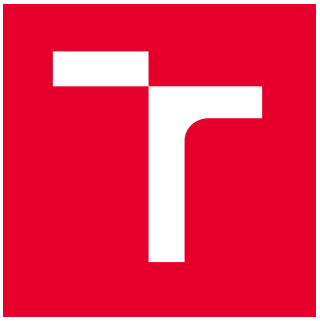
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2023 Bc. Filip Jašek



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY

DEPARTMENT OF RADIO ELECTRONICS

DETEKCE DEFEKTŮ DESEK VE VÝROBĚ POLOVODIČŮ

CLASSIFICATION OF BOARD DEFECTS IN SEMICONDUCTOR MANUFACTURING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Filip Jašek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jiří Dřínovský, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Elektronika a komunikační technologie**

Ústav radioelektroniky

Student: Bc. Filip Jašek ***ID:*** 211562

Ročník: 2 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Detekce defektů desek ve výrobě polovodičů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s problematikou identifikace chybných čipů při výrobě polovodičů a s metodami kontroly a řízení výtěžnosti při výrobě polovodičů. Dále se seznamte s metodami strojového učení pro rozpoznávání obrazu. Na základě získaných poznatků proveďte vyhodnocení parametrů existujícího řešení využívajícího metody strojového učení RESNET18 pro klasifikaci defektů při výrobě polovodičů. Ověřte možnosti zlepšení těchto řešení pomocí zvýšení množství vstupních dat, změnou topologie modelu a změnou parametrů modelu. Vyhodnoťte efektivitu jednotlivých metod zlepšení.

Navrhněte systém pro poloautomatizovanou klasifikaci defektů ve výrobním procesu polovodičů. Identifikujte jednotlivé bloky systému a specifikujte jejich parametry. Realizujte bloky pro transformace vstupních dat a trénování modelů a jejich verzování. Dále realizujte blok umožňující predikci přes rozhraní REST API, který bude distribuovaný jako Docker obraz.

DOPORUČENÁ LITERATURA:

[1] CHALLOT, Francois. Deep Learning v jazyce Python: knihovny Keras, Tensorflow. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN-978-80-247-3100-1.

[2] GOODFELLOW, Ian, Yoshua BENGIO, Aaron COURVILLE. Deep learning. Cambridge: MIT Press, [2016]. Adaptive computation and machine learning (MIT Press). ISBN 978-0262035613

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 8.8.2023

Vedoucí práce: Ing. Jiří Dřínovský, Ph.D.

doc. Ing. Lucie Hudcová, Ph.D.
předseda rady studijního programu

Abstrakt

Tato diplomová práce se zabývá problematikou detekce defektů desek ve výrobě polovodičů. V rámci této práce byly zkoumány metody identifikace defektních čipů a kontroly řízení výtěžnosti při výrobě polovodičů. Práce se rovněž zabývá metodami strojového učení pro rozpoznání obrazu s cílem klasifikace defektů ve výrobním procesu. První zvolený přístup využíval k inferenci sítě ResNet18, avšak ukázalo se, že jeho přesnost nedosahovala vysokých hodnot sledovaných metrik z důvodu nedostatečného množství vstupních dat. Pro tento sledovaný dataset tak bylo vyzkoušeno použití předtrénovaných sítí využívající topologie ResNet50v2. K navýšení metrik však došlo až s použitím jiného datasetu. Pomocí ladění hyperparametrů sítě a augmentací byly zkoumány další možnosti zlepšení výkonnosti sítě. V práci se také ukázalo, že použití autoenkodérů pro redukci datového toku při inferenci může navýšit rychlost samotné inference, avšak s degradací evaluačních metrik.

Klíčová slova

Umělá inteligence, detekce defektů, identifikace chybných čipů, výtěžnost výroby, strojové učení, počítačové vidění, klasifikace defektů, sítě ResNet, předtrénované sítě, redukce toku dat, inference

Abstract

This diploma thesis focuses on detecting defects in semiconductor wafer manufacturing. It explores methods for identifying faulty chips and controlling yield during production. To classify defects machine learning techniques are used. Initially, ResNet18 architecture was used for inference, but low accuracy was attributed to limited input data. Transfer learning with ResNet50v2 was then attempted, resulting in improved metric with different dataset. Hyperparameter tuning and data augmentations were also explored. The study found that autoencoders for data compression during inference increased speed but led to degraded evaluation metrics.

Keywords

Artificial intelligence, defect detection, faulty chip identification, yield control, machine learning, computer vision, defect classification, ResNet networks, transfer learning, data flow reduction, inference

Bibliografická citace

JAŠEK, Filip. *Detekce defektů desek ve výrobě polovodičů.* Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/151749. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce Jiří Dřínovský.

Pozn.: Bibliografická citace je generována informačním systémem.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Bc. Filip Jašek*
VUT ID studenta:	*211562*
Typ práce:	Diplomová práce
Akademický rok:	*2022/23*
Téma závěrečné práce:	Detekce defektů desek při výrobě polovodičů

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 17.5.2023

podpis autora

Poděkování

Děkuji vedoucímu diplomové (bakalářské) práce Ing Pavlu Raškovi a Ing. Jiřímu Dřínovskému, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: 9.5.2023

podpis autora

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

Úvod

Detekce a klasifikace defektů u polovodičových technologií je velice důležitou kapitolou při jejich výrobě. Defekty samotné pak mohou vznikat v průběhu jednotlivých etap výroby – při přípravě struktur, zpracování hotových desek a v neposlední řadě při zapouzdření čipu. Důvody, proč se touto problematikou zabývat jsou jednoznačné. Jedná se především o finanční aspekty, jelikož všechny fáze výroby jsou nákladné.

Inspekční metody používané v současnosti si lze představit jako síto. Podle hustoty použité mřížky lze vybrané čipy z další fáze výroby vyloučit. Toto nastavení však představuje určitý kompromis, jelikož vyřazeny mohou být i takové čipy, které nejsou defektního charakteru. Při příliš hrubém nastavení dochází k obrácenému problému a v praxi se přiklání k více přísnější metodě. Vyřazením funkčních čipů však klesá celková výtěžnost.

U této práce je kladeno za cíl prozkoumat možnosti využití strojového učení jako doplňkový nástroj pro současné inspekční metody, přičemž bude vyhledávat čipy, které byly vyřazeny prvotní inspekcí. Nedílnou součástí strojového vidění je také implementace konvolučních neuronových sítí, popř. algoritmů hlubokého učení, což v současnosti představuje nejrozšířenější metodu realizace. V poslední době je především kladen velký důraz na kvalitu dat, proto je součástí každého strojově se učícího projektu rovněž jejich sběr, příprava a čištění. Tyto metody se snaží maximalizovat entropii a předat k trénovaní maximálně reprezentativní datovou strukturu. Součástí této práce není rozbor všech používaných metod při trénování modelů. Především jsou zde uvedeny postupy, které se při řešení tohoto konkrétního úkolu empiricky ověřily.

Text lze strukturovat do tří ucelených celků. První část je věnována rozboru architektury a algoritmům neuronových sítí. Součástí další části práce jsou metody analýzy a vyhodnocení vstupních datasetů. V poslední kapitole je nastíněn způsob inference natrénované sítě s daty získanými při inspekci jednotlivých waferů.

1. ALGORITMY STROJOVÉHO UČENÍ

Tato kapitola rozebírá základní větve strojového učení, které stojí na počátku každého strojově učícího se algoritmu. Čtenář tak získá vhled do řešené problematiky klasifikačních úloh. Nejčastěji vyskytujícím se algoritmem je řízené učení (nebo také učení s učitelem). Tento způsob učení je však pouze jednou ze čtyř kategorií [1]. Dále se tato kapitola zabývá rozdělením problematiky na dvě základní části: Klasifikace, regrese. V neposlední řadě je zde naznačen matematický aparát používaný při výpočtech a optimalizaci učebních algoritmů.

1.1 Kategorie strojového učení

1.1.1 Řízené učení (učení s učitelem)

Jedná se o nejčastěji implementovaný algoritmus při řešení úkolů strojového učení, který je rovněž majoritně zastoupený v této práci. Princip takového učení spočívá v mapování vstupních příznaků na předem známé cíle. Výstup takto natrénované sítě bude deterministický. V řízeném učení se nejčastěji setkáváme s klasifikací (výstupní predikce má diskrétní charakter – např. kočka nebo pes) nebo s regresí (výstupní predikce má spojitý charakter – např. kalkulačka ceny pojištění). Lze se však setkat i s jinými realizacemi učení s učitelem: Detekce objektů, segmentace obrazu, vytváření sekvencí [1]. Součástí každé implementace řízeného učení je důraz na samotnou aktuálnost dat. Je zřejmé, že statistická distribuce dat se může časem měnit – růst ceny bytů v důsledku inflace. Proto by měl model umět na takové změny reagovat, třeba přetrénováním modelu na aktuálních datech – takový přístup se potom dá označit jako samořízené učení (viz 1.1.3) [1] [2].

1.1.2 Neřízené učení (učení bez učitele)

Neřízené učení (*unsupervised learning)* spočívá v transformaci dat, avšak nemá zadané jednoznačné cíle. Tento algoritmus by měl být součástí každého předzpracování dat, jelikož může být nápomocný při hledání různých reprezentací a pochopení korelací vstupních dat [1]. Taková realizace může být součástí šumové analýzy nebo jako součást *pipeline* při automatickém přetrénování [2]. Mezi další známé kategorie neřízeného učení patří redukce dimenzionality – snižování počtu vstupních parametrů na základě vzájemné korelace příznaků – a shluková analýza – hledání shluků ve vstupním souboru dat. Shluková analýza se mimo jiné používá při hledání parametrů velikosti ohraničujících rámečků při detekci objektů.

1.1.3 Samořízené učení

Samořízené učení (*self-supervised learning*) je speciálním případem řízeného učení. Podobně jako u řízeného učení potřebuje anotovaná vstupní data. V tomto případě je

však samotná anotace prováděna bez zásahu učitele. Označení tříd jsou obvykle prováděna pomocí heuristických metod [1].

1.1.4 Posilované učení

Posilované učení (*reinforcement learning*) je typ učení, které má základ v behaviorální psychologii, kdy počítač zpravidla získává informace o prostředí, ve kterém se vyskytuje (např. počítačová hra) a prostřednictvím dostupných akcí se pokouší maximalizovat skóre. Implementace posilovaného učení se v budoucnu očekává například při autonomním řízení vozidel [1]. V neposlední řadě jsou zde naznačeny procesy používané při nastavování parametrů neuronových sítí, především algoritmem zpětného šíření (v angl. literatuře *backpropagation algorithm*).

1.2 Případy strojového učení

Následující podkapitoly rozebírají nejběžnější případy použití strojového učení: klasifikace a regrese.

1.2.1 Klasifikační úloha

Klasifikační problémy spočívají ve zpracování vstupů za účelem získání diskrétních výstupů. Podle typu výstupu lze klasifikaci dále dělit na binární (výstupy jsou např. charakteru: trpí/netrpí osoba onemocněním) a klasifikaci do více tříd (např.: o který typ vozidla se jedná: dodávka/osobní automobil/nákladní automobil). Jako klasifikační úlohu lze rovněž považovat algoritmy pro predikci defektů, které jsou obsaženy v této práci. V závislosti na úloze je potřeba vhodně zvolit ztrátovou funkci – ta určuje, jak konkrétní odhady klasifikátoru reprodukují správné cíle [1]. V případě klasifikace nejčastěji používáme následující ztrátové funkce.

1.2.1.1 Křížová entropie

Spočívá v penalizaci špatných predikcí. Jestliže je správný cíl predikován s pravděpodobností 1, potom je žádoucí, aby ztrátová funkce byla minimální. Pokud by byl naopak správný cíl predikován s malou pravděpodobností, je potřeba, aby ztrátová funkce byla vysoká. Tento předpoklad vychází z teorie entropie. Pokud je entropie nulová, znamená to, že systém nemá žádnou vypovídací hodnotu (všechny cíle jsou stejné). Problémem je, že výstupní distribuce systému je neznámá, avšak vstupní distribuce je známá, proto lze použít křížovou entropii [3].

$$H_p(Q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log\big(p(y_i)\big) + (1 - y_i)\log\big(1 - p(y_i)\big). \tag{1.1}$$

kde y_i je označení třídy (1 nebo 0) a $p(y_i)$ je pravděpodobnost dané třídy.

Podle druhu klasifikační úlohy se používají různé ztrátové funkce. V případě klasifikace do 2 tříd se využívá binární křížová entropie (*Binary crossentropy*). Problematika řešená v této práci však pokaždé zahrnuje větší množství tříd, proto je využíváno kategorické křížové entropie (*Categorical Crossentropy*). Ve frameworku *keras* se volí typ ztrátové funkce přidáním argumentu *loss* do funkce *compile*.
Zjednodušeně lze problém učení interpretovat jako hledání takové distribuce architektury strojového učení, která minimalizuje rozdílnost vůči původní distribuci (resp. vůči vstupnímu datasetu). Tento poznatek však může vést k dalším nástrahám učení, jelikož distribuce vstupních dat nemusí mít nutně generalizující charakter (viz kapitola 3). Za předpokladu mnohonásobné klasifikace lze použít stejné mechanismy jako u binární klasifikace. V tomto případě se pokaždé nahlíží na jednu třídu vůči všem ostatním. Tento algoritmus se nazývá *One-Vs-All.*

1.2.2 Regresní úloha

Problematika regrese nemá má na rozdíl od klasifikace spojitý charakter.
V závislosti na tomto faktu je potřeba zvolit jinou ztrátovou funkci. Typicky se pro regresi používá střední kvadratická chyba (*mean squared error*, MSE). Jak název napovídá v tomto případě je spočítán průměrný rozdíl čtverců predikovaných cílů a očekávaných hodnot. Stejný algoritmus využívá také metoda nejmenších čtverců. V tomto případě je rovněž potřeba minimalizovat chyby mezi předpoklady a predikcí.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \widetilde{y_i})^2. \qquad (1.2)$$

kde N je množství vzorků y_i je korektní označení třídy a $\widetilde{y_i}$ je výstup predikce klasifikátoru.

1.3 Optimizéry

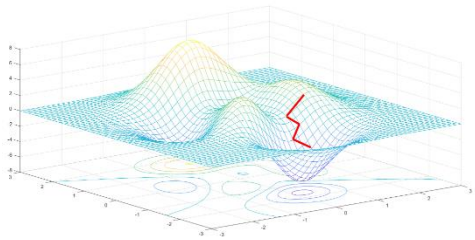
Součástí minulé podkapitoly byl rozbor základních typů úloh řešených při implementaci strojově učících se modelů, přičemž byly zmíněny algoritmy ztrátových funkcí, jež jsou základním pilířem k posouzení performance konkrétní sítě. Pokud je hodnota ztrátové funkce vysoká, je vhodným algoritmem potřeba zajistit její snížení. Toho je možné dosáhnout pomocí optimizérů. Součástí této kapitoly jsou algoritmy založené na gradientu. Gradientem se dá označit derivace tenzorových operací [1]. Proto podobně jako u derivací spočívá tato úloha v hledání takových parametrů neuronové sítě, které minimalizují hodnotu ztráty. Hodnoty pro optimalizaci jsou: váhy (*weights,* w), zkreslení (*bias,* b), mohou být patrné z rovnice (1.3) [1].

$$\bar{v} = \mathrm{relu}\left(\bar{w} + \bar{b}\right). \qquad (1.3)$$

Řešení gradientu nejčastěji probíhá za pomocí numerických metod, proto je potřeba volit krok učení (*learning rate,* lr), jež specifikuje posuv vah sítě. Musí být dostatečně malý na to, aby nedocházelo k divergenci při hledání minima, avšak pokud bude zvolena příliš malá hodnota, funkce nikdy nebude konvergovat ke globálnímu minimu [1]. V praxi je využíváno výhradně numerických metod. Výjimkou nejsou ani algoritmy implementované v této práci. Volby kroku učení jsou však často standardizovány pro různé typy úloh strojového učení. V práci se nejčastěji vyskytuje krok učení s hodnotou $1 \cdot 10^{-3}$. Dalším možným řešením by bylo použití analytických metod, které je na rozdíl od numerických metod explicitní, avšak výpočetně náročné s rostoucím množstvím vah. Obecně se jej nedoporučuje používat, je-li $w > 10^4$.

1.3.1 Stochastický gradientní sestup

Problémem řešení gradientu může být výpočetní náročnost, jelikož velikost vstupních dat bývá zpravidla v řádech tisíců. V praxi se proto data rozdělí do náhodných dávek (označované jako *batches*) o menších velikostech a parametry jsou upravovány na základě ztrátové funkce na těchto datových dávkách [1]. Jelikož je funkce diferencovatelná, lze vypočítat její gradient. Gradient však určuje směr růstu, proto jsou váhy aktualizovány v opačném směru od gradientu s nastaveným krokem (viz výše). Ve svém minimu je derivace funkce minimální, proto je možné postupně krok učení snižovat, aby se hodnota ztrátové funkce maximálně minimalizovala. Název stochastický reflektuje skutečnost náhodného poklesu směrem k lokálnímu minimu ztrátové funkce [1]. Algoritmus SGD (*Stochastic Gradient Descent*) zachycuje obrázek 1.1 – červená křivka kopíruje sestup pro jednotlivé dávky dat.



obrázek 1.1) Gradientní sestup pro 3D ztrátovou funkci

Výběr vhodné velikosti datové dávky je rovněž stěžejní. Kritická je zpravidla jeho dolní hranice. Je-li totiž zadána příliš nízká hodnota datové dávky, bude docházet k pomalé konvergenci k lokálnímu minimu. Obecně je doporučeno používat alespoň velikost datové dávky o hodnotě 64 [1]. Tato skutečnost je rovněž respektována v této práci, kde je použito velikostí 64/128 podle výpočetní náročnosti dané úlohy.

1.3.2 SGD s hybností

Problémem gradientu může být skutečnost, že namísto globálního minima bude konvergovat k minimu lokálnímu. Tento fenomén rovněž zachycuje obrázek 1.1, kde jsou patrná dvě minima. Tato komplikace může být vyřešena použitím SGD s hybností, který nezohledňuje pouze hodnotu gradientu, ale přidává do rovnice další proměnnou – hybnost. Tento algoritmus našel inspiraci ve fyzice. Interpretovat se dá pomocí kuličky, která je schopna sledovat křivku ztrátové funkce a je velmi málo pravděpodobné, že se při své cestě zastaví v lokálním extrému. Toho je dosaženo nejenom podle aktuální hodnoty gradientu, ale také podle aktuální rychlosti, která je odvozena od předešlých hodnot. Prakticky to znamená, že krok učení je upravován na základě dat z aktuální i minulé iterace [1]. Ve frameworku *keras* lze najít optimizér implementující SGD pod názvy: *Adam, Adagrad, RMSprop, atd.* Při implementaci modelů strojového učení byl převážně použit optimizér s hybností Adam. Experimenty byly také prováděny optimizérem SGD, avšak výkonnost sítí s jeho použitím dosahovala nižších evaluačních metrik výsledného modelu.

1.3.3 Algoritmus zpětného šíření

V úvodu kapitoly 1.3 byla opodstatněna důležitost výpočtu gradientu při hledání vah sítě. Při výpočtu derivace na absolutně diferencovatelném intervalu nelze jednoduše použít analytické metody, které jsou sice explicitní, ale s rostoucím množstvím parametrů neuronové sítě (*Neural Network,* NN) jsou značně náročné na výpočetní výkon. K výpočtu parametrů se proto používají numerické metody. Pomocí ztrátové funkce lze vypočítat ztráty systému při průchodu trénovacích dat neuronovou sítí. Toto ztrátové číslo předává informaci o způsobu nastavení vah. To v principu znamená úpravu vah sítě od poslední vrstvy (odtud název *Backpropagation Algorithm*). Samotný algoritmus potom využívá řetízkové pravidlo derivací, jelikož jednotlivé neurony ve vrstvě *n+1* jsou závislé na parametrech vrstvy *n*. Řetízkové pravidlo interpretuje rovnice (1.4).

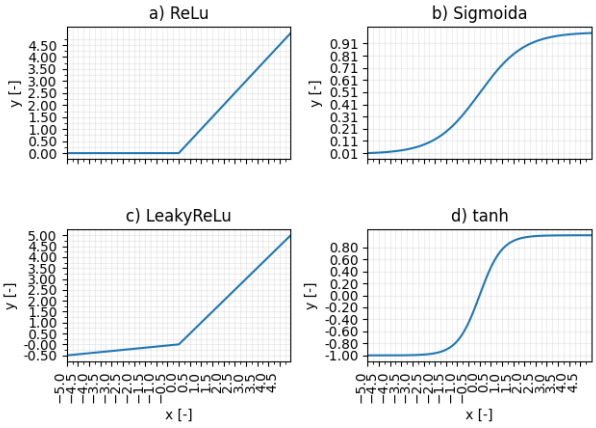
$$\frac{\partial}{\partial x} f\big(g(x)\big) = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}. \tag{1.4}$$

kde $f(x)$ reprezentuje funkci poslední vrstvy sítě a $g(x)$ reprezentuje n – 1 vrstvu sítě. Mechanismus zpětného šíření je součástí základních knihoven realizující strojové učení. Prostřednictvím frameworku *keras* je tento algoritmus spuštěn voláním funkce *fit*

1.3.4 Aktivační funkce

Aktivační funkce tvoří základní stavební bloky NN. Konkrétní implementaci ukazuje rovnice (1.3), kde je jako aktivační funkce použita lineárně lomená funkce (*Rectified Linear Unit*, ReLU), která je v úlohách strojového učení hojně používána.

Dalšími funkcemi jsou např. LeakyRelu, Sigmoida nebo hyperbolický tangens (viz obrázek 1.2a-d). Tyto funkce mají za úkol reprodukovat nelinearitu systému. Použití lomených lineárních funkcí (viz obrázek 1.2a,c) má své opodstatnění při hledání derivací těchto funkcí. Jednak je samotný výpočet derivací výpočetně jednodušší a jednak je vyšší pravděpodobnost nalezení globálního minima těchto funkcí [1]. Použití jednodušších lineárních funkcí bylo rovněž upřednostněno při řešení klasifikátoru na predikci defektů.



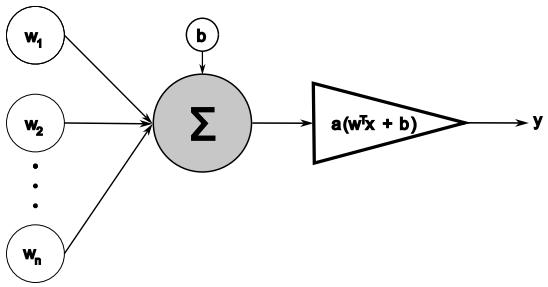
obrázek 1.2) Aktivační funkce

2. NEURONOVÉ SÍTĚ

Tato kapitola je soustředěna na problematiku neuronových sítí a rozebírá konvoluční neuronové sítě, které jsou hojně využívány při řešení úkolů z oblasti počítačového vidění (*computer vision*, CV). Dále jsou zde řešeny základní stavební bloky – vrstvy – konvoluční neuronové sítě (CNN). Realizace těchto bloků je v práci řešena prostřednictvím frameworku *keras* programovacího jazyka python.

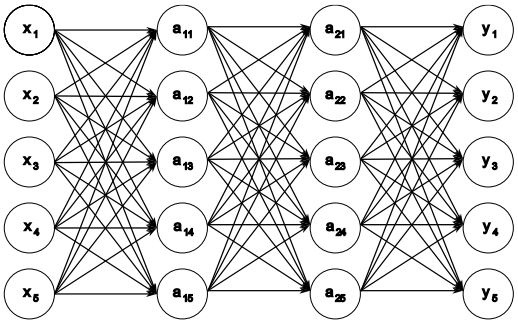
2.1 Neuronové sítě

Neuronové sítě mají funkci nástroje transformace reprezentací. Na svém vstupu přijímají data a pomocí vhodně nastavených vah je upravují na užitečnější reprezentace [1]. Základním stavebním blokem NN je perceptron – model neuronu (viz obrázek 2.1, kde *a* značí aktivační funkci – kapitola 1.3.4).



obrázek 2.1) Model neuronu [4]

Skládáním a propojováním základních bloků je definována architektura neuronové sítě. Každá architektura obsahuje vstupní a výstupní vrstvu a libovolný počet skrytých vrstev (viz obrázek 2.2). Jsou-li všechny perceptrony mezi vrstvami navzájem propojeny, lze hovořit o hustě propojené síti (*fully connected*, FC) [1]. Výstupním vrstvám jsou nastaveny aktivační funkce vhodné pro konkrétní implementaci (např. pro výběr 1 z 10 kategorií je vhodné použít funkci softmax). Použití FC vrstev se zpravidla na úkoly strojového vidění příliš nehodí. Jsou však v práci použity v posledních vrstvách modelu pro transformaci reprezentací před konečnou predikci. Lze s nimi dosáhnout uspokojivé přesnosti sítě i v úlohách počítačového vidění, ale mnohem lepších výsledků je dosaženo použitím konvolučních neuronových sítí (*convolutional neural networks*, CNN) [1].



obrázek 2.2) Příklad hustě propojené sítě (FC)

2.2 Konvoluční neuronové sítě

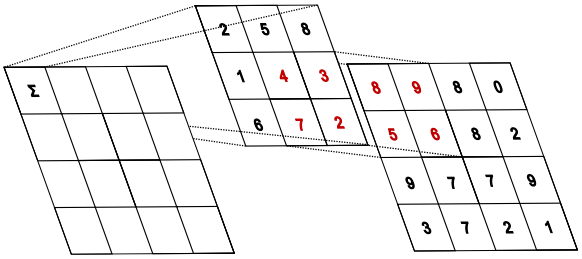
Konvoluční neuronové sítě jsou využívány v problematice strojového vidění. Spadají do podmnožiny neuronových sítí [4]. Běžně implementovanou architekturou je použití CNN k předzpracování obrazu a tyto výsledky jsou dále zpracovány hustě propojenou sítí. Příkladem může být architektura ResNet, která byla použita při klasifikačních úlohách této práce. CNN má na vstupu tenzor tvaru (výška obrazu × šířka obrazu × kanály) [1]. CNN nejčastěji používají následující vrstvy: konvoluční vrstva, dekonvoluční vrstva, maxpool a unpool vrstvy. Vzhledem k velikosti obrazových dat už není u CNN možné propojovat všechny neurony [5].

2.2.1 Konvoluční vrstva

Hlavní výhodou použití CNN proti hustě propojené síti je skutečnost, že FC se učí globální vzory ze vstupních dat, zatímco CNN se učí vzory lokální [1]. Konvoluční vrstva je popsána rovnicí (2.1:

$$Y = X * F \Leftrightarrow y_{i,j} = \sum_{k=1}^{n} \sum_{l=1}^{n} f_{k,l} x_{i+k,j+l}. \qquad (2.1)$$

kde *F* je konvoluční jádro a X tvoří prostor, kde je konvoluce prováděna. Velikost konvolučního jádra se zpravidla volí jako čtverec o velikosti hrany 3, 5 nebo 7. Tato jádra v sobě nesou informaci o hranách objektu, kontrastu, atd. Rovnici (2.1 ilustruje obrázek 2.3. Obráceným postupem lze získat dekonvoluční vrstvu.

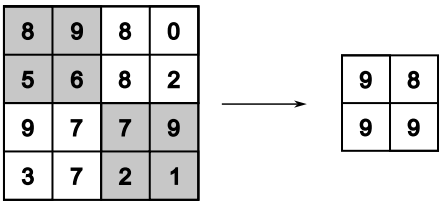


obrázek 2.3) Konvoluční vrstva

CNN vrstvy jsou podobně jako hustě propojené sítě charakterizovány množstvím filtrů obsažených v dané vrstvě. CNN jsou stěžejní součástí topologií ResNet, jež jsou v této práci hojně využívány.

2.2.2 Sdružovací vrstva (*maxpool vrstva*)

Tento typ vrstvy slouží k redukci vstupních dat. S redukcí vstupů se přímo pojí rychlost zpracování těchto dat. Podobně jako u konvoluce se u maxpool vrstvy určí velikost jádra. Dále musí být stanoveny parametry horizontálního a vertikálního kroku. Nejčastěji je možné se setkat s jádrem velikosti 2 × 2 s krokem o stejné velikosti. Při zvolení takového jádra se velikost vstupních dat sníží na polovinu. Obecně lze stanovit, že pro vstup velikosti $a \times b$ bude výstup po průchodu touto vrstvou $\frac{a}{w} \times \frac{b}{h}$, kde w a h jsou rozměry jádra. Princip ilustruje obrázek 2.4. Maxpool vrstva vrátí na výstupu číslo s maximální hodnotou nacházející se oknu jádra.
V případě této komprese dochází k nenávratné ztrátě dat, distribuce informace se však nezmění [4].

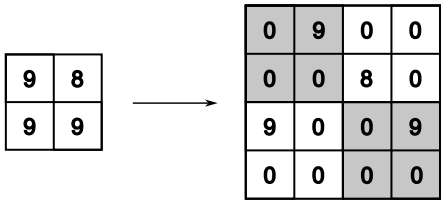


obrázek 2.4) znázornění maxpool vrstvy s jádrem 2 × 2 [4]

Mimo maxpool založeném na sdružování na základě maximální hodnoty v jádře, existuje ještě alternativa založená na průměrování hodnot v okně jádra (*average pooling*).
Maxpool vrstva má své opodstatnění při redukci množství parametrů sítě. V sítích ResNet, které jsou v této práci implementovány se tato vrstva vyskytuje na začátcích přeskakujících bloků (viz 6). Společně s touto vrstvou je realizováno rozšíření filtrů CNN na dvojnásobek původní velikosti.

2.2.3 Zpětně sdružovací vrstva (*unpooling*)

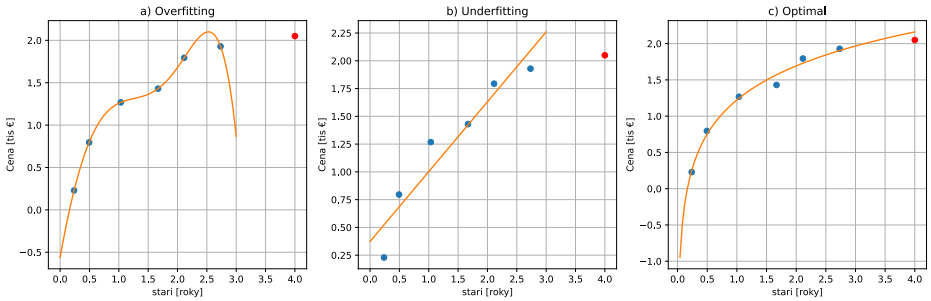
Tento typ vrstvy provádí inverzní operaci k sdružování, tzn. vstupní data jsou zvětšována. Podobně jako u maxpooling vrstvy je definována velikost jádra. Dimenze výstupních dat bude s uvážením jádra velikosti w × h rovna $wa \times hb$. Při zvětšení jsou neznámá místa zpětně doplněna nulovými hodnotami. Tento princip znázorňuje obrázek 2.5 [4].
Využití unpool vrstev má význam u reziduálních sítí, kde se aditivní metodou přičítá signál z předchozích vrstev k vrstvě aktuální. Často je totiž nutné zmenšený obraz zvětšit, aby rozměry si rozměry sčítaných vrstev odpovídaly. Pro operaci sečtení je potřeba stejná dimenze vstupních tenzorů, proto je nutné, např. operací unpool, dimenze vstupů vyrovnat.



obrázek 2.5) Znázornění unpool vrstvy [4]

3. PROBLÉMY PŘI UČENÍ

Při procesu učení sítí může dojít k celé řadě problémů. Již v kapitole 1 bylo naznačeno, že volbou datasetu se špatnou generalizující distribucí bude docházet ke špatným predikcím na nových datech. Tato situace může být jednou z příčin podučení (*underfitting*), kdy síť z dostupného datasetu není schopna extrahovat další příznaky. Obrácená situace se nazývá přeučení (*overfitting*). Součástí této kapitoly je rozbor těchto fenoménů a způsoby, kterými jim lze předejít.



obrázek 3.1) Příklad a) přeučení (*overfitting*) b) podučení (*underfitting*) a c) optimální predikce

3.1 Přeučení

Přeučení vzniká v případech, kdy je model příliš přizpůsoben trénovacímu datasetu a ztrácí schopnost generalizace a správné predikce na nových (neviděných) datech. Problém přeučení lze jednoduše ilustrovat na regresní problematice (viz obrázek 3.1a). Zde je patrný trend prediktoru ceny zboží v závislosti na jeho staří. Při použití polynomické rovnice vyššího řádu lze pozorovat špatnou predikci na nových datech (viz obrázek 3.1a – červeně). Přeučení se rovněž projeví na křivkách pamatování (viz obrázek 3.2a). Zde se projevuje degradace ztráty od 3. epochy. Další častou příčinou přeučení může být nedostatečná velikost trénovacích dat. V tomto případě není síť schopná extrahovat dostatečné množství příznaků, které by měly generalizující charakter. Zvýšení pamatovací kapacity (složitosti) sítě nebude mít na její performanci výraznější vliv [1]. V praxi je proto nejprve konkrétní dataset otestován na ověřené síti a následně vyzkoušen na nových architekturách. Součástí této práce však nebylo vytvoření vlastní architektury pro zpracování vstupních dat. Především byly použité již ověřené architektury, a tak nebylo nutné tuto dodatečnou kontrolu realizovat. Postupy pro zmírnění přeučení jsou následující:

3.1.1 Zjednodušení architektury

Jedná se o nejjednodušší způsob, kterým lze přeučení zabránit. Spočívá ve zjednodušení architektury systému, což znamená snížení množství učebních parametrů, které se v hlubokém učení také označují jako kapacita modelu [1].

Takový model má svou pamatovací kapacitu, tzn. schopnost modelu učit se příznaky z dat. Je-li pamatovací kapacita vysoká, pak se model naučí pamatovat trénovací příklady s jejich cíli. Takový model je však nepoužitelný [1]. Pro obrázek 3.1a by takové zmírnění znamenalo snížení stupně polynomu. Je-li však architektura zjednodušena příliš, může docházet k opačnému problému – podučení (viz obrázek 3.1b).

3.1.2 Váhová regularizace (*weight decay*)

Problematika regularizace spočívá v penalizaci vah sítě, což vede k jejich pravidelnější distribuci [1]. Tento způsob penalizace se nazývá regularizace vah a je implementován adicí ztrátové funkce, jež je napojená na váhy sítě. Lze obecně předpokládat, že jednodušší architektury budou na přeučení méně náchylné.
V závislosti na implementaci ztrátové funkce jsou rozlišovány následující případy [1]:

- L1 regularizace: vztah mezi hodnotou penalizace a váhových koeficientů je přímo úměrný [1]
- L2 regularizace: vztah mezi hodnotou penalizace a váhových koeficientů je kvadraticky úměrný [1]

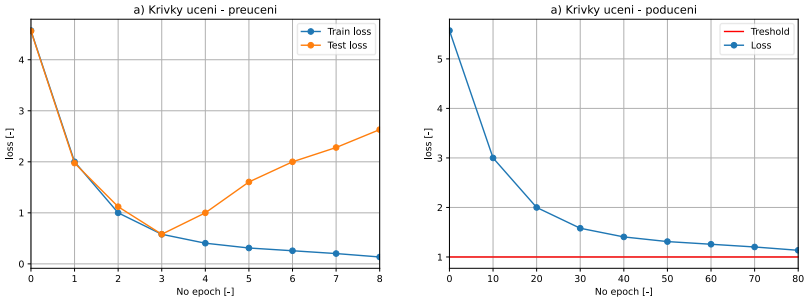
3.1.3 Přidání výpadku (*dropout*)

Výpadek je realizován náhodným vynecháváním skupiny vstupních příznaků během trénování s danou pravděpodobností. To v konečném důsledku mění distribuci vlivu vah přispívajících do ztrátové funkce. Přidání výpadku je nejefektivnější a nejčastěji implementovaný regularizační způsob při zvládání přeučení [1] [4].

3.2 Podučení

Podučení (*underfitting*) je protipólem k přeučení. Nejčastěji je způsobeno nízkou pamatovací kapacitou sítě. Tento fenomén může být opět ilustrován regresí (viz obrázek 3.1b). Z obrázku je patrné, že lineární funkce nebude vhodná volba pro modelování nelinearit systému. Za těchto okolností je potřeba zvolit složitější architekturu sítě. Podučení ukazuje obrázek 3.2b, kdy je patrné, že i po osmdesáti epochách nedochází k zlepšení ztráty pod požadovanou úroveň [1]. Podučení se projeví v testovací i trénovací fázi.

Sítě ResNet obsahují velké množství učebních parameterů. Lze tak očekávat možnost případného přeučení této sítě. Při řešení jednotlivých klasifikačních úloh v této práci bylo právě přeučení datasetu jednou z hlavních nástrah. Jeho kompenzace byla realizována přidáním výpadku nebo L2 regularizace. Toto řešení se ukázalo jako dostačující.



obrázek 3.2) přeučení **a)** a podučení **b)** na křivkách učení

4. PŘÍPRAVA A VYHODNOCENÍ DAT

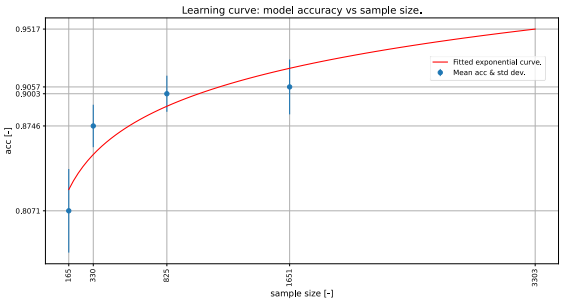
Při trénování NN nebo CNN je volba trénovacích dat zásadní jednak pro následné správné vyhodnocení a jednak pro možnost nasazení natrénovaného modelu do provozu. Součástí této kapitoly je rozbor metod používaných v problematice strojového vidění.

4.1 Distribuce a velikost datasetu

První otázkou jakéhokoliv strojově učícího se algoritmu bude velikost dat, potřebná k úspěšnému natrénování modelu. Odpověď na ni se však může lišit v závislosti na řešeném problému (např. velikost datasetu se bude odlišná v případě binární klasifikace a v případě klasifikace do deseti tříd). V praxi se lze často potkat s empirickými pravidly, používanými především při prvotním odhadu velikosti datasetu. Jedno z nich hovoří o minimálním množství 1000 vzorků na třídu. Takové řešení však nemusí být dostatečně robustní a v některých případech může být takové množství nedostatečné. Jedním z možných přístupů je postupné vzorkování datasetu [6].

4.1.1 Postupné vzorkování datasetu (*Sample-Size Methodology*)

Tato metoda je poměrně jednoduchá na implementaci, zároveň je však velice efektivní. Spočívá v trénování několika modelů při postupném zvyšování množství vzorků ve vybraném datasetu (např. při použití 5 %, 10 %, 50 %, atd. původního datasetu). Poté je vypočítána průměrná hodnota přesnosti pro všechny podmnožiny původního datasetu. Tímto způsobem lze poté pozorovat trend přesnosti datasetu (viz obrázek 4.1) [7].



obrázek 4.1) Závislost přesnosti modelu na velikosti datasetu [7].

Metoda postupného vzorkování datasetu je v práci použita pro vyhodnocení datasetu, kde bylo z křivek učení patrné nedostatečné množství dat k trénování (viz 6.2). Aplikací tohoto algoritmu se tato teze potvrdila.

4.1.2 Distribuce dat

Kontrola distribuce vstupního datasetu je rovněž jedním z nezbytných kroků před samotným trénováním modelu. Problém nastává především, je-li jedna z tříd zastoupena výrazně více než všechny ostatní (např. 1 % ku 99 %). Taková distribuce může přinést další nástrahy při trénování sítě. Především se síť vůbec nemusí naučit rozpoznávat příznaky menší třídy. Způsobů řešení je několik. Ty nejpoužívanější jsou následující:

- Slučování tříd: V tomto případě se dochází ke sloučení vizuálně podobných tříd. Takový zásah může být v některých případech nežádoucí (např. při klasifikaci kočkovitých šelem je žádoucí od sebe jednotlivé druhy oddělit).
- Převzorkování: Vzorky z méně zastoupené třídy jsou pomocí augmentačních metod rozšířeny (viz 4.3).
- Podvzorkování: Vzorky z majoritní skupiny jsou náhodně selektovány a z datasetu odstraněny

V praxi se často používají kombinace některých z výše uvedených metod (nejčastěji převzorkování s podvzorkováním). Kontrola distribuce má dále své opodstatnění při implementaci přetrénovacích algoritmů. Dochází-li ke změnám distribuce, je nutné identifikovat jejich původ a zahrnout jej do algoritmu [2].
Pro úlohy počítačového vidění je generace syntetických dat pro převzorkování některých tříd problematická. K tvorbě nových dat se zpravidla využívá generativní učení. Jeho implementace a ladění však může být časově náročné (viz 8.4.1) [8].
Podvzorkování majoritních tříd se rovněž ukázalo jako problematické, jelikož jeho implementací v této práci docházelo k významným poklesům sledovaných metrik v průběhu trénování. Vyrovnání distribuce v průběhu učení je tak řešeno váhováním jednotlivých tříd. Algoritmus je založený na přidělení vyšší významnosti méně zastoupeným třídám v průběhu učení. Tento mechanismus je rovněž jednoduchý na implementaci, jelikož je součástí základních knihoven programovacího jazyka python (*keras*, *scikit-learn*)

4.2 Čištění dat

Čištění dat je soubor operací, který si klade za cíl odstranění chybných nebo nadbytečných dat. Do této skupiny spadají především špatně označené skupiny vzorků, zašuměné vzorky nebo podobné vzorky.

4.2.1 Kontrola chybných označení

Jedná se o inspekci správnosti označení cílů u obrázkového datasetu. Tato inspekce může probíhat jednak manuálně a jednak automaticky. Nevýhody manuální kontroly jsou jednoznačné. Zahrnují především zdlouhavost procesu a také chybu lidského

úsudku (lidský faktor). Manuální metoda je rovněž finančně nákladnější ve srovnání s automatickou.

Řešení pro automatickou kontrolu značení je algoritmus *labelfix* [9]. Labelfix je inspekční nástroj, obsahující na svém vstupu kontrolovaný dataset a procento vzorků, které má při inspekci odstranit. Jedná se o X % vzorků, které jsou velmi pravděpodobně špatně označené. Metoda filtrace je intuitivní a lze ji shrnout do 4 kroků [9]:

1. Natrénování klasifikátoru na celém datasetu (bez oddělení na trénovací a testovací set)
2. Inference na tomtéž datasetu
3. Výpočet skalárního součinu $\langle y_n, y_n'' \rangle$, kde y_n je označení třídy dané datasetem a y_n'' je vektor pravděpodobností klasifikátoru
4. Seřazení hodnot z bodu 3 a extrakce zvolených X %

Implementace této architektury na vstupních datech byla provedena, avšak skalární součin jednotlivých dat byl minimální i pro nejhorších 10 % výsledků. Z tohoto závěru lze dedukovat vysoce kvalitní označení všech trénovacích dat. Jelikož algoritmus si klade vyšší výpočetní nároky v průběhu trénovací sekvence bylo jeho spuštění v průběhu trénovací sekvence na sledovaných datech deaktivováno. Artefakty tohoto kódu jsou však patrné v souboru *libs/dataProcessing/preprocessing.py*

4.2.2 Duplicitní obrázky

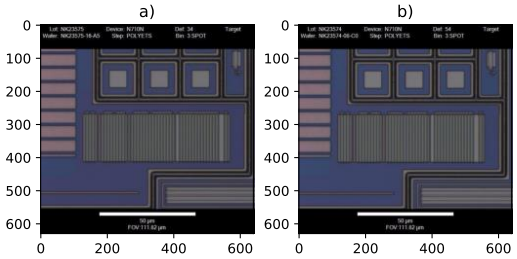
Ponecháním podobných obrázků v datasetu může docházet k nahodilým chybám při vyhodnocení natrénovaného modelu. Tyto chyby jsou především spojeny s duálním výskytem vzorků (velmi podobné obrázky v trénovacím a testovacím setu). Jejich vliv bude s rostoucím množstvím duplicit významnější.

Řešení podobnosti obrázků založené na porovnávání jednotlivých pixelů by bylo časově náročné, proto je výhodné použít předtrénované sítě (např. MobileNet), jež transformují vstupní příznaky na užitečnější reprezentace menších velikostí.

Z těchto výstupů se spočítá kosinova podobnost (viz rovnice (4.1)), jejíž výsledkem je matice vzájemných podobností obrázků. Výpočet matice lze zjednodušit, jelikož je symetrická podle hlavní diagonály (např. je-li řešena podobnost prvního obrázku vůči ostatním, pak podobnosti ostatních obrázků vůči prvnímu jsou rovnocenné, atd.) [6].

$$S_C(A,B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{N} A_i B_i}{\sqrt{\sum_{i=1}^{N} A_i^2}\sqrt{\sum_{i=1}^{N} B_i^2}}. \quad (4.1)$$

kde A značí výběr reprezentací obrázku A a B je výběr reprezentací obrázku B. Obecně lze považovat obrázky s vyšší podobností jak 90 % jako vhodné k filtraci. Vzájemná podobnost je také patrná z vizuální inspekce (viz obrázek 4.2).



obrázek 4.2) Obrázky s podobností $S_C > 90$ %

Vyhodnocení dat prostřednictvím selekce duplicitních obrázků bylo v původním řešení této práce očekáváno. Po jeho realizaci však bylo zjištěno, že množství duplicitních obrázků bylo minimální (< 1 %). Za těchto okolností lze považovat algoritmus hledání duplicit za redundantní. V průběhu trénovací sekvence tento algoritmus použit nebyl. Artefakty jeho implementace jsou patrné v souboru *similarity.py*.

4.3 Rozšíření dat (augmentace)

Cílem rozšíření dat je schopnost generovat nová data ze stávajícího datasetu. Rozšíření dat rovněž snižuje pravděpodobnost přeučení sítě, jelikož model má stále k dispozici nové příznaky, z kterých se síť může učit. Je-li v ideálním případě množina vstupních dat nekonečná, pak je pravděpodobnost přeučení téměř nulová. V praxi ale nelze zaručit nekonečnou množinu dat. Existují však způsoby, kterými lze dataset dostatečně rozšířit, a zmírnit tak vliv přeučení. Mezi ty mohou patřit následující úpravy: rotace, přiblížení, zrcadlové otočení nebo náhodný střih obrázků [1].

Transformací obrázků nelze získat nebo přidat nové informace. Metoda augmentací proto nemusí být dostačující pro úplné odstranění přeučení. Často ji bude potřeba doplnit dalšími algoritmy minimalizující přeučení (dropout, váhová regulace) [1]. Volbou špatných augmentací může docházet k poklesům metrik natrénovaných modelů. Součástí této práce byl implementován algoritmus hledání takových augmentačních vrstev, které maximalizují přesnost modelu (viz 7.2).

4.4 Vyhodnocení

V předešlých kapitolách byly zmíněny 2 základní metriky vyhodnocení dat: přesnost a ztrátovost (*accuracy, loss*). Tyto metriky však nemusí být vždy dostačující pro správné posouzení výkonnosti neuronové sítě, zejména v případech nevyrovnaných predikcí (např. pouze 1 % lidí trpí danou chorobou). Lze proto konstatovat, že metriky založené na přesnosti nepříliš penalizují špatné predikce, proto se častěji lze setkat s metrikami založenými na matici záměn (*confusion matrix*). Příklad takové

matice ilustruje tabulka 4.1 [4]. V tabulce jsou zvýrazněné shody s prediktorem šedou barvou.

tabulka 4.1) Matice záměn pro kategorie a, b, c

		Ref.		
		a	*b*	*c*
Pred.	*a*	*3*	*1*	*1*
	b	*1*	*2*	*2*
	c	*0*	*1*	*15*

Z matice záměn lze definovat další hodnotící metriky: *precision, recall.* Metrika precision udává poměr správně označených vůči součtu prvků predikcí v dané kategorii. Metrika recall naopak udává poměr správných predikcí vůči všem prvkům, jež mají referenci v dané kategorii. Výpočet precision a recall se provádí pro každou kategorii zvlášť, průměrováním přes všechny kategorie lze získat průměrnou hodnotu precision a recall celé sítě.

4.4.1 F1 skóre

F_1 skóre je často používaná metrika, shrnující performanci natrénované sítě. Představuje kompromis mezi precision a recall (viz rovnice (4.2)) [10].

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{4.2}$$

Jelikož $precision, recall \in \langle 0; 1 \rangle$ lze prohlásit, že F_1 bude nabývat taktéž hodnot z tohoto intervalu. Nejlepší možné skóre, které síť může dosáhnout je 1 [10].

4.4.2 ROC skóre

ROC (*Receiver Operating Characteristics*, ROC) křivka je způsob evaluace dat pro binární modely. Křivky určují poměr mezi korektní predikcí (*True Positive Rate*, TPR) vůči falešně pozitivní predikcí (*False Positive Rate*, FPR).
V ideálním případě je poměr falešně pozitivních predikcí nulový a korektních predikcí maximální. Tuto situaci ilustruje obrázek 4.3. V ideálním případě prochází křivka bodem $A = [0; 1]$. Jestliže je ROC křivka přímo úměrná (se úhlem sklonu $\pi/4$), potom lze klasifikátor označit jako náhodný.

obrázek 4.3) ROC křivka

Plocha pod křivkou ROC představuje metriku AUC (*Area Under Curve*). Tato metrika poskytuje dodatečnou informaci k natrénovanému modelu. Lze usoudit, že model s modrým průběhem bude vhodnější použít než model se zeleným průběhem (viz obrázek 4.3). V ideálním případě se AUC = 1. Avšak jakýkoliv model s AUC > 0,9 lze považovat za velmi uspokojivý. Metriku AUC je výhodné použít především při vyhodnocování trénování na nevybalancovaných datasetech a v situacích, kde není znám práh predikce [8].

Součástí této práce je rovněž rozbor distribuce defektů jednotlivých tříd. Obecně lze tvrdit, že jejich distribuce není rovnoměrná. Některé defekty vykazují ojedinělý charakter, zatímco jiné jsou mnohem četnější. Z tohoto důvodu nelze považovat metriku přesnosti za dostačující při vyhodnocení výkonnosti sítě. Z hlediska zpětné vazby je pro operátory rovněž nepřípustné slučování jednotlivých tříd za účelem vytvoření super-třídy (viz 4.1.2). V této práci jsou tak naznačeny další metriky, které dokáží nerovnoměrný charakter distribucí reflektovat. V textu se tak při vyhodnocování sítě kromě přesnosti používají metriky F1 skóre a AUC skóre.

5. DETEKCE DEFEKTŮ U POLOVODIČŮ

Problematika detekce defektů při výrobě polovodičových technologií je komplexní záležitost zahrnující nejen přímé sledování defektů, ale kontrolu samotných zařízení a prostor, ve kterých jsou polovodiče vyráběny (*tool monitoring*). Defekty na deskách (*waferech*) vznikají v jednotlivých etapách samotné výroby (např. při leštění, přípravě struktur). Tyto defekty se dají detekovat optickou metodou nebo měřením na sondách.

5.1 Druhy inspekcí

5.1.1 Inspekce zařízení

Kontrola zařízení používaných v průběhu výrobního procesu je stěžejní pro výrobu samotnou. Tato zařízení je potřeba kontrolovat při uvedení do provozu (*dark field inspection*) a v případech změny rozložení distribuce defektů. Kontroly se provádí na testovacích waferech, které mají determinované parametry. Po průchodu zařízením dochází k hledání rozdílů výstupních waferů se vstupní referencí.

5.1.2 Inspekce místností

Při výrobě polovodičů je potřeba dbát na čistotu výrobních hal a místností, jelikož prachové částice na waferech způsobují zvýšenou defektivitu – přičemž defekt vzniklý prachovou částicí se může v průběhu dalšího zpracování zvětšovat. Norma ISO 14644-1 stanovuje specifika čistých místností (viz tabulka 5.1).

tabulka 5.1) ISO 14644-1 Specifika čistých místností [11]

Třída	Maximum částic / m^3					
	$\geq 0.1[\mu m]$	$\geq 0.2[\mu m]$	$\geq 0.3[\mu m]$	$\geq 0.5[\mu m]$	$\geq 1[\mu m]$	$\geq 5[\mu m]$
ISO 1	10	-	-	-	-	-
ISO 2	100	24	10	-	-	-
ISO 3	1 000	237	102	35	-	-
ISO 4	10 000	2 370	1 020	352	83	-
ISO 5	100 000	23 700	10 200	3 520	832	-
ISO 6	1 000 000	237 00	102 000	35 200	8 320	293
ISO 7	-	-	-	3 520 00	83 200	2 930
ISO 8	-	-	-	3 520 000	832 000	29 300
ISO 9	-	-	-	35 200 000	8 320 000	293 000

5.1.3 Inspekce waferů

Tyto inspekce lze rozdělit do 2 kategorií: Pravidelné a nepravidelné. Pravidelné inspekce jsou časově nenáročné a finančně nenákladné. Pravidelnými inspekcemi se zjišťuje především následující skutečnost:

- Posuv distribuce defektů: v důsledku incidentů – zařízení není v kondici, lidský faktor nebo při výjimečných situacích špatná situace v místnosti (vysoká prašnost)

K nepravidelným inspekcím se přistupuje v rámci experimentování, popř. při nasazení nových strojů do výroby. K nepravidelným kontrolám lze zařadit také inspekce na elektronovém mikroskopu (SEM). Výhodou této inspekce je možnost zjištění původu konkrétního defektu (např. obsah částic titanu – defekt vznikl v poslední etapě výroby)

Ve sledovaných závodech má inspekce 7 úrovní (1-7). Tyto kontroly probíhají v jednotlivých etapách výroby polovodičů. Vyhodnocení probíhá na základě statistické analýzy. Jsou kontrolovány následující parametry:

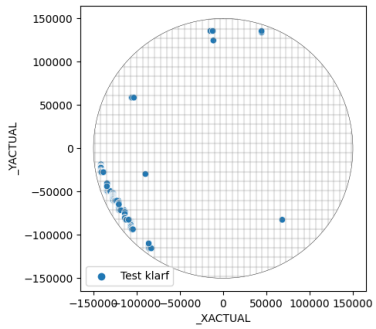
- Z hlediska posuvů distribuce se zjišťuje hustota pravděpodobnosti jednotlivých defektů (srovnání s různými deskami)
- Defekty, jež způsobí úplné zničení součástky (*defect of interest,* DOI) – aspekt dlouhodobé životnosti (např. koroze – postupně se rozrůstá)
- Oblastní defekty (*pattern issue*) – týkající se poškození oblastí (např. škrábance na deskách, špatná expozice)

Výstupní inspekce je posledním krokem předtím, než wafery opustí závod. Tato kontrola probíhá optickou metodou a měřením na sondách. Data získaná pro trénování a následnou inferenci jsou získána právě optickou metodou. Jednotlivé obrázky defektních waferů jsou pořízeny prostřednictvím rozhraní od společnosti KLA. Součástí této kontroly je rovněž zakapání chybných čipů na desce inkoustem. Kontrolu lze interpretovat jako síto s nastavenou hrubostí. Čím jemnější bude, tím nižší bude propustnost vyhovujících čipů. To má za následek zvýšení ztrátové výtěžnosti. Cílem této práce je navrhnout algoritmus, využívající metod strojového učení, který bude pracovat jako nástroj sekundární inspekce výstupní kontroly. Při této konfiguraci bude možné nastavit jemnější filtr a vyřazené čipy budou dotřiďovány natrénovaným modelem.

5.2 Sběr dat

Jak uvádí podkapitola 5.1.3, obrázky jsou získávány prostřednictvím softwaru od společnosti KLA. Při pořizování jednotlivých snímků lze použít různých přibližovacích čoček. Pro strojově učící se model je však konzistence dat velmi důležitá, neboť při rozdílné distribuci vstupních obrazových dat může docházet k nahodilým chybám při trénování modelů. Rovněž charaktery defektů pro různá

přiblížení mohou vykazovat vysoké znaky odlišností. Granularita jednotlivých snímků pořízených softwarem nemusí vzhledem k povaze defektů přímo korelovat s velikostí jednoho čipu na waferu (viz obrázek 5.1).

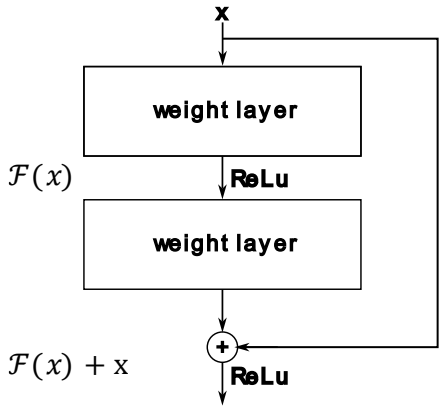


obrázek 5.1) Grafické znázornění waferu s defekty (modře)

.

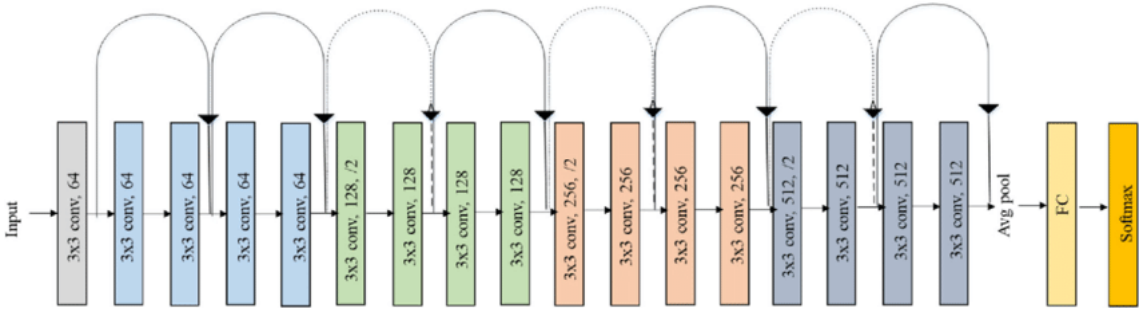
6. KLASIFIKÁTOR POMOCÍ RESNET SÍTÍ

Problémem použití NN nebo CNN při implementaci modelů využívajících hluboké učení je jejich omezení. Intuitivně by se dalo předpokládat, že zvýšením počtu vrstev dojde ke zlepšení performance sítě. Tento předpoklad je však mylný, neboť se zvyšujícím se množstvím vrstev nedochází k dalším zlepšením, a tak jednodušší síť může dosáhnout lepších parametrů než síť komplexnější [12]. Dalším problémem, souvisejícím s vyšší složitostí modelu, je explodující gradient (*exploding gradient*). Jedná se o situaci, při které jsou chyby gradientu kumulovány a váhy sítě jsou následně měněny s vysokým krokem.
Možným řešením výše popsaných skutečností je použití reziduálních bloků. Základem těchto sítí jsou tzv. reziduální stavební bloky: váhované vrstvy těchto bloků jsou propojeny s daty na vstupu těchto bloků (*skip connection*). Základní blok zobrazuje obrázek 6.1 [12].



obrázek 6.1) Základní stavební blok ResNet [12]

Podle množství použitých bloků *n* lze definovat počet vrstev sítě. Počet vrstev specifikuje konkrétní realizaci sítě *ResNet* (např. *ResNet18* obsahuje 18 vrstev). Sledovaný dataset byl nejprve natrénován 18vrstvou reziduální sítí [12], později byly použity 50vrstvé sítě. Základní architektura sítí *ResNet* vychází ze sítě *VGG16*, která byla obohacena právě o přeskakující bloky [12].



obrázek 6.2) Architektura sítě ResNet18 [12]

Architekturu sítě ResNet18 ukazuje obrázek 6.2. Jsou zde patrné zmíněné přeskakující bloky (šipkami). Množství filtrů v jednotlivých vrstvách je předem určené, přičemž dochází při operaci maxpool k jejich zdvojnásobení. Při dalším trénování byla tato architektura zachována, včetně velikosti filtrů jednotlivých konvolučních vrstev. V programové implementaci byl pouze změněn výstup této neuronové sítě, který je spojen se skupinou hustě propojených vrstev. Poslední hustě propojená vrstva této architektury obsahuje stejný počet jednotek jako je počet výstupních tříd.

6.1 Parametry trénování

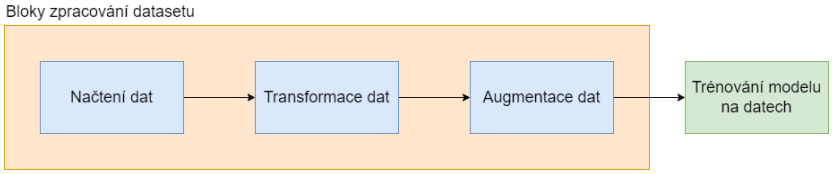
Předpokladem pro správnou performanci sítě je správné nastavení jejich hyper-parametrů. Tímto pojmem lze označit takové parametry sítě, které se v průběhu trénování modelu nemění. Jejich správné nastavení je prováděno ve fázi ladění modelu. Tyto parametry mohou být častou příčinou slabé performance sítě [8]. Jejich změna by proto měla být vždy dostatečně odůvodněna. Seznam vybraných hyper-parametrů, jež byly použity v 6.2 a 6.3 uvádí tabulka 6.1.

tabulka 6.1) hyper-parametry použité k trénování

Parametr	Data Belgie		Data EFK	
	Iterace I.	Iterace II.	Iterace I	Iterace II.
krok učení	0,001	0,001	0,001	0,0001
Optimizér	adam	adam	adam	adam
L2_regularizace	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$	$1 \cdot 10^{-6}$
Velikost dávky	64	64	128	128
Předčasné ukončení (trpělivost)	10	7	10	7

Dalším požadavkem z hlediska trénování je opakovatelnost výsledků. Nástrahou každého strojově učícího se modelu je nahodilost procesů v průběhu trénování (např. náhodná inicializace vah sítě). Tato nahodilost lze eliminovat použitím jader v průběhu trénování. Použití jader zajistí deterministický charakter výstupů trénování, proto jsou součástí kódu implementovány. Dalším aspektem, ovlivňujícím opakovatelnost výsledků neuronových sítí, jsou vstupní data. Jakákoliv manipulace s daty má přímý vliv na performanci sítě. V práci je proto použito jejich verzováním prostřednictvím databáze, jejíž poskytovatelem je ML studio od společnosti Microsoft. Verzovány jsou rovněž trénované modely sítě. Ty mohou být

následně využívány inferenčním zařízením při klasifikaci defektů. Nutno podotknout, že tyto kroky jsou v realizaci prováděny manuálně [2].



obrázek 6.3) Blokové schéma trénovací sekvence

Z hlediska přehlednosti je vhodné rozdělení kódu do ucelených bloků podle jejich funkce. Tyto bloky jsou následně sekvenčně spouštěny. Výstupy jednotlivých bloků jsou propojovány se vstupy dalších (tzv. *pipeline*). Blokové schéma trénovací sekvence ilustruje obrázek 6.3. Jednotlivé bloky zastávají následující funkce:

- **Načtení dat**: Součástí tohoto kroku dochází k načtení obrazových dat a jejich převodu do maticového zápisu. Načítání dat je implementováno skriptem *upload_data.py*
- **Transformace dat**: Tímto krokem se rozumí předzpracování dat. Především zde dochází k odstranění chybných dat, anomálií nebo chybně značených dat. Transformace dat je prováděna skriptem *preproces_train_data.py*
- **Augmentace dat**: Provedení různých geometrických/kontrastních úprav vstupních dat. Tyto úpravy jsou prováděny nahodile. Krok je důležitý především z hlediska redukce rizika přeučení modelu ve fázi trénování. Augmentace jsou realizovány skriptem *augment_data.py*
- **Trénování modelu na datech**: nastavení vah sítě prostřednictvím gradientního sestupu. Zde realizováno pomocí knihovny *keras*. Trénování probíhá ve skriptu *train_and_validate_model.py*

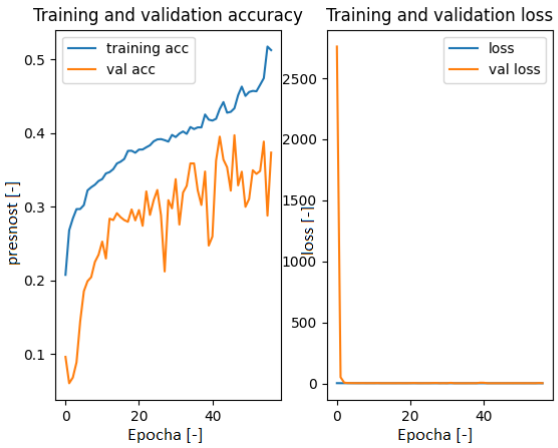
Výstupem běhu této kódové sekvence je soubor natrénovaných modelů z jednotlivých fází učení. Trénování bylo ve všech případech realizováno prostřednictvím K-násobné validace[1]. Natrénované modely jsou tak výsledkem dílčích přehybů této validace. Výstup rovněž obsahuje metriky průběhu trénování v jednotlivých epochách (tzv. křivky učení). Pro vyhodnocení bylo použito průměrových vrstev z jednotlivých přehybů učení. Rozbor těchto výsledků shrnují kapitoly 6.2 a 6.3.

6.1.1 Data k trénování

Data pro klasifikaci byla obdržena při optické inspekci waferů. Jednotlivé obrázky jsou komprimovány formátem JPEG. Pro použití v inferenci je potřeba převést

[1] K-násobná validace – dataset je rozdělen na K ucelených bloků (přehybů), kde trénování probíhá ve formě 1 blok pro validaci a K-1 pro trénování

obrazové body na číselné zastoupení. Toho je docíleno prostřednictvím kroku *načtení dat*. Načtená data mají rozměr 3D matice (x, y, z), kde souřadnice x a y určují horizontální a vertikální složky vstupního obrázku a souřadnice z určuje barevné zastoupení RGB (v případě ČB je souřadnice z = 1). Načtením dat dochází k převedení těchto dat do numpy[2] polí. Hodnota každého pixelu souřadnicové soustavy je reprezentována osmibitovým číslem (0 – 255).
Pro rychlejší konvergenci a nižší hodnoty ztrátové funkce je doporučeno jednotlivé hodnoty normovat (0 – 1) [1]. V nativní implementaci je tento krok proveden před zahájením samotného trénování (ve skriptu *train_and_validate_model.py*).
Před touto normalizací je rovněž provedena jejich augmentace, která zabraňuje přeučení modelu před jeho trénováním [1]. Použité augmentační vrstvy jsou popsané pro jednotlivé datasety (viz 6.2 a 6.3). Evaluaci augmentací se zabývá kapitola 7.2).
Před zahájením samotného trénování je nutné eliminovat taková data, která by mohla snižovat výkonnost natrénované sítě. Možné způsoby selekce nevhodných dat jsou naznačeny v kapitole 4.2. Součástí této práce je implementace algoritmu *labelfix*, jež dokáže vytřídit špatně označené třídy. Bylo však zjištěno, že tento krok není nutný, jelikož jeho vynechání nemělo významný vliv na zlepšení metrik sítě. Pro neznámé datasety je však ponechána možnost jeho aktivace.



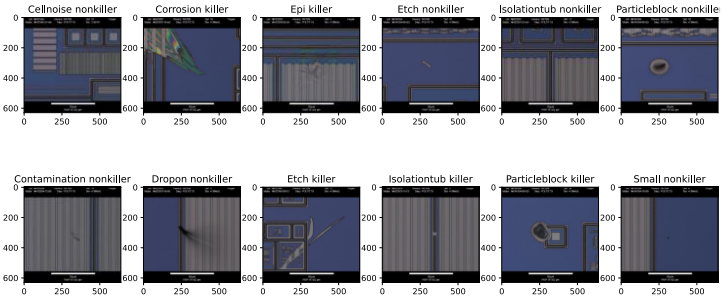
obrázek 6.4) Křivky učení – trénovací data I. iterace

6.2 Vyhodnocení dat (FAB Belgie)

Data pro trénování byla získána z belgického závodu na výrobu polovodičů. Přičemž byly provedeny augmentace obrazových dat. V nativní implementaci byly

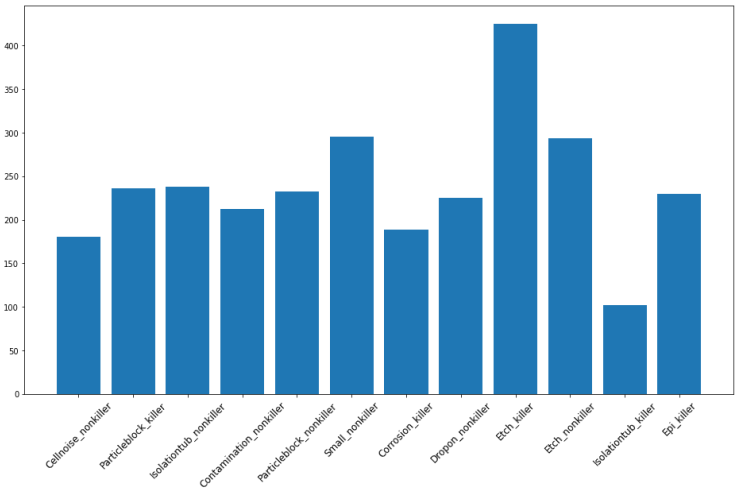
[2] numpy – knihovna programovacího jazyku python, která umožňuje práci s maticemi

použity následující modifikace: rotace, zrcadlení (horizontální, vertikální), náhodný střih. Distribuci kategorií defektů ukazuje obrázek 6.6. Vyváženost tříd lze považovat za uspokojující, proto nebyla potřeba implementace algoritmů popsaných v kapitole 4.1. Celkové množství dat pro trénování je cca 2500 vzorků (200 vzorků/třída).
Sledované třídy se mohou lišit v závislosti potřeb procesních inženýrů v každém závodě a v závislosti na výrobní technologii. Defekty jsou blíže určeny katalogem defektů.



obrázek 6.5) Defekty sledované klasifikátorem

Jako součást algoritmů zabraňujících přeučení je implementována regularizace vah a přidání *dropout* vrstev (viz 3.1.2). Dále bylo realizováno předčasné ukončení, což je mechanismus ukončující učení, jestliže dochází ke zvyšování testovací chyby. Její fluktuace však není předvídatelná, proto se nastavuje trpělivost systému (tzn. počet iterací, kdy nedochází ke zlepšení) [4]. Zde byla nastavena trpělivost systému na hodnotu 10.



obrázek 6.6) Distribuce tříd defektů

Z křivek učení (viz obrázek 6.4) je patrný monotónní růst přesnosti na trénovaných datech. Data samotná byla rozdělena na 3 podmnožiny. Kromě trénovací a testovací je zde k dispozici ještě validační množina (*cross-validation*, CV). Její funkce

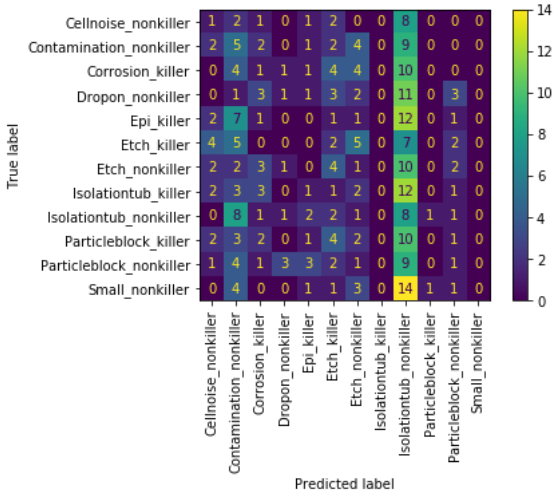
představuje sekundární validaci natrénovaného modelu – ověřuje generalizační sílu natrénované architektury.

6.2.1 Výsledky trénování z I. iterace

Pro trénování byla použita síť ResNet18 s náhodně inicializovanými váhami. Implementace této sítě je rovněž součástí přílohy této práce (viz cesta: *libs/train/Resnet.py*). Průběh trénování vystihuje obrázek 6.4. Z těchto křivek je patrné, že přesnost modelu na validačních datech vykazuje vysokou fluktuaci. Tento jev může být způsoben několika aspekty [1] [8]:

- Malá velikost validačního setu: ve validačním setu je pravděpodobná variace vzorků použitá pro validaci
- Šum v datech: může být zdrojem problémů nízké performance modelu. V tomto případě je potřeba provést dodatečné předzpracování dat před trénováním
- Špatně nastavené parametry neuronové sítě: pravděpodobně chybně zvolený krok učení (zde byl použit: 0,001) nebo velikost datové dávky (zde 64)
- Algoritmus předčasného ukončení: pro trénování byl tento algoritmus použit. Síť totiž v průběhu trénování usiluje o dosažení ideálního záchytného bodu.

Z výše uvedených možností je nejvíce pravděpodobná možnost nízké velikosti validačního setu. Jelikož je zde implementovaný algoritmus K-násobné validace s počtem rozdělení 5, lze spočítat průměrnou velikost validačních dat pro jedno přeložení na cca 170 vzorků (na třídu). Další zkoumanou možností je rovněž šum v datech, avšak z charakteru obrázků (viz obrázek 6.5) této teorii nic nenasvědčuje.

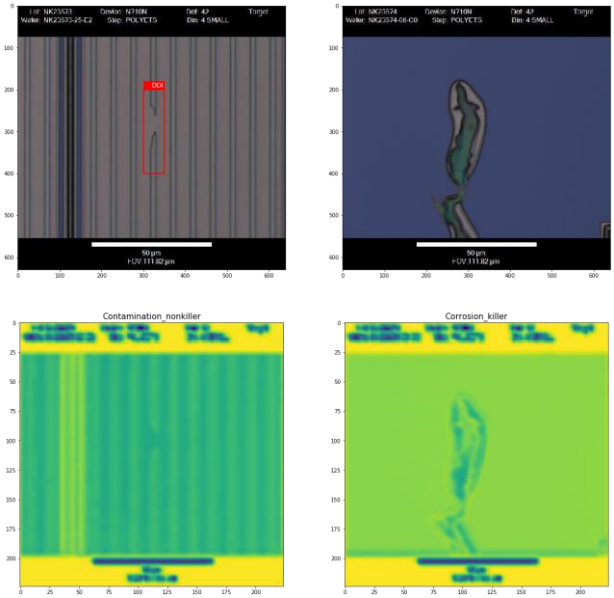


obrázek 6.7) Matice záměn I. iterace

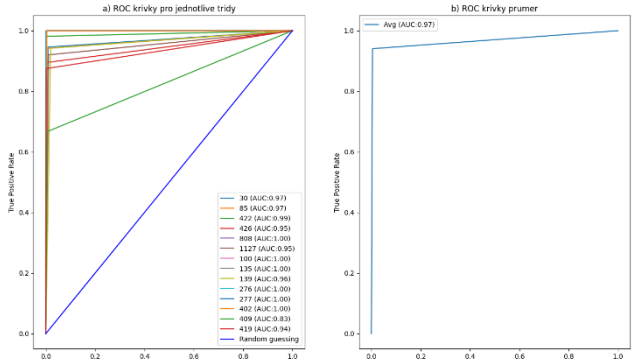
Lze předpokládat, že takto natrénovaný model nemá generalizační sílu, proto je nasazení takového modelu do produkce nepřípustné. Z matice záměn (viz obrázek 6.7) je patrné, že prediktor predikuje výhradně třídu „*Isolationtub nonkiller*“. Zde se rovněž nabízí několik vysvětlení [8]:

- Špatná extrakce příznaků: obrázky pro trénování jsou špatně transformovány (špatná volba augmentací může změnit výslednou interpretaci defektu)
- Malá velikost trénovacího datasetu: pokud je dataset příliš malý, je pravděpodobné, že klasifikátor špatně monitoruje příznaky jednotlivých tříd
- Špatná volba parametrů sítě

Jednou z uvažovaných možností je špatná extrakce příznaků. Extrahované příznaky z poslední konvoluční vrstvy ukazuje obrázek 6.8. Zde jsou patrné výstupy z poslední konvoluční vrstvy sítě ResNet18. Pro poruchu „*Corrosion killer*“ je zřejmé, že model je schopen dostatečně rozlišit příznaky dané kategorie defektů. Naopak pro třídu „*Contamination nonkiller*“ je zřejmé, že příznaky nejsou rozlišeny (viz obrázek 6.8 – červeně). Uvedené příklady náhodně zvolených defektů ilustrují slabou generalizační sílu modelu. Naměřené F1 skóre pro I. iteraci je 6,1 %.

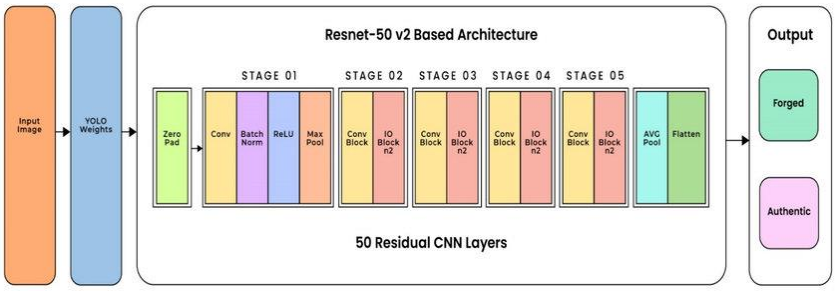


obrázek 6.8) Výstupy z poslední konvoluční vrstvy na náhodně vybraných datech



obrázek 6.9) ROC křivka - I. iterace (II. krok)

Porovnáním poměrů predikcí lze obdržet ROC křivku pro jednotlivé třídy (viz obrázek 6.9). Pomocí těchto výsledků lze interpretovat, že natrénovaná architektura kopíruje chování náhodného klasifikátoru [8]. Porovnání výsledků s II. iterací ukazuje tabulka 6.2.

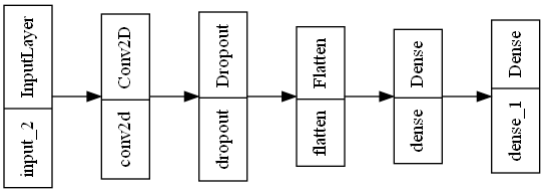


obrázek 6.10) Architektura sítě ResNet50v2 [13]

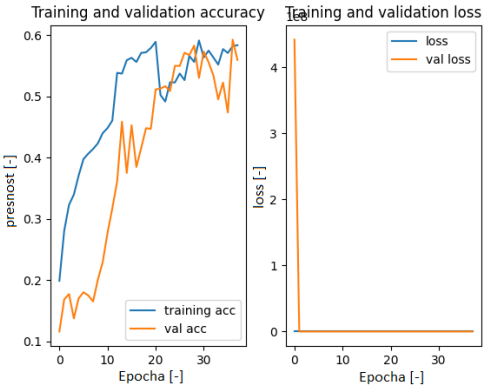
6.2.2 Výsledky trénování II. iterace

Pro II. iteraci byla použita předtrénovaná síť. Použití předtrénovaných sítí (*transfer learning,* TL) spočívá v použití natrénovaných sítí na jiných datových množinách. Výhodou tohoto přístupu je především jednodušší extrakce příznaků (okraje, kontrast, atp.) dané datové množiny, jelikož parametry předtrénované sítě jsou již určeny. Další výhodu představuje čas potřebný pro naučení dané sítě. Zde stojí za zmínění, že čas potřebný pro natrénování takové sítě byl poloviční ve srovnání s I. iterací [8]. Pro trénování sítě byla použita síť ResNet50v2 (viz obrázek 6.10), která byla natrénována pro datovou množinu „*imagenet*“. Jedná se o síť, jejíž výstupem je 1 000 různých tříd (např. kohouti, ryby, atd.). Výstup této sítě je nadále zpracováván hustě propojenou sítí, jejíž schéma ilustruje obrázek 6.11 . Kódová realizace této předtrénované sítě je přímo součástí trénovacího skriptu (*train_and_validate_model.py*).

Výstup této sítě byl přizpůsoben pro účely klasifikování defektů na 12 tříd. Nativní implementace trénování spočívá v uzamčení konkrétních vrstev pro trénování (do 120 vrstvy). Tyto parametry sítě nebudou v první části sítě upravovány algoritmem zpětného šíření. V další fázi budou vyhodnocovány výsledky z tohoto učení. Jestliže budou uspokojivé, budou všechny vrstvy modelu nastavené jako trénovací a s jemnějším krokem učení přeučeny.

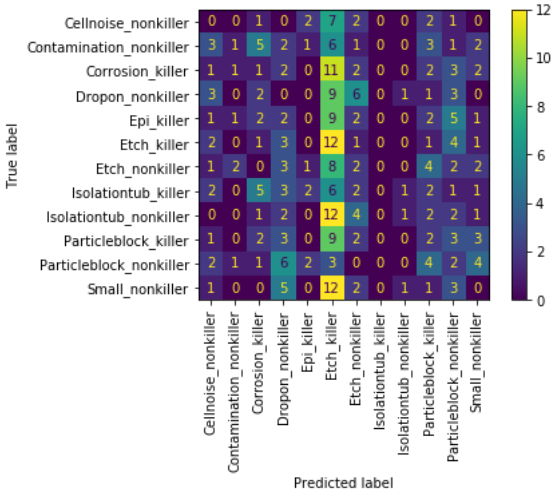


obrázek 6.11) Sít hustě propojených vrstev a dropout vrstev připojených za základním blokem ResNet50

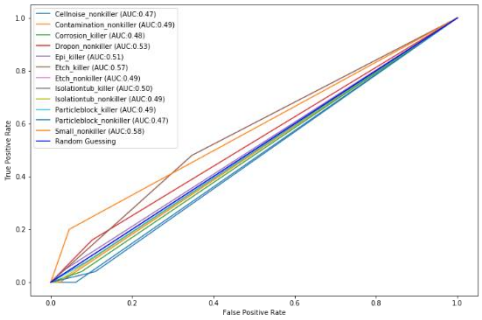


obrázek 6.12) Křivky učení – trénovací data II. iterace

Zatímco průběhy ztrátové funkce jsou pro obě iterace podobné se strmě klesajícím charakterem, pro trénovací přesnost lze sledovat u II. iterace vysoké fluktuace jednak pro trénovací množinu a jednak pro validační množinu (viz obrázek 6.12). Tento fenomén lze vysvětlit přeučením datasetu. V tomto případě se může jednat o situaci, že architektura sítě je příliš komplexní a své výstupy tak přizpůsobuje trénovacím datům [1].

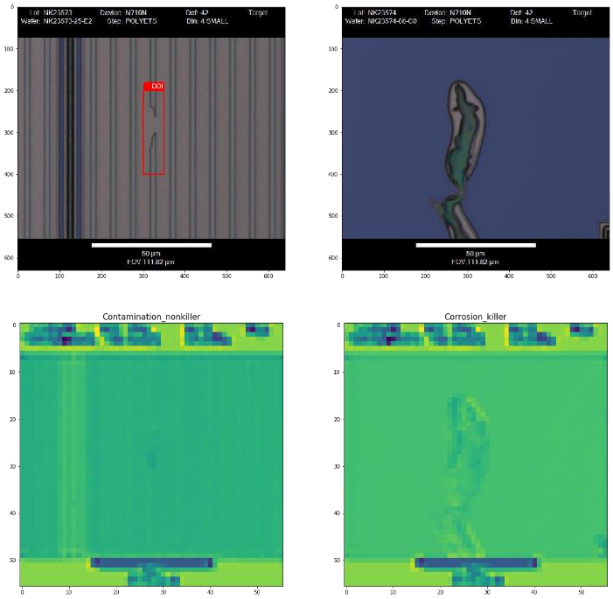


obrázek 6.13) Matice záměn II. iterace



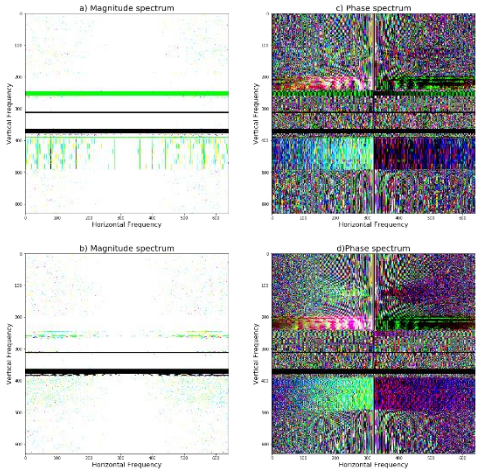
obrázek 6.14) ROC křivka – II. iterace

Tezi o možném přeučení datasetu lze rovněž podpořit maticí záměn (viz obrázek 6.13), která jasně implikuje slabou generalizační sílu modelu. F1 skóre této architektury dosahuje nižšího výsledku než v I. iteraci cca 5,6 %. Z výstupu konvoluční vrstvy lze opět dedukovat špatnou extrakci parametrů pro některé třídy (viz obrázek 6.15).



obrázek 6.15) Výstupy z 19. konvoluční vrstvy II s náhodným výběrem dat

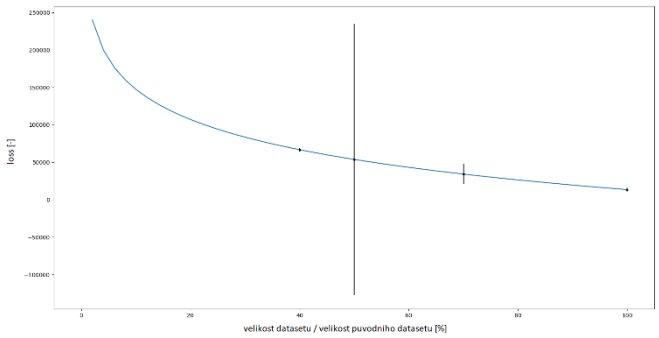
Z výstupů konvoluční vrstvy lze však pozorovat výhody použití předučených sítí. Síť totiž byla schopná extrakce některých příznaků na datech, které dosud nebyla modelem monitorována. Poměr korektních a falešně pozitivních predikcí zobrazuje obrázek 6.14. Podobně jako v případě I. iterace je zde patrné náhodné chování klasifikátoru.



obrázek 6.16) FFT náhodně vybraného vzorku: a) magnituda obrázku před filtrací, b) magnituda obrázku po filtraci, c) a d) fáze obrázku před a po filtraci

tabulka 6.2) Porovnání metrik jednotlivých iterací

metrika		I. iterace	II. iterace
AUC	[-]	0,498	0,498
F1	[%]	2,955	5,988
Přesnost	[%]	5,862	7,933



obrázek 6.17) Metoda postupného vzorkování datasetu

6.2.3 Shrnutí

Sledovaný dataset byl postupně natrénován pomocí 2 odlišných architektur. Srovnáním metrik jednotlivých iterací (viz tabulka 4.1) lze dedukovat, že síť ResNet50v2 dosahuje lepších výsledků v metrikách přesnosti a F1 skóre, avšak výsledek AUC skóre je srovnatelný se sítí ResNet18. Vzhledem ke skutečnosti, že obě natrénované sítě vykazují charakter náhodného klasifikátoru, je jejich použití v jakékoliv produkční aplikaci nemyslitelné.

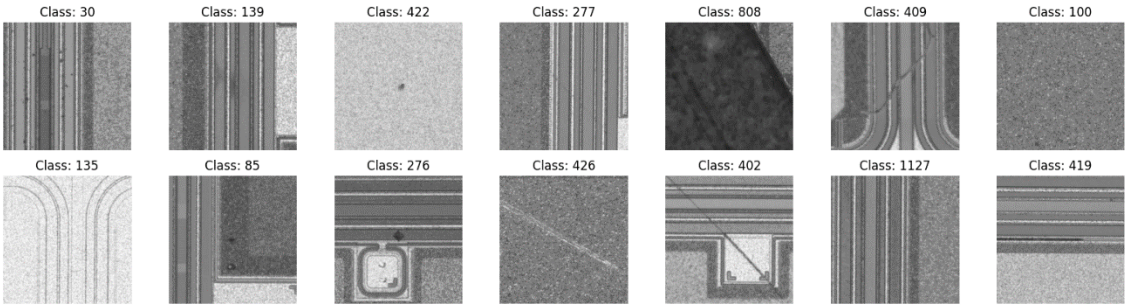
Kvůli nízké performanci a přeučení sítě ResNet50v2, které patrně souvisí s nízkou velikostí vstupních dat, nebyla tato architektura natrénována s jemnějším krokem učení, jak je popsáno v 6.2.2.

Jako nejpravděpodobnější příčina nízké performance u obou trénovacích iterací se jeví nízké množství vstupních dat, potřebných pro správné natrénování sítě a jeho vysokou generalizační sílu [1]. Tato teorie byla potvrzená metodou postupného vzorkování datasetu, jejíž výsledky prezentuje obrázek 6.17. Z obrázku je patrný sestupný trend ztrátové funkce v průběhu trénování, který se asymptoticky blíží 0. Lze tak očekávat, že rozšířením datasetu by mohlo dojít k poklesu ztrátové funkce v průběhu trénování. Možnost špatného nastavení parametrů sítě byla označena jako nepravděpodobná, jelikož model byl přetrénován s adaptivním krokem učení (v

rozsahu $1 \cdot 10^{-3} \div 1 \cdot 10^{-5}$) a s dvojnásobnou velikostí dávky (velikost dávky byla 128), přičemž výsledky trénování byly srovnatelné. Další pravděpodobnou chybou mohou být neodstraněná popisová pole jednotlivých obrázků. Této hypotéze by mohla nasvědčovat skutečnost vysokého množství predikcí pro jednu třídu (viz obrázek 6.7 a obrázek 6.13) a pro třídy s méně patrnými příznaky (viz obrázek 6.15 – červeně). Pro další iterace jsou tyto redundantní informace odstraněny.
Další uvažovanou možností nízké performance je šum obsažený v datech. Tento šum může být patrný z Fourierovy transformace (obrázek 6.16a). Tmavé regiony reprezentují popisové pole obrázků. Světle zelený region rovnoměrně zastoupený v horizontálních frekvenčních složkách je rovněž obsažen pouze v popisovém poli. Pro filtraci šumu byl implementován mechanismus průměrující obrazové body v okolním regionu. Tuto filtraci ilustruje obrázek 6.16b, zde je patrné rozptýlení vertikálních složek spektra.
V poslední řadě může problém spočívat ve zpracovávaném datasetu. Konkrétní modifikace a úpravy poskytnutých dat však nejsou známy, a proto tuto hypotézu nelze potvrdit ani vyvrátit.

6.3 Vyhodnocení dat (FAB EFK)

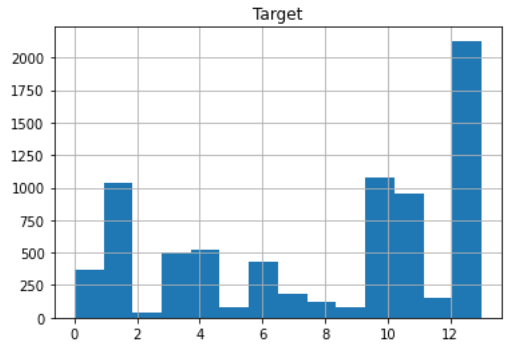
Data poskytnutá závodem na výrobu polovodičů v EFK obsahují přes 7 000 obrazových souborů defektů. Data byla předem rozdělena do 3 kategorií: trénovací, validační, testovací. Validační a testovací podmnožině náleží menší množství dat. Sledovanou skupinu defektů zde tvoří 14 tříd s kódovými značeními (viz. obrázek 6.18).



obrázek 6.18) Třídy defektů – FAB EFK

Distribuce jednotlivých tříd, jak ukazuje obrázek 6.19, je problematické pro úlohu strojového učení. Překážka nastává v průběhu trénování, kdy se klasifikátor naučí monitorovat příznaky majoritní třídy. To může vyústit v chybné predikce pouze této třídy [8]. Jelikož rozdíly mezi nejčetnější třídou a nejméně četnou je padesátinásobný, je potřeba tyto nerovnováhy korigovat. V nativní implementaci je problém řešen přidáním váhových parametrů jednotlivých tříd do trénovací funkce.

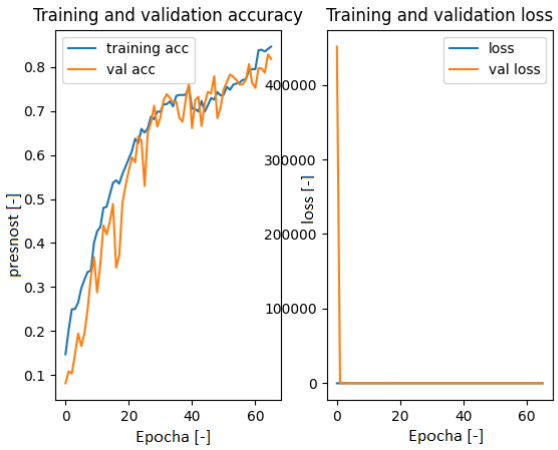
Váhování tříd je patrné v kódu (*train_and_validate_model.py*) přidáním parametru *class weight* do trénovací funkce.



obrázek 6.19) Distribuce tříd defektů

6.3.1 Výsledky trénování z I. iterace

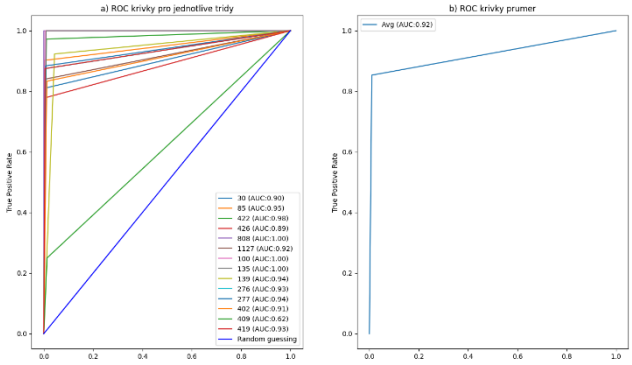
Pro trénování sítě na datech z EFK byla použita síť popsaná v kapitole 6.2.2. Zde byl opět použit podobný přístup pro trénování modelu na dostupných datech. Model je trénovaný na síti ResNet50v2 ve dvou krocích, přičemž rozdíly mezi kroky jsou pouze v množství vrstev, jež mají povolené trénování, a velikost kroku učení. Krok učení byl ve druhém kroku 10× zmenšen a použit pro natrénování celé sítě. Výsledná performance sítě je následně porovnána se současným produkčním řešením, které využívá platformu *AutoML* od společnosti Google.



obrázek 6.20) Křivky učení – I. iterace (1. krok)

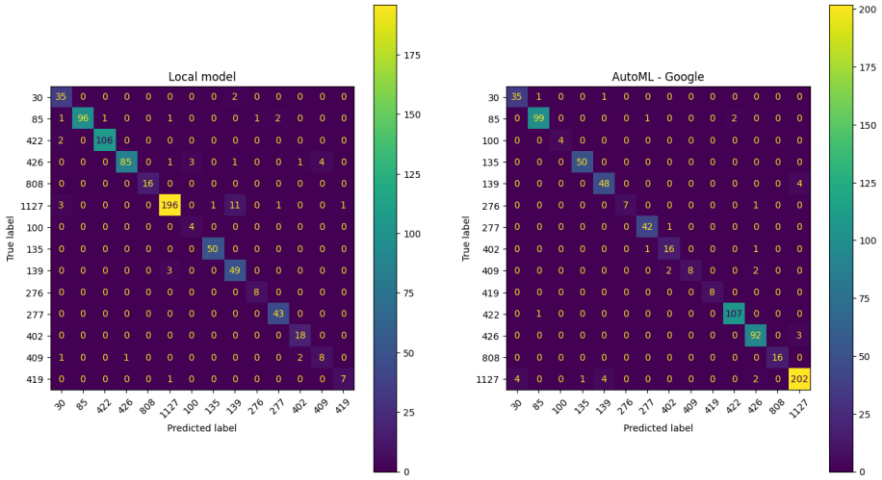
Výsledky trénování v jednotlivých epochách ilustruje obrázek 6.20. Zde je patrné monotónní stoupání trénovací přesnosti a strmý pád ztrátové funkce. Po 20 epoše je rovněž patrné, že validační přesnost začíná v některých krocích přerůstat trénovací. Fluktuace pro prvních 50 epoch je rovněž značná. Tyto aspekty by mohly naznačovat možnost přeučení modelu [1]. Avšak pro poslední epochy je patrná stabilizace validační přesnosti. Pro ověření generalizační síly modelu byla implementována ROC křivka (viz obrázek 6.21). Z této křivky lze dedukovat

vysokou úspěšnost modelu na testovacích datech. Průměrná hodnota AUC skóre atakuje hodnotu 0,92. Pouze pro třídu s označením 409 lze pozorovat značné přiblížení skóre k náhodnému klasifikátoru (viz obrázek 6.21). Metriky z prvního kroku jsou shrnuje tabulka 6.3.



obrázek 6.21) ROC křivka – I. iterace (1. krok)

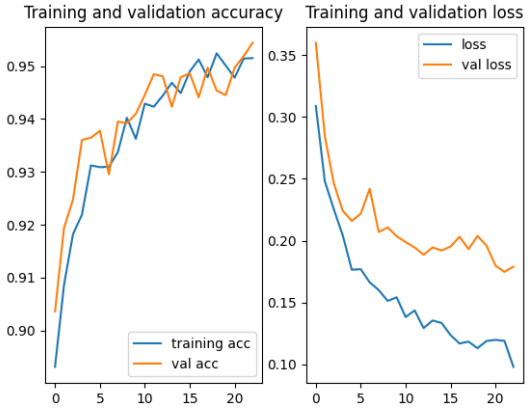
Z tabulky je rovněž patrné, že v prvním kroku se podařilo model natrénovat s horšími metrikami, než jsou dostupné v produkčním modelu. Z AUC skóre však vyplívá vysoká rozlišovací schopnost mezi jednotlivými třídami. Pro metriku F1 lze sledovat vysoký rozdíl performancí mezi produkčním a implementovaným řešením.



obrázek 6.22) Srovnání matic záměn produkčního modelu (vpravo) a implementovaného řešní (vlevo)

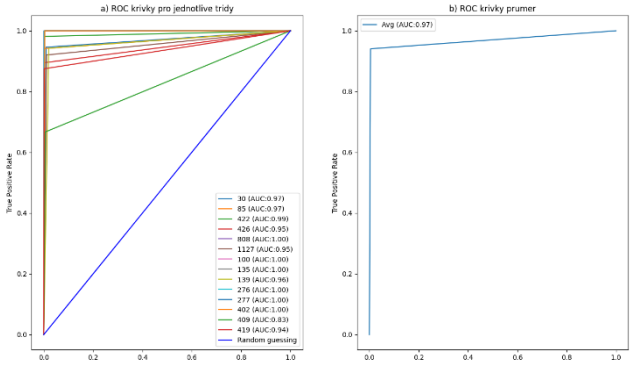
Tyto rozdíly minimalizuje druhý krok s jemnějším krokem učení pro všechny vrstvy modelu. Srovnání matic záměn s produkčním modelem ilustruje obrázek 6.22. Z obrázku lze dedukovat lepší performanci produkčního řešení pro určité skupiny tříd. Tento pokles může být vysvětlen mírným přeučením modelu, které je patrné z křivek učení (viz obrázek 6.23). Zde je patrná fluktuace validační ztrát, která od určité epochy vykazuje rovněž rozkol od trénovací ztráty. Výsledný efekt přeučení však není na metrikách příliš patrný. Porovnáním metrik produkčního a implementovaného řešení (viz tabulka 6.3) lze dospět k závěru, že realizací

prostřednictvím sítí ResNet50v2 lze dosáhnout vysoké testovací přesnosti na nových datech. Zásadní rozdíl lze sledovat především pro F1 skóre, kde pozorujeme rozdíl 4 %.



obrázek 6.23) Křivky učení – I. iterace (2. krok)

Srovnáním ROC křivek jednotlivých kategorií (viz obrázek 6.24) lze pozorovat značné zlepšení problematické skóre třídy 409 z 1. kroku trénování. Z křivek je rovněž patrný posuv poměrů správně označených tříd (*true positive rate*, TPR) nad hodnotu 0,9.



obrázek 6.24) ROC křivka – I. iterace (2. krok)

6.3.2 Shrnutí

Vyhodnocení trénování na datech z EFK lze interpretovat kladně. Z metrik výsledného modelu je patrná jeho generalizační síla. Pro porovnání s produkčním řešením bylo využito tří metrik: AUC, F1 a Přesnosti (viz tabulka 6.3). Z těch lze pozorovat lepší výkonost právě produkčního řešení. Tuto odchylku lze vysvětlit možným přeučením modelu v 2. kroku I. iterace, jež je patrné z křivek učení (viz obrázek 6.23). Toto přeučení lze vysvětlit možnou komplexitou použité *ResNet* sítě, u které je pravděpodobné přílišné učení příznaků z trénovacích dat. Jako nástroj zmírnění přeučení se nabízí nasbírání vyššího množství dat nebo přidáním dropout vrstev. Kolekce dat je však časově a finančně náročná. Další možností zmírňující

přeučení je zefektivnění způsobu výběru augmentačních vrstev, jelikož v nativní implementaci byly augmentace vybírány náhodně. Způsoby evaluace augmentací jsou podrobně řešeny v kapitole 6.4. Příčinou nižší výkonosti sítě ve srovnání s řešením AutoML může být nižší rozlišení obrázků, které bylo v průběhu trénování použito (300 × 300 pro AutoML a 224 × 224 pro realizované řešení). Snížením rozlišení obrázků však bylo docíleno redukce celkové velikosti na polovinu. To může být výhodou v následné online inferenci, kdy dochází k redukci množství datového toku, což s sebou nese nároky na časy jednotlivých predikcí.

tabulka 6.3) Srovnání produkčního a natrénovaného modelu

metrika		I. iterace		Produkční řešení
		1. krok	2. krok	AutoML
AUC	[-]	0,921	0,968	0,968
F1	[%]	78,98	90,48	94,73
Přesnost	[%]	82,51	94,13	95,82

6.4 Dílčí závěr

Pro evaluaci této kapitoly byly použity algoritmy sítí ResNet18 a ResNet50v2. Při výběru druhé srovnávací sítě byl především kladen důraz na její nízkou komplexitu a složitost podobně jako je v síti ResNet18 a zároveň na dostupnost předtrénovaných sítí ve frameworku tensorflow a možnosti aplikace metody *transfer learning*. Při implementaci algoritmu ResNet18 byly použity odladěné fragmenty kódu dostupné z [14].

7. EVALUACE HYPERPARAMETRŮ

Výčet některých hyper-parametrů je popsán v kapitole 6.1. Součástí této kapitoly je nalezení takových hyperparametrů, které maximalizují metriky natrénované sítě. Patří mezi ně například množství hustě propojených vrstev na výstupu předtrénované sítě nebo počet vrstev předučené sítě určených k trénování modelu.

Jako další z hyperparametrů lze vnímat datovou augmentaci. Ta je jedním z nástrojů používaných ve strojovém vidění. Jejich podstata je objasněna v 4.3. Je nezpochybnitelné, že jakákoliv transformace dat má přímý důsledek na performanci sítě. Chybnou volbou augmentačních vrstev tak dochází k její degradaci. Tato kapitola si klade za cíl nalezení takových augmentací, které maximalizují performanci sítě. Výsledné metriky budou srovnány s metrikami získanými v 6.2.

7.1 Rozbor vrstev sítě

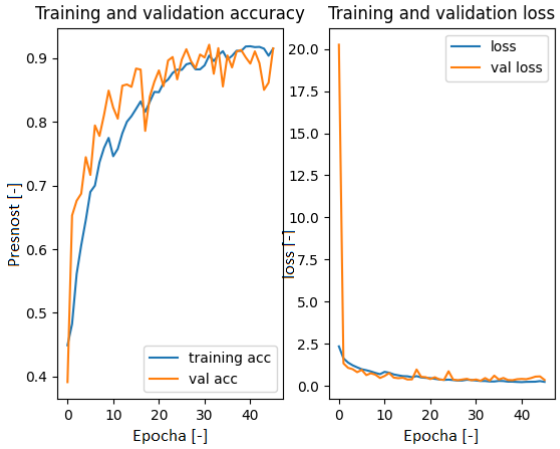
V kapitole 6.3 byla použita předtrénovaná ResNet50v2 síť. Její výstup byl však modifikován přidáním skupiny hustě propojených vrstev a přizpůsobení poslední vrstvy na požadovaný počet tříd. Volba přídavných vrstev však byla zvolena náhodně [8]. Možné zlepšení performance sítě se nabízí laděním hyperparametrů těchto vrstev. Mezi rozmítané parametry je uvažován: počet přídavných vrstev a jejich typ, počet jednotek v hustě propojených vrstvách. Volbu rozmítání popisuje tabulka 7.1.

tabulka 7.1) parametry rozmítání

Parametr rozmítání		hodnota min.	hodnota max.	krok	Optim.
přídavné vrstvy	For smyčka	1	3	1	1
FCN vrstvy	Jednotky	32	512	3	161
Dropout vrstva	Koeficient	0.1	0.5	0.1	0,5
Conv2D vrstva	Filtry	32	256	32	-
	Jádro	3	7	2	-
MaxPooling	Velikost redukce	2	4	1	-
Neaktivních vrstev (netrénovatelých)	-	0	20	1	12

Z tabulky je rovněž patrný optimální počet vrstev pro trénování včetně množství jednotek ve vrstvách. Pro hledání hyperparametrů byl použit algoritmus náhodného hledání. Jedná se o metodu, kde je pro nalezení ideálních parametrů sítě použito

náhodného výběru, kdy dochází k porovnání výsledků mezi jednotlivými iteracemi [8]. Výhodou použití tohoto přístupu je jeho jednoduchá implementace a vysoká variabilita při použití na různé typy neuronových sítí [8].
Síť s optimálními hyperparametry je následně použita pro natrénování na celém datasetu. Podmínky trénování jsou uvedeny v kapitole 6.3. Pro účely této evaluace byl vytvořený skript, který proces ladění realizuje (*evaluate_augmentations.py*).



obrázek 7.1) Křivky učení – ladění hyperparametrů

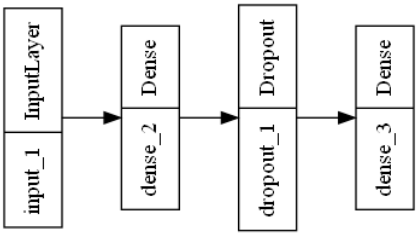
Z křivek učení (viz obrázek 7.1) je patrná rychlá konvergence validační přesnosti k 90 %. Přičemž v rámci jednotek procent je zde patrná její fluktuace. Porovnání s modelem, uvedeným v kapitole 6.3, uvádí tabulka 7.2. Zde je zřejmé nepatrné zlepšení metrik AUC a F1. Odchylka performancí těchto dvou modelů je však minimální, a nelze tak výsledky odladěného modelu interpretovat jako zlepšení. Vrstvy přidané optimalizací ilustruje obrázek 7.2.

tabulka 7.2) Vyhodnocení laděného modelu a stávajícího řešení

metrika		řešení 1.	ladění HP
AUC	[-]	0,968	0,969
F1	[%]	90,48	91,98
Přesnost	[%]	94,13	93,99

Vzhledem k výpočetní náročnosti ladícího procesu bylo potřeba regulovat některé z hyperparametrů sítě. Bylo tak přistoupeno ke snížení velikosti dávky dat o čtyřnásobek (32 – původně 128). Tato redukce mohla způsobit pomalou konvergenci modelu v průběhu ladění, což mohlo zapříčinit podobnou performanci sítě jako u již natrénované architektury. Součástí této implementace bylo rovněž vynechání K-násobné validace, což rovněž mohlo přispět k podobným metrikám

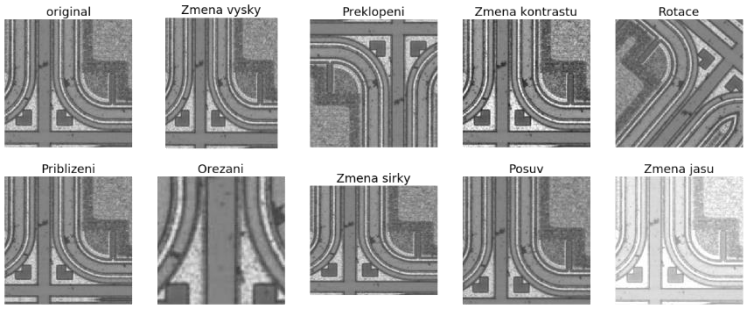
srovnávaných modelů. Jako problematický lze rovněž vnímat algoritmus náhodného hledání hyperparametrů. Jelikož parametry sítě jsou vybírány náhodně, nelze jednoznačně potvrdit, že byla zvolena jejich nejlepší kombinace. Této skutečnosti lze předejít provedením několika iterací ladění [8]. Tento přístup byl aplikován součástí této práce, přičemž je prezentován model s nejlepší performancí.



obrázek 7.2) Přidané vrstvy – HP ladění

7.2 Rozbor augmentačních vrstev

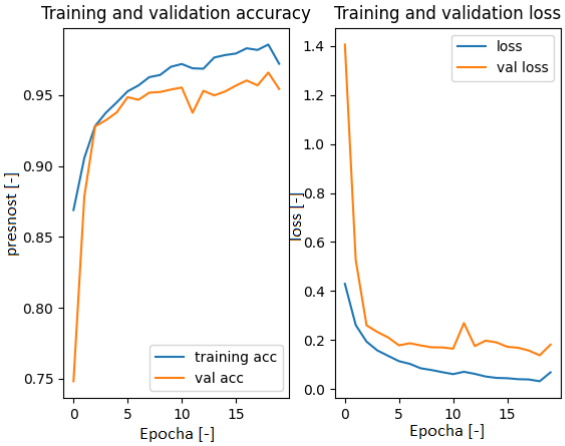
Augmentace lze ze své podstaty rozdělit na 2 kategorie: geometrické a hloubkové. Geometrickými augmentacemi se rozumí taková úprava obrazu, jejíž výsledkem je jeho prostorová úprava. Naopak hloubkové augmentace mění jednotlivé kanály obrazového vstupu. Příklady těchto transformací ilustruje obrázek 7.3. Tyto vrstvy jsou podporovány frameworkem *keras*.



obrázek 7.3) Augmentační vrstvy ve frameworku keras

Z vrstev je patrné, že separátním použitím nedochází ke ztrátě informace z obrazu, jelikož defekty jsou na všech snímcích patrné. Problematické může být mnohonásobné použití různých vrstev. Další nástrahou je samotné nastavení vrstev. V případě velkého rozptylu hodnot již může být obraz špatně čitelný (např. pro změnu jasu nebo vysoké hodnoty ořezu). Pro ladění budou požity pouze takové augmentační vrstvy, které neprovádějí ořez, jelikož by mohlo dojít k ořezání informace o defektu. Vyhodnocení augmentací je rovněž prováděno skriptem *evaluate_augmentations.py*

Z výsledné evaluace je patrné, že nejlepší performance dosahuje model pouze pro kombinace augmentací horizontálního a vertikálního překlopení. Pro tento experiment bylo rovněž využito algoritmu náhodného hledání hyperparametrů, a proto byl proveden ve třech iteracích. Tím je zvýšena jistota správného výběru příznaků [8].
Z vybraných vrstev lze usoudit, že ladicí algoritmus vybral nejméně invazivní geometrické transformace. Horizontální a vertikální překlopení se totiž objevilo v každé z ladicích iteraci. Informace o vybraných augmentacích byly následně použity v trénovacím algoritmu, popsaném v kapitole 6.3.



obrázek 7.4) Křivky učení – evaluace augmentací

Výstup trénovací sekvence ukazuje obrázek 7.4. Zde je patrné, že přesnost validačních ztrát konverguje k 95 %. Na křivkách není patrná žádná fluktuace přesnosti (ztrát) v průběhu učení, jež byla patrná při všech předchozích trénovacích iteracích. Z křivek lze usoudit, že model nevykazuje žádné nástrahy učení. Vyhodnocení natrénovaného modelu na testovacích datech vykazuje vysoké hodnoty všech sledovaných metrik. Porovnáním s původním řešením (viz Tabulka 7.3) lze pozorovat nárůst v metrikách přesnosti a F1 skóre. Jedná se však pouze o nárůsty v jednotkách procent. Přínosem této evaluace je však zjištění, že správnou volbou augmentací dochází eliminaci fluktuací validační přesnosti (ztrát) v průběhu trénování.

Tabulka 7.3) Porovnání metrik původního řešení a řešení po ladění

metrika		řešení 1.	ladění HP
AUC	[-]	0,968	0,963
F1	[%]	90,48	92,20
Přesnost	[%]	94,13	94,91

7.3 Shrnutí

Z výsledků uvedených v podkapitolách 7.1 a 7.2 je zřejmé, že laděním jednotlivých hyperparametrů sítě nedochází k významnému nárůstu sledovaných metrik u modelu. Základní nastavení sítě *ResNet* tak lze považovat za dostačující v řešení problematiky klasifikace defektů. Toto zjištění může být interpretováno kladně, jelikož při potřebě aplikace klasifikátoru pro podobnou úlohu predikce není potřeba tyto ladicí mechanismy pokaždé spouštět, což snižuje potřebu režijních nákladů na natrénování sítě.

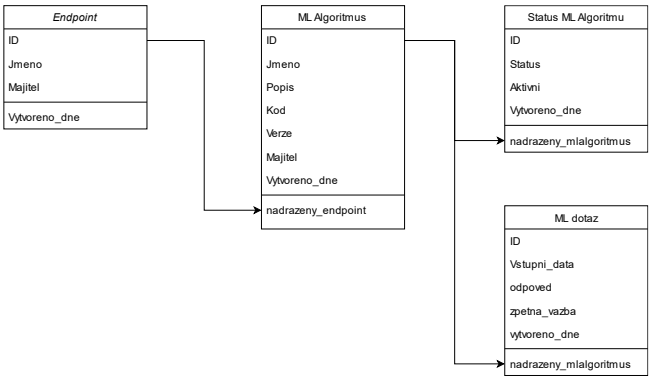
Při vyhodnocování augmentačních vrstev bylo rovněž zjištěno, že při jejich správné volbě dochází k minimalizaci fluktuací v průběhu trénovací sekvence. Z tohoto důvodu je nadále doporučeno používat při trénování výhradně tyto augmentace.

8. INFERENCE

Inference je posledním krokem životního cyklu v průběhu aplikace strojového učení. Princip spočívá v poskytnutí zpětné vazby uživateli s využitím natrénovaného modelu. Tato kapitola si klade za cíl vytvoření inferenčního zařízení využívajícího REST API rozhraní. Data pro inferenční zařízení jsou získávána v průběhu optické kontroly waferů. Jednotlivé popisy sledovaných defektů jsou uvedeny v kapitole 5.1.3. Komunikace mezi uživatelem a zařízením je realizována pomocí http protokolu. Nároky tohoto rozhraní jsou především kladeny na rychlost odezvy. Ta může být v případě inference obrazových dat značně dlouhá, proto je součástí této kapitoly rovněž hledání způsobu algoritmizace, která by byla schopna data před jejich odesláním redukovat s minimálním dopadem na správnost predikcí.
Dalším z požadavků na zařízení je jeho přenositelnost mezi jednotlivými závody. Z tohoto hlediska lze poté rozdělit inferenci do dvou skupin:

- online: inference probíhá na vzdáleném serveru
- off-line: inference probíhá lokálně v rámci zařízení

Z hlediska přenositelnosti se jako nejvíce výhodná nabízí online inference. Výhody použití jsou zřejmé především v možnostech zpětné vazby, sběru dat a aplikace předučených sítí (*transfer learning*, TL). Problémem online prostředí jsou především náklady spojené s datovým tokem, který vyžaduje vysokou rychlost přenosu. Ta je však často limitovaná, což omezuje rychlost komunikace a snižuje výrobní výtěžnost. Jednotlivá rozhraní jsou v této kapitole srovnána s off-line inferencí, která neřeší problematiku transportu dat.



obrázek 8.1) Struktura databáze

8.1 Struktura databáze

Schéma relační databáze ilustruje obrázek 8.1. Ze schématu je patrné, že nejvýše v hierarchii je umístěn *endpoint*. Ten lze interpretovat jako bránu mezi uživatelem a inferenčním zařízením. Uvažovaný scénář předpokládá alespoň jeden *endpoint* pro každý výrobní závod. Granularita koncového rozhraní není blíže specifikována.

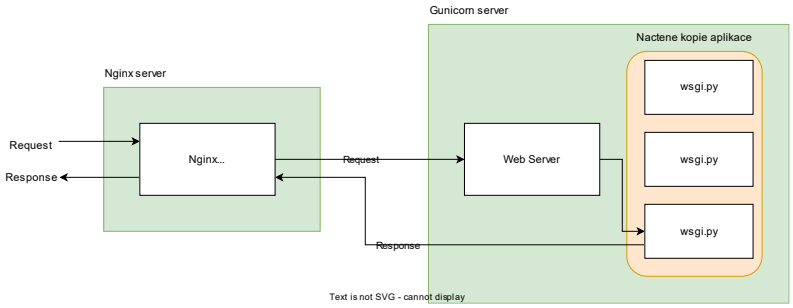
Očekává se rozlišení koncového API rozhraní jednak na urovni výrobního závodu (FABu) a jednak v závislosti na úkolu predikce. Ve vztahu 1:N k *endpointu* je implementován ML algoritmus. Ten představuje konkrétní kódovou implementaci s natrénovanými váhami sítě. Pro váhy sítě není uvažována separátní tabulka v databázi. S rostoucí velikostí zápisů v databázi lze však zařadit tabulku s váhami ve vztahu 1:N k tabulce s ML algoritmem. V neposlední řadě jsou ukládána data s ML dotazy pro konkrétní algoritmus. Přičemž dotazu lze zadat parametry zpětné vazby. Ty mohou být využity pro testovací účely k porovnávání metrik mezi jednotlivými modely. Kódová implementace databáze je k dispozici v souboru `backend/server/apps/endpoints/models.py`

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "endpoints": "http://127.0.0.1:8000/api/v1/endpoints",
    "mlalgorithms": "http://127.0.0.1:8000/api/v1/mlalgorithms",
    "mlalgorithmstatuses": "http://127.0.0.1:8000/api/v1/mlalgorithmstatuses",
    "mlrequests": "http://127.0.0.1:8000/api/v1/mlrequests",
    "abtests": "http://127.0.0.1:8000/api/v1/abtests"
}
```

obrázek 8.2) URL adresy serializované databáze

Všechny tabulky relační databáze jsou serializovány a zpřístupněny prostřednictvím REST API rozhraní (viz `backend/server/apps/endpoints/serializers.py`). Toto rozhraní je implementováno prostřednictvím *Django* frameworku realizované prostřednictvím programovaho jazyku Python. Jednotlivé URL ukazuje obrázek 8.2.



obrázek 8.3) Architektura webové aplikace

8.2 Struktura webové aplikace

Jak již bylo zmíněno v 8.1 URL adresy kopírují schéma databáze, přičemž realizace webového rozhraní je provedena prostřednictvím frameworku *Django* a jeho modifikacemi pro REST API rozhraní (*djangorestframework*). Právě realizace rozhraní prostřednictvím programovacího jazyku Python může znamenat komplikace z hlediska přidané latence, způsobené spouštěním Python skriptů. Z tohoto důvodu dochází v produkčním řešení ke striktnímu oddělení aplikace do 2 částí (viz obrázek 8.3). Výhodou této implementace je možnost oddělení výpočetně náročných aplikací od webového rozhraní. Server realizovaný prostřednictvím *nginx*

tak v této implementaci představuje proxy, jelikož pouze přeposílá požadavky uživatele na výpočetní server [15]. Použití *wsgi* serveru pro výpočetní část aplikace rovněž představuje výhodu z hlediska možnosti ukládání chodu aplikace do cache paměti (*nginx* tuto možnost nenabízí). Parametry serveru jsou určeny v adresáři *docker/nginx*

8.2.1 Docker distribuce

Docker je open-source platforma, jež umožňuje kontejnerizaci aplikací. Výhodou je snadná tvorba, spouštění a správa kontejnerů, které představují lehké a samostatné běhové prostředí, která obsahují všechny potřebné proměnné pro běh aplikace, včetně knihoven a konfigurací [16].
Použití této platformy nabízí možnost rychlého a snadného vytváření verzí produkční aplikace, proto byl vytvořen produkční obraz výše popsané architektury (viz 8.1 a 8.2). Databáze je realizována pomocí open-source databázového systému MySQL, jež umožňuje organizaci a správu uložených dat a představuje korporátní standart. Proces kontejnerizace je specifikován prostřednictvím souboru `docker-compose.yml`.

tabulka 8.1) Parametry serveru

Server	Odezva [ms]	R [Mbps]
Onprem	175	100

8.3 Obsluha aplikace

Aplikaci lze z hlediska obsluhy rozdělit na servisní a inferenční část. Servisní část lze interpretovat jako soubor nástrojů umožňující správný běh backendového prostředí webové aplikace a aktuálnost kódu, zatímco inferenční část se zabývá vztahem klient-server a umožňuje jejich vzájemnou komunikaci.

8.3.1 Servisní část

Program je z hlediska programové implementace obsluhován pomocí platformy Docker (viz 8.2.1). Výhodou tohoto přístupu je především přenositelnost kódu mezi jednotlivými koncovými stanicemi. Takto kontejnerizovaný software lze následně pouhým spuštěním kódu nepřetržitě aktualizovat v provozu (*Continuous Delivery*, CD). Spuštění kontejnerizace lze provést prostřednictvím příkazové řádky příkazem `docker-compose build`. Soubor `docker-compose.yml` pak definuje jednotlivé služby realizované prostřednictvím rozhraní docker. Součástí této práce jsou realizovány služby *nginx* a *wsgi* serveru (viz 8.2). Dále jsou zde implementované služby zprostředkovávající databázi *mysql* a uživatelské rozhraní *phpmyadmin*, jež práci s databází usnadňuje. Spuštění kompilovaného programu lze

realizovat prostřednictvím uživatelského rozhraní softwaru *Docker Desktop* nebo pomocí příkazu `docker-compose up`.

8.3.2 Inferenční část

Inferencí se z hlediska této práce rozumí predikce třídy na základě obrazového vstupu. Ta je realizovaná prostřednictvím http protokolu. Základní rozcestník webové aplikace lze získat při zadání URL adresy `http://127.0.0.1/api/v1/`. IP adresa 127.0.0.1 v tomto případě odkazuje na lokální server hostujícího počítače (localhost). V reálné aplikaci se pak bude lišit. Odpovědi na dotazy s obrázky určenými k predikci lze získat dotazem POST na adresu:

```
http://<DNS_adresa_serveru>/api/v1/<site>/<endpoint>/predict?query
```

přičemž:

- `DNS_adresa_serveru`: představuje konkrétní přeložitelnou adresu respektující konvence jednak firemní a jednak správce DNS serveru (nic.cz pro ČR). Např.: mapdowngrade.onsemi.com
- `/api/v1`: odkazuje na konkrétní verzi webové aplikace
- `site`: specifikuje výrobní závod, pro který je sada natrénovaných modelů určena.
- `endpoint`: specifikuje konkrétní natrénovaný model. Modely se totiž v závislosti na použité technologii mohou lišit (jiné budou použity pro substrát SiC a jiné pro Si)
- `query`: je nositelem doplňujících informací o modelu. Zadána může být verze (klíčovým slovem *version*) nebo status (dev, prod, QA, atp.) použitého modelu (klíčovým slovem *status*). Použití tak může vypadat následovně: `…/predict?version=1.0.0&status=prod`

Inferenční část nevyužívá autentizace uživatelů a rovněž nepředpokládá zabezpečenou komunikaci prostřednictvím https protokolu, jelikož její použití je omezeno pouze v rámci firemní VPN sítě. Některá funkcionalita kódu, jež je součástí této práce, může být z tohoto důvodu do značné míry omezena. Pro správnou funkcionalitu spuštěné aplikace se rovněž předpokládá neobsazenost portu 80 (http port) na cílové stanici.

8.4 Datový tok

Datový tok představuje jedno z omezení aplikace strojově učícího se algoritmu do produkčního řešení. Nástraha souvisí především s přidanou latencí při predikci kategorií. Toto zpoždění lze v případě online inference rozdělit na transportní zpoždění t_{tran} a zpoždění, způsobené samotnou predikcí t_{pred}. Výsledné zpoždění t_{Σ} lze pak určit pomocí rovnice(8.1). Transportní čas lze určit jako součet doby

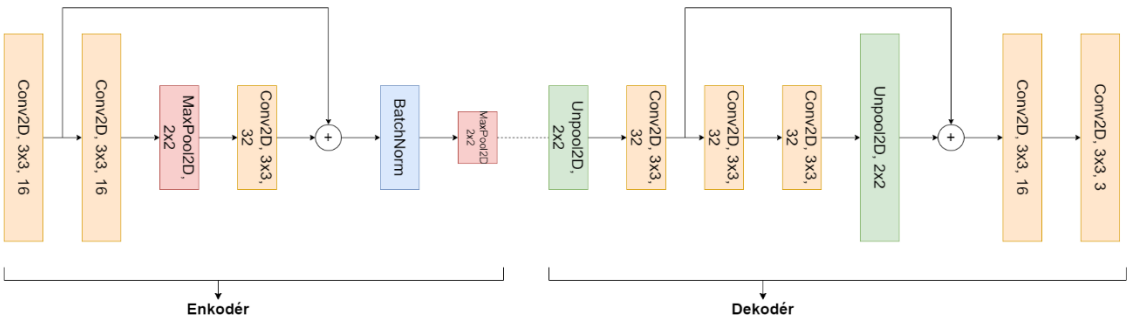
obrátky (*Round Trip Time,* RTT) a podíl délky paketu L ku přenosové rychlosti R [15].

$$t_{\Sigma} = t_{pred} + t_{tran} = t_{pred} + \left(RTT + \frac{L}{R}\right). \quad (8.1)$$

Data pro monitorování rychlosti přenosu jsou popsána v podkapitole 6.3. Jedná se o testovací podmnožinu datasetu, která čítá 790 obrázků defektů. Průměrná velikost jednoho obrázku je cca 17 kB. Parametry vzdáleného serveru popisuje tabulka 8.1. Pro výpočty byla uvažována přenosová rychlost 100 Mbps. Minimální latenci linky lze zajistit snížením doby obrátky a zvýšením přenosové rychlosti nebo snížením délky paketu. Změnit parametry přenosové linky je spjaté se změnou celkové infrastruktury. Takový krok by však znamenal vysoké ekonomické náklady. Proto se jako nejméně invazivní krok dá označit snížení velikosti paketu, což představuje redukci dat na straně uživatele. Té může být dosaženo následujícími prostředky:

- Redukce dimenzionality: omezení počtu kanálů na černou a bílou
- Odstranění redundantních informací: Z charakteristiky defektů vyplívá jejich rozložení pouze v malé části vstupních obrázků (viz obrázek 6.15 – vlevo nahoře). Takový úkol však vyžaduje segmentaci/detekci vstupního obrazu, což by mohlo krok samotné predikce na serveru udělat nadbytečným. Další možností je snížení velikosti vstupních obrázků. Avšak redukce velikosti byla již provedena před trénováním samotné sítě. Snižování velikostí by mohlo mít značný dopad na její performanci.
- Využití komprese: komprese je jeden z možných způsobů snížení množství datového toku. Obrázky jsou však už v základní podobě upraveny standartním kompresním formátem JPEG. Snížení objemu dat je z tohoto hlediska téměř nereálné.
- Extrakce příznaků: Prostřednictvím strojového učení extrahovat příznaky ze vstupních dat, tyto příznaky budou poslány přenosovým médiem a na straně serveru budou opět rekonstruovány v původní obraz.

Z výše uvedených možností se jako nejefektivnější nabízí extrakce příznaků. Příznaky lze z výchozích dat extrahovat pomocí generativního učení s využitím auto-enkodérů.

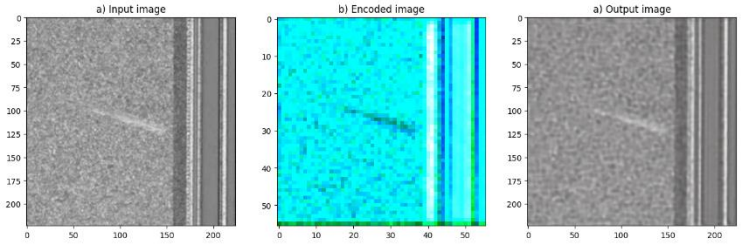


obrázek 8.4) Architektura autoenkodéru

8.4.1 Autoenkodéry

Jedná se o typ generativního učení, jež se používá pro převod dat z jednoho prostoru do druhého, s cílem naučit se reprezentaci vstupních dat v novém prostoru s menším počtem dimenzí. Tento typ učení se řadí do kategorie učení bez učitele (1.1.2). Cílem autoenkodérů je tak komprimace dat do menšího a efektnějšího formátu [8]. Hlavní idejí této podkapitoly je použití autoenkodérů pro komprimaci dat před jejich posláním na linku. V serverové části jsou pak rekonstruovány původní obrazy pomocí dekodéru a následně provedena predikce. Cílem této kapitoly je snaha o redukci datového toku za cenu zachování přesnosti predikce natrénované sítě (viz kapitola 6.3). Adresář s vlastní implementací kódu (název adresáře *autoencoders*) je součástí příloh této práce.

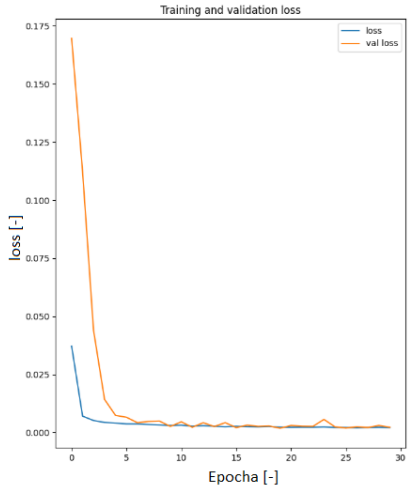
Architekturu sítě ilustruje obrázek 8.4. Zde je patrné rozdělení sítě na část enkodéru a dekodéru, přičemž data jsou zpracována na straně klienta pomocí enkodéru a následně rekonstruována serverem v dekódovací části. Tyto průchody ilustruje obrázek 8.5. S pomocí použité architektury lze očekávat 4× nižší datový tok skrz linku ve srovnání se stavem, kdy není aplikována žádná datová transformace. Přidáním modelu autoenkodéru mezi prediktor lze očekávat snížení performance klasifikačního modelu, jelikož samotný model vykazuje jistou hodnotu ztrát [8].



obrázek 8.5) Autoenkodér náhodného vzorku: a) původní obraz, b) kódovaný obraz, c) výstup autoenkodéru

Křivku ztrát v průběhu jednotlivých trénovacích epoch zobrazuje obrázek 8.6. Zde je možné sledovat monotónní pokles pro průběh trénovací a validační ztráty v průběhu třiceti epoch. Z této křivky není patrné přeučení modelu. Výsledná hodnota ztrát začíná po 8 epoše oscilovat kolem hodnoty 0,02.

Porovnáním vstupu a výstupu autoenkodéru (viz obrázek 8.5a, c) lze dedukovat dobrý výsledek rekonstrukční funkce. Z kódovaného obrázku (viz obrázek 8.5b) lze pozorovat správnou extrakci příslušného defektu (v tomto případě se jedná o škrábanec na waferu).



obrázek 8.6) Průběh trénovací a validační ztráty

Důležitým aspektem z hlediska inference jsou predikce konkrétních tříd na vstupních datech. Zde je model porovnán s modelem popsaným v 6.3. Výsledky shrnuje tabulka 8.2. Z výsledků je patrný značný pokles všech sledovaných metrik. Ten lze vysvětlit přidáním ztrát autoenkodéru. Opětovné navýšení těchto metrik by mohlo být možné přetrénováním modelu autoenkodéru za dosažení nižší střední kvadratické chyby (*mean squared error*, MSE). Toho může být dosaženo zvýšením množství trénovacích dat, popř. použitím vhodných augmentací. Kroky ladění autoenkodéru však přesahují zadání této práce.

tabulka 8.2) Porovnání performance po redukci dat

metrika		Výsledky z 6.3	Použití enkodéru
AUC	[-]	0,968	0,916
F1	[%]	90,48	74,7
Přesnost	[%]	94,13	83,03

Dalším důležitou evaluací je měření rychlosti odezvy serveru v případě aplikace autoenkodérů. Je zřejmé, že doba odezvy (viz rovnice (8.1)) bude obohacena o latence enkodéru a dekodéru. Tato skutečnost je prezentována následující rovnici [15]:

$$\mathrm{T}_{\Sigma}(N) = T_{pred}(N) + N \cdot \left(RTT + \frac{L}{R}\right) + T_{auc}(N), \tag{8.2}$$

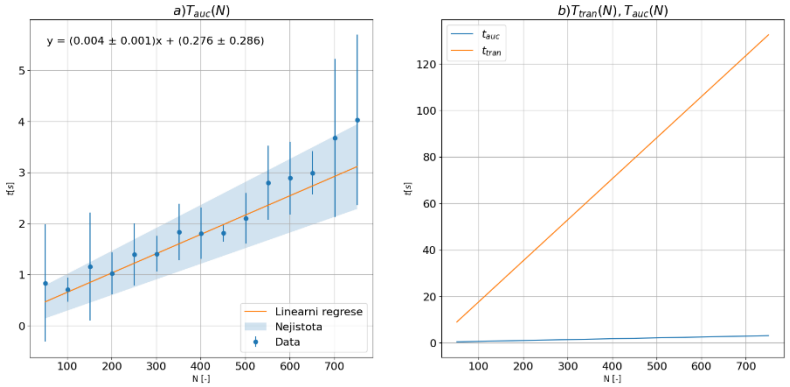
kde N značí množství paketů v požadavku. Z rovnice lze odvodit, že pro jednu sledovanou predikci je $t_{auc} \leq 1{,}08$ ms (odvození vztahu (8.3) viz A.2).

Předpokládaná hodnota přenosové rychlosti je $R = 100\ \mathrm{Mb/s}$ a průměrná velikost jednoho obrázku činí $L = 18\mathrm{kB}$. Z odvození je rovněž patrná přímá úměra na počtu vzorků N.

$$t_{auc} \leq \frac{N}{R} \cdot \Delta L. \tag{8.3}$$

Průměrná rychlost zpracování jednoho vzorku auto-enkodéru (v režimu podpory grafických karet) je $t_{auc} = 322{,}84\ \mathrm{ms}$. Lze rovněž předpokládat závislost doby zpracování enkodéru na počtu vstupních příznaků. Tato závislost je rovněž reflektována v rovnici (8.2). Experimentálně bylo zjištěno, že doba zpracování auto-enkodérem je přímo úměrná množství vstupních dat (viz obrázek 8.7a). Experiment byl proveden s předem definovaným rozměrem obrázků, popsaným v kapitole 6.3 (odvození výpočtů viz A.2). Srovnáním průběhů transportního času t_{tran} a času potřebného pro zpracování dat auto-enkodérem t_{auc} (viz obrázek 8.7b) lze pozorovat zanedbatelný vliv příspěvku auto-enkodéru. Pro výpočty je tedy uvažováno $T_{auc}(N) = konst. = t_{auc}$. Tuto skutečnost prezentuje rovnice (8.3).

Dosazením hodnoty do vztahu (8.3) lze získat minimální množství odeslaných vzorků, kdy doba zpracování dat s enkodérem je nižší než doba zpracování bez této komprimace. Tuto hodnotu lze po zaokrouhlení stanovit jako $N_{min} \doteq 299$. Pro skupinu defektů s nižší hodnotou představuje doba t_{auc} zvýšení latence odpovědi. Využití komprimace pomocí autoenkodérů se jeví jako velice výhodné, jelikož množství zpracovávaných dat je obvykle v řádech tisíců. Avšak pokles performance o 15 % představuje vysokou degradaci predikovaných hodnot. Pro nižší střední kvadratickou chybu bude nejspíše potřeba sesbírat další množství vstupních dat nebo snížit velikost komprese. Další problém v aplikaci enkodérů jako kompresního nástroje spočívá v jejich jedno účelném použití. Je-li změna vstupních příznaků vysoká, je potřeba provést přetrénování tohoto modelu na nových datech. Tento krok značně zvyšuje dobu trénování sítě jako celku. Při aplikaci enkodérů lze rovněž pokaždé očekávat určitou degradaci predikcí, jelikož část informace s příznaky může být při této kompresi ztracena.



obrázek 8.7) Závislost rychlosti zpracování dat a) auto-enkodéru a b) srovnání s přenosovou cestou

8.5 Dílčí závěr

Při realizaci inferenční části diplomové práce byly použity open-source platformy *Django* a *Django REST Framework* dostupné z [17] [18]. Kontejnerizační SW docker je dostupný z [16]. Funkční části webové aplikace jako je struktura databáze a část programové implementace byly převzány z [19]. Zdrojové kódy byly upraveny pro potřeby predikce založené na rozpoznání obrazu. Tyto zásahy jsou výlučně spojeny s úpravou nastavení django serveru pro ukládání médií (obrázků) určených k inferenci. Struktura vybraných tabulek v databázi byla rovněž pozměněna. Především tabulka *endpoint* byla obohacena o sloupec site, který specifikuje výrobní závod a tabulka *mlalgorithm* využívá pro identifikaci modelu odkaz na soubor formátu *h5* namísto ukládání kódu pomocí nástroje *inspect*. Kódová implementace souboru `backend/server/apps/ml/registry.py` byla dále obohacena o součást, jež je schopna identifikovat nově zaregistrovaný model dostupný ve službě azure ML studio od společnosti Microsoft. Tato funkcionalita je však dostupná pouze v rámci intranetu společnosti Onsemi. Kódová realizace kontejnerizačního softwaru Docker musela být v rámci revize odladěna. Soubor `docker-compose.yml` byl obohacen o služby spojené s SQL databází, včetně uživatelského rozhraní k databázi. Wsgi server vykazoval při skládání obrazu chyby spojené s použitými verzemi aplikace. Tyto chyby byly odstraněny. Pro webové rozhraní inferenčního zařízení byl zaměněn poslouchací port aplikace na 80, jež je pro http protokol typický a nemusí být při zadávání do prohlížeče explicitně definován.

9. ZÁVĚR

Z výstupů práce lze označit hodnoty metrik jako uspokojivé, za předpokladu minimálního množství dat potřebného k trénování o velikosti alespoň 6 000. Pro data s třetinovou velikostí je z křivek učení patrné přeučení, jež souvisí s nízkým množstvím vstupních dat. Takový klasifikátor se následně vykazoval náhodnou charakteristiku predikcí. Porovnáním jednotlivých iterací lze také konstatovat vysokou efektivitu použití předučených sítí (*transfer learning*, TL). Jejich přínos spočívá jednak ve zkrácení času potřebném pro přetrénování sítě a jednak ve vyšších hodnotách F1 skóre a přesnosti.

Z evaluace hyperparametrů je zřejmé, že jejich laděním nedochází k výraznému zlepšení výkonnosti sítě. Tuto vlastnost však lze interpretovat kladně, jelikož při řešení klasifikátorů pro úlohu predikce defektů je použití sítí *ResNet* v základní konfiguraci dostačující. Vyhodnocením augmentačních vrstev byla zjištěna pravděpodobná spojitost fluktuací na validační přesnosti v průběhu trénování s volbou různých augmentačních vrstev. Ladicím algoritmem bylo zjištěno, že nejlepší výkonnosti dosahuje síť pro horizontální a vertikální překlopení. Pro toto nastavení rovněž dochází ke značné redukci fluktuací.

Zařízení je z hlediska inference založené na predikci hodnot po jednotlivých vzorcích. Využití inferenčního zařízení bude rovněž použito v produkční verzi této aplikace. Obrázky s defekty jsou na straně serveru zpracovávány jednotlivě. Záznamy těchto dotazů jsou k dalšímu zpracování ukládány do databáze. Tento přístup však značně prodlužuje inspekci samotných waferů, proto je zde naznačen způsob komprimace dat pomocí autoenkodérů. Přidáním komprimace pomocí auto-enkodéru se sice zvyšují režijní náklady na přenos dat, avšak s vyšším množstvím dat redukují čas potřebný k jejich inferenci, proto je tato implementace pro vysoké množství dat výhodná. Vyhodnocení predikcí s použitím tohoto druhu komprimace však značně degraduje přesnost a F1 skóre. Avšak při zpracovávání vysokého množství vstupních dat může ušetřit značné množství času, potřebného ke klasifikaci. Vylepšení tohoto algoritmu bude pravděpodobně vyžadovat vyšší množství vstupních dat pro učení. Mírnou úpravou inferenčního zařízení je možné dosáhnout skládáním vstupních dat vedle sebe. Za tohoto předpokladu by došlo ke značné redukci požadavků na server, což by zkrátilo dobu klasifikace, především z hlediska doby obrátky RTT a to v řádech jednotek minut za předpokladu tisíců vstupních dat.

Zadání práce lze považovat za splněné. Z jejího výstupu je rovněž možné pozorovat možnosti dalších vylepšení. Především z hlediska inference, kterou lze obohatit o lepší možnosti zpětné vazby systému a také možnosti redukce datového toku.

10. REFERENCE

[1] CHOLLET, François. *Deep learning v jazyku Python: knihovny Keras, Tensorflow*. První vydání. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1.

[2] *MLOps: Continuous delivery and automation pipelines in machine learning* [online]. Neuvedeno: Neuvedeno, 2020 [cit. 2022-11-14]. Dostupné z: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning

[3] MARŠÁLEK, Roman. *Teorie rádiové komunikace*. Vyd. 1. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2012. ISBN 978-80-214-4503-1.

[4] ZACHA, Jiří. *Konvoluční neuronové sítě pro klasifikaci objektů z LiDARových dat*. Praha, 2019. Bakalářská práce. ČVUT. Vedoucí práce Ing. Patrik Vacek.

[5] MITRENGA, Michal. *KONVOLUČNÍ NEURONOVÁ SÍŤ PRO SEGMETACI OBRAZU*. Brno, 2018. Bakalářská práce. VUT. Vedoucí práce Doc. Ing. Václav Jirsík, CSc.

[6] *How to Ensure Image Dataset Quality In Image Classification* [online]. Neuvedeno: Picsellia, 2022 [cit. 2022-12-28]. Dostupné z: https://www.picsellia.com/post/image-data-quality-for-image-classification

[7] VERSTER, Jaco. *Estimating required sample size for model training: Modeling the relationship between training set size and model accuracy.* [online]. Neuvedeno: keras.io, 2021 [cit. 2022-12-28]. Dostupné z: https://keras.io/examples/keras_recipes/sample_size_estimate/

[8] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge: MIT Press, 2016. Adaptive computation and machine learning (MIT Press). ISBN 978-0262035613.

[9] MULLER, Nicolas a Karla MARKERT. Identifying Mislabeled Instances in Classification Datasets. *2019 International Joint Conference on Neural Networks (IJCNN)* [online]. IEEE, 2019, 1-8 [cit. 2022-12-28]. ISBN 978-1-7281-1985-4. Dostupné z: doi:10.1109/IJCNN.2019.8851920

[10] JAŠEK, Filip. *Predikce a analýza dopravní situace pomocí metod strojového učení*. Brno, 2021. Bakalářská práce. VUT. Vedoucí práce Ing. Jakub Götthans.

[11] *Čisté prostory a příslušné řízené prostředí - Část 1: Klasifikace čistoty vzduchu*. 11/2000. Neuvedeno: Neuvedeno, 2000. ČSN EN ISO 14644-1

(125301).

[12] HE, Kaiming, Xiangyu ZHANG, Shaoqing REN a Jian SUN. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2016, 770-778 [cit. 2022-11-14]. ISBN 978-1-4673-8851-1. Dostupné z: doi:10.1109/CVPR.2016.90

[13] QAZI, Emad Ul Haq, Tanveer ZIA a Abdulrazaq ALMORJAN. Deep Learning-Based Digital Image Forgery Detection System. *Applied Sciences* [online]. 2022, **12**(6) [cit. 2023-04-14]. ISSN 2076-3417. Dostupné z: doi:10.3390/app12062851

[14] How-to-build-a-resnet-from-scratch-with-tensorflow-2-and-keras. In: *Github* [online]. Neuvedeno: Neuvedeno, 2022 [cit. 2023-08-05]. Dostupné z: https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-build-a-resnet-from-scratch-with-tensorflow-2-and-keras.md

[15] KOLKA, Zdeněk. *Počítačové a komunikační sítě: Skripta FEKT VUT*. Brno, 2007.

[16] *Docker docs* [online]. Neuvedeno: Docker, Inc, 2013 [cit. 2023-05-15]. Dostupné z: https://docs.docker.com/

[17] Django. In: *Django: Django makes it easier to build better web apps more quickly and with less code.* [online]. Neuvedeno: Django Software Foundation, 2005 [cit. 2023-08-03]. Dostupné z: https://www.djangoproject.com/

[18] Django REST Framework. In: *Django REST Framework* [online]. Neuvedeno: Encode OSS Ltd, 2011 [cit. 2023-08-03]. Dostupné z: https://www.django-rest-framework.org/

[19] Machine Learning with Django. In: *ML With Django* [online]. Neuvedeno: MLJAR, Sp. z o.o., 2022 [cit. 2023-08-03]. Dostupné z: https://www.deploymachinelearning.com/

SEZNAM SYMBOLŮ A ZKRATEK

Zkratky:

CNN	Konvoluční neuronové sítě
CV	Computer Vision – strojové vidění
FC	Hustě propojené vrstvy
SGD	Stochastický gradientní sestup
SEM	Elektronový mikroskop
ROC	Receiver Operating Characteristics
FAB	Závod na výrobu polovodičů
DOI	Defect of Interest – sledovaný defekt
TL	Transfer Learning

Symboly:

F_1	F_1 skóre sítě	[-]
AUC	Plocha pod ROC křivkou	[-]
MSE	Střední kvadratická odchylka	[-]
H	Entropie	[bit]
S_C	Kosinova podobnost	[-]
RTT	Čas obrátky	[s]
L	Velikost paketu	[bit]
R	Rychlost přenosu	[b/s]
t_{auc}	*Čas zpracování autoenkodéru*	[s]
N	*Množství dat*	[-]

SEZNAM PŘÍLOH

Příloha A - Odvození výpočtů

A.1 Výpočty nejistot:

Číslo odměru	x_i	1	2	3	4	5
prodleva	[ms]	320	380	2000	420	705

Výpočty pro uvedené odměry byly získány pro počet vzorků 50 určených k následné inferenci. Pro všechny ostatní množství vzorků byl proveden stejný výpočet.
Průměrná hodnota:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{5} x_i = \frac{320 + 380 + \cdots + 305}{5} = 765$$

Výpočet nejistoty typu A:

$$u_A = s_x \cdot \frac{1}{\sqrt{n}} = \sqrt{\frac{1}{n(n-1)}\sum_{i=1}^{n}(x_i - \bar{x})^2} = \sqrt{\frac{1}{20}[(320-765)^2 + \cdots + (705-765)^2]} = 315{,}76$$

Předpokládá se alespoň provedení 10 odečtů, ze kterých se nejistota typu A vypočítá. Je nutno dopočítat správnou hodnotu pomocí opravných koeficientů:

$$u_A = k_s \cdot s_{\bar{x}} = 1{,}4 \cdot 315{,}76 = 442{,}06$$

Hodnota $k_s = 1{,}4$ je tabelována pro počet pěti odměrů

A.2 Odvození výpočtu odezvy linky:

$$t_{pred} + N \cdot \left(RTT + \frac{L_1}{R}\right) + t_{auc} \leq t_{pred} + N \cdot \left(RTT + \frac{L_2}{R}\right)$$

$$t_{auc} \leq N\left(RTT + \frac{L_2}{R}\right) - N\left(RTT + \frac{L_1}{R}\right)$$

$$t_{auc} \leq \frac{N}{R}(L_2 - L_1)$$

$$t_{auc} \leq \frac{N}{R} \cdot \Delta L$$

Předpoklad: $N = 1 \ \wedge R = 100\ \text{Mbps} \wedge L_1 = \frac{1}{4}L_2 = L = 144 \cdot 10^3 \text{kb}$

$$t_{auc} \leq \frac{1}{100 \cdot 10^6} \cdot \frac{3}{4} \cdot 144 \cdot 10^3$$

$$t_{auc} \leq \frac{108}{100 \cdot 10^3} = 1{,}08\ \text{ms}$$

Rozměrová zkouška:

$$t_{auc} = \frac{1}{\text{kb} \cdot s^{-1}} \cdot \text{kb} = \text{s}$$

Příloha B - Zdrojové kódy

Zdrojové kódy jsou součástí komprimovaného souboru `code.zip`. Stromová struktura adresáře je uvedena níže.

```
[code]
    ab_testing.ipynb
    augment_data.py
    dataPreparation.py
    docker-compose.yml
    evaluate_augmentations.py
    nn_config.yml
    preprocess_train_data.py
    similarity.py
    train_and_validate_model.py
    upload_data.py
    [autoencoders]
        main.py
        test.ipynb
        [models]
    [backend]
        [mediafiles]
            [mediafiles]
        [server]
            db.sqlite3
            manage.py
            [apps]
                [endpoints]
                    admin.py
                    apps.py
                    models.py
                    serializers.py
                    tests.py
                    urls.py
                    views.py
                    __init__.py
                    [migrations]
                        0001_initial.py
                        0002_rename_descritpion_mlalgorithm_description.py
                        0003_alter_mlrequest_input_data.py
                        0004_alter_mlrequest_input_data.py
                        0005_abtest.py
                        0006_alter_abtest_created_at.py
                        __init__.py
                [ml]
                    registry.py
                    tests.py
                    __init__.py
                    [classifier]
                        resnet.py
            [server]
                settings.py
                urls.py
                wsgi.py
                __init__.py
    [docker]
        [backend]
            Dockerfile
            wsgi-entrypoint.sh
        [nginx]
            default.conf
            Dockerfile
    [libs]
        [dataProcessing]
            .amlignore
            .amlignore.amltmp
            augment.py
            balance.py
            preprocessing.py
            transform.py
            __init__.py
        [misc]
            model.py
            parser.py
            __init__.py
        [train]
            Resnet.py
            __init__.py
```