TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Využití autoenkodérů pro detekci anomálií v obraze

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Albert Myšák**
Vedoucí práce: Ing. Karel Paleček, Ph.D.
Konzultant: Ing. Lukáš Matějů

Liberec 2018

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Using autoencoders for anomaly detection in images

Bachelor thesis

Study programme: B2646 – Information technology
Study branch: 1802R007 – Information technology

Author: **Albert Myšák**
Supervisor: Ing. Karel Paleček, Ph.D.
Consultant: Ing. Lukáš Matějů

Liberec 2018

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2018/2019**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Albert Myšák**
Osobní číslo:	**M14000055**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Využití autoenkodérů pro detekci anomálií v obraze**
Zadávající katedra:	**Ústav informačních technologií a elektroniky**

Zásady pro vypracování:

1. Seznamte se s problematikou neuronových sítí a autoenkodérů.
2. Sestavte množinu trénovacích a testovacích dat.
3. Navrhněte algoritmus pro detekci anomálií na textiliích.
4. Vyhodnoťte úspěšnost navrženého řešení na testovacích datech.

Rozsah grafických prací:	**Dle potřeby dokumentace**
Rozsah pracovní zprávy:	**cca 30-40 stran**
Forma zpracování bakalářské práce:	**tištěná/elektronická**

Seznam odborné literatury:

[1] **Goodfellow, I., Bengio, Y., Courville, A. Deep learning. MIT Press, 2016**

[2] **Bishop, C. Pattern Recognition and Machine Learning. 2006. ISBN 13: 978-038731073**

[3] **Karpathy, A., Johnson, J., Li, F. Convolutional neural neworks for visual recognition.**

[4] **dostupné online: http://cs231n.stanford.edu/**

Vedoucí bakalářské práce:	**Ing. Karel Paleček, Ph.D.** Ústav informačních technologií a elektroniky
Konzultant bakalářské práce:	**Ing. Lukáš Matějů** Ústav informačních technologií a elektroniky
Datum zadání bakalářské práce:	**18. října 2018**
Termín odevzdání bakalářské práce:	**30. dubna 2019**

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 18. října 2018

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstrakt

Tato bakalářská práce se zabývá využitím autoenkodéru pro detekci anomálií na snímcích textílie. V práci je uvedeno strojové vidění a software pro strojové vidění využivající autoenkodéry. Dále je popsán autoenkodér a neuronové sítě. Pro otestování různých modelů autoenkodérů byla nasnímána sada 291 snímků textílie rozdělena na data trénovací, validační a testovací. V práci je popsán návrh autoenkodérů a faktory ovlivňující jejich funkčnost. Vybrané 3 modely byly vyhodnoceny na testovací sadě dvěma metodami. Nejlepších výsledků dosáhl model kombinující konvoluční i plně propojené vrstvy. Při hodnocení jednotlivých pixelů má nejlepší model TPR 0.81 při FPR 0.003. Při vyhodnocení celých snímků má nejlepší model TPR 1 a FPR 0. Výsledný algoritmus je implementován v prostředí LabView s využitím jazyka Python.

Abstract

This paper describes the use of autoencoders in an anomaly detection of images. Paper presents machine vision and machine vision software that uses autoencoders. Autoencoder and neural network is described and to test the presented models, dataset of fabric images has been acquired. Dataset has been divided into 3 parts to be: trained, validated and tested. Paper describes development of autoencoder and factors that effect results. Chosen 3 models were compared on test dataset using two methods. Best results were achieved by model, that combined convolutional and fully connected layers. Performance is measured by two metrics. Performance on every pixel of best model has TPR 0.81 with FPR 0.003. Performance on whole image is TPR 1 with FPR 0. Algorithm is implemented using LabView and Python.

Poděkování

Rád bych poděkoval panu Palečkovi za jeho čas. Také bych rád poděkoval své přítelkyni, za korekturu překlepů a mé rodině za podporu.

Obsah

Seznam zkratek 10

1 Úvod do problematiky **11**
1.1 Strojové vidění 11
1.2 Kontrola kvality v průmyslu pomocí kamerových systémů 11
1.3 Defektoskopie 13
1.4 Dostupný software pro strojové vidění 13
1.4.1 LabView 13
1.4.2 ViDi 14
1.4.3 Pekat vision 15
1.4.4 API Keras 15

2 Rešerše autoenkodérů **16**
2.1 Co to je autoenkodér 16

3 Neuronová síť a její části **18**
3.1 Obecný popis vrstev 18
3.2 Neuron 18
3.3 Vstupní vrstva 19
3.4 Plně propojená vrstva 19
3.5 Konvoluční vrstva 20
3.6 Aktivační vrstva 21
3.7 MaxPooling vrstva 22
3.8 Upsampling vrstva 22
3.9 Trénování neuronových sítí 22

4 Popis datasetu **24**
4.1 Předzpracování obrazu 24
4.2 Rozdělení snímků 26

5 Návrh modelu autoenkodéru **28**
5.1 Plně propojené vrstvy 29
5.2 Model pouze konvolučních vrstvev 29
5.3 Kombinace plně propojených a konvolučních vrstev 31

6 Vyhodnocení modelů **33**
6.1 Vyhodnocení algoritmu po pixelech 33
6.2 Vyhodnocení algoritmu na úrovni snímku 35

7 Integrace modelu pro reálný provoz v prostředí LabView **37**
7.1 LabView VI 37

8 Závěr **40**

Použitá literatura **42**

Seznam zkratek

SV	Strojové vidění
NN	Neuronová síť
CNN	Konvoluční neuronová síť
TF	TensorFlow
CNTK	Microsoft Cognitive Toolkit
LV	LabView
ROI	Region zájmu
FPGA	Programovatelné hradlové pole
PLC	Programovatelný logický automat
TUL	Technická Univerzita v Liberci
FM	Fakulta mechatroniky, informatiky a mezioborových studií
VI	Virtual Instrument

1 Úvod do problematiky

Tato kapitola představuje strojové vidění, jeho obecné využití v průmyslu a využití analýzy obrazu pro detekci vad na textilu. Dále budou popsány nástroje umožnující využití této technologie.

1.1 Strojové vidění

Definice pojmu strojové vidění (SV) se často liší, avšak vždy se jedná o rozpoznání požadované informace z obrazu výpočetním systémem. Strojové vidění se používá například při třídění červených a zelených jablek nebo nalezení poškození povrchu. Zpracování vizuálních dat je důležitou součástí nejen průmyslu, ale i v oblastech samořiditelných automobilů, obchodů bez pokladen, či sledování pohybu a identifikace osob. K rozvoji tohoto vědního oboru přispívají nízké ceny hardware: především kamer, procesorů, grafických karet a též rozvoj software od frameworků, optimalizovaných ovladačů grafických karet, až po kompletní systémy řešící strojové vidění.

Strojové vidění se dá rozdělit na části: sběr obrazu, korekce vad, detekce příznaků a detekce objektů na základě zjištěných příznaků.

Ve strojovém vidění se používají anotace. Jedná se o přidané informace ke snímkku.

1. Informace je jedna pro celý snímek. Např zdali snímek obsahuje vadnou textílii, nebo textílii bez vady.

2. Informace je pro každý pixel v obraze. Např pixely v oblastech, ve kterých je textílie vadná.

Přiložená informace může mít různou podobu např. pojmenování souboru, text v souboru, vypsané body ohraničující oblast nebo binární maska obrazu.

1.2 Kontrola kvality v průmyslu pomocí kamerových systémů

V této práci bude popsáno strojové vidění v průmyslu, aby později byla zjevná výhoda využití autoenkodéru oproti ostatním metodám analýzy obrazu.

Použití strojového vidění v průmyslu má výhody i nevýhody oproti využití kontroly lidmi. Tyto výhody i nevýhody je potřeba brát v potaz, při zvažování náhrady

lidské síly systémem strojového vidění. Faktory při posuzování vhodnosti využití strojového vidění:

- cena zavedení kontroly do provozu
- cena provozu kontroly na určité časové období
- cena při využití dalších stanic
- přesnost a opakovatelnost kontroly
- náročnost úpravy kontrolního procesu

Porovnání kontroly kvality (popraskání) a kvantity chleba strojovým viděním oproti lidské kontrole:

- Cena zavedení kontroly do provozu - pořizovací náklady kamerového systému jsou v řádu statisíců, až jednotek milionů korun. Největší část nákladů pokrývá platy aplikačních inženýrů, kteří navrhují kontrolní stanici a její software. Další část nákladů pokrývá hardware a softwarové licence. U operátora výroby je pořizovací náklad řádově nižší, často se jedá o nástupní bonus v řádu desetitisíců korun a režijní náklady na nového zaměstnance.
- Cena provozu kontroly na určité časové období - náklady na provoz jsou u systému strojového vidění minimální. Údržba systému spočívá často pouze v čištění objektivů, či nahrazení nefunkční části. Při použití průmyslových komponentů, je však životnost součástí systému v řádu desítek let. Například kamera Imperx má udávanou životnost cca 77 let.
- Cena při využití dalších stanic - cena na rozšíření o novou kontrolní stanici je u systému strojového vidění často pouze cena za hardware. U operátorů výroby je potřeba najmout a platit dalšímu zaměstnanci. Obecně tedy platí, že systémy strojového vidění se vyplatí při velkém objemu výroby u výrobků vyráběných po delší období.
- Přesnost a opakovatelnost kontroly - stroj má výhodu v konzistentním výstupu. Při stejném vstupu vytvoří počítačový program vždy stejný výstup. Oproti tomu člověk při změně nálady může dva stejné bochníky chleba vyhodnotit různě. Jeden den jako správně vyrobený, avšak jiný den jako chybně vyrobený. Při využití lidské kontroly bývá kvalita kontroly nekonzistentní.
- Náročnost úpravy kontrolního procesu - člověk je snáze ”programovatelný”. Pokud kontrolu linky v pekárně požádáme o vyřazení popraskaných bochníků, člověk si přirozeně z mnohaleté životní zkušenosti uvědomí jak popraskaný chléb vypadá a ihned dokáže provádět kontrolu. V porovnání s tím vytvořit systém na detekci popraskaných bochníků chleba vyžaduje mnoho kroků od přesného popsání problému, přes navrhnutí snímací soustavy a navrhnutí algoritmu až po jeho implementaci. Tato doba je mnohem delší, než zaškolení operátora výroby, jaké bochníky má vyřazovat.

Strojovým viděním lze měřit vzálenosti, úhly, velikost plochy, počítat kusy, počítat části, detekovat přítomnost objektů, detekovat vadné výrobky a další variace těchto úloh.

1.3 Defektoskopie

Častým požadavkem výrobního průmyslu bývá testování výrobků na vady. Některé z nich nemusí být přímo viditelné, ale mohou se projevovat jistými fyzikálními vlastnostmi, které mohou být pozorovatelné kamerou. Hledání příznaků těchto vad se nazývá "nedestruktivní testování" (NDT), neboli "Defektoskopie". Výhoda nedestruktivního testování spočívá v rychlosti odhalení závady (vyfotit výrobek je rychlejší než rozřezat kovový výrobek a provést chemickou analýzu) a také nepoškozením výrobku, který může být použit pro jeho účel. Pro defektoskopii je klíčem nalezení vhodných přiznaků, které vhodně popisují vadu. Při hledání příznaků vad je potřeba porovnat výskyt příznaku na vadných a správných výrobcích. Pro defektoskopii je ideální příznak takový, který se vyskytuje pouze u špatných výrobků, ale nevyskytuje se u výrobků správně vyrobených.

Postupy popsané v této práci mohou být aplikovány pro defektoskopii.

1.4 Dostupný software pro strojové vidění

V této části je prezentován vybraný software pro strojové vidění a jsou představeny dva nástroje, které využívají autoenkodér pro průmyslové zpracování obrazu. Dále je popsáno API Keras pro návrh neuronové sítě.

1.4.1 LabView

Software LabView[1] nabízí dlouholetou integraci počítačových zařízení v průmyslu. LabView umožnuje vytvořit program pomocí grafického programování. Spojováním bloků lze vytvořit měřící a řídící hardware, analyzovat data a prezentovat výsledky. Licence LabView je oproti konkurenci levná, podpora hardware je široká a integrace je velmi snadná, proto je tento nástroj vhodný pro použití v průmyslových aplikacích. Výsledek měření lze snadno dále komunikovat s PLC, či například využít k ovládání servomotorů, které roztřídí výrobky na špatné a dobré. Množství kompatibilního poskytovaného hardware (průmyslové počítače, světla, kamery, senzory, rozvaděče, elektromotory a další), udělalo z LabView často používatný nástroj v průmyslové výrobě. Tvůrce tohoto software, společnost National Instruments nabízí mnoho modulů pro svůj LabView ekosystém, mezi které patří i "Vision and Motion" modul, který poskytuje nástroje pro strojové vidění a pomocí kterého lze vykonávat tyto operace:

- nastavit kameru a akvizici snímků
- načítat, manipulovat a ukládat snímky ve vhodných formátech pro fotografie

- zobrazovat výsledky analýzy obrazu - práce s okny, management obrazovek, použití nástroje pro prohlížení snímků
- definovat a použít ROI - region zájmů
- vykreslit údaje do obrazu
- kalibrovat obraz
- pracovat s videem v reálném čase
- nastavit FPGA, aby provádělo analýzu obrazu
- zpracovávat obraz pomocí: úpravy jasové složky obrazu, filtrace, morfologie, analýzy částic, úprav barevných kanálů, práce ve frekvenční oblasti, nalezení hran
- nalezt vzor, klasifikovat obraz, analyzovat konturu, porovnat obraz se 'zlatým vzorem', opticky rozpoznat text, analyzovat geometrii, sledovat objekt, párovat příznaky

Ve výčtu možností LabView se nenachází modul zpracování obrazu pomocí neuronových sítí, proto bylo potřeba najít jiný software, který neuronové sítě využívá.

1.4.2 ViDi

Software ViDi je první nástroj na učení hlubokých neuronových sítí pro průmyslovou automatizaci. ViDi pomocí grafického uživatelského rozhraní umožnuje detekci anomálií (ta je založená na autoenkodéru), detekci objektů, klasifikaci objektů a strojové čtení textu. K natrénování neuronové sítě má ViDi vhodné rozhraní, kam je možno nahrát snímky, anotovat je, rozdělit je na snímky trénovací, testovací a validační. ViDi má 3 nástroje (učení s učitelem, učení bez učitele, OCR), které za sebe lze řetězit.

ViDi z logických důvodů přesně neříká jak funguje a nepopisuje algorimy, které využívá. Nelze tedy výstup tohoto programu porovnat s výstupem této práce vytvořené pomocí jiného nástroje, jelikož z ViDi ani nelze získat výstup ve formátu, který by umožnoval statistické vyhodnocení. Výstupem ViDi je ohodnocení pro každý snímek od 0 do 100. U této hodnoty lze nastavit ruční hranici, která snímky rozdělí do tří kategorií: "OK"/"NOK"/"nelze rozhodnout". Shrnutí ukáže, kolik snímků patří do každé kategorie, a kolik snímků by v kategorii být mělo dle anotace. Po porovnání s anotací program ve svém GUI zvýrazní špatně vyhodnocené snímky. ViDi bylo odkoupeno společností Cognex, která obaluje ViDi svým software zahrnujícím podporu hardware a řídících procesů. Cena jedné licence na jeden počítač je dle využítitého počtu nástrojů mezi 10000-20000 euro.

1.4.3 Pekat vision

Pekat [2] je brněnský software podobný ViDi. Hlavním rozdílem je nižší cena - kolem 100000 kč za licenci. Jako ViDi Obsahuje anotační nástroj, bohužel nelze vidět počet správně a špatně vyhodnocených snímků. Obsahuje několik nástrojů (klasifikace, detekce anomálií, kontrola povrchu), které lze řetězit jak paralelně, tak sekvenčně. Lze vložit Python kód, který může manipulovat se vstupem nebo výstupem nástrojů. Otevřený je datový formát, ve kterém si bloky předávají vstupy a výstupy. Pro klasifikaci jsou využívány veřejně dostupné předtrénované neuronvé sítě jako Resnet V2 [3] vytvořený pomocí API Keras.

1.4.4 API Keras

Pro zpracování dat neuronovou sítí existují knihovny a frameworky, které optimalizují použité výpočty a algoritmy. Těchto knihoven a frameworků vzniklo v posledních letech mnoho. Nabízejí různé funkce, a práce s nimi je odlišná. API Keras [4] vzniklo jako vysokoúrovňové rozhraní sjednocující práci s TensoFlow [5], CNTK a Theano. Toto API je implementované v jazyku Python. Keras umožňuje s minimálním počtem řádků kódu vytvořit neuronovou síť se vším, co je popsáno v kapitole Neuronová síť a její části.

2 Rešerše autoenkodérů

Cílem této kapitoly je definovat autoenkodér, popsat jeho výhody i nevýhody a odůvodnit proč je vhodný pro detekci anomálií.

2.1 Co to je autoenkodér

Autoenkoding je metoda komprimace dat, která je:

1. ztrátová
2. určená na pevně daný vstupní formát dat
3. nastavena s parametry, které jsou určeny daty

Termín Autoenkodér je používán pro označení neuronové sítě o dvou funkcích využívajících vlastností autoenkodingu. První funkce $f_1(\mathbf{x}) = \mathbf{z}$ zkomprimuje (angl. encode) vstupní data. Druhá funkce $f_2(\mathbf{z}) = \mathbf{y}$ dekomprimuje (angl. decode) zkomprimovaná data. Výstup první funkce je vstupem druhé funkce. Cílem výstupu druhé části $\mathbf{y}$ je dosáhnout vstupních dat části první $\mathbf{x}$.

Datový typ a velikost vstupu první funkce f_1 je pevně dán při návrhu autoenkodéru. Pomocí komprimace PNG lze zkomprimovat obraz o šířce 100 pixelů i 200 pixelů. Pokud vytvoříme autoenkodér pro obraz šířky 100 pixelů, nebude moci přijmout obraz šířky 200 pixelů.

Komprimace PNG je lidmi navržený algoritmus, oproti tomu komprimace pomocí autoenkodéru je navržená z trénovacích dat. Stejný autoenkodér při různých trénovacích datech bude provádět jinou komprimaci. Pokud tedy naučíme autoenkodér komprimovat snímky textílie, na kterých je text, nebude schopen komprimovat snímky letadel. Toto je zásadní vlastnost, pro kterou byl zvolen autoenkodér v této práci jako detektor anomálií. Při správně navrženém modelu autoenkodéru bude autoenkodér schopen správně zkomprimovat a dekomprimovat pouze snímky, na kterých byl natrénován a jim velmi podobné. Pro natrénování autoenkodéru není potřeba na snímcích anotovat (označovat) vadné oblasti. Tato vlastnost činí autoenkodér výborným nástrojem pro velké datasety, kde by bylo časově náročné snímky anotovat.

Autoenkodéry jsou zřídka používány v praxi, jelikož jejich hlavní funkcí je komprimace. Lidmi navržené algoritmy v současnosti komprimují univerzálněji (jak rozměr dat, tak jejich obsah) a nevyžadují natrénování. Při velkém omezení obsahu

vstupních dat je možno dosáhnout lepší komprimace pomocí autoenkodéru, než pomocí lidmi navržených algoritmů.

Autoenkdéry lze použít k odstranění šumu [6], nebo pro redukci dimenze dat [7].

Aktivní výzkum autoenkodérů probíhá v oblasti 3D deformace modelů [8] a jiných velmi specificky zaměřených využití.

Zopakujme, že autoenkodér se skládá ze dvou funkcí, první komprimuje data (také nazývána enkodér) a druhá funkce zkomprimovaná data dekomprimuje (také nazývaná dekodér). Mezi těmito funkcemi tedy vznikne vektor $\mathbf{z}$. Velikost tohoto vektoru udává, jak moc budou data zkomprimována. Navržení velikosti tohoto vektoru je jedním ze zásadních bodů návrhu celého autoenkodéru. Čím menší je tento vektor $\mathbf{z}$, tím více budou data zkomprimována, a tím více musí být ostatní části autoenkodéru schopny generalizovat data.

3 Neuronová síť a její části

Cílem této kapitoly je popsat části, ze kterých se neuronové sítě skládají. Tyto části budou využity při návrhu autoenkodéru.

První (enkodér) i druhá (dekodér) funkce autoenkodéru je neuronová síť. Popis následujících částí tedy platí pro enkodér i dekodér.

Neuronová síť v informatice, je matematický model, který se fungováním připodobňuje lidskému mozku. Lidský mozek se skládá z mnoha na sebe napojených částí, takzvaných neuronů, odtud tedy název a základní vlastnosti tohoto matematického modelu. Mnoho biologických principů fungujících v mozku dostupné nástroje na vytváření programů nesimulují, proto je vhodné se vyhnout bližsímu spojování lidského mozku a počítačového programu. Dále bude pojmem 'neuronové sítě' v této práci označen pouze model počítačového programu. Tato práce se zabývá pouze dopřednými neuronovými sítěmi [9].

Další detaily teorie jsou v knize [10].

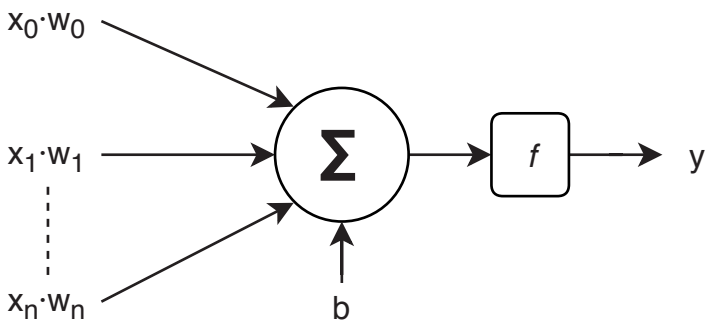
Topologie neuronové sítě a její parametry se nazývají modelem, tento model lze natrénovat (nalezení parametrů sítě) a uložit. Pokud následně chceme využít neuronovou síť, načteme model a jeho parametry.

3.1 Obecný popis vrstev

Neuronové sítě se skládají z vrstev, které se skládají z neuronů. Neuron 3.1 je matematická funkce, která bude popsána v náslefující sekci Neuron . Vstupem první vrstvy je matice obrazových dat. Vstupem dalších vrstev je výstup vrstev předchozích. Výstup poslední vrstvy je výstupem celé neuronové sítě. Jisté varianty neuronových sítí využívají napojení nejen na vrstvu předchozí nebo vrstvu následující, ale i libovolné vrstvy, které předcházejí nebo následují. Využití tohoto složitějšího napojení je mimo rozsah této práce. Skrytou vrstvou bude nazývána vrstva, která není na vstupu ani na výstupu sítě. Jelikož se tato práce zabývá zpracováním obrazu, budou představeny pouze vrstvy pro práci s obrazem. Vrstvy se dělí podle zapojení neuronů a matematické funkce, kterou používají.

3.2 Neuron

Neuron je matematická funkce. Vstup neuronu určuje typ vrstvy, ve které se neuron nachází. Vstupem mohou být krom výstupů předchozí vrstvy i tzv. váhy. Váha je



Obrázek 3.1: Model umělého neuronu $\mathbf{y} = f(\mathbf{x} \cdot \mathbf{w} + \mathbf{b})$ který má na vstupu vektor x velikosti $n+1$. Vstupy vynásobí příslušnými vahami w a výsledky sečte. Výsledek součtu je vstupem aktivační funkce f, která vypočte výstup neuronu y.

trénovatelný parametr, který je přiřazen právě každé dvojici výstup-vstup. Dalším vstupem může být tzv. bias, což je taktéž trénovatelná konstanta, která se přičte k hodnotě výstupu matematické operace. Je jedna pro jeden neuron a pokud nebude zmíněno jinak, bude vždy vstupem neuronu. Trénovatelnost konstant znamená změnu hodnoty konstanty algoritmem zpětné propagace chyb [11]. Neuron poskytuje svůj výstup jako vstup na jeden nebo více neuronů v další vrstvě. Model umělého neuronu je zobrazen na 3.1.

3.3 Vstupní vrstva

Tato vrstva je specifická pro API Keras. Vstupem je matice obrazových dat. Výstupem je identická matice obrazových dat. Vrstva má parametry: počet sloupců w vstupní matice, počet řádků h vstupní matice a hloubku c vstupní matice. Při vkládání vstupních dat modelu neuronové sítě si Keras zkontoluje, zdali rozměry vstupu odpovídají rozměrům vstupní vrstvy.

3.4 Plně propojená vrstva

Plně propojená vrstva má formu

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{W} + \mathbf{b}$$

kde $\mathbf{x}$ je vstupní vektor vrstvy o rozměru k, $\mathbf{W}$ je trénovatelná matice vah s rozměry $k \times g$, $\mathbf{y}$ je výstupní vektor a g je počet neuronů ve vrstvě.

Tyto vrstvy jsou využívány, pokud chceme aby každý neuron zpracovával informace z celé předchozí vrstvy. Plně propojené vrstvy se při zpracování obrazu používají jako finální vrstvy po vrstvách konvolučních.

3.5 Konvoluční vrstva

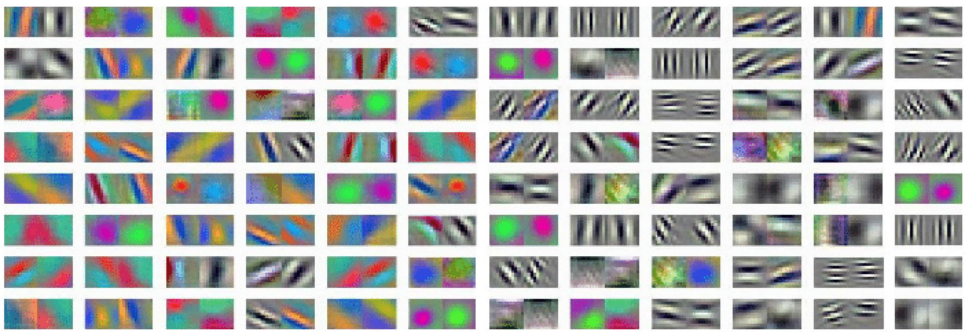
Dvourozměrná konvoluce pro každý výstupní pixel:

$$z_{uv} = \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} \mathbf{w_{ij}} \mathbf{x_{u+i,v+j}} + b$$

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

Váhy konvoluční vrstvy je vhodné si představit jako filtr. Filtr je posouván nad vstupem a každá konvoluce vypočte jednu hodnotu. Posouváním tedy dostaneme pro každý filtr dvourozměrné pole hodnot. Parametry vrstvy jsou:

1. velikost filtru w, h: často používané rozměry jsou 11x11, 5x5, 3x3, 1x1.
2. počet filtrů k: jedna vrstva může mít více filtrů, například AlexNet [12] má v první vrstvě filtrů 96 zobrazených na 3.2. Tyto filtry se při vizualizaci hodnot do RGB prostrotu podobají Gáborovým filtrům.



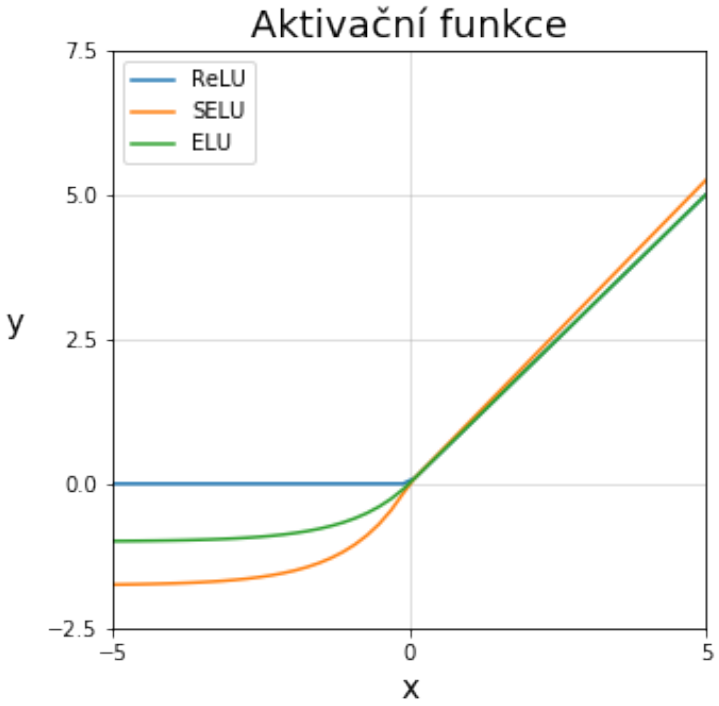
Obrázek 3.2: Vizualizované první filtry sítě AlexNet [12]

3. velikost kroku (angl. stride): konvoluci nemusíme provádět nad každou možnou částí vstupu, avšak můžeme posouvat filtr o libovolný počet pixelů.
4. rozšíření vstupu (angl. padding): pokud rozšíříme vstup, výsledek konvoluce bude mít stejný rozměr jako vstup konvoluce. Rozšiřovat vstup lze několika metodami např. použitím konstanty, použitím hodnoty nejbližšího pole nebo zrcadlením.

Pro rozměr výstupu platí závistlost:

$$Q = \frac{M - K + 2 * O}{P} + 1$$

kde Q je velikost výstupu (pro jeden rozměr), M je velikost vstupu, K je velikost filtru, O je velikost okraje a P je posun filtru.



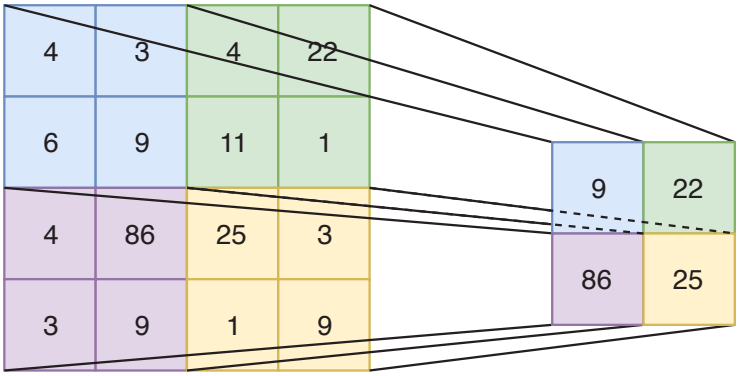
Obrázek 3.3: Porovnání tří aktivačních funkcí v jednom grafu.

3.6 Aktivační vrstva

Tato vrstva aplikuje matematickou funkci na výstupní vektor předchozí vrstvy. V této práci byly využity funkce:

- ReLU:
$$y(x) = \begin{cases} x & : x \geq 0 \\ 0 & : x < 0 \end{cases}$$
Vlastnosti ReLU jsou popsány v [13].
- ELU:
$$y(x) = \begin{cases} x & : x \geq 0 \\ \alpha(exp(x)-1) & : x < 0 \end{cases}$$
Vlastnosti ELU a porovnání s ReLU je popsáno v [14]. ELU má jeden parametr α. Tento parametr má v API Keras přednastavenou hodnotu 1.
- SELU:
$$y(x) = \begin{cases} \lambda x & : x \geq 0 \\ \lambda\alpha(exp(x)-1) & : x < 0 \end{cases}$$
Vlastnosti SELU a porovnání s ReLU je popsáno v [14]. SELU má dva parametry λ, α. λ má v API Keras přednastavenou hodnotu 1.0507. α má v API Keras přednastavenou hodnotu 1.6733.

Funkce jsou zobrazeny na 3.3



Obrázek 3.4: Příklad MaxPooling vrstvy s parametry $2, 2$. Vstupní vektor o rozměru $4, 4$ je zredukován do rozměru $2, 2$. Na výstupu jsou maximální hodnoty z částí vstupu. Příslušné části vstupu a výstupu jsou odlišeny barvou.

3.7 MaxPooling vrstva

Tato vrstva redukuje rozměr dat prostupujících sítí. MaxPooling má dva parametry - horizontální dělitel a a vertikální dělitel s. Vstupem je vektor $\mathbf{x}$ o rozměru w, h, c. Vstupní vektor je rozdělen na části o velikosti a, s a v každé části je nalezena maximální hodnota. Výstupem je vektor $\mathbf{y}$ o rozměru $\frac{w}{s}, \frac{h}{a}, c$. Tato vyrstva je využita v autoenkodéru, v jeho kódovací části. Příklad MaxPooling vrstvy je na 3.4.

3.8 Upsampling vrstva

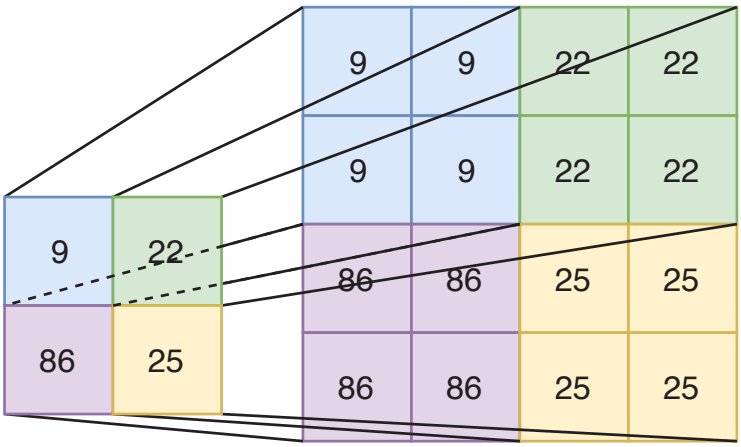
Upsampling vrstva replikuje vstup. Upsampling má dva parametry - horizontální násobek a a vertikální násobek s. Vstupem je matice $\mathbf{x}$ o rozměru w, h, e. Výstupem je matice $\mathbf{y}$ o rozměru $q \times a, w \times s, e$. Tato vrstva zvětšuje rozměr vstupních dat a je proto využita v autoenkodéru v jeho dekódovací části. Příklad Upsampling vrstvy je na 3.5.

3.9 Trénování neuronových sítí

Trénováním neuronové sítě nazýváme nastavení parametrů $\mathbf{w}$ a b vrstev neuronové sítě. Neuronová síť má chybovou funkci, kterou volíme. V této práci byla použita střední kvadratická chyba vyjádřena:

$$e = \sum(\mathbf{Y} - \mathbf{X})^2$$

kde $\mathbf{X}$ je vstupní obraz neuronové sítě a $\mathbf{Y}$ je výstupní obraz neuronové sítě. Nastavení parametrů probíhá algoritmem *zpětné propagace chyb* s využitím optimalizačního algoritmu adam [15]. Algoritmus nastavuje parametry sítě tak, aby hodnota chybová funkce byla co nejmenší. API Keras implementuje tento algoritmus funkce *fit*.



Obrázek 3.5: Příklad Upsampling vrstvy s parametry $2, 2$. Vstupní vektor o rozměru $2, 2$ je rozšířen na rozměr $4, 4$. Příslušné části vstupu a výstupu jsou odlišeny barvou.

4 Popis datasetu

Pro tuto práci jsem vytvořil sadu snímků stuhy. Stuhu jsem nasnímal plošnou kamerou Blue Fox Matrix Vision 2089C. Pod kameru jsem manuálním postupným posunem umístil stuhu. Pozice a rotace stuhy je na snímcích variabilní. Stuha obsahuje opakující se text vyšitý bílou barvou. V jednom segmentu je text vyšit ve dvou orientacích. Písmena textu mají být oddělena. Při tkaní se mohou vyskytnou tyto vady:

1. Přebývající nit - nastane pokud stroj mezi písmeny nepřeruší nit a propojí písmena. Vadné nítě mají při rozlišení 600x600 pixelů velikost 9 pixelů.

2. Nedokončené či chybějící písmeno - nastane pokud je na písmeno použita nit černé barvy nebo písmeno není přítomné.

3. Nečistota - nastane pokud černá část obsahuje nečistoty

Cílem této práce je tyto vady detekovat.

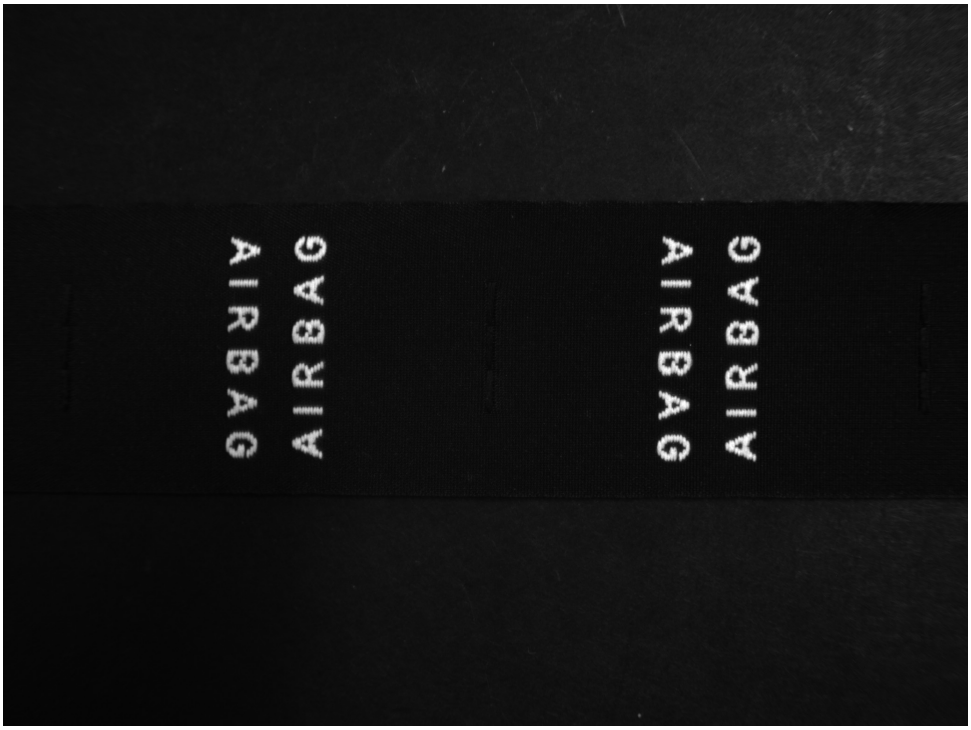
Nezpracované snímky z kamery obsahují v horní a spodní části pozadí. Na nezpracovaném snímku jsou vyfoceny dva segmenty. Nezpracovaných snímků jsem nasnímal 154. Nezpracované snímky mají rozlišení 2064x1544.

4.1 Předzpracování obrazu

Cílem předzpracování je získat snímky, které obsahují pouze jeden segment, kde vyšitý text je přibližně na středu obrazu.

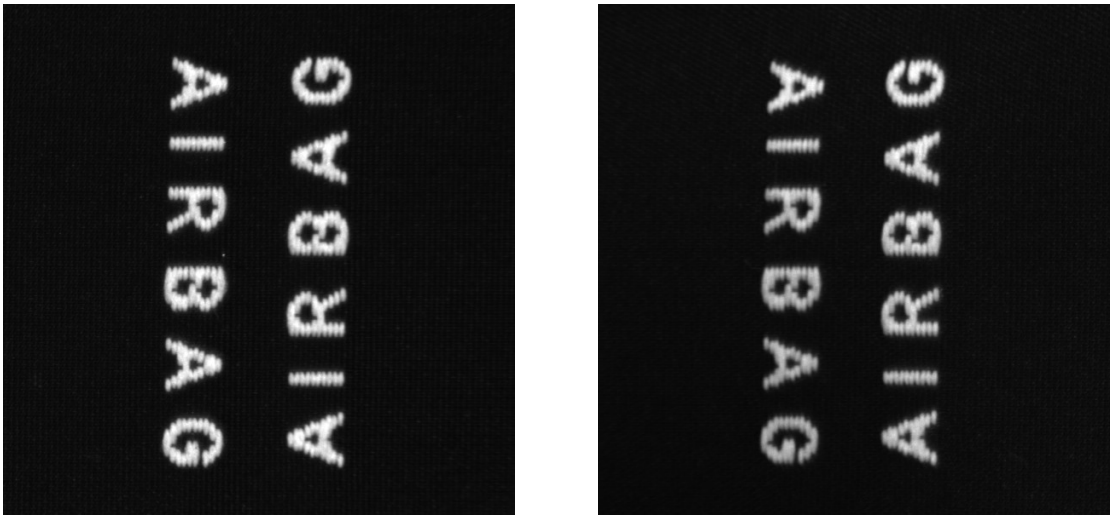
Ze snímku 4.1 jsem manuálně vyřízl šablonu 4.2 o rozměru 600x600 pixelů. Výřez šablony jsem zvolil tak, aby zřetězen pokryl celou plochu stuhy. Čtvercový rozměr je vhodný pro autoenkodér, kdy dekodér může využít MaxPooling se stejnými parametry. Prvočíselný rozklad čísla 600 obsahuje tři násobky čísla dva. Je možné využít tři MaxPooling vrstvy bez rozšiřování obrazu o krajní pixely.

Šedotónovou šablonu 4.2 jsem pomocí funkce `cv2.matchTemplate` našel ve snímcích a tyto části uložil. Tato funkce snímky nerotuje, neprovádí korekci perspektivy a nemusí najít přesnou polohu vyšitého textu. Vyšitý text je na výsledných snímcích různě posunutý, natočený, roztažený a zvlněný. Deformace mají různé kombinace, které nelze triviálně popsat. Lze najít klíčové body a párováním najít transformaci (v opencv funkcí `findHomography`), která by danou deformaci aproximovala. Porovnáním šablony se vzorkem, by bylo možné najít vadné části. Tato práce poskytuje alternativní řešení k nalezení vadných částí.



Obrázek 4.1: Originální snímek - nezpracovaný výstup z kamery

Obrázek 4.2: Šablona pro nalezení přibližné polohy segmentů



Obrázek 4.3: Dva vybrané OK snímky z vytvořené datové sady

Data je nutné před začátkem trénování sítě přetypovat na datový typ *float32* a každý snímek transformovat do pole velikosti 600x600x1.

Ze 154 nezpracovaných snímků po vyřazení rozmazaných snímků vzniklo 291 výřezů podobných šabloně. Snímky jsem normalizoval dle

$$\mathbf{Y} = \frac{\mathbf{X} - \mu}{\sigma}$$

kde $\mathbf{X}$ jsou vstupní data, μ je střední hodnota $\mathbf{X}$, σ je rozptyl $\mathbf{X}$ a $\mathbf{Y}$ jsou normalizovaná data. Takto upravená data mají střední hodnotu 0 a rozptyl 1, což jsou obecně vhodné vlastnosti pro trénování neuronových sítí.

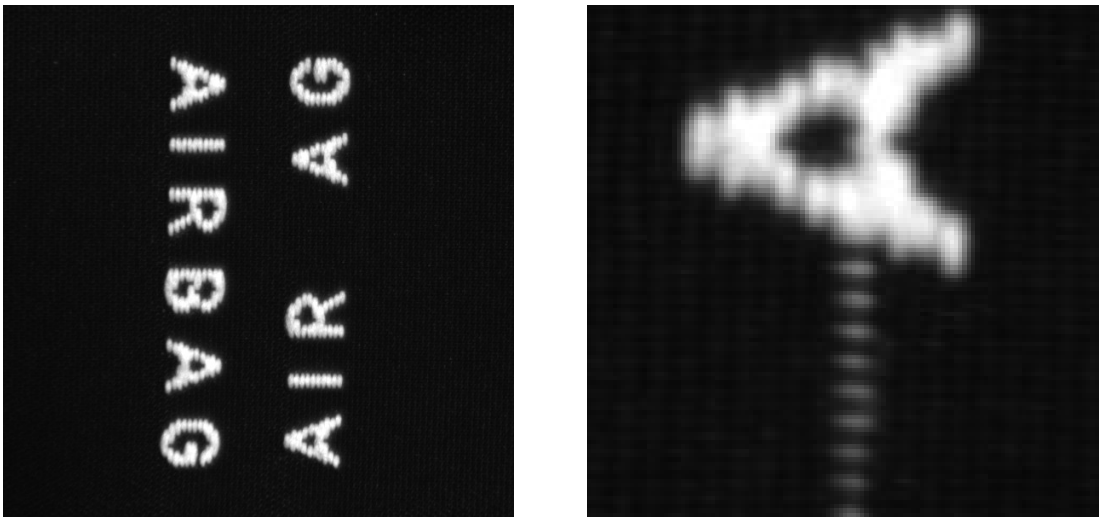
4.2 Rozdělení snímků

291 normalizovaných snímků jsem rozdělil do tří sad:

1. Trénovací data (248 snímků) - data, na kteých se model učí. Na datech je spočtena rekonstrukční odchylka. Tato sada obsahuje pouze bezchybné snímky, aby bylo možno hodnotit model dle rekonstrukční odchylky.

2. Validační data pro rekonstrukční odchylku (19 snímků) - data, ze kterých se odvozuje chybovost modelu. Porovnáním úspěšnosti na trénovací a validační sadě lze poznat, kdy se již síť neučí. Na této sadě jsem ladil hyperparametry, takže se dá říci, že jsem nastavoval hyperparametry tak, aby výsledek na validační sadě byl co nejlepší. Tato sada obsahuje pouze bezchybné snímky, aby bylo možno hodnotit model dle rekonstrukční odchylky.

3. Testovací data (24 snímků) - data, ze kterých je prezentována úspěšnost modelu v této práci. Použitím testovací sady snímků zamezíme možnosti, že by



(a) Chybějící písmeno (b) Detail snímku s přebývající nití

Obrázek 4.4: Dva vybrané NOK snímky z datové sady

model byl nastaven podle validační sady. Model se s těmito snímky setká pouze jednou. Výsledek na této sadě odpovídá reálnému testovacímu provozu a lze tento výslek prezentovat jako úspěšnost modelu, se kterou model bude vykovávat svůj úkon. K těmto snímkům jsem vytvořil anotace vadných oblastí pro následné vyhodnocení.

Pojem validační a testovací sada je v některých pramenech prohozen, důležitá je však podstata neprezentovat výsledky validační sady.

5 Návrh modelu autoenkodéru

Pro trénování vybraných modelů jsem využil jupyter notebook [16] pro jeho přehlednost. Modely neuronových sítí jsem navrhoval v API Keras. Tento software poskytuje jednoduché rozhraní pro práci s různými frameworky pro neuronové sítě a usnadňuje práci při návrhu modelu. API Keras jsem využil pro framework Tensorflow. Software Keras umožňuje trénování neuronové sítě přerušit, zobrazit vysledky a poté pokračovat v trénování.

Pro možnost reprodukce výsledků jsem zvolil seed numpy na hodnotu 1 a seed Tensorflow na hodnotu 2.

Návrh neuronové sítě probíhal v cyklech: Navrhl jsem model, který jsem natrénoval a vyhodnotil jeho úspěšnost na validační sadě, identifikoval jsem klíčové faktory ovlivňující jeho fungování a na základě zjištěných informací jsem navrhl nový model. Tento proces jsem opakoval.

Keras vypočte druhou mocninu rozdílů vstupu a výstupu. Tato funkce je mnou zvoleným kritériem. Neuronová síť iteračními kroky postupuje k minimu této funkce.

Během návrhu bylo potřeba nastavovat počet, typ a parametry vrstev a rychlost učení. Toto jsou tzv. hyperparametry neuronových sítí.

Pokud se při trénování chyba nesnižuje, ukončuji trénování a analyzuji výsledek. Pokud se snižuje chyba na trénovací sadě, ale na sadě validační se chyba nesnižuje, indikuje se tím přetrénovávání sítě, což je jev při kterém se model přizpůsobí pouze datům, na kterých byl natrénován, ale nedokáže generalizovat data obdobná.

Od vstupních dat jsem odečetl jejich střední hodnotu a vydělil je rozptylem, takže data, která jsou vstupem sítě mají průměrnou hodnotu 0 a rozptyl 1. Jedná se tedy o data normalizovaná.

U vadných snímků je potřeba, aby síť přebývající nitě nepřekreslila, naopak je potřeba chybějící písmena překreslit pro následné vyhodnocení vad.

Přehled všech natrénovaných a vyhodnocených modelů je v příloze *BP modely.xls*. Každý model má přiřazený identifikační kód. K modelu jsou vypsány vrstvy, ze kterých se model skládá, včetně jejich parametrů a rozměru výstupu těchto vrstev. Je specifikováno, jaká optimalizační funkce byla použita včetně nastavení rychlosti učení. Při trénování modelů jsem nastavoval počet současně vyhodnocovaných snímků (batch size) vždy na největší možný počet. Tento počet je omezen velikostí paměti grafické karty. Pro trénování jsem využil grafickou kartu NVIDIA GTX 1060 3GB. Tato velikost paměti stačila na 4-16 naráz trénovaných snímků. Modernější grafické karty vhodněji uzpůsobené trénování neuronových sítí, by umožňily trénovat více snímků naráz. Změřil jsem výsledek modelu při trénování 16 snímků a 8 snímků naráz. Dle výsledků z validační sady 0.0242 resp 0.0238 soudím, že vyšším počtem

současně trénovaných snímků bych větší přesnosti modelů nedosáhl.

Na různých modelech jsem porovnal optimalizační funkce SDG, adagrad a adam. Nejlepšího výsledku dosáhla opakovaně funkce adam.

Důležité bylo vynechání aktivační funkce před výstupem, aby výstup mohl dosahovat libovolné hodnoty.

V následujících sekcích popíšu nejpřínosnější modely.

5.1 Plně propojené vrstvy

Jako první jsem chtěl vyzkoušet plně propojené vrstvy, avšak jsem si spočítal počet parametrů a usoudil, že model nemůže mít schopnost generalizace. Vezměme v potaz obraz o rozměrech 600x600. Model je následovný: vstupní obraz -> skrytá plně propojená vrstva (relu) -> výstupní plně propojená vrstva. Pokud bychom měli pouze jednu skrytou plně propojednou vrstvu s pouze jedním neuronem, počet parametrů plně propojené vrstvy je 360001 (počítáme bias) a počet parametrů výstupní plně propojené sítě je 720000. Tato síť s pouze jednou skrytou vrstvou má 1080001 parametrů. Pokud dám do skryté vrstvy n neuronů, počet parametrů bude $2 * n * 360000 + 360000 + n$. Obecně platí, čím větší počet parametrů síť má, tím větší musí být trénovací sada, aby se síť nepřetrénovala.

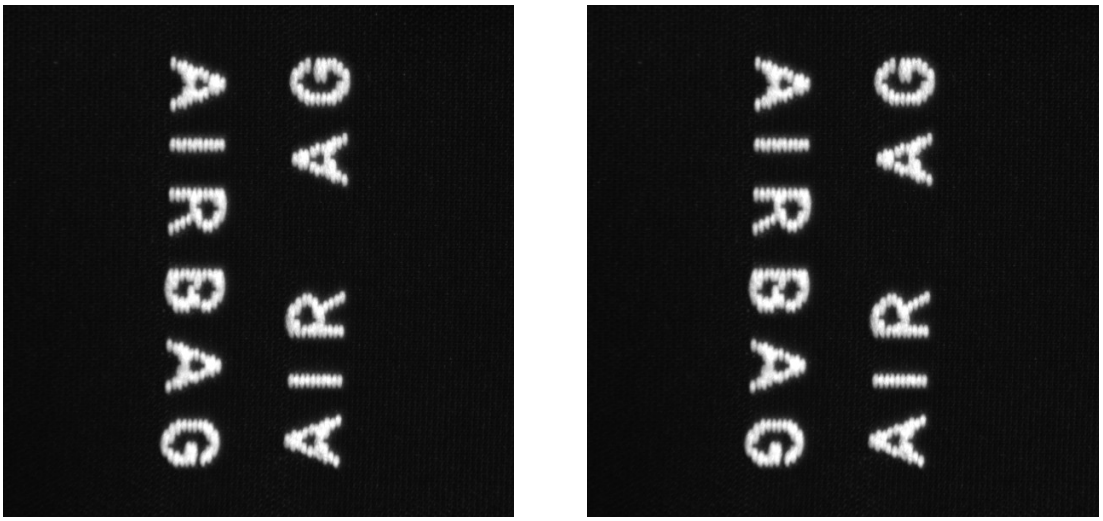
5.2 Model pouze konvolučních vrstvev

Vlastnosti konvoluční vrstvy plynou z vlastnosí vstupních dat. Vstupem konvoluce je určitá oblast vrstvy předchozí. Tato vlastnost má výhodu například pokud chceme detekovat písmena. Filtr tvaru písmena "A" bude mít vysokou odezvu ve všech místech, kde se písmeno "A" vyskytuje. Nebere ovšem v potaz, že by zde písmeno "A" být nemělo. Filtr nezná svojí polohu v prostoru a tato poloha není do výsledku zanesena. Konvoluční vrstvy nemají tolik parametrů jako plně propojené vrstvy.

Autoenkodér se skládá ze tří konvolučních vrstev, následovanými relu vrstvami a maxpooling vrstvami. Dekonvoluční síť (dekodér) má dvě konvoluční vrstvy, a místo Maxpooling vrstev jsem využil UpSampling vrstvy. Počet parametrů sítě je 1,905.

Dekodér:
Konvoluce 3x3 - 8 filtrů Relu Maxpooling 2x2
Konvoluce 3x3 - 4 filtry Relu Maxpooling 2x2
Konvoluce 3x3 - 2 filtry Relu
Enkodér:
Upsampling 2x2
Konvoluce 3x3 - 8 filtrů Relu Upsampling 2x2
Konvoluce 3x3 - 1 filtr

V epoše č.130 se rekonstrukční chyba ustálí na hodnotě 0.0043. Výsledné snímky mají správné deformace a neostré okraje pro detailní rozlišení jednotlivých nití. U snímků, kde na vstupu byla přebývající nit, síť nit nepřekreslila na výstup. U snímků, kde chybělo písmeno, síť písmeno nepřekreslila. Výstupní vrstva dekodéru má



(a) Snímek před průchodem sítí c1

(b) Snímek po průchodu sítí c1

Obrázek 5.1: Snímek před a po průchodu konvoluční sítí (c1). Síť do snímku nedokreslí chybějící písmeno, pouze rozmaže vstupní snímek.

vstupy z oblasti 7x7 a nedokáže tedy rozlišit, zda je v oblasti mimo text, či v oblasti kde text být má.

Pro nalezení minima ztrátové funkce jsem použil algoritmus adam [15] se základním nastavením, kromě rychlosti učení, kterou jsem nastavil na hodnotu 0.005.

Experimentoval jsem s různou velikostí filtrů, počtem vrstev, počtem filtrů, velikostí upsamplingu.

U konvolučních sítí klasifikujících objekty je žádoucí najít možnosti sítě nastavením velkého množství parametrů, aby se síť přetrénovávala. Poté se síť optimalizuje, aby se nepřetrénovávala, ale zároveň aby měla co největší množství parametrů (s množstvím parametrů roste přesnost sítě). Tuto metodu jsem také vyzkoušel, ovšem při nastavení většího množství filtrů, se síť naučila překreslovat i drobné detaily (které jsou často vadou), což je nežádoucí.

Autoenkodér komprimuje data. Data jsou nejvíce zkomprimována v nejužším bodě, zde se tedy jedná o komprimaci snímku do rozměru 150x150x(2+1(bias)). Při nastavení počtu filtrů na hodnotu 2 bylo dosaženo rekonstrukční chyby 0.0014.

Pro možnost detekce špatné polohy písmene musí konvoluční jádro v nejužším bodě autoenkodéru mít vstupní data z velké části vstupního obrazu. Toho lze dosáhnout zvětšením filtrů, nebo přidáním více vrstev. Při zvětšení filtrů byly výstupní snímky velmi rozmazané. Při přidávání dalších konvolučních a maxpooling vrstev se síť přestala učit. Použití normalizačních vrstev nebo ELU jednotek nepomohlo, Síť, která by byla schopna detekovat chybějcící písmeno, jsem nebyl schopen natrénovat.

5.3 Kombinace plně propojených a konvolučních vrstev

Mezi enkodér a dekodér jsem vložil plně propojenou vrstvu. Při použití pouze dvou maxpooling vrstev je počet parametrů v přechodu po maxpooling vrstvě (rozměr 150x150x8) na vrstvu plně propojenou (10) 1800010. Tento počet parametrů je vhodné redukovat. Redukovat tuto hodnotu lze: zmenšením plně propojené vrstvy, snížením počtu parametrů poslední maxpooling vrstvy nebo použitím další maxpoling vrstvy.

Model b1:
Konvoluce 3x3 - 16 filtrů Relu Maxpooling 2x2
Konvoluce 3x3 - 4 filtry Relu Maxpooling 2x2
Konvoluce 3x3 - 8 filtrů Relu Maxpooling 2x2
Plně propojená vrstva - 10 neuronů
Plně propojená vrstva - 22500 neuronů
Konvoluce 3x3 - 8 filtrů Relu Upsampling 2x2
Konvoluce 3x3 - 8 filtrů Relu Upsampling 2x2
Konvoluce 3x3 - 16 filtrů Relu Upsampling 2x2
Konvoluce 3x3 - 1 filtr

Tento model využívá optimalizační funkci adam, rychlost učení 0.01. Při trénování se hodnota optimalizační funkce ustálila na 0.0135 pro trénovací data, 0.0139 pro data validační. V další kapitole tento model vyhodnotím na testovacích datech.

Oproti pouze konvoluční síti tento model zachycuje v plně propojených vrstvách snímek jako celek. Jak zobrazuje snímek 5.2b, model překreslí chybějící písmeno na výstupní obraz. Výstup je rozmazaný, jelikož síť není dostečně hluboká. Redukci rozmazání snímku jsem dosáhl modelem *b22*.

Model b22:
Konvoluce 3x3 - 16 filtrů Relu Maxpooling 2x2
Konvoluce 3x3 - 8 filtrů Relu Maxpooling 2x2
Konvoluce 3x3 - 4 filtry Relu Maxpooling 2x2
Konvoluce 3x3 - 4 filtry Relu Maxpooling 2x2
Plně propojená vrstva - 10 neuronů - bez bias
Plně propojená vrstva - 360000 neuronů - bez bias
Konvoluce 3x3 - 8 filtrů Relu
Konvoluce 3x3 - 16 filtrů Relu Konvoluce 3x3 - 1 filtr

Tento model má 2929361 parametrů. Použil jsem optimalizační funkci adam s ryhlostí učení 0.01. Při trénování se hodnota optimalizační funkce ustálila na 0.0093 pro trénovací data a 0.0133 pro data validační.

U podobných modelů docházelo k vytváření výstupu, který nebyl závislý na vstupních datech. Výstup byl vždy průměrný snímek. Tuto vlastnost jsem odstranil nepoužitím bias u plně propojených vrstev. Model byl tak nucen vytvářet výstup na základě vstupu. Tento model má ostřejší výstup 5.3b oproti modelu b1 5.2b.

(a) Snímek před průchodem sítí b1 (b) Snímek po průchodu sítí b1

Obrázek 5.2: Snímek před a po průchodu kombinovanou neuronovou sítí (b1). Síť do snímku dokreslí chybějící písmeno

(a) Snímek před průchodem sítí b22 (b) Snímek po průchodu sítí b22

Obrázek 5.3: Snímek před a po průchodu kombinovanou neuronovou sítí (b22). Síť do snímku dokreslí chybějící písmeno

6 Vyhodnocení modelů

V této kapitole vyhodnotím tři vybrané modely (c1, b1, b22) dvěma metodami:

1. Vyhodnocení algoritmu po pixelech - ohodnotím, zdali každý pixel je správně zařazen do kategorie OK/NOK.

2. Vyhodnocení celých snímků - vyhodnotím, zdali snímky jsou správně zařazeny do kategorie OK/NOK.

6.1 Vyhodnocení algoritmu po pixelech

Nástrojem labelme [17] jsem označil oblasti, kde se vyskytují vady. Labelme jsem spouštěl s parametrem *autosave*. Díky tomuto parametru není potřeba ukládat anotace pro každý snímek zvlášť, ale ukládají se automaticky. Program labelme vytvoří soubory .json obsahující souřadnice bodů ohraničujících vady a snímky.

Vytvořil jsem pomocný skript *jsonstodataset.ipynb*, který převede anotace z formátu *.json* do formátu *.png*. Výsledkem je snímek, kde anotovaný objekt má nenulovou hodnotu. Tento snímek lze načíst pomocí knihovny *opencv* a užitím operátoru $>$ jsem získal binární masku (dále nazývaná *anotace*), kterou dále využiji.

Pro vyhodnocení výsledků výpočtu

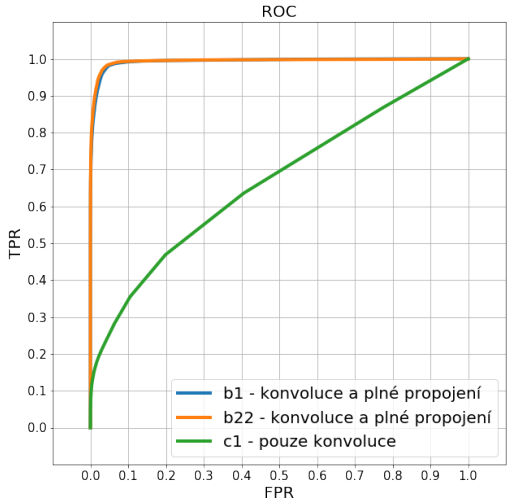
$$\mathbf{D} = |\mathbf{Y} - \mathbf{X}|$$

kde $\mathbf{D}$ je absolutní rozdíl snímků (dále diference), $\mathbf{X}$ je vstupní snímek sítě a $\mathbf{Y}$ je výstupní snímek sítě. Diference $\mathbf{D}$ má velkou hodnotu tam, kde je velký rozdíl mezi snímkem vstupujícím do sítě a snímkem vystupujícím z ní . Pro rovnoměrně rozložených 255 hodnot t z intervalu $[min(\mathbf{D}), max(\mathbf{D})]$, jsem nalezl oblasti $\mathbf{D} > t$ nazvanou *predikce*. Porovnal jsem těchto 255 predikcí s anotací a vypočetl:

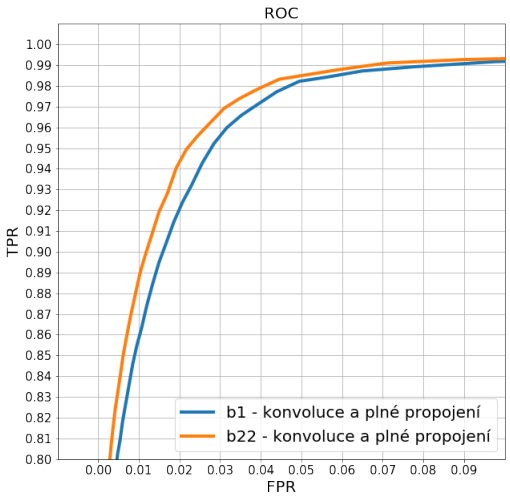
1. True-Pozitive(TP) - pixely, kde anotace a predikce je 1. Vada na snímku je a algoritmus jí našel.

2. False-Pozitive(FN) - pixely, kde anotace je 0 a predikce je 1. Vada zde byla detekována, ale vada zde ve skutečnosti není.

3. True-Negative(TN) - pixely, kde anotace a predikce je 0 . Není zde vada, a vada zde detekována nebyla.



(a) ROC křivky modelů b1, b22 a c1.



(b) Detail ROC křivky pro modely b1 a b22.

Obrázek 6.1: ROC křivka zobrazuje závislost TPR na FPR.

4. False-Negative(FN) - pixely, kde anotace je 1 a predikce je 0. Vada zde je, ale nebyla detekována.

Následně jsem provedl výpočet TPR (true positive rate) a FPR (false pozitive rate).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Hodnota TPR je jak velká část vad byla nalezena. Hodnota FPR je jak velká část snímků bez vady byla označena, jako by vadu měla.

V ROC křivce 6.1a je zobrazena závislost TPR na FPR tří modelů. Modely b1 a b22 mají podobné výsledky, proto je v 6.1b zobrazen výřez ROC křivky pro zvýraznění rozdílů

Cílem algoritmu je dosažení co nejvyššího TPR (maximální hodnota je 1) a co nejnižšího FPR (minimální hodnota je 0).

TPR a FPR je ovlivněnou chybou ruční anotace.

Ideální klasifikátor má TPR 1 při FPR 0. Vzdálenost od tohoto bodu je distance v kde,

$$v = \sqrt{(TPR - 1)^2 + (FPR)^2}$$

Pixelů bez vady je 1380 krát více, než pixelů s vadou. Proto jsem metriku distance upravil na:

$$u = \sqrt{(TPR - 1)^2 + (FPR)^2 * 1380}$$

Tabulka 6.1 zobrazuje výsledky distance u pro vybrané modely. Výsledky odpovídají grafu 6.1a. Výsledky distance u nekorelují s rekonstrukční odchylkou. Konvo-

luční model c1 má výrazně nižší rekonstrukční odchylku, jelikož nedokáže detekovat chybějící písmena, která ovlivňují distanci u,

Model	Rekonstrukční odchylka	Distance u	TPR	FPR
c1	0.0032	0.893	0.127	0.005
b1	0.0139	0.255	0.779	0.003
b22	0.0133	0.227	0.808	0.003

Tabulka 6.1: Tabulka distance u pro vybrané modely

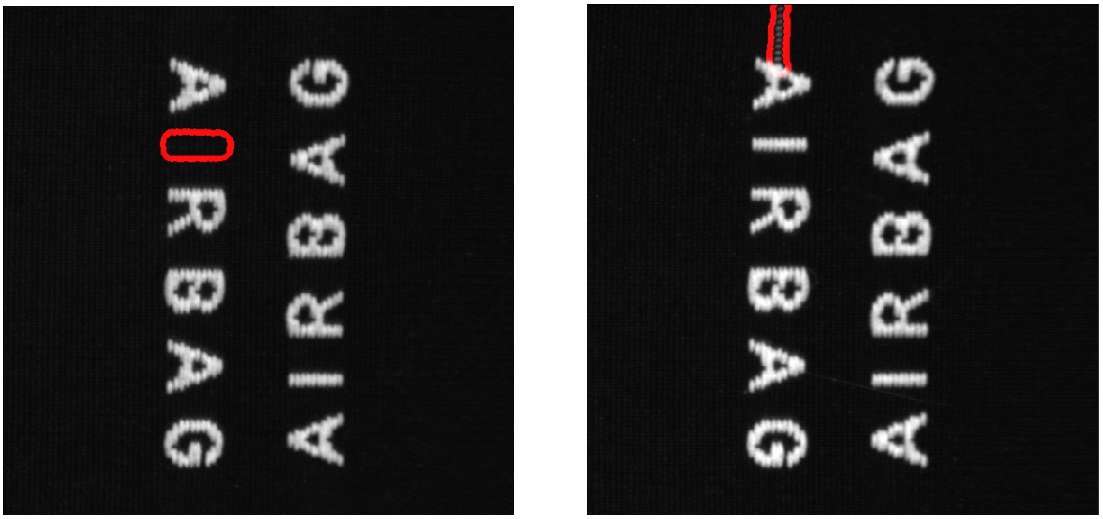
6.2 Vyhodnocení algoritmu na úrovni snímku

Snímky jsem rozdělil do dvou skupin, podle toho zda na nich chyba je, či nikoli. Nalezl jsem práh, pro který je distance u minimální. Diferenci vstupu a výstupu sítě jsem oprahoval, morfologicky zavřel s kruhovým jádrem 15,15 a našel největší propojenou komponentu v obraze. V tabulce 6.2 jsou hodnoty TPR a FPR. Za vadné snímky jsou považovány snímky, kde velikost největší propojené komponenty je větší jak 80 px.

Model	TPR	FPR
c1	0.22	0.4
b1	0.88	0
b22	1	0

Tabulka 6.2: Tabulka TPR a FPR při vyhodnocení celého snímku.

Celkově je v trénovacích datech pouze 9 vadných snímků. Perfektní výsledek modelu 22 může být tedy zkreslen, avšak všechny bezchybné snímky mají velikost největší oblasti menší jak 47 px. Snímky s vadou mají velikost oblasti od 107 px.



(a) Vadná textílie s chybějícím písmenem. Nalezená vada je zvýrazněná červeným ohraničením.

(b) Přebývající nit. Nalezená vada je zvýrazněná červeným ohraničením.

Obrázek 6.2: Snímky po vyhodnocení

7 Integrace modelu pro reálný provoz v prostředí LabView

V této kapitole jsem vytvořil příklad, jak je možné model aplikovat v praxi. Jazyk Python není běžnou součástí výpočetních systémů při průmylové výrobě, proto pro jsem pro využil prostředí LabView, které nabízí grafické rozhraní pro operátory výroby, ovladače pro vstupně výstupní zařízení včetně kamery a zároveň je LabView vhodné pro použití na průmyslových výpočetních systémech (např. NI IC).

Základní schéma aplikace je následovné:

1. Inicializace modelu autoenkodéru v Pythonu
2. Inicializace kamery
3. Získání obrazu
4. Vyhodnocení obrazu v neuronové síti - V pythonu
5. Prahování a morfologická operace obrazu - v Pythonu
6. Prahování velikosti oblasti - v Pythonu
7. Zobrazení výsledku defektoskopie

7.1 LabView VI

LabView VI, neboli Virtual Instrument se skládá ze dvou částí: Grafické rozhraní a aplikační logika. Následující popis bloků patří do aplikační logiky.

Akvizice a zpracování obrazu probíhá ve While cyklu. Před cyklem inicializuji kameru a model autoenkodéru. V cyklu se vyfotí snímek, který se zpracuje a zobrazí. Při ukončení cyklu bude ukončena práce s kamerou a modelem.

Pro akvizici obrazu jsem využil blok `IMAQdx Grab`. Tento blok potřebuje jako vstup odkaz na místo v paměti pro obraz, a session, neboli odkaz na kameru. Nejprve jsem zapojil odkaz na místo v paměti pro obraz, vytvořený pomocí bloku `IMAQdx Create`. Vstupem bloku `IMAQdx Create` je řetězec s libovolným pojmenováním. Druhým vstupem bloku `IMAQdx Grab` je odkaz na kameru. Tento obraz je vytvořen pomocí bloku `IMAQdx Open Camera` a bloku `IMAQdx Configure Grab`.

Po akvizici je snímek pomocí bloku `ImageToArray` převeden z formátu `image` do dvourozměrného pole. Dvourozměrné pole je formát kompatibilní s jazykem Python. K volání funkcí jazyka `Python` používám blok `Python Node`, jehož vstupem jsou:

- reference na `Python` interpret
- cesta k souboru, ze kterého se má funkce vykonat
- jméno funkce, která se má vykonat
- návratový datový typ - datový typ, který bude přiveden na tento vstup, bude očekáván na výstupu tohoto bloku
- vstupní data - lze zvolit libovolný počet vstupních dat. Tyto data budou předána `Python` skriptu

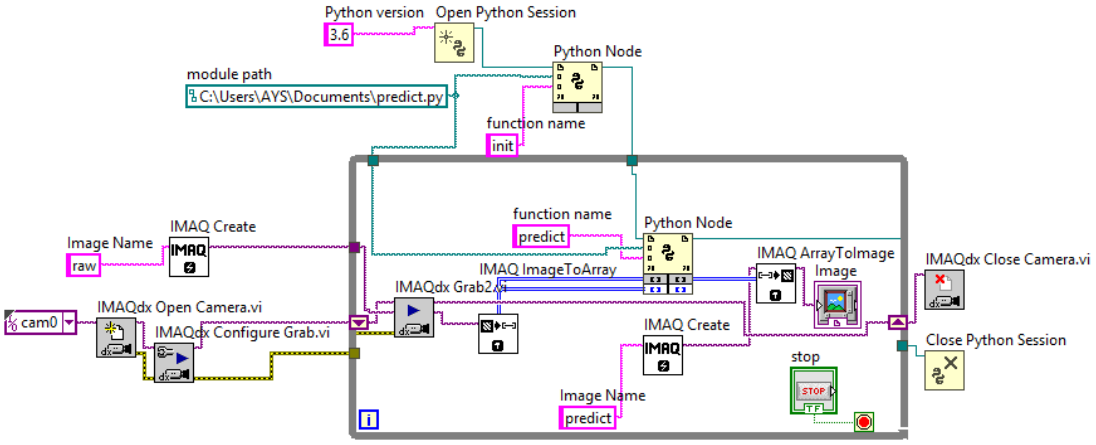
Blok `Python Node` vrátí referenci na datový typ a návratovou hodnotu vybrané `Python` funkce.

Navrácené pole převedu pomocí bloku `IMAQ ArrayToImage` z dvojrozměrného pole do datového typu `Image`. Blok `IMAQ ArrayToImage` potřebuje na vstupu odkaz na místo v paměti pro obraz. Tento odkaz jsem vytvožil opět pomocí bloku `IMAQdx Create`. Je důležité bloku `IMAQdx Create` přivést na vstup jiné jméno, než které používáme pro akvizici snímku. Snímek by v tomto případě mohl být zobrazen pouze do přijmutí dalšího snímku.

Výstup bloku `IMAQ ArrayToImage` napojím na `Image Display`, což je grafický prvek v uživatelském rozhraní. Tento prvek zobrazuje obraz. Umožňuje přibližování a oddalování obrazu. Ve `while` bloku je také stop tlačítko, které po stisknutí přeruší akvizici a zpracování obrazu. Po stisknutí tohoto tlačítka smyčka `while` dokončí aktuální iteraci a již novou nezačne. Následně se vykonají dva bloky pro korektní ukončení programu:

- `IMAQdx Close Camera` - ukončí práci s kamerou. Při nevykonání tohoto bloku by jiné programy, požadující snímky z využité kamery, nemusely být schopny komunikovat s kamerou.
- `Close Python Session` - blok ukončí komunikaci s `Python` interpretem

Vykonávání defektoskopie spouštím tlačítkem `Run`.
Rychlost programu je 15 snímků za sekundu.



(a) VI 1

Obrázek 7.1: LabView VI

8 Závěr

V bakalářské práci jsem popsal strojové vidění, neuronové sítě a autoenkodér. Nasnímal jsem 291 snímků textílií, které obsahují snímky s vadou i snímky bez vady. Snímky jsem rozdělil na data trénovací, validační a testovací. Na snímcích bez vady jsem natrénoval 34 modelů, vyhodnotil jejich rekonstrukční odchylku na validační sadě a popsal, co jí ovlivňuje. Vybral jsem tři modely pro vyhodnocení na testovacích datech. Pouze konvoluční model se ukázal jako nevyhovující pro detekci velkých vad. Při hodnocení jednotlivých pixelů má nejlepší model, kombinující konvoluční a plně propojené vrstvy, TPR 0.81 při FPR 0.003. Při vyhodnocení celých snímků má nejlepší model TPR 1 a FPR 0. V poslední kapitole je ukázka, jak je možno implementovat navržený algoritmus pro kontinuální vyhodnocování snímků. Pro přesnější evaluaci modelů by bylo potřeba více snímků se složitější strukturou.

Použitá literatura

[1] INSTRUMENTS, National. *LabView* [LabView]. 2018. Verze 15. Dostupné také z: http://czech.ni.com/labview.

[2] *Pekatvision*. Verze 8. Dostupné také z: https://www.pekatvision.com/.

[3] HE, Kaiming; ZHANG, Xiangyu; REN, Shaoqing; SUN, Jian. Identity Mappings in Deep Residual Networks. *CoRR*. 2016, roč. abs/1603.05027. Dostupné z arXiv: 1603.05027.

[4] CHOLLET, François et al. *Keras* [https://keras.io]. 2015.

[5] MARTÍN ABADI et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Dostupné také z: https://www.tensorflow.org/. Software available from tensorflow.org.

[6] PASCAL, Vincent; HUGO, Larochelle; PIERRE-ANTOINE, Manzagol; ISABELLE, Lajoie; YOSHUA, Bengio. Stacked Denoising Autoencoders: Learning Useful Representationsina Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research 11 (2010) 3371-3408*. Dostupné také z: http://www.jmlr.org/papers/volume11/vincent10a/vincent10a.pdf.

[7] G. E., Hinton; R. R., Salakhutdinov. Reducing the Dimensionality ofData with Neural Networks. *www.sciencemag.org SCIENCE*. 2006. Dostupné také z: https://www.cs.toronto.edu/~hinton/science.pdf.

[8] QINGYANG, Tan; LIN, Gao; YU-KUN, Lai; SHIHONG, Xia. Variational Autoencoders for Deforming 3D Mesh Models. In: Salt Lake City, Utah, USA, 2018. Verze 5. Dostupné také z: http://openaccess.thecvf.com/content_cvpr_2018/papers/Tan_Variational_Autoencoders_for_CVPR_2018_paper.pdf.

[9] SAZLI, Murat. A brief review of feed-forward neural networks. *Communications, Faculty Of Science, University of Ankara*. 2006, roč. 50, s. 11–17. Dostupné z DOI: 10.1501/0003168.

[10] IAN GOODFELLOW, Yoshua Bengio; COURVILLE, Aaron. *Deep Learning*. 2016. Dostupné také z: http://www.deeplearningbook.org. Book in preparation for MIT Press.

[11] LECUN, Yann. *A theoretical framework for back-propagation.* Verze 7. Dostupné také z: https://www.researchgate.net/profile/Yann_Lecun/publication/2360531_A_Theoretical_Framework_for_Back-Propagation/links/0deec519dfa297eac1000000/A-Theoretical-Framework-for-Back-Propagation.pdf.

[12] KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: PEREIRA, F.; BURGES, C. J. C.; BOTTOU, L.; WEINBERGER, K. Q. (ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, s. 1097–1105. Dostupné také z: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[13] AGARAP, Abien Fred. Deep Learning using Rectified Linear Units (ReLU). *CoRR*. 2018, roč. abs/1803.08375. Dostupné z arXiv: 1803.08375.

[14] PEDAMONTI, Dabal. Comparison of non-linear activation functions for deep neural networks on MNIST classification task. *CoRR*. 2018, roč. abs/1804.02763. Dostupné z arXiv: 1804.02763.

[15] KINGMA, Diederik P.; BA, Jimmy. Adam: A Method for Stochastic Optimization. *CoRR*. 2014, roč. abs/1412.6980. Dostupné z arXiv: 1412.6980.

[16] *Jupyter*. Verze 14. Dostupné také z: http://jupyter.org/.

[17] TORRALBA, A.; RUSSELL, B. C.; YUEN, J. LabelMe: Online Image Annotation and Applications. *Proceedings of the IEEE*. 2010, roč. 98, č. 8, s. 1467–1484. ISSN 0018-9219. Dostupné z DOI: 10.1109/JPROC.2010.2050290.