frac{TP_{iou}}{TP_{iou} + FP_{iou}} = \frac{TP_{iou}}{all\ detections} \tag{3.7}$$

Průměrná přesnost modelu AP_{iou} se pak rovná aritmetickému průměru přesností přes všechny detekované třídy:

$$AP_{iou} = \frac{1}{C} \sum_{i}^{C} Precission_{iou}(i) \tag{3.8}$$

kde C je počet detekovaných tříd.

Kapitola 4

Realizace

V této kapitole se budeme věnovat implementační části práce. V první části vybereme informace, které budeme extrahovat z produktových stránek. V druhé části se zaměříme na tvorbu datasetu. Vytvoříme nástroj pro vytváření obrazové podoby stránky a pro tyto obrázky vytvoříme anotace. Nakonec na vytvořeném datasetu natrénujeme model pro detekci objektů a změříme jeho úspěšnost.

4.1 Výběr informací, které budeme extrahovat

Před přípravou datasetu je vhodné provést analýzu, na základě které zvolíme informace, jež budeme extrahovat. Inspiraci pro volbu vhodných informací k extrakci jsme se rozhodli čerpat u produktových srovnávačů, mimo jiné proto, že jedním z potenciálních využití této práce je právě srovnání s konkurencí. Srovnávače, ze kterých jsme čerpali, jsme vybrali na základě návštěvnosti. Podle [42] jsou nejnavštěvovanější české srovnávače heureka.cz a zbozi.cz[2]. Pro analýzu využijeme specifikace pro XML feed, které jsou k dispozici na stránkách `https://sluzby.heureka.cz/napoveda/xml-feed/` a `https://napoveda.zbozi.cz/xml-feed/specifikace/`.

Do tabulek 4.1 a 4.2 jsme doplnili relevantní atributy ze specifikací pro srovnávače zbozi.cz a heureka.cz. Vyřadili jsme atributy, které jsou potřebné pouze pro fungování srovnávače nebo se nezobrazují na produktové stránce. Jde například o unikátní identifikátor produktu v rámci e-shopu nebo atributy pro úpravu zobrazení ve srovnávači, jako je cena za proklik. Na základě atributů z tabulek jsme k extrakci vybrali následující atributy: název a popis produktu, cenu včetně DPH, dostupnost a hlavní obrázek.

[2]Vycházeli jsme z celkové měsíční návštěvnosti za říjen 2020.

Název atributu	**popis atributu**	**vyžadováno**
PRODUCTNAME	název nabídky	povinné
DESCRIPTION	popis nabídky	povinné
URL	adresa nabídky v e-shopu	povinné
PRICE_VAT	cena	povinné
DELIVERY_DATE	dostupnost	povinné
IMGURL	adresa obrázku	doporučené
PRODUCT	název nabídky ve vyhledávání	doporučené
MANUFACTURER	výrobce produktu	doporučené
DELIVERY	dopravce a síť výdejních míst	doporučené
ITEM_ID	identifikátor nabídky v e-shopu	doporučené
CATEGORYTEXT	kategorie dle Zboží.cz	doporučené
EXTRA_MESSAGE	doplňkové informace o nabídce	doporučené
EAN	čárový kód	doporučené
PARAM	parametry nabídky	doporučené
EROTIC	označení erotických nabídek	doporučené
PRODUCTNO	produktový kód výrobce	doporučené
ISBN	identifikační číslo knihy	doporučené
ITEMGROUP_ID	označení skupiny nabídek	doporučené
BRAND	značka produktu	doplňkové

Tabulka 4.1: Specifikace XML feedu pro srovnávač zbozi.cz

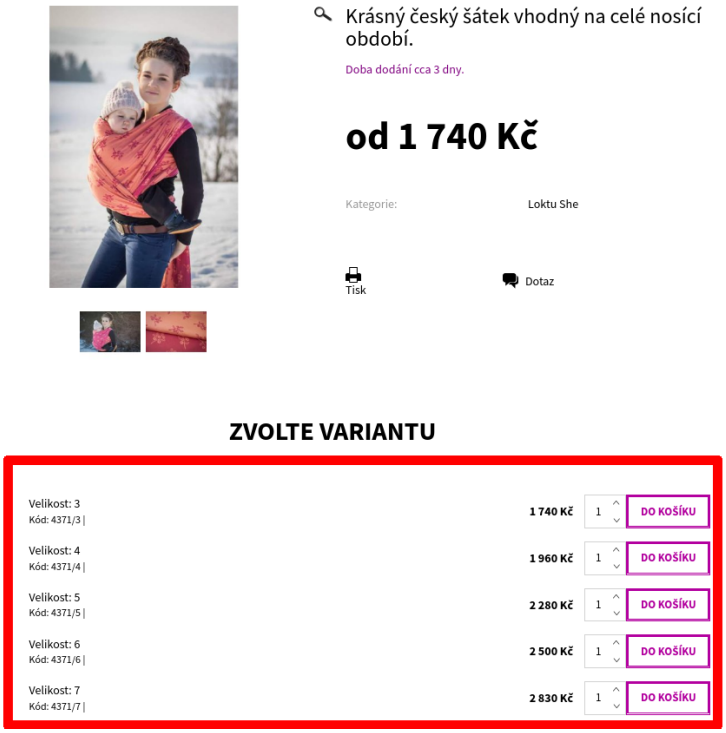
Název atributu	**popis atributu**
PRODUCTNAME	název produktu
PRODUCT	název produktu
DESCRIPTION	popis produktu
URL	adresa produktu
IMGURL	adresa obrázku
IMGURL_ALTERNATIVE	adresa dalšího obrázku produktu
VIDEO_URL	adresa videa produktu
PRICE_VAT	prodejní cena s DPH
MANUFACTURER	výrobce
CATEGORYTEXT	kategorie produktu
EAN	EAN kód produktu
PARAM	parametry produktu
DELIVERY_DATE	skladovost produktu

Tabulka 4.2: Specifikace XML feedu pro srovnávač heureka.cz

Při anotování datasetu jsme narazili na několik situací, nad kterými jsme při analýze neuvažovali:

- **Chybějící informace:** Některé e-shopy nezobrazují všechny informace, které jsme si dali za cíl extrahovat. V případě, že bude chybět informace o dostupnosti produktu, budeme tento produkt považovat za dostupný (tzn. že jej e-shop má skladem). V případě, že bude chybět cena produktu, budeme naopak produkt považovat za nedostupný. V ostatních případech budeme chybějící údaje ignorovat.
- **Krátký a dlouhý popis produktu:** Velmi často jsme se setkali s tím, že e-shop rozděluje popis produktu na krátký a dlouhý popis. Z toho důvodu jsme popis produktu v extrahovaných atributech rozdělili na dlouhý a krátký popis.
- **Varianty produktů:** Některé e-shopy zobrazují v detailu produktu varianty, které je možné zakoupit. Typickým příkladem jsou různé velikosti produktu, kdy se varianty liší názvem, cenou nebo dostupností (příklad uvádíme na obrázku 4.1). Do extrahovaných informací jsme tedy varianty přidali.
- **Drobečková navigace:** V prvotní analýze jsme s kategoriemi produktů nepočítali, protože informace o kategorii produktu nebývá na produktové stránce uvedená. Velká část e-shopů ale zobrazuje drobečkovou navigaci, ve které jsou uvedené kategorie produktu, tak jsme se rozhodli této informace využít a přidat jí do extrahovaných informací.

Finální podobu informací, které se budeme snažit ze stránek extrahovat uvádíme na ukázce kódu 7.



Obrázek 4.1: Příklad zobrazení produktu, který má více variant. Sekci s variantami jsme zvýraznili červeně.

```
{
  "name": "Název produktu",
  "availability": "Skladem",
  "price": "Cena s DPH",
  "shortDescription": "Krátký popisek",
  "longDescription": "Dlouhý popisek",
  "category": [
    "kategorie produktu",
    "podkategorie produktu"
  ],
  "variants": [
    {
      "name": "Název varianty",
      "price": "Cena varianty s DPH",
      "availability": "Dostupnost varianty"
    }
  ]
}
```

Ukázka kódu 7: Extrahovaná data

4.2 Tvorba obrázků webových stránek

Abychom mohli vytvořit dataset, potřebujeme z webových stránek vytvořit obrázky, na kterých pak natrénujeme náš model. Mimo obrazové podoby stránky ještě potřebujeme seznam vykreslených elementů (který bude obsahovat údaje o pozicích a velikostech elementů) a zdrojový kód stránky. Aby byl výsledek co nejpřesnější, rozhodli jsme se, že ke zhotovení obrázků a seznamu elementů využijeme možností webových prohlížečů. Vyzkoušeli jsme dva přístupy: rozšíření do prohlížeče a nástroje pro automatizaci prohlížečů Selenium.

4.2.1 Tvorba obrázků pomocí rozšíření pro prohlížeč

Prvním pokusem o vytvoření obrázkové podoby stránky bylo vytvoření rozšíření pro prohlížeč, který po kliknutí na tlačítko rozšíření nabídne ke stažení obrázek stránky a soubor, ve kterém jsou zapsané pozice všech prvků na stránce.

Pro vytvoření obrazové podoby stránky nabízí prohlížeč API, které vrátí obrazová data pro aktuálně vykreslenou oblast. Pokud bychom chtěli vytvořit obrázek z celé stránky, musíme si napsat obsluhu, která stránkou bude postupně posouvat a následně nafocené oblasti spojí do jednoho velkého obrázku. To je nepraktické, pokud jsou na stránce prvky, které reagují na posun

stránky (například menu, které se po určitém posunu přichytí na horní část obrazovky).

Dalším krokem bylo vytvoření seznamu veškerých prvků na stránce. Pro získání všech elementů jsme využili API pro manipulaci s DOM, jak uvádíme v ukázce kódu 8. Pro každý element necháme přepočítat aktuální hodnoty zobrazení (tím vynutíme správné hodnoty i u prvků, které ještě nejsou vykreslené) a vypočtené hodnoty uložíme pro pozdější zpracování. Pro natrénování modelu potřebujeme znát výšku, šířku, pozici a jednoznačný identifikátor, pomocí kterého budeme schopni zpětně dohledat prvek v HTML kódu. Pro účely této práce jsme za jednoznačný identifikátor zvolili *XPath* selektor, pro jehož výpočet jsme využili implementaci z projektu *Chromium*[43]. Do seznamu elementů jsme pro ladící účely přidali navíc informace o použitém HTML tagu, hodnoty HTML atributů `class` a `id` a textový obsah elementu.

Při testování naší implementace jsme narazili na několik problémů. První problém byl nesoulad v obrazové podobě stránky a získaným seznamem elementů, kdy prvky v seznamu měly evidovanou jinou pozici, než měly na obrázku. Problém jsme se pokusili odstranit korekcí, ale nepodařilo se nám zjistit, jestli jsou námi zjištěné korekce platné pro všechny webové stránky. Další problém spočíval v obtížné automatizaci daného přístupu – rozšíření je třeba aktivovat kliknutím na tlačítko. Poslední problém jsme již zmínili v odstavci o tvorbě obrazové podoby stránky, a to je absence podpory pro vytváření obrázku z celé stránky, nikoliv z aktuálního pohledu. Vzhledem k těmto okolnostem jsme rozšíření pro prohlížeč dále nerozvíjeli a pro tvorbu obrázků jsme využili nástroj *Selenium*.

4.2.2 Tvorba obrázků pomocí nástroje Selenium

Další pokus o tvorbu obrázků webových stránek jsem udělali s pomocí nástroje *Selenium*, který slouží pro automatizaci prohlížečů. Pro implementaci jsme zvolili platformu *node.js* v kombinaci s prohlížečem Mozilla Firefox[44]. Implementační kroky jsou stejné jako v případě rozšíření prohlížeče: Vytvoření obrazové podoby stránky a získání seznamu elementů.

Pro tvorbu obrazové podoby stránky nabízí *Selenium* API, které umožňuje, stejně jako v případě rozšíření prohlížeče, vytvořit obrázek z aktuálně viditelné oblasti. Kromě tvorby obrázku z vykreslené oblasti umožňuje *Selenium* pořizovat obrazovou podobu i z jednotlivých elementů na stránce. Pokud využijeme toto API na HTML elementu `body`, získáme obrazovou podobu celé stránky bez nutnosti dalšího skládání obrázku z vykreslených oblastí, jako tomu bylo v případě rozšíření pro prohlížeč. Pro získání seznamu elementů jsme použili stejný kód, jaký je uvedený v ukázce 8. Oproti uvedenému kódu jsme odstranili korekci pozic elementů (řádky 28 a 29), protože již nebyla potřeba. Implementaci nástroje na tvorbu obrazové podoby stránky pomocí nástroje *Selenium* naleznete v přiložených souborech ve složce `selenium-screenshots`.

```
function getOffset(el) {
  var x = 0;
  var y = 0;
  while (el && !isNaN(el.offsetLeft) && !isNaN(el.offsetTop)) {
    x += el.offsetLeft - el.scrollLeft;
    y += el.offsetTop - el.scrollTop;
    el = el.offsetParent;
  }
  return {top: y, left: x};
}

var result = [];
var all = document.getElementsByTagName("*");
for (var i = 0, max = all.length; i < max; i++) {
  var style = window.getComputedStyle(all[i]);
  var rect = all[i].getBoundingClientRect();

  // Ignore invisible elements
  if (style.display === 'none') continue;
  position = getOffset(all[i]);

  result.push({
    xpath: Elements.DOMPath.xPath(all[i], false),
    element: all[i].tagName,
    class: all[i].className,
    id: all[i].id,
    text: all[i].innerText,
    x: position.left + 6, // Pixel corrections
    y: position.top - 2, // Pixel corrections
    height: rect.height,
    width: rect.width,
  });
}
```

Ukázka kódu 8: Získání všech prvků na stránce.

4.2.2.1 Problémy při tvorbě obrázků

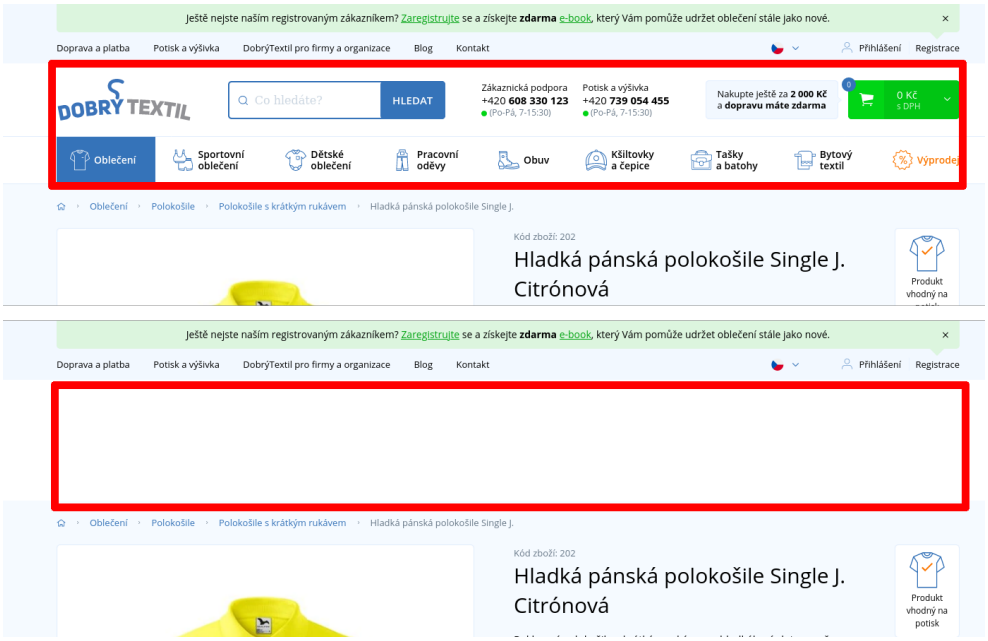
V průběhu vytváření obrázků pro dataset jsme u některých stránek pozorovali problémy, které mohly ovlivnit kvalitu našeho modelu:

- **Stránky, které chtějí posílat notifikace** – prohlížeč vykreslí žádost o povolení notifikací přes obsah, což by mohlo zakrýt důležité informace (obrázek 4.2). Řešení je v tomto případě jednoduché, prohlížeči stačí vypnout podporu notifikací a tyto dialogy zmizí.
- **Prvky, které reagují na posun stránky** – typický příklad je navigační lišta, která se při posunu stránky přichytí na horní okraj obrazovky, aby ji uživatel neztratil z dohledu. Při pořizování obrazové podoby stránky se tyto elementy přesunou do spodní části zachyceného obrázku, jak je znázorněno na obrázcích 4.3 a 4.4. Z toho usuzujeme, že při pořizování obrázku elementu prohlížeč posune stránku na konec tohoto elementu. Problém jsme se neúspěšně pokusili vyřešit tím, že před samotným pořízením obrázku posuneme okno prohlížeče na začátek stránky. Při hlubším rozboru stránek s těmito navigačními lištami jsme zjistili, že tyto prvky jsou obsluhovány pomocí javascriptové události `scroll`. Pokud zaregistrujeme vlastní obsluhu této události, která zastaví její zpracování, problém se tím vyřeší.
- **Prvky, které mají nastavenou fixní pozici** – podobně jako v předchozím případě, se prvky s fixní pozicí na stránce posunuly ke spodní části obrázku a překrývaly důležitý obsah (ukázka je na obrázku 4.6). Na rozdíl od předchozího případu jsou ale tyto prvky ovládány pomocí CSS, kde můžeme jednoduše hodnotu pozice přepsat a problém vyřešit.
- **Stránky, které omezují velikost elementu body** – některé stránky nastavují výšku elementu `body` na hodnotu `100%` a posouvání stránky pak realizují na vlastním vnitřním elementu. Běžný uživatel nepozná rozdíl, ale v našem případě to omezuje velikost výsledného obrázku na velikost obrazovky. V tomto případě je oprava jednoduchá. Stačí přenastavit výšku elementu `body` na hodnotu `auto`.
- **Líné načítání obrázků** – některé stránky optimalizují rychlost tím, že odloží stažení a zobrazení obrázků do momentu, než je uživatel potřebuje. V našem případě to způsobilo, že některé obrázky na stránce chybí (zpravidla ty, které byly umístěny za ohybem stránky). V současné době se pro implementaci líného načítání používají dva mechanismy: První způsob, kde se pomocí javascriptu kontroluje posun stránky a v závislosti na tom, jak moc je stránka posunuta, se upravují atributy obrázkům na stránce (typická úprava atributů je přepis hodnoty data atributu `data-src` do atributu `src`, čímž se vynutí stažení obrázku). Druhý způsob využívá nativní podpory prohlížečů, kdy stačí obrázkům

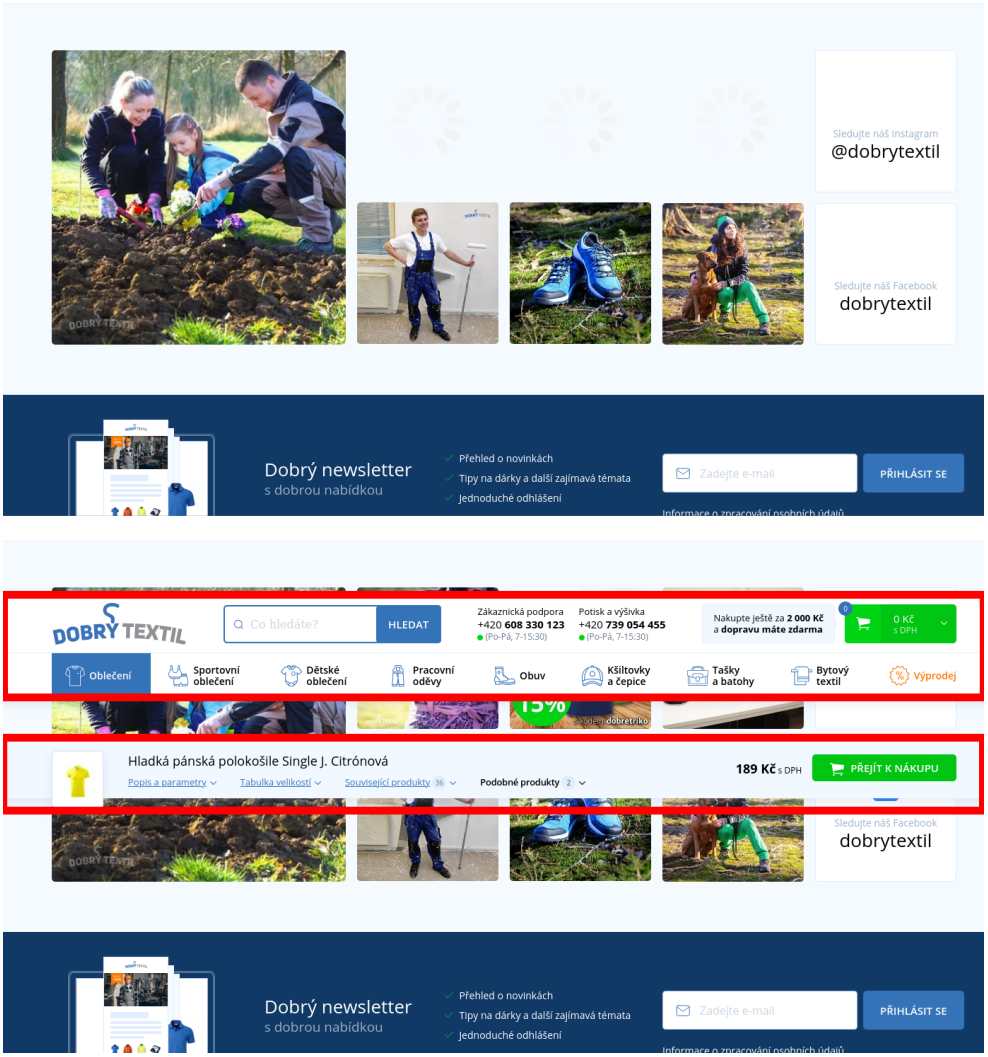
přiřadit atribut `loading=lazy` a prohlížeč se o líné načtení postará sám. Naše řešení spočívá v úpravě stránky tak, aby líné obrázky neobsahovala: Javascriptem procházíme všechny obrázky na stránce a odstraňujeme HTML atribut `loading` a v případě, že má obrázek nastavený HTML atribut `data-src`, tak jeho obsah přesuneme to atributu `src`.

- **Stránky s vyskakujícími okny** – některé stránky zobrazují vyskakující okna, která překrývají část nebo dokonce celý obsah (ukázka je na obrázku 4.5). Tyto stránky jsme z našeho datasetu odstranili, protože jsme nenalezli žádný spolehlivý způsob, jak vyskakovací okna automatizovaně zavřít.

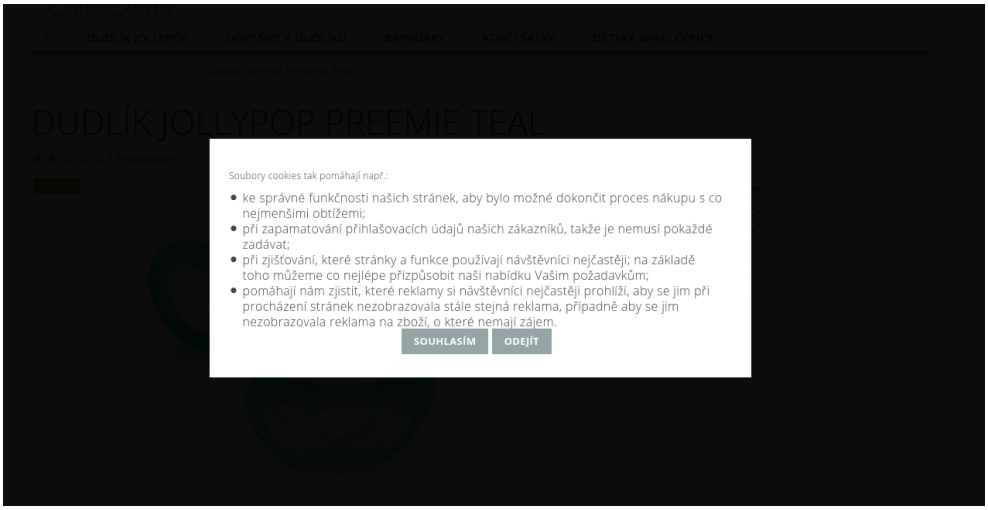
Obrázek 4.2: Žádost o povolení notifikací na webu mesec.cz.



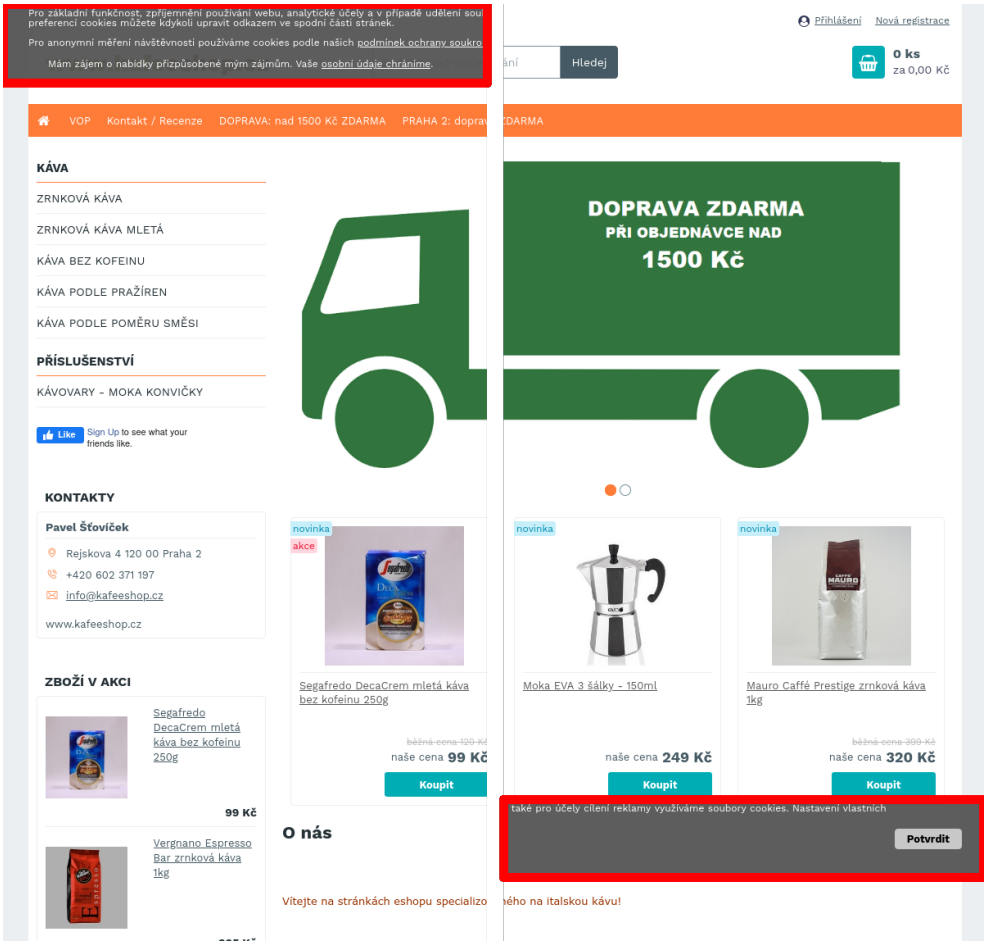
Obrázek 4.3: Ukázka navigační lišty, která se při pořizování obrázku posunula. Horní polovina obsahuje správný stav (tak, jak jej vidí uživatel v prohlížeči), spodní polovina obsahuje stav, který jsme zachytili před opravou.



Obrázek 4.4: Ukázka navigační lišty, která se při pořizování obrázku posunula. Horní polovina obsahuje patičku webu tak, jak ji vidí uživatel. Spodní polovina obsahuje stav, který jsme zachytili před opravou (navigační lišta se přesunula do patičky webu).



Obrázek 4.5: Vyskakovací okno, které zakrývá celý obsah.



Obrázek 4.6: Ukázka cookie lišty, která má nastavenou fixní pozici. Vlevo je zobrazen správný stav tak, jak jej vidí uživatel v prohlížeči. Vpravo je pak stav, který jsme zachytili před opravou.

4.3 Tvorba datasetu

Aby model správně fungoval, musíme ho natrénovat na kvalitním datasetu. Jelikož jsme vhodný dataset nenalezli, museli jsme ho vytvořit. Tvorbu datasetu rozdělíme do tří fází:

1. Příprava URL adres produktových stránek.
2. Zpracování produktových stránek do obrazové podoby.
3. Vyznačení relevantních oblastí v obrázku – anotace datasetu.

4.3.1 Příprava URL adres produktových stránek

Seznam URL adres jsme vytvořili ze dvou zdrojů. Hlavním zdrojem byl seznam XML feedů, který nám poskytla společnost Simplia s. r. o., která se zabývá pronajímáním e-shopového řešení svým klientům. Výhoda tohoto řešení je v unikátnosti šablon – každý internetový obchod na tomto řešení má vlastní šablonu. Díky tomu můžeme využít všechny e-shopy pro tvorbu našeho datasetu, aniž bychom se museli obávat, že se model přeučí na konkrétní šablonu. Další výhodou pro nás je forma zpřístupněných dat – pro vytvoření seznamu adres produktů z XML feedu nám stačí zpracovat XML soubor a adresy produktů si z něho jednoduše vytáhnout. Druhou část datasetu jsme vytvořili z náhodných e-shopů nalezených na internetu – internetové obchody jsme ručně proklikali a náhodně vybrali deset produktů. Celkově jsme na seznamu URL adres měli 229 e-shopů od společnosti Simplia a 203 náhodných e-shopů, od každého e-shopu pak deset obrázků (pokud bylo alespoň deset produktů v nabídce).

4.3.2 Zpracování produktových stránek do obrazové podoby

Pro vytvoření obrazové podoby stránek ze seznamu jsme využili nástroj, který byl detailně popsán v kapitole 4.2.

4.3.3 Vyznačení relevantních oblastí v obrázku – anotace datasetu

Anotace datasetu spočívá v označení relevantních oblastí v obrázku. Standardně se datasety anotují ručně pomocí nástrojů pro anotaci (populární jsou například nástroje *LabelImg*[45] nebo *labelme*[46]), což je ale velice časově náročné. Proto jsme hledali způsoby, jakými bychom mohli anotování datasetu urychlit.

4.3.3.1 Automatizované anotování datasetu

Anotování obrázků webových stránek má oproti anotaci běžných obrázků určité výhody: Anotované objekty jsou vždy ve tvaru obdélníku, který není nijak pootočený (hrany obdélníku jsou vodorovné s okraji obrázku), nemusíme proto řešit polygonální anotace. Ke každému obrázku již máme metadata v podobě seznamu elementů na stránce, které můžeme využít k automatizaci anotování datasetu.

Pro každý e-shop z našeho datasetu jsme vybrali vzorek, u kterého jsme ručně identifikovali prvky, jež označovaly anotovaný atribut. Pro tyto prvky jsme si zapsali HTML atributy `class` a `id` a také informaci, jaký HTML tag byl pro daný element použit. Na základě těchto informací jsme následně hledali v dalších vzorcích stejně vypadající elementy, ze kterých jsme automatizovaně vytvořili anotace – rozměry oblastí jsme přejali z pozic a velikostí HTML elementů. Díky tomuto přístupu se nám podařilo tvorbu datasetu urychlit, nicméně automatizované anotování nebylo bez chyb:

- **Hledání elementů ve vzorcích fungovalo i pokud obsah stránky nebyl čitelný** – tento automatizovaný způsob označil oblast i v případě, že obsah stránky překrylo vyskakovací okno (což by zkreslovalo model).
- **Velikost elementů neodpovídala velikosti obsahu** – to je dáno tím, jak prohlížeč vykresluje elementy: Blokové prvky roztáhne na celou šířku a do rozměrů započítá i hodnoty CSS vlastnosti `padding`.
- **Problematické označení elementů, které neměly vlastní HTML tag** – tato metoda nedokázala vybrat správný element, pokud e-shop vypisoval texty bez umístění do vlastních HTML tagů.
- **Vícenásobné označení elementů se stejnými vlastnostmi** – v případě, že na stránce existovalo více elementů se stejnými HTML atributy, bylo problematické zvolení správného elementu (typický problém u e-shopů, které v detailu produktu zobrazují podobné produkty).

Na základě těchto skutečností jsme se rozhodli, že automatizovaný dataset podrobíme ruční kontrole, při které z datasetu odstraníme nevhodné vzorky (například pokud byly překryté vyskakovacím oknem) a upravíme špatně označené oblasti.

4.3.3.2 Korekce anotací

Abychom do datasetu nezanášeli šum v podobě nesprávně označených oblastí, rozhodli jsme se anotované oblasti zmenšit podle velikosti jejich obsahu. Pro tyto účely jsme vytvořili krátký program, který za pomocí knihovny *opencv-python*[47] projde všechny vzorky a aplikuje na ně algoritmus 1. Ukázku, jak tento program upravuje anotace vzorků uvádíme na obrázku 4.7.

Obrázek 4.7: Korekce anotace pomocí algoritmu 1. Červeně jsme označili oblasti, které vznikly pomocí automatizované anotace. Zeleně jsme označili stejné oblasti po korekci.

Algoritmus 1: Korekce anotací v datasetu

```
forall annotation in sample do
    crop image by annotation;
    find edges in cropped image using Canny algorithm;
    find countours of those edges;
    calculate convex hull of found countours;
    update annotation position and dimensions;
end
```

4.3.3.3 Výsledná podoba datasetu

Celý dataset jsem uspořádali podle toho, z jakého e-shopu jsme vzorek pořídili. Popis struktury složek datasetu uvádíme na obrázku 4.8.

Pro uložení anotací jsme zvolili vlastní formát dat, který nám umožňuje s datasetem pracovat automatizovaně, aniž bychom se museli bát, že anotace v průběhu manipulace poškodíme. Konverzi anotací do formátu, se kterým pracuje model pak provádíme těsně před trénováním modelu. Z toho důvodu také anotace ukládáme do každé skupiny vzorků zvlášť – s menším množstvím anotací se lépe pracuje. Ukázku souboru s anotacemi uvádíme na ukázce kódu 9.

Poslední krok přípravy datasetu spočíval v rozdělení na trénovací a testovací množinu. Vzhledem k povaze vzorků v datasetu jsme se rozhodli jej rozdělit ručně podle následujících kritérií:

Dataset obsahuje několik různých vzorků z jednoho e-shopu, které jsou si vizuálně velice podobné (liší se jen v zobrazených hodnotách). Proto, pokud jsme se rozhodli vzorek přesunout do testovací množiny, přesunuli jsme tam

celou skupinu vzorků (všechny vzorky z daného e-shopu). Ze stejného důvodu jsme do testovací množiny přesunuli i všechny jazykové mutace daného e-shopu. Posledním kritériem bylo, abychom v testovací množině měli nejen vzorky moderních e-shopů, ale i vzorky starších e-shopů, které neodpovídají dnešním standardům na prezentaci informací. Cílem bylo otestování našeho modelu i na těchto starších e-shopech. Výslednou bilanci počtu vzorků a jejich anotovaných oblastí uvádíme v tabulce 4.3.

```
train ..... vzorky z trénovací množiny
  adresa-e-shopu.cz ..... název e-shopu, ze kterého vzorky pocházejí
    1.json ..... seznam elementů pro první vzorek
    1.html ..... HTML kód stránky pro první vzorek
    1.png ..... obrázek kód stránky pro první vzorek
    metadata.json ..... soubor s anotacemi
val ..... vzorky z testovací množiny
  e-shop.cz ..... název e-shopu, ze kterého vzorky pocházejí
    1.json ..... seznam elementů pro první vzorek
    1.html ..... HTML kód stránky pro první vzorek
    1.png ..... obrázek kód stránky pro první vzorek
    metadata.json ..... soubor s anotacemi
```

Obrázek 4.8: Struktura složek v datasetu.

Atribut	**Počet instancí v trénovací množině**	**Počet instancí v testovací množině**
Název produktu	3279	285
Cena	3217	290
Dostupnost	2961	271
Obrázek	3241	284
Krátký popisek	1680	160
Dlouhý popisek	2784	263
Drobečková navigace	3054	274
Název varianty	901	66
Cena variatny	858	65
Dostupnost varianty	842	62
Celkem	22817	2020
Celkem vzorků	3271	285

Tabulka 4.3: Rozložení tříd v datasetu.

```
{
  "1": [
    {
      "type": "name",
      "top": 346,
      "left": 680,
      "width": 487,
      "height": 36
    },
    {
      "type": "mainPrice",
      "top": 576,
      "left": 1159,
      "width": 81,
      "height": 29
    },
    {
      "type": "image",
      "top": 396,
      "left": 681,
      "width": 218,
      "height": 282
    }
  ]
}
```

Ukázka kódu 9: Formát anotací.

4.4 Tvorba a trénování modelu pro detekci a klasifikaci objektů

Model pro detekci a klasifikaci objektů jsme se rozhodli vytvořit pomocí nástroje *detectron2*[48], který implementuje nejmodernější přístupy pro řešení detekce objektů. Tento nástroj jsme si vybrali, protože má dle [49] nejrychlejší trénování, obsahuje předtrénované modely pro řešení detekce objektů a jeho použití je velmi jednoduché.

Model můžeme vytvořit buď implementováním vlastních komponent, nebo můžeme využít již předpřipravených modulů v podobě konfigurace, které můžeme nalézt na stránce `https://github.com/facebookresearch/detectron2/tree/master/configs`. My jsme pro realizaci našeho modelu zvolili způsob s využitím konfiguračních souborů. K dispozici jsou připravené konfigurace pro různé typy detekce objektů (například segmentace nebo hledání klíčových bodů), my ale využijeme pouze konfigurace pro detekci a klasifikaci.

Abychom mohli natrénovat náš model, musíme nejprve zaregistrovat náš dataset, aby ho *detectron2* mohl použít. Námi vytvořený dataset ukládá anotace ve formátu, který jsme definovali v kapitole 4.3.3.3, proto musíme implementovat vlastní *dataloader*, který anotace překonvertuje do formátu, který dokáže *detectron2* zpracovat (nástroj využívá anotace ve formátu COCO[50]). Nástroj v anotacích potřebuje velikosti obrázků, takže je musíme před načtením spočítat pomocí knihovny *opencv*. Implementaci *dataloaderu* naleznete v přiložených souborech pod názvem `data_loader.py`.

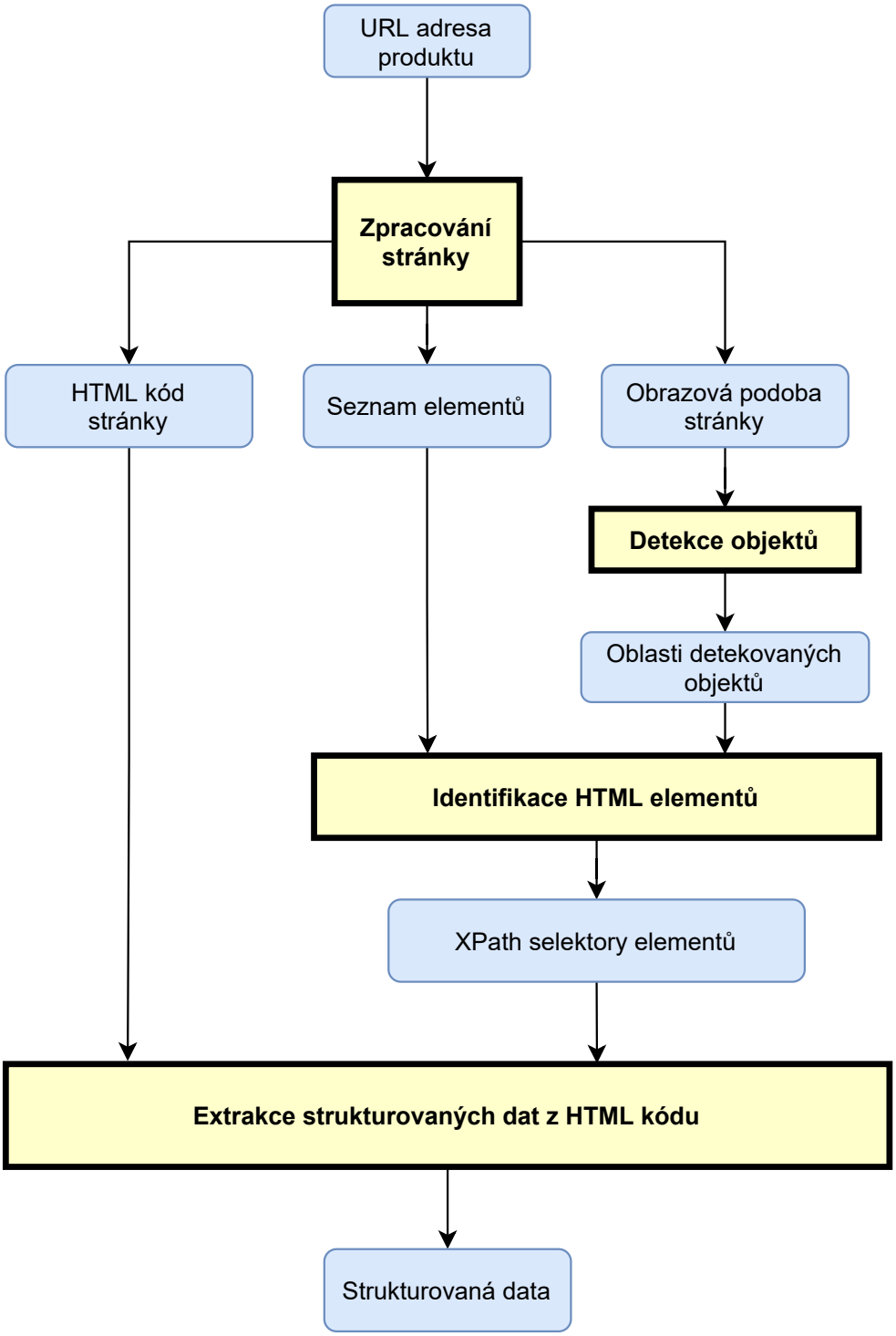
Trénování a použití modelu uvádíme na ukázkách kódu 10 a 11. Předpřipravené modely se parametrizují pomocí konfiguračního objektu, který následně předáme do některé z komponent nástroje. My v této práci využíváme komponenty *Evaluator*, *Predictor* a *Trainer*. Použitými konfiguracemi modelů a jejich výsledky se detailně zabýváme v kapitole 5.

4.5 Extrakce strukturovaných dat

Nyní si popíšeme, jak výše uvedené komponenty zapojit do procesu extrakce strukturovaných dat (diagram celého procesu extrakce uvádíme na obrázku 4.9). Na vstupu očekáváme URL adresu produktové stránky, kterou nejprve zpracujeme pomocí nástroje popsaného v kapitole 4.2. Nástroj vygeneruje tři výstupy: HTML kód stránky, seznam elementů na stránce a obrazovou podobu stránky. Na obrázku následně provedeme detekci relevantních oblastí pomocí modelu popsaného v kapitole 4.4. Posledním krokem procesu je využití těchto relevantních oblastí k identifikaci elementů v HTML kódu.

4.5.0.1 Identifikace HTML elementů na základě detekovaných oblastí

Pro identifikaci HTML elementů na základě detekovaných oblastí jsme navrhli algoritmus, který využívá metriky *intersection over union* pro nalezení správného elementu. Nejdříve pro všechny detekované oblasti vybereme element, pro který má deteková oblast nejvyšší hodnotu metriky *intersection over union*. Abychom předešli vybrání prázdných elementů, vybíráme pouze z elementů, které mají textový obsah. Pokud je detekovaná oblast označena jako obrázek, omezíme vyhledávání pouze na elementy s obrázkem, abychom ve výsledcích nedostávali obalové elementy obrázku. Z takto vybraných elementů následně vybereme pro každou třídu jeden element s nejvyšší hodnotou skóre v dané třídě. Posledním krokem je extrakce obsahu z HTML elementů – k lokalizaci elementů v HTML kódu využíváme XPath selektorů, které jsme si uložili do seznamu prvků na stránce. V tomto kroku také provádíme normalizace výstupních dat, jako je převedení relativních adres na absolutní nebo odstranění nadměrného počtu bílých znaků. Takto zpracovaný výstup vracíme ve formátu json, příklad uvádíme na ukázce kódu 12. Implementaci tohoto algoritmu naleznete v přiložených souborech pod názvem `extractor.py`.



Obrázek 4.9: Proces extrakce strukturovaných dat z webu.

```
import dataset_loader

outputDir = "trained-models/faster_rcnn_R_101_FPN_3x"

setup_logger(os.path.join(outputDir, 'training.log'))

for d in ["train", "val"]:
  DatasetCatalog.register(
    "websites_" + d,
    lambda d=d: dataset_loader.get_data(os.path.join("data", d))
  )
  MetadataCatalog.get("websites_" + d)
    .set(thing_classes=dataset_loader.categories)

baseConfiguration = "COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"

cfg = get_cfg()
cfg.OUTPUT_DIR = outputDir
cfg.merge_from_file(model_zoo.get_config_file(baseConfiguration))
cfg.DATASETS.TRAIN = ("websites_train",)
cfg.DATASETS.TEST = ("websites_val",)
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url(baseConfiguration)
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025
cfg.SOLVER.MAX_ITER = 30000
cfg.SOLVER.STEPS = []
cfg.SOLVER.CHECKPOINT_PERIOD = 1000
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(dataset_loader.categories)


os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=True)
trainer.train()
```

Ukázka kódu 10: Trénování modelu pomocí nástroje *detectron2*.

```
import extractor

baseConfiguration = "COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml";

cfg = get_cfg()
cfg.OUTPUT_DIR = "trained-models/faster_rcnn_R_101_FPN_3x"
cfg.merge_from_file(model_zoo.get_config_file(baseConfiguration))
cfg.DATALOADER.NUM_WORKERS = 1
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 512
cfg.MODEL.ROI_HEADS.NUM_CLASSES = len(dataset_loader.categories)

cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.50
predictor = DefaultPredictor(cfg)

im = cv2.imread(sys.argv[1] + '.png')
outputs = predictor(im)
instances = outputs["instances"].to("cpu")

formattedInstances = []

# Format predictions and use extractor to extract values
# from HTML code
for k, className in enumerate(instances.pred_classes):
  box = instances.pred_boxes[k][0].tensor[0]

  formattedInstances.append({
    'type': dataset_loader.categories[className],
    'score': instances.scores[k].item(),
    'x': box[0].item(),
    'y': box[1].item(),
    'width': box[2].item() - box[0].item(),
    'height': box[3].item() - box[1].item(),
  })

extracted = extractor.extract(sys.argv[1], formattedInstances)

# Output extracted data in JSON format
print(json.dumps(extracted))
```

Ukázka kódu 11: Použití modelu *detectron2*.

```
{
  "mainPrice": "49 Kč",
  "shortDescription": "Překrásná sponka s velkým květem a ...",
  "name": "SPONKA - VELKÝ KVĚT S PUNTÍKY",
  "breadcrumbs": "SPONKA - VELKÝ KVĚT S PUNTÍKY",
  "availability": "Skladem"
}
```

Ukázka kódu 12: Příklad výstupu.

Kapitola 5

Experimenty

5.1 Použité modely pro detekci objektů

Pro detekci objektů jsme využili předpřipravených modelů z nástroje *Detectron2*, které už jsou předtrénované na datasetu COCO[50]. Nástroj můžeme využít i bez použití předtrénovaných variant modelů, ale využití předtrénovaných modelů přináší rychlejší trénování.

Nástroj *Detectron2* nabízí pro detekci objektů několik modelů založených na architektuře *Faster R-CNN* nebo *RetinaNet*. U každé architektury existuje několik variant, které se liší podle použité páteřní sítě. Pro náš model jsme vybrali varianty *Faster R-CNN* s páteřními sítěmi *ResNet-50*, *ResNet-101* a *ResNeXt-101* (s parametry kardinality 32 a šířkou úzkého hrdla 8), a varianty *RetinaNet* s páteřními sítěmi *ResNet-50*, *ResNet-101*. Všechny použité páteřní sítě využívali *feature pyramid network*, číslo za pomlčkou pak značí počet vrstev v dané síti.

Před samotným použitím modelů jsme ještě upravili transformace, které se používají při trénování sítě. Při řešení úloh klasifikace a detekce obrázků se při trénování obrázek často náhodně převrací, aby byl dataset co nejobsáhlejší – obrázek kočky, který bude horizontálně převrácený bude stále vypadat jako obrázek kočky. V našem případě je toto ale nežádoucí transformace, protože obrázky webových stránek budou všechny orientované stejně (sice existují jazyky, kde se píše zprava doleva, ale těm se v této práci nevěnujeme). Proto jsme tuto transformaci vypnuli.

5.2 Trénování modelů pro detekci objektů

Všechny výpočty jsme prováděli na stolním počítači s procesorem Intel Core i5-4590 (základní takt 3,3 GHz), 16 GB operační paměti a grafické kartě Nvidia GTX 1080 (karta disponuje 2560 výpočetními jádry a 8 GB grafické paměti). Využívali jsme modely, které byly předtrénované na datasetu COCO[50], které

jsme trénovali po 30 000 iterací. Trénovací a vybavovací časy jednotlivých modelů uvádíme v tabulce 5.1.

model	**páteřní síť**	**celkový čas tréninku**	**vybavovací čas na obrázek**
faster_rcnn/R_101	*ResNet-101*	6 h 10 m	0,1078 s
faster_rcnn/R_50	*ResNet-50*	4 h 11 m	0,0792 s
faster_rcnn/X_101	*ResNeXt-101*	6 h 45 m	0,1991 s
retinanet/R_101	*ResNet-101*	3 h 45 m	0,1053 s
retinanet/R_50	*ResNet-50*	2 h 40 m	0,0754 s

Tabulka 5.1: Časy potřebné na trénování a používání modelů.

5.3 Výsledky modelů pro detekci objektů

V této části práce prezentujeme výsledky modelů pro detekci a klasifikaci objetků, které jsme natrénovali na našem datasetu. Zaměříme se na metriku průměrné přesnoti modelu, kterou jsme definovali v kapitole 3.3.3. Hodnotu prahu pro evaluaci jsme zvolili 50 %, jelikož detekované objekty dále párujeme s HTML elementy na stránce, což nám provádí korekci detekovaných objetků.

Na obrázku 5.2 uvádíme vývoj průměrných přesností modelů AP_{50} a AP_{75} v závislosti na délce trénování. Na obrázcích 5.3 a 5.4 uvádíme vývoj průměrných přesností modelů AP_{50} pro jednotlivé kategorie objektů. Z naměřených hodnot můžeme pozorovat, že modely konvergují po přibližně 15 000 iteracích. Dále můžeme pozorovat, že nejlepšího průměrného výsledku dosáhl model *Faster R-CNN* s páteřní sítí *ResNeXt-101*. Když se podíváme na grafy výsledků jednotlivých kategorií, zjistíme, že nejlépe dopadly kategorie obrázek a název produktu (přesnost se pohybuje okolo 70 %) a dále cena produktu (přesnost necelých 60 %).

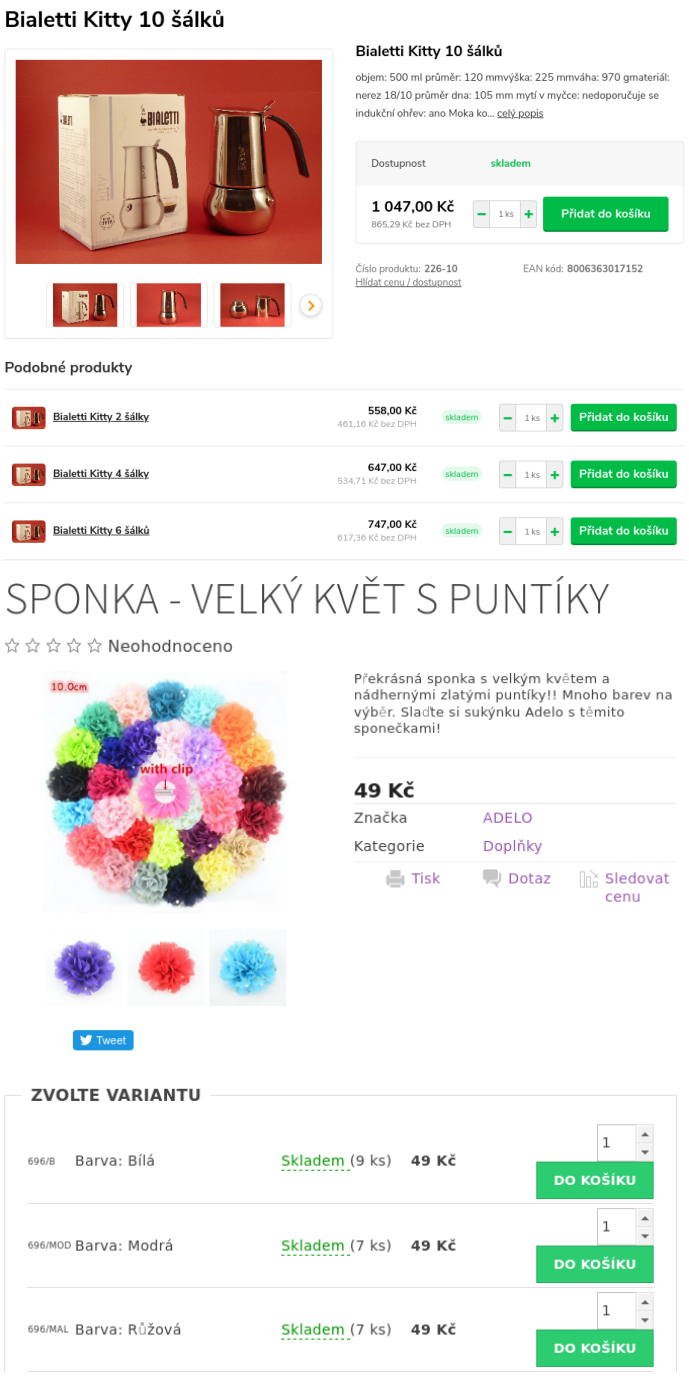
Naopak kategorie dostupnost produktu a kategorie, které značí varianty produktů, dopadly nejhůře. V případě dostupnosti produktu stojí za špatným výsledkem různorodé označení na e-shopech, kdy se používá několik různých forem pro označení dostupnosti. Pokud e-shop využíval pro označení dostupnosti zelený text „skladem“, model jej ve většině případů detekoval správně. Dostupnost pak byla špatně detekována v případě, že e-shop pro označení dostupnosti používal texty jako „dostupné za X dní“, „odesíláme za X dní“ nebo jen číslo označující počet položek skladem.

Špatný výsledek detekce oblastí variant přisuzujeme několika faktorům. Jak uvádíme na obrázku 5.1, na některých e-shopech se na místě variant vykreslují podobné produkty, které jsou vizuálně velmi podobné s variantami produktu. V datasetu jsme oblasti s podobnými produkty neoznačily, proto po-

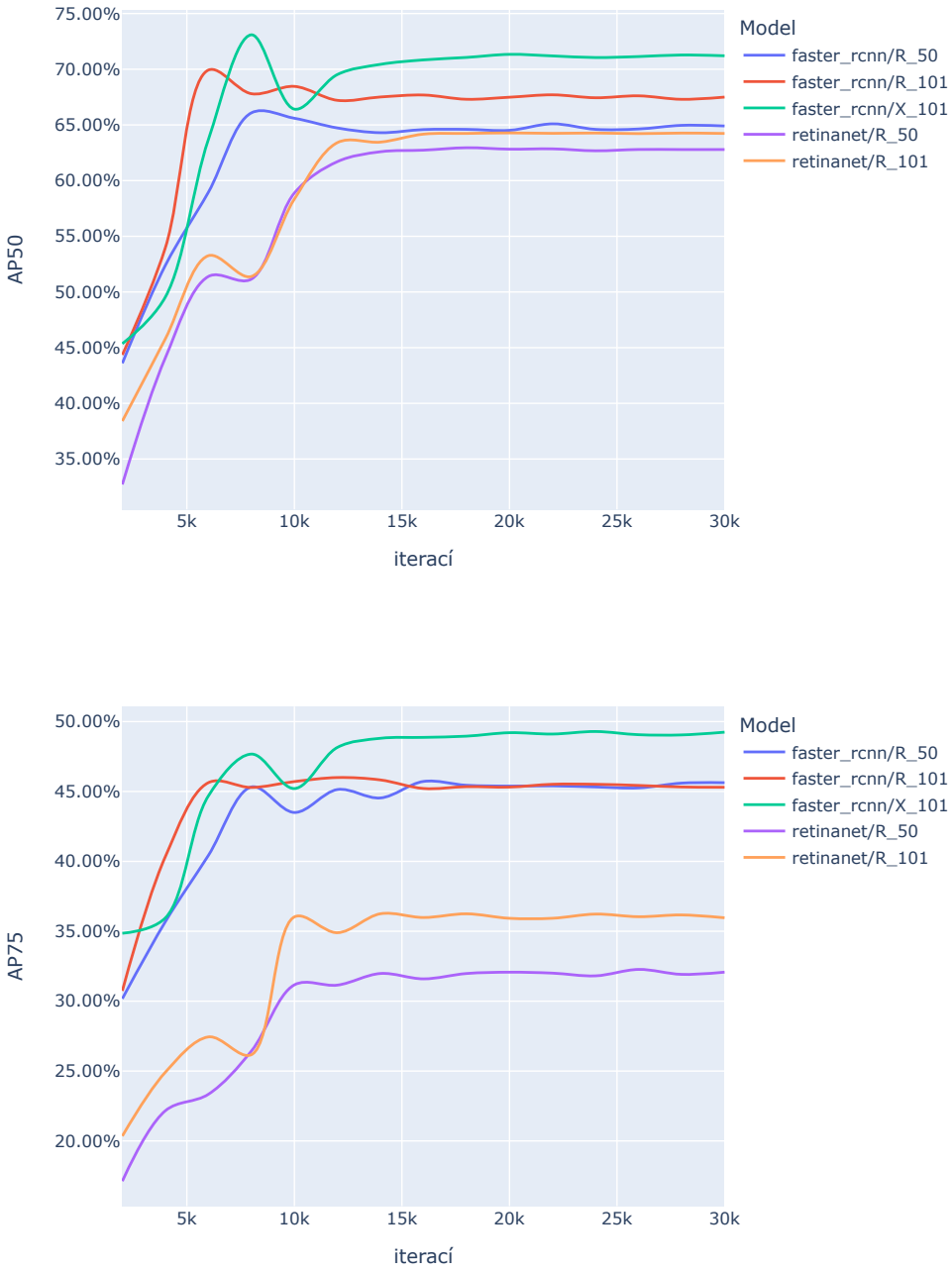
kud je model detekoval jako varianty, došlo ke snížení přesnosti predikce. Další příčinou, kterou jsme pozorovali, byly částečné detekce, kdy model správně detekoval část varianty (většinou správně detekoval cenu varianty) ale zbylé atributy varianty již nedetekoval, nebo nedetekoval všechny varianty. V neposlední řadě přisuzujeme špatný výsledek nedostatku vzorků s variantami. Jak můžeme vidět v tabulce 4.3, vzorků s variantami je výrazně méně, než ostatních vzorků (počet instancí v tabulce značí počet oblastí, ale na jednom vzorku se může vyskytovat více variant).

Na obrázku 5.5 uvádíme výsledky pro vybrané e-shopy z testovacího datasetu (vybrali jsme 3 e-shopy s nejlepšími výsledky a 3 e-shopy s nejhoršími výsledky). V tabulce 5.2 uvádíme výsledky pro všechny e-shopy z testovacího datasetu. Zde můžeme pozorovat, že na většině stránkách proběhla detekce velmi dobře (s výsledkem 70 % a lépe).

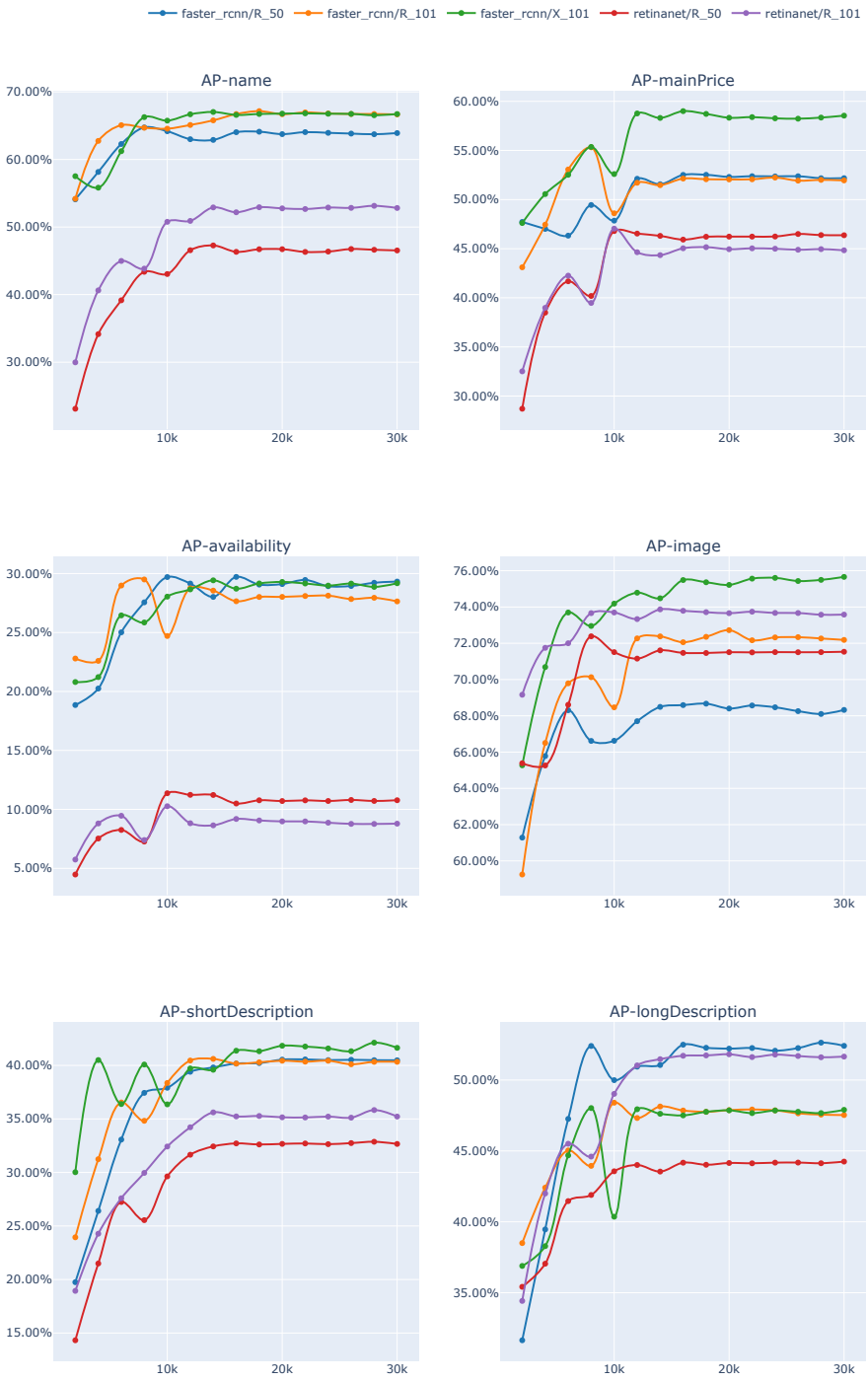
Pokud porovnáme vizuální podobu e-shopu, který dopadl nejlépe (obrázek 5.6), s e-shopem, který dopadl nejhůře (obrázek 5.7), zjistíme, že mezi stránkami jsou obrovské rozdíly. V prvním případě je e-shop velice zdařile zpracovaný, důležité prvky jsou barevně odlišené, kdežto v druhém případě je e-shop velice špatně navržený a vlivem nízkého kontrastu mezi pozadím a textem je čtení této stránky náročné i pro člověka. Nutno podotknout, že e-shop v prvním případě využívá sdílenou šablonu platformy `eshop-rychle.cz`, kdy stejnou šablonu využívá i několik e-shopů z trénovací množiny, což přispělo k dobrému výsledku našeho modelu. Záměrně jsme tyto vzorky nevyřazovali z testovací množiny, jelikož je tento způsob provozování e-shopů v České republice velice rozšířený a nás zajímalo, jakých výsledků náš model dosáhne, pokud bude použit na stejnou šablonu (je velice pravděpodobné, že bychom na takovou šablonu narazili při použití našeho modelu pro reálná data).



Obrázek 5.1: Vykreslení podobných produktů (horní obrázek) v podobném vizuálním zpracování, jako varianty produktu (spodní obrázek).



Obrázek 5.2: Průměrné přesnosti modelů AP_{50} (horní obrázek) a AP_{75} (spodní obrázek) v závislosti na délce trénování.



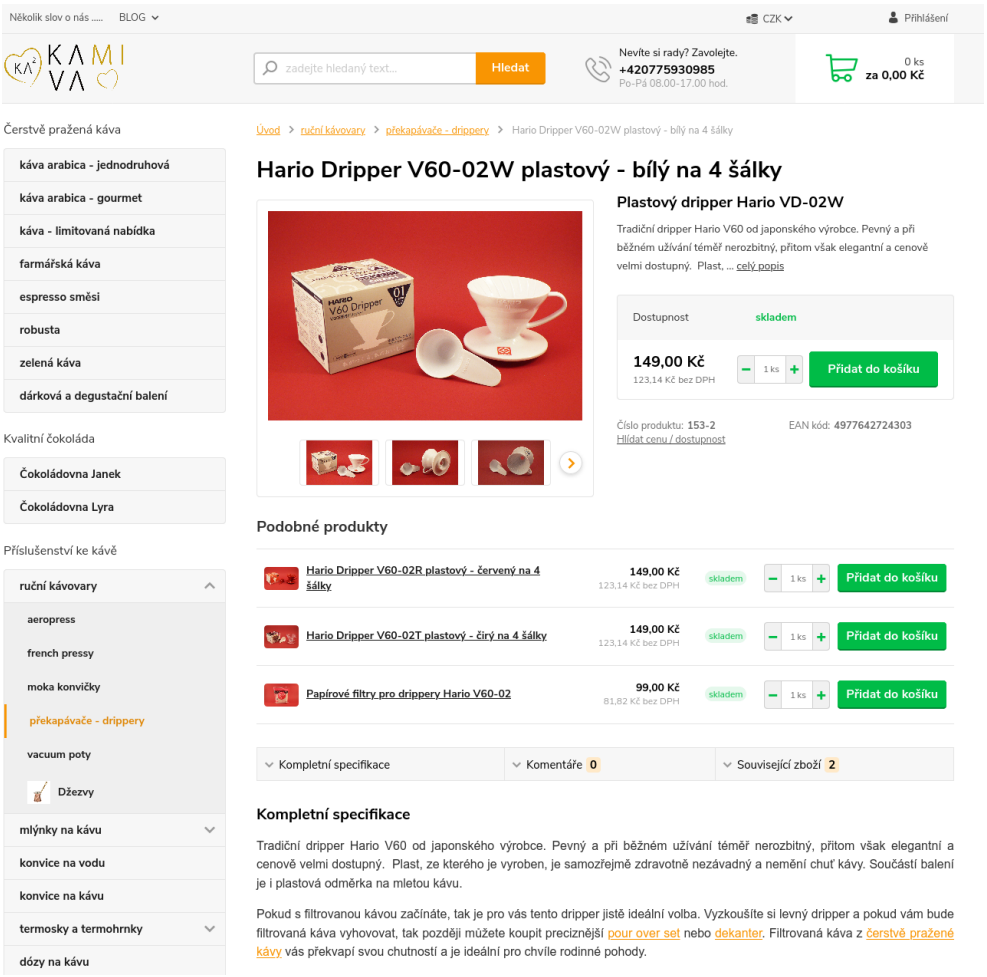
Obrázek 5.3: Průměrné přesnosti modelů AP_{50} v závislosti na délce trénování pro jednotlivé kategorie objektů.

Obrázek 5.4: Průměrné přesnosti modelů AP_{50} v závislosti na délce trénování pro jednotlivé kategorie objektů.

Obrázek 5.5: Průměrné přesnosti modelů AP_{50} v závislosti na délce trénování pro jednotlivé e-shopy. Vlevo jsou uvedené e-shopy, které náš model detekoval nejlépe, vpravo jsou e-shopy, které náš model detekoval nejhůře.

e-shop	faster_rcnn/R_50	faster_rcnn/R_101	faster_rcnn/X_101	retinanet/R_50	retinanet/R_101
aaasamolepkynazed.cz	38,02	43,85	47,80	62,12	54,47
adelo.cz	80,64	76,92	84,91	81,74	84,70
bici.cz	67,69	58,67	70,30	59,73	71,25
chcigril.cz	78,22	77,70	85,15	78,81	74,30
cocochocokeratin.sk	95,08	87,66	96,39	79,47	86,60
cosmeticos.cz	85,99	81,08	83,03	76,98	84,47
drakkaria.cz	74,26	77,62	74,26	77,12	76,19
ebal.cz	62,59	59,91	67,71	54,55	72,65
ebubak.cz	81,25	83,28	84,49	82,30	76,73
e-cs.cz	52,56	53,13	54,76	65,14	63,10
e-manikury.cz	88,86	78,52	86,77	72,63	77,15
m.e-manikury.cz	80,03	82,84	77,89	81,02	84,97
fesakov.cz	80,91	76,99	74,51	82,52	79,67
fesnakocka.cz	69,52	70,27	75,22	79,86	80,08
gsm-market.cz	77,85	89,12	81,43	73,15	72,11
healer.cz	87,64	86,50	92,16	85,50	88,12
kamikava.cz	98,66	96,86	96,65	95,49	85,70
nakupzdomu.eu	56,79	80,00	72,69	60,80	63,16
nej-lekarna.cz	93,99	97,77	95,51	72,17	80,66
nejlepsirumy.cz	14,00	19,47	21,32	10,09	18,49
nemrznoucismesi.cz	58,47	53,44	54,04	57,92	53,45
pazba.cz	88,27	88,41	91,82	83,89	86,65
pohary.com	67,65	69,21	68,81	65,87	67,13
prodort.cz	83,74	83,76	85,26	81,75	84,50
profimed.cz	92,36	87,92	89,83	86,56	87,92
svet-her.cz	76,23	77,78	78,87	71,92	73,50
svet-hier.sk	77,94	81,02	81,53	74,79	72,05
wulflund.com	68,91	89,11	92,22	82,88	80,09

Tabulka 5.2: Průměrné přesnosti modelů AP_{50} pro jednotlivé e-shopy (uvedené hodnoty jsou v %).



Obrázek 5.6: Ukázka e-shopu, který je graficky zpracovaný kvalitně – ze všech prvků je na první pohled jasné, co znamenají.

Obrázek 5.7: Ukázka e-shopu, který je graficky zpracovaný velice nepřehledně – pozadí splývá s důležitým textem a e-shop je velice náročný na čtení.

5.4 Výsledky modelu pro extrakci strukturovaných informací

V předchozí části jsme se věnovali vyhodnocením modelů pro detekci a klasifikaci objektů. Nyní tyto modely využijeme společně s naším algoritmem pro propojení detekovaných oblastí s HTML elementy, čímž sestrojíme model pro extrakci strukturovaných dat z webu. Pro detekci objektů jsme vybrali model *Faster R-CNN* s páteřní sítí *ResNeXt-101*, který dosáhl nejlepších výsledků detekce objektů.

Vybrali jsme 4 e-shopy, na kterých si ukážeme silné a slabé stránky našeho modelu. U těchto e-shopů vykreslíme detekované oblasti, a následně zanalyzujeme extrahované informace. Na závěr náš model zhodnotíme pro všechny e-shopy z testovací množiny.

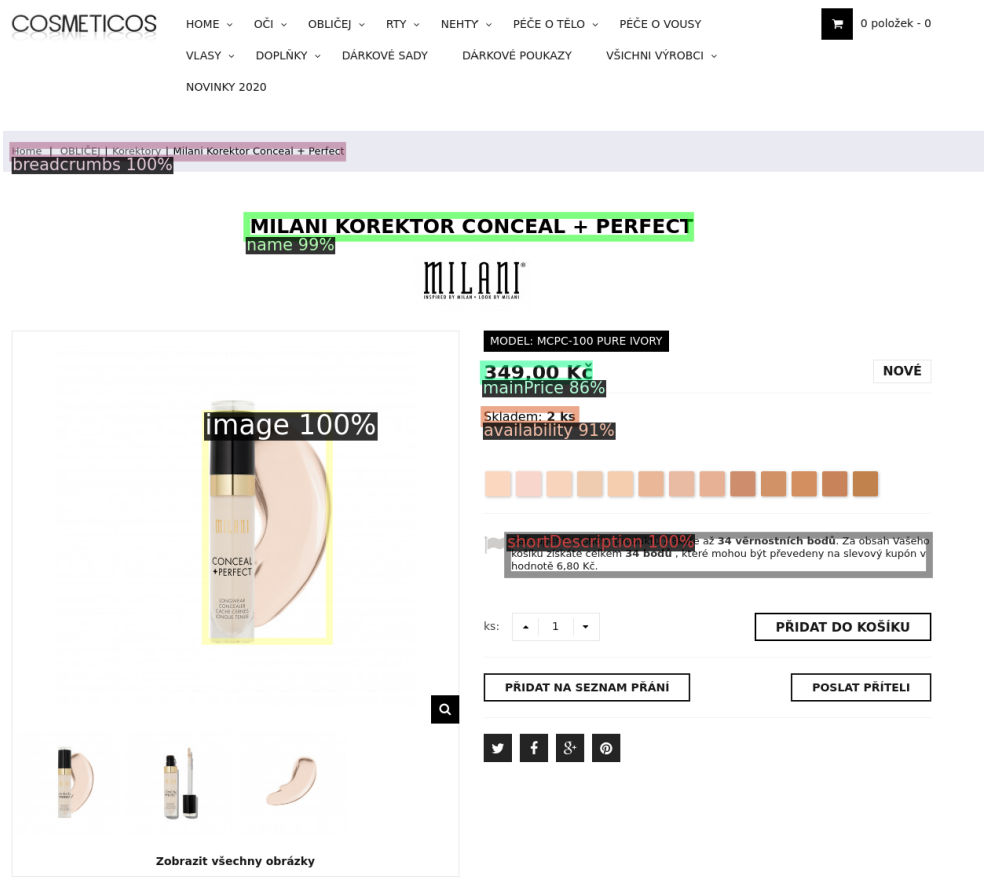
5.4.1 Výsledky extrakce na vybraných e-shopech

5.4.1.1 Výsledky extrakce na e-shopu cosmeticos.cz

První e-shop z našeho výběru je e-shop `cosmeticos.cz`, kde můžeme pozorovat téměř perfektní extrakci. Na obrázku 5.8 jsme vyznačili oblasti, které model detekoval. Jak si můžeme všimnout, všechny informace jsou detekovány správně. Na ukázce kódu 13 uvádíme kompletní výstup našeho modelu pro extrakci. Jedinou vadou je špatně zpracovaná dostupnost, kdy náš algoritmus pro propojení detekovaných oblastí s HTML elementy špatně idenfikoval výstupní element a informaci o počtu dostupných kusů zahodil.


```
{
  "category": [
    "Home",
    "OBLIČEJ",
    "Korektory"
  ],
  "shortDescription": "Zakoupením tohoto produktu získáte až 34...",
  "image": "https://cosmeticos.cz/4937-large_default/milani-ko...",
  "name": "Milani Korektor Conceal + Perfect",
  "longDescription": "Veškeré vady pleti jsou pryč! Tento...",
  "availability": "Skladem:",
  "mainPrice": "349,00 Kč"
}
```


Ukázka kódu 13: Extrahovaná data z webu `cosmeticos.cz`, uvedeného na obrázku 5.8. Výpis jsme pro lepší přehlednost zkrátili (zkrácené pasáže jsme nahradili třemi tečkami).



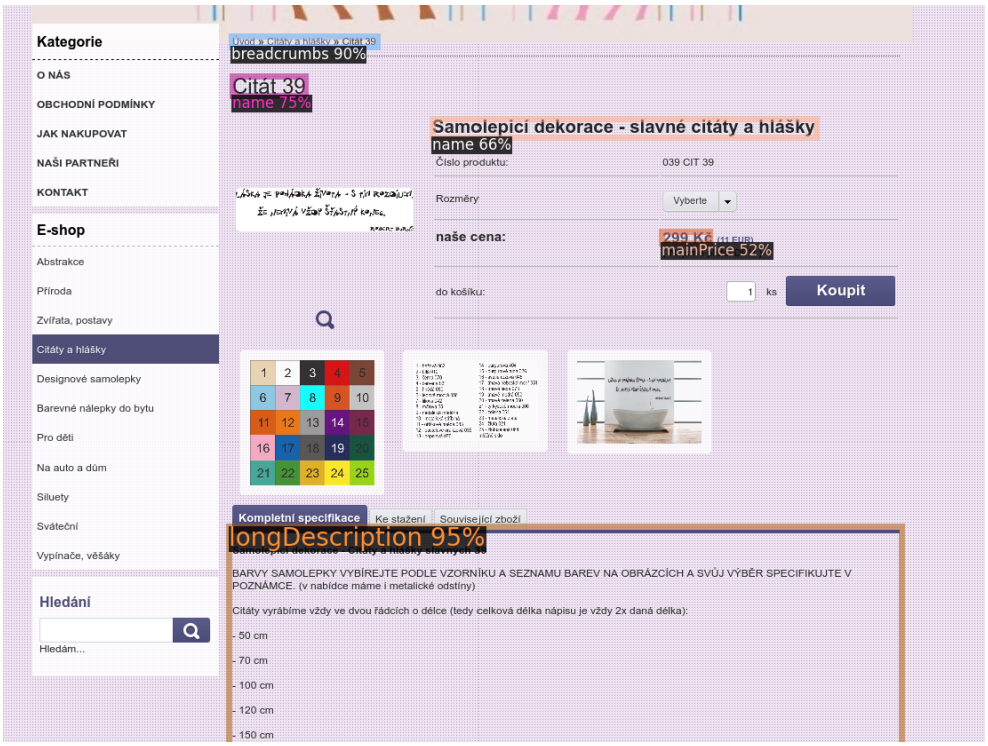
Obrázek 5.8: Web `cosmeticos.cz` s vyznačenými detekovanými oblastmi.

5.4.1.2 Výsledky e-shopu aaasamolepkynazed.cz

E-shop `aaasamolepkynazed.cz` jsme vybrali k vyhodnocení, abychom ověřili, jaké výsledky bude náš model poskytovat v případech, kdy model pro detekci oblastí špatně detekuje oblasti (e-shop `aaasamolepkynazed.cz` měl druhý nejhorší výsledek při vyhodnocování modelu pro detekci oblastí). Na obrázku 5.9 můžeme pozorovat několik nedostatků při detekování oblastí: prvním nedostatkem je absence oblasti pro obrázek, druhým nedostatkem jsou pak 2 oblasti pro název produktu. Jak můžeme vidět v extrahovaných datech na ukázce kódu 14, našemu algoritmu se povedlo druhý nedostatek vyřešit – správně vybral název s větší pravdepodobností, byť v tomto případě je nejednoznačné, co je skutečný název produktu.

```
{
  "longDescription":
  ↪  "Samolepicí dekorace - Citáty a hlášky slavných 39...",
  "category": [
    "Úvod",
    "Citáty a hlášky"
  ],
  "name": "Citát 39",
  "mainPrice": "299 Kč (11 EUR)"
}
```


Ukázka kódu 14: Extrahovaná data z webu `aaasamolepkynazed.cz`, uvedeného na obrázku 5.9. Výpis jsme pro lepší přehlednost zkrátili (zkrácené pasáže jsme nahradili třemi tečkami).



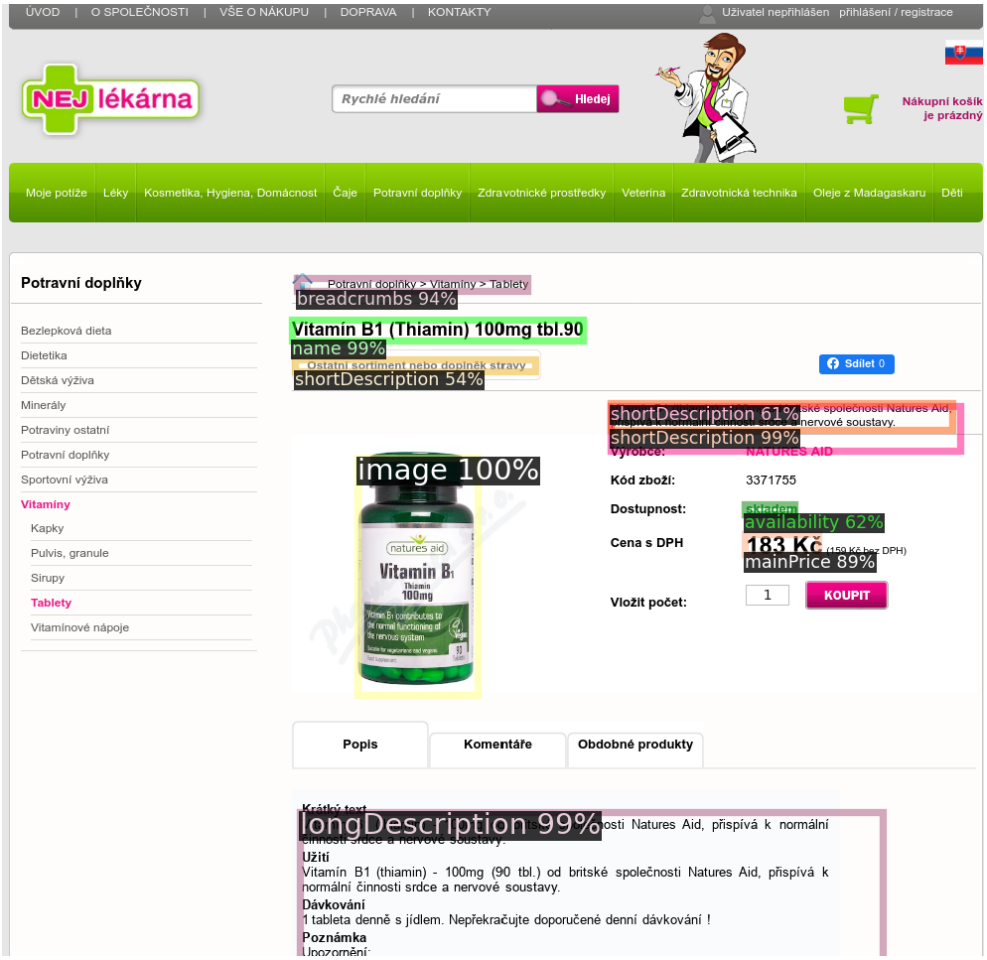
Obrázek 5.9: Web `aaasamolepkynazed.cz` s vyznačenými detekovanými oblastmi.

5.4.1.3 Výsledky e-shopu nej-lekarna.cz

E-shop `nej-lekarna.cz` jsme vybrali k vyhodnocení, abychom demonstrovali slabá místa našeho modelu. V tomto případě model pro detekci objektů detekoval oblasti správně, jak můžeme vidět na obrázku 5.10, ale náš algoritmus nedokázal oblasti správně přiřadit k HTML elementům. Výsledek extrakce uvádíme na ukázce kódu 15, kde můžeme pozorovat, že výsledkem extrakce je celá textová reprezentace stránky. Příčinou špatné extrakce je nestandardně naprogramovaná šablona e-shopu – většina HTML elementů na stránce má záporné souřadnice a správně by se neměly zobrazit (u ostatních e-shopů jsme na tento problém nenarazili). Tento příklad názorně ukazuje slabé místo našeho modelu, a to závislost na struktuře HTML kódu. Algoritmus pro propojení detekovaných oblastí s HTML elementy dokáže propojit pouze konkrétní elementy, pokud bude detekovaná oblast obsahovat více elementů, nedokáže si s tím náš algoritmus poradit a vybere element s nejlepší shodou.

```
{
  "longDescription":
  ↪  "NavigaceÚvodO společnostiAktualityKontaktyVše o...",
  "name":
  ↪  "Vitamín B1 (Thiamin) 100mg ... .fb_hidden{ position:abs...",
  "shortDescription":
  ↪  "Vitamín B1 (Thia ... .fb_hidden{position:abs...",
  "category": [
    "ÚVOD",
    "O SPOLEČNOSTI",
    "...",
    "Reklamační řád",
    "Rady při nákupu",
    "Registrace",
    "Přihlášení",
    "Zapomenuté heslo",
    "e-shop internetové lékárny"
  ],
  "mainPrice": "Vitamín B1 (Thiamin) 100mg tbl...",
  "availability": "Vitamín B1 (Thiamin) 100mg tbl..."
}
```

Ukázka kódu 15: Nesprávně extrahovaná data z webu `nej-lekarna.cz`, uvedeného na obrázku 5.8. Výpis jsme pro lepší přehlednost zkrátili (zkrácené pasáže jsme nahradili třemi tečkami).



Obrázek 5.10: Web `nej-lekarna.cz` s vyznačenými detekovanými oblastmi.

5.4.1.4 Výsledky e-shopu czc.cz

E-shop `czc.cz` jsme vybrali k vyhodnocení, abychom demonstrovali zajímavé vlastnosti našeho modelu. Tento e-shop je naprogramovaný na zakázku (šablonu nesdílí více e-shopů) a my jej v datasetu nemáme. Můžeme tedy pozorovat, jakých výsledků může náš model dosáhnout, pokud bude extrahovat data z neznámého e-shopu.

Zajímavou vlastností, kterou můžeme pozorovat, je schopnost korekce detekovaných oblastí. V horní části obrázku 5.11 je oblast názvu produktu příliš velká, přičemž obsahuje i nesouvisející prvky. Náš model propojuje detekované oblasti s HTML elementy, díky čemuž zvládne zpracovat i detekované oblasti, které nejsou optimálně umístěné. V tomto případě se název produktu extrahoval správně, jak uvádíme na ukázce kódu 16.

Další vlastnost modelu, kterou můžeme pozorovat je citlivost modelu na velikost vstupního obrázku. Původní obrázek má rozměry 1910×8639 pixelů, a jak můžeme vidět na horní části obrázku 5.11 a na horní části kódu 16, výsledky extrakce nejsou příliš uspokojivé – detekované oblasti neodpovídají prvkům na stránce a extrahovaná data postrádají důležité informace. Pokud jsme stejný obrázek ořízneme na velikost 1910×2784 pixelů (spodní část obrázku 5.11 a spodní část kódu 16), predikované oblasti se skokově zlepší, stejně tak jako extrahovaná data.


```
{
  "name": "Dell Inspiron 15 (3501), černá",
  "image":
  ↪  "https://iczc.cz/bi31c090hsiqmb8bumjjhq74fc-1_1/obrazek",
  "shortDescription":
  ↪  "Prodloužená záruka 1 rok za 1 199 Kč 1 199 Kč Náh...",
  "longDescription":
  ↪  "Ve Vašem prohlížeči není aktuálně povoleno spouštění Javas..."
}

{
  "mainPrice": "14 990 Kč",
  "image":
  ↪  "https://iczc.cz/bi31c090hsiqmb8bumjjhq74fc-1_1/obrazek",
  "name": "Dell Inspiron 15 (3501), černá",
  "availability": "Skladem 5 a více kusů Kdy zboží dostanu?",
  "category": [
    "Úvodní stránka",
    "Homepage",
    "CZC.klub",
    "CZC.Klub - Odměny pro naše zákazníky",
    "PC, notebooky a software",
    "Dell",
    "Dell Inspiron 15 (3501), černá"
  ],
  "shortDescription":
  ↪  "Stylový notebook s vyváženým výkonem pro nekončí..."
}
```


Ukázka kódu 16: Extrahovaná data z webu `czc.cz`, uvedeného na obrázku 5.11.
Horní data jsou výsledkem zpracování obrázku v původní velikosti, spodní data jsou výsledkem zpracování zmenšeného obrázku.

Obrázek 5.11: Web `czc.cz` s vyznačenými detekovanými oblastmi. Horní obrázek je v původní velikosti 1910×8639 pixelů, spodní je oříznutý na velikost 1910×2784 pixelů.

5.4.2 Výsledky extrakce na e-shopech z testovací množiny

V této části prezentujeme výsledky našeho modelu na všech e-shopech z testovací množiny. Jelikož náš dataset neobsahuje extrahované informace (obsahuje pouze vyznačené oblasti pro detekci objektů), vyhodnotíme náš model na vybraných vzorcích ručně. Z každého e-shopu z testovací množiny jsme náhodně zvolili jeden vzorek, na kterém jsme provedli extrakci a výsledky jsme uvedli do tabulky 5.4.2. Z vyhodnocení jsme vyřadili kategorie extrahovaných informací, které označují varianty produktu. Rozhodli jsme tak proto, abychom nepenalizovali výsledky extrakce u e-shopů, které varianty používají (u kterých extrakce základních parametrů proběhla dobře, ale varianty se neextrahovali správně). Z výsledků modelů pro detekci objektů navíc můžeme usoudit, že bychom stejně nedosáhli uspokojivých výsledků.

Úspěšnost jednotlivých kategorií jsme vyhodnocovali podle extrahovaných informací: Pokud extrahovaná informace byla úplná a správná, vyhodnotili jsme ji jako úspěšnou extrakci. Extrakci jsme považovali za neúspěšnou v případě, že model informaci neextrahoval, extrahoval špatně (například místo ceny produktu uvedl skladovost) nebo extrahoval neúplně (popis byl příliš krátký, v kategoriích některé kategorie chyběly, apod.). Pokud model vyhodnotíme průměrně přes všechny testované e-shopy, získáme úspěšnost extrakce 77,31 %, což považujeme za velice uspokojivý výsledek.

Průměrnou úspěšnost pro jednotlivé kategorie uvádíme v tabulce 5.4.2. Nejúspěšněji extrahovanou kategorií jsou obrázky s průměrnou úspěšností 96,43 % (je to dáno úspěšnou detekcí objektů a speciálním zpracováním v našem algoritmu). Nejhůře dopadla kategorie dostupnosti produktů s úspěšností 56,52 %, což je dáno především velkou rozmanitostí v používaných textových repreznatacích pro uvedení dostupnosti.

Kategorie	**Úspěšnost kategorie**
Název	85,71 %
Cena	78,57 %
Dostupnost	56,52 %
Krátký popis	70,59 %
Dlouhý popis	79,17 %
Kategorie	64,29 %
Obrázek	96,43 %

Tabulka 5.3: Průměrná úspěšnost extrakce podle jednotlivých kategorií. Vyhodnoceno na e-shopech z testovací množiny.

E-eshop	Název	Cena	Dostupnost	Krátký popis	Dlouhý popis	Kategorie	Obrázek	Úspěšnost extrakce
aaasamolepkynazed.cz	1	0	–	–	1	1	1	80,00 %
adelo.cz	1	1	–	1	–	1	1	100,00 %
bici.cz	1	1	0	1	1	1	1	85,71 %
chcigril.cz	1	1	1	1	–	1	1	100,00 %
cocochocokeratin.sk	1	1	1	1	1	1	1	100,00 %
cosmeticos.cz	1	1	0	0	1	1	1	71,43 %
drakkaria.cz	1	1	–	–	1	1	1	100,00 %
ebal.cz	1	1	1	–	1	1	1	100,00 %
ebubak.cz	1	1	1	1	1	1	1	100,00 %
e-cs.cz	1	1	0	0	1	1	1	71,43 %
e-manikury.cz	1	1	–	0	1	0	1	66,67 %
m.e-manikury.cz	0	1	1	–	1	1	1	83,33 %
fesakov.cz	1	1	0	0	0	0	1	42,86 %
fesnakocka.cz	1	1	0	1	0	0	1	57,14 %
gsm-market.cz	1	1	0	1	–	1	1	83,33 %
healer.cz	1	1	1	1	0	1	1	85,71 %
kamikava.cz	1	1	1	1	1	1	1	100,00 %
nakupzdomu.eu	1	0	1	–	1	1	1	83,33 %
nej-lekarna.cz	0	0	0	0	0	0	0	0,00 %
nejlepsirumy.cz	0	0	0	–	1	0	1	33,33 %
nemrznoucismesi.cz	0	0	0	1	–	1	1	50,00 %
pazba.cz	1	1	1	–	1	0	1	83,33 %
pohary.com	1	0	1	1	0	1	1	71,43 %
prodort.cz	1	1	1	–	1	1	1	100,00 %
profimed.cz	1	1	1	1	1	0	1	85,71 %
svet-her.cz	1	1	1	–	1	0	1	83,33 %
svet-hier.sk	1	1	0	–	1	0	1	66,67 %
wulflund.com	1	1	–	–	1	0	1	80,00 %

Tabulka 5.4: Výsledky extrakce pro e-shopy z testovací skupiny. 1 značí úspěšnou extrakci, 0 značí neúspěšnou extrakci (data buď chyběla, nebo neobsahovala správné hodnoty). Pomlčka značí případy, kdy e-shop daný atribut nepoužívá.

5.4.3 Vyhodnocení modelu pro extrakci informací

Námi natrénované modely *Faster R-CNN* pro detekci objektů dosahovali průměrné přesnosti AP_{50} 67,5 % a 64,91 % (modely s páteřními sítěmi *ResNet-101* a *ResNet-50*), model *Faster R-CNN* s páteřní sítí *ResNeXt-101* dosáhl dokonce průměrné přesnosti AP_{50} 71,21 %. Modely *RetinaNet* dosahovaly menší průměrné přesnosti AP_{50}, konkrétně 64,23 % pro model s páteřní sítí *ResNet-101* a 64,91 % pro model s páteřní sítí *ResNet-50*, ale jejich trénování zabralo zhruba poloviční čas oproti modelům *Faster R-CNN*. Pro srovnání, nejlepší výsledek průměrné přesnosti AP_{50} detekce objektů na datasetu COCO je 77 %[50].

Model pro extrakci informací dosáhl průměrné úspěšnosti extrakci základních parametrů 77,31 %, což považujeme za velice uspokojivý výsledek. Naopak neuspokojivého výsledku jsme dosáhli při zpracování e-shopů s variantami, kde je velký prostor pro zlepšení. Model můžeme dále vylepšovat ve dvou rovinách – zlepšováním modelů pro detekci objektů, nebo vylepšováním algoritmu na extrakci nalezených oblastí. My jsme pro extrakci informací využívali propojení s HTML elementy, což má výhodu v podobě větší tolerance k detekovaným oblastem, ale výsledná extrakce je závislá na struktuře HTML kódu. Jednou z možných alternativ může být řešení extrakce informací pomocí nástrojů pro optické rozpoznávání textu (OCR).

I přes uspokojivé výsledky musíme zmínit hlavní nevýhodou našeho modelu oproti jiným způsobům extrakce informací, a tou je náročnost na systémové prostředky, jelikož v našem modelu využíváme hluboké konvoluční sítě, které vyžadují velký výpočetní výkon. Pro zpracovávání velkého množství stránek náš model nebude nejvhodnější, zvláště když uvážíme existenci strukturovaných metadat, jejichž zpracování je několikanásobně rychlejší. Nicméně si myslíme, že náš model může být úspěšnou alternativou, vhodnou zejména pro zpracování stránek, které strukturovaná metadata neobsahují.

Závěr

V této práci jsme se zabývali možnostmi extrakce informací z webu pomocí strojového zpracování obrazové podoby stránek. Cíli práce bylo provedení rešerše existujích přístupů pro extrakci informací z webových stránek, čemuž se věnujeme v kapitole 1, a rešerše přístupů pro strojové zpracování obrázků, kterým se věnujeme v kapitole 3. Dalšími cíli bylo navrhnutí způsobu zachytávání obrazové podoby stránky, nalezení nebo vytvoření datasetu a navrhnutí modelu pro extrakci informací. Těmto cílům se věnujeme v kapitole 4. Posledním cílem práce bylo provedení experimentů, které jsme popsali v kapitole 5.

Při implementaci nástroje pro tvorbu obrazové podoby stránek jsme odhalili a vyřešili spoustu problémů, které se vážou k tvorbě obrazové podoby stránky a část práce zabývající se tímto nástrojem může sloužit jako příručka pro každého, kdo bude řešit problém zachytávání obrazové podoby stránek.

V rámci této práce jsme vytvořili obrázkový dataset produktových stránek, čítající 3 556 vzorků z 375 e-shopů. Tento dataset obsahuje označené oblasti s produktovým názvem, cenou, dostupností, kategorií (v podobě drobečkové navigace) a obrázku. V případě, že produktová stránka obsahuje více variant produktu, jsou označené i název, cena a dostupnost varianty.

Modely, které jsme natrénovali pro detekci objektů dosáhly uspokojivých výsledků, kdy nejlepší model dosáhl průměrné přesnosti AP_{50} 71,21 %. Natrénované modely jsme pak využili společně s námi navrženým algoritmem pro vytvoření modelu pro extrakci strukturovaných informací z produktových stránek, který jsme otestovali na 28 e-shopech, při čemž jsme naměřili 77,31 % úspěšnost extrakce.

Možná budoucí rozšíření této práce by mohla spočívat v aplikování stejných principů i pro jiné úlohy zpracovávání webových stránek, například vytvoření klasifikátoru, který na základě obrazové podoby určí, co daná stránka zobrazuje (například článek na blogu nebo produktová stránka). Dalším možným rozšířením práce je rozvoj datasetu, například doplnění anotací jednotlivých částí stránek, které se pak využijí k detekci a klasifikaci různých komponent webu, jako jsou navigační lišty nebo reklamní okna.

Literatura

[1] Understanding the components :: Documentation for Selenium. c2021. Dostupné z: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/

[2] Activation Functions: A Short Summary. c2018. Dostupné z: https://medium.com/hyunjulie/activation-functions-a-short-summary-8450c1b1d426

[3] A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. c2018. Dostupné z: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[4] Understanding Deep Self-attention Mechanism in Convolution Neural Networks. c2020. Dostupné z: https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251

[5] CS231n Convolutional Neural Networks for Visual Recognition. c2021. Dostupné z: https://cs231n.github.io/convolutional-networks/

[6] R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. c2018. Dostupné z: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[7] Ren, S.; He, K.; Girshick, R. B.; aj.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*, ročník abs/1506.01497, 2015, 1506.01497. Dostupné z: http://arxiv.org/abs/1506.01497

[8] Lin, T.; Dollár, P.; Girshick, R. B.; aj.: Feature Pyramid Networks for Object Detection. *CoRR*, ročník abs/1612.03144, 2016.

[9] Redmon, J.; Divvala, S. K.; Girshick, R. B.; aj.: You Only Look Once: Unified, Real-Time Object Detection. *CoRR*, ročník abs/1506.02640, 2015.

[10] Lin, T.; Goyal, P.; Girshick, R. B.; aj.: Focal Loss for Dense Object Detection. *CoRR*, ročník abs/1708.02002, 2017.

[11] He, K.; Zhang, X.; Ren, S.; aj.: Deep Residual Learning for Image Recognition. *CoRR*, ročník abs/1512.03385, 2015.

[12] He, K.; Zhang, X.; Ren, S.; aj.: Identity Mappings in Deep Residual Networks. *CoRR*, ročník abs/1603.05027, 2016.

[13] Xie, S.; Girshick, R. B.; Dollár, P.; aj.: Aggregated Residual Transformations for Deep Neural Networks. *CoRR*, ročník abs/1611.05431, 2016.

[14] Hand, D.; Mannila, H.; Smyth, P.: *Principles of data mining*. Massachusetts: MIT Press, vyd. 1 vydání, 2001, ISBN 026208290x.

[15] Liu, B.: *Web data mining*. Berlin: Springer, first edition vydání, c2007, ISBN 978-3-540-37881-5.

[16] Markov, Z.; Larose, D. T.: *Data mining the web*. Hoboken: John Wiley & Sons, 2007, ISBN 978-0-471-66655-4.

[17] HTTrack. c2021. Dostupné z: `https://www.httrack.com/`

[18] GNU Wget. c2017. Dostupné z: `https://www.gnu.org/software/wget/`

[19] *wget(1) - Linux man page*. 2020.

[20] WebDriver. c2020. Dostupné z: `https://www.w3.org/TR/webdriver/`

[21] SeleniumHQ Browser Automation. c2021. Dostupné z: `https://www.selenium.dev/`

[22] SeleniumHQ: Podporované prohlížeče. c2021. Dostupné z: `https://www.selenium.dev/documentation/en/getting_started_with_webdriver/browsers/`

[23] Beautiful Soup. c1996-2021. Dostupné z: `https://www.crummy.com/software/BeautifulSoup/`

[24] Readability 1.0. c2021. Dostupné z: `https://github.com/masukomi/arc90-readability`

[25] Readability.js. c2021. Dostupné z: `https://github.com/mozilla/readability`

[26] Dragnet. c2021. Dostupné z: https://github.com/dragnet-org/dragnet

[27] Python-Goose - Article Extractor. c2021. Dostupné z: https://github.com/grangier/python-goose

[28] Understand how structured data works. c2021. Dostupné z: https://developers.google.com/search/docs/guides/intro-structured-data

[29] Schema.org. c2021. Dostupné z: https://schema.org/

[30] microdata-node. c2020. Dostupné z: https://www.npmjs.com/package/microdata-node

[31] RDFa Core 1.1 - Third Edition. c2015. Dostupné z: https://www.w3.org/TR/rdfa-core/

[32] RDFa Lite 1.1 - Second Edition. c2015. Dostupné z: https://www.w3.org/TR/rdfa-lite/

[33] JSON for Linking Data. c2021. Dostupné z: https://json-ld.org/

[34] Goodfellow, I.; Bengio, Y.; Courville, A.: *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[35] Girshick, R.; Donahue, J.; Darrell, T.; aj.: Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, s. 580–587, doi:10.1109/CVPR.2014.81.

[36] Girshick, R. B.: Fast R-CNN. *CoRR*, ročník abs/1504.08083, 2015, 1504.08083. Dostupné z: http://arxiv.org/abs/1504.08083

[37] Uijlings, J.; Sande, K.; Gevers, T.; aj.: Selective Search for Object Recognition. *International Journal of Computer Vision*, ročník 104, 09 2013: s. 154–171, doi:10.1007/s11263-013-0620-5.

[38] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, ročník 25, editace F. Pereira; C. J. C. Burges; L. Bottou; K. Q. Weinberger, Curran Associates, Inc., 2012. Dostupné z: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[39] Huang, G.; Liu, Z.; Weinberger, K. Q.: Densely Connected Convolutional Networks. *CoRR*, ročník abs/1608.06993, 2016.

[40] Padilla, R.; Netto, S. L.; da Silva, E. A. B.: A Survey on Performance Metrics for Object-Detection Algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, s. 237–242.

[41] Rezatofighi, H.; Tsoi, N.; Gwak, J.; aj.: Generalized Intersection over Union. June 2019.

[42] Meet your Real Users. Dostupné z: `https://rating.gemius.com/`

[43] The Chromium Projects. c2021. Dostupné z: `https://www.chromium.org/`

[44] Firefox – buďte na internetu v bezpečí díky produktům, které dbají na vaše soukromí — Mozilla. c2021. Dostupné z: `https://www.mozilla.org/cs/firefox/`

[45] LabelImg - a graphical image annotation tool. c2021. Dostupné z: `https://github.com/tzutalin/labelImg`

[46] labelme - Image Polygonal Annotation with Python. c2021. Dostupné z: `https://github.com/wkentaro/labelme`

[47] OpenCV on Wheels. c2021. Dostupné z: `https://pypi.org/project/opencv-python/`

[48] Wu, Y.; Kirillov, A.; Massa, F.; aj.: Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[49] Benchmarks — detectron2 0.4 documentation. c2020. Dostupné z: `https://detectron2.readthedocs.io/en/latest/notes/benchmarks.html`

[50] COCO - Common Objects in Context. c2021. Dostupné z: `https://cocodataset.org/`

[51] Requests: HTTP for Humans. c2021. Dostupné z: `https://requests.readthedocs.io/en/master/`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CNN Konvoluční neuronová síť

DOM Document Object Model

FL Focal Loss

FP False positive

FPN Feature pyramid network

GT Ground truth

IoU Intersection over Union

KDD Knowledge discovery in data

OCR Optical Character Recognition

TN False negative

TN True negative

TP True positive

JSON-LD JavaScript Object Notation for Linked Data

JSON JavaScript Object Notation

MPL Vícevrstvá perceptronová síť

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

RPN Region Proposal Network

URL Uniform Resource Locator

Příloha B

Obsah přiloženého CD

- `README.md` ..... stručný popis obsahu CD
- `src`
  - `dataset` ..... dataset vytvořený v rámci práce
    - `train` ..... trénovací množina
    - `val` ..... testovací množina
  - `model` ..... implementace modelu
    - `extraction` ..... implementace modelu pro extrakci
    - `object-detection` ..... implementace modelu pro detekci objektů
    - `trained-models` ..... váhy naučených modelů
    - `results` ..... výsledky experimentů
  - `selenium-screenshots` .. nástroj pro tvorbu obrazové podoby stránek
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Implementace crawleru v jazyku python

Pro vytvoření crawleru v jazyku python můžeme použít nepřeberné množství knihoven. Níže uvádíme ukázkovou implementaci sekvenčního crawleru naimplementovaného pomocí knihoven *requests*[51] (pro práci s HTTP požadavky) a *Beautiful Soup*[23] (pro zpracování HTML kódu).

```
1  from bs4 import BeautifulSoup
2  import requests
3  import sqlite3
4
5  def createDatabase():
6    connection = sqlite3.connect('links.db')
7    cursor = connection.cursor()
8
9    cursor.execute('''
10   CREATE TABLE IF NOT EXISTS links
11   (
12     id integer PRIMARY KEY AUTOINCREMENT,
13     link varchar(255) NOT NULL,
14     done tinyint DEFAULT 0 NOT NULL,
15     parent integer NULL
16   );
17   ''')
18
19   cursor.execute('''
20   CREATE UNIQUE INDEX IF NOT EXISTS links_link_uindex ON links (link)
21   ''')
22
23   return connection
```

```
24
25 class Crawler:
26   def __init__(self, connection, baseUrl):
27     self.baseUrl = baseUrl
28     self.connection = connection
29     self.c = connection.cursor()
30
31   def run(self):
32     self.c.execute('''
33     INSERT OR IGNORE INTO links (id, link, parent) VALUES (?, ?, ?)
34     ''', (None, self.baseUrl, None))
35
36     while True:
37       for row in c.execute('SELECT * FROM links WHERE done = 0 ORDER BY id'):
38         print('Parsing url ' + row[1])
39
40         self.parseUrl(row[1], row[0])
41
42   def parseUrl(self, url, parent):
43
44     try:
45       r = requests.get(url, verify=True, timeout=5.000)
46       # Zkontrolujeme, že stránka vrátila správný stavový kód
47       # neexistující stránky zpracovávat nebudeme
48       if r.status_code != 200:
49         return
50
51       soup = BeautifulSoup(r.text, 'html.parser')
52
53       # Uložení kódu stránky
54
55       for link in soup.find_all('a'):
56         href = link.get('href')
57         if href[0] == '/':
58           href = url + href
59
60         if href.startswith('http'):
61           self.c.execute('''
62           INSERT OR IGNORE INTO links (id, link, parent) VALUES (?, ?, ?)
63           ''', (None, href, parent))
64
65       # Sleep()
66
67     except Exception as inst:
```

```
68        print(inst)
69        pass
70
71      finally:
72        # Nakonec aktuální stránku označíme za zpracovanou
73        self.c.execute('UPDATE links SET done = 1 WHERE id = ?', (parent, ))
74        self.connection.commit()
75
76
77
78
79  baseUrl = 'https://fit.cvut.cz'
80
81  connection = createDatabase()
82  scrapper = Scrapper(connection, baseUrl)
83  scrapper.run()
```