FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Bakalářská práce

Rapidní vývoj algoritmů strojového vidění

Studijní program: B0613A140005 – Informační technologie
Studijní obor: B0613A140005AI – Aplikovaná informatika
Autor práce: **Kevin Daněk**
Vedoucí práce: Ing. Igor Kopetschke
Konzultant: Bc. Albert Myšák

Liberec 2024

FAKULTA MECHATRONIKY,
INFORMATIKY A MEZIOBOROVÝCH
STUDIÍ TUL

Zadání bakalářské práce

Rapidní vývoj algoritmů strojového vidění

Jméno a příjmení:	**Kevin Daněk**
Osobní číslo:	M21000099
Studijní program:	B0613A140005 Informační technologie
Specializace:	Aplikovaná informatika
Zadávající katedra:	Ústav nových technologií a aplikované informatiky
Akademický rok:	2023/2024

Zásady pro vypracování:

1. Proveďte rešerši na téma rapidního vývoje algoritmů strojového vidění včetně souvisejících nástrojů s důrazem na ImageNets, PEKAT Vision, Vision Builder a případné další.
2. Demonstrujte nedostatky nástrojů diskutovaných v rešerši na praktických situacích. Vycházejte z aktuálního stavu a potřeb společnosti CLOUDCODE s.r.o.
3. Analyzujte a definujte požadavky na vlastní řešení dle potřeb společnosti CLOUDCODE s.r.o.
4. Na základě předchozích bodů navrhněte vlastní řešení s důrazem na vyvážení přístupnostni expertního systému. V rámci návrhu celkové architektury realizujte propojení se systémem CCM (CLOUDCODE Manager) s důrazem na rozšiřitelnost.
5. Implementujte svůj návrh formou webové aplikace, implementujte grafické rozhraní za využití knihoven OpenCV, TensorFlow, Plotly, a interních knihoven CLOUDCODE.
6. Nasaďte výsledný produkt do produkčního prostředí, získejte zpětnou vazbu od uživatelů, analyzujte jí a navrhněte další směry vývoje a odstranění případných nedostatků.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30 – 40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	čeština

Seznam odborné literatury:

[1] WATHAN, Adam a Steve SCHOGER. Refactoring UI [online]. Adam Wathan & Steve Schoger, 2019. Dostupné z: https://www.refactoringui.com

[2] SOMMERVILLE, Ian. Software Enginnering [online]. 9th ed. Addison-Wesley, 2010. ISBN 978-0-13-703515-1. Dostupné z: https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th–Edition-by-Ian-Sommerville.pdf

[3] PRINCE, Simon J.D. Understanding Deep Learning [online]. August 6th, 2023 Edition. MIT Press, 2023. Dostupné z: https://udlbook.github.io/udlbook/

Vedoucí práce:	Ing. Igor Kopetschke Ústav nových technologií a aplikované informatiky
Konzultant práce:	Bc. Albert Myšák CLOUDCODE s.r.o., Paní Zdislavy 418, 470 01 Česká Lípa
Datum zadání práce:	12. října 2023
Předpokládaný termín odevzdání:	14. května 2024

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Chaloupka, Ph.D.
garant studijního programu

V Liberci dne 19. října 2023

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

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

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

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

9. 5. 2024 Kevin Daněk

RAPIDNÍ VÝVOJ ALGORITMŮ STROJOVÉHO VIDĚNÍ

ABSTRAKT

Cílem této práce je pro společnost CLOUDCODE s.r.o. provést rešerši existujících nástrojů pro rapidní vývoj algoritmů strojového vidění, zjistit jejich nedostatky pro potřeby společnosti a navrhnout vlastní řešení, bude splňovat požadavky dané společností CLOUDCODE a bude integrováno do softwaru CLOUDCODE Manager. Práce má za cíl řešení nasadit do produkce a získat zpětnou vazbu od uživatelů.

Klíčová slova: Neuronové sítě, Strojové vidění, Rapidní vývoj, Automobilový průmysl, Rapidní prototypování

RAPID DEVELOPMENT OF MACHINE VISION ALGORITHMS

ABSTRACT

This thesis aims to research available tools for rapid development of machine vision algorithms, summarize their disadvantages and point out areas they are lacking in for usage in CLOUDCODE and design own solution that satisfies requirements set by CLOUDCODE and integrates with CLOUDCODE Manager and already existing codebase. Thesis also aims to deploy the solution to production and gather feedback from users.

Keywords: Neural networks, Machine vision, Rapid development, Automotive, Rapid Prototyping

PODĚKOVÁNÍ

Rád bych poděkoval panu Ing. Igorovi Kopetschkemu, který mě provedl nejenom procesem přípravy tématu, ale byl mi nápomocný kdykoliv jsem ho potřeboval a nechal mě pracovat mým vlastním, poněkud zvláštním tempem. Díky také patří společnosti CLOUDCODE s.r.o. a Albertovi Myšákovi, který se mnou konzultoval problémy firmy a pomáhal mi s rešerší.

Pro zlepšení srozumitelnosti, přístupnosti a čitelnosti textu jsem využil modely *ChatGPT 3.5* a *Gemini* k upravení některých formulací. Tímto prohlašuji, že výstupy jsou v textu použity v souladu se směrnicí rektora č. 4/2023 Technické univerzity v Liberci a jsem si vědom následků, které vyplývají z jejího porušení.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM VZORCŮ

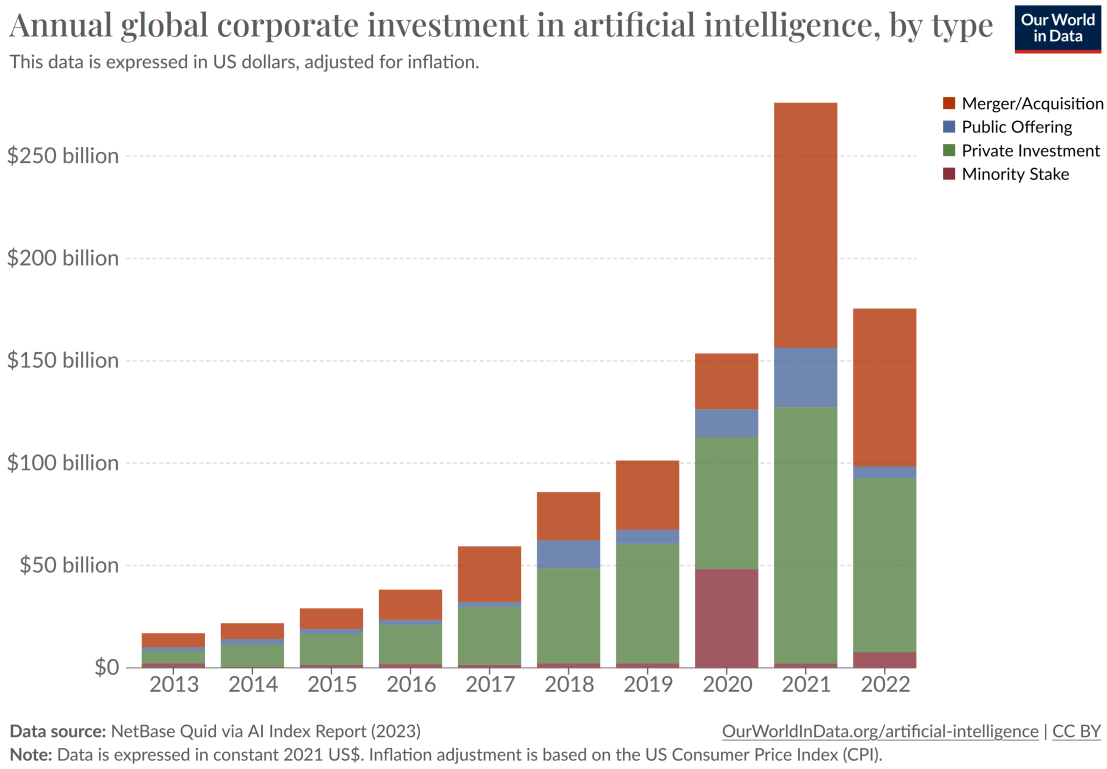
SEZNAM ZKRATEK

AD převodník	Převodník anologového signálu na digitální
API	Rozhraní, které umožňuje různým softwarovým aplikacím komunikovat a interagovat mezi sebou.
CCM	Cloudcode Manager
CMYK	Barevný model *Cyan*, *Magenta*, *Yellow* & *Black*
HTTP	Hyper-text Transfer Protocol, protokol používaný pro komunikaci s webovými servery.
RGB	Barevný model *Red*, *Green* & *Blue*
SDK	Sada nástrojů pro vývoj a integraci software (kompilátor, debugger či framework)

1 ÚVOD

Strojové učení a umělá inteligence jsou tu s námi již od šedesátých let minulého století. Jejich vývoj probíhal za oponami výzkumu a nebyl, až na pár výjimek, moc veřejně diskutovaný. Neomezený potenciál těchto technologií se do širšího povědomí veřejnosti dostal až v nedávné době s příchodem modelů jako DALL-E či ChatGPT. (Roser, 2022)

Ačkoliv jsou výstupy takto obřích modelů dechberoucí, jsou často uplatnitelné jako bagr na dětském pískovišti. Pro většinu problémů se zkrátka jedná o příliš komplexní a obecné nástroje. Proto se již dávno před příchodem těchto populárních modelů investovalo v korporátní sféře do vývoje jednoúčelových neuronových sítí. (Roser, 2023)



Obrázek 1.1: Korporátní investice do technologie umělé inteligence v čase (Roser, 2023)

Podniky mají zpravidla jeden cíl: být úspěšným, a tím pádem i výdělečným, podnikem. Jedním ze způsobů, jak úspěchu na trhu dosáhnout, je být lepší než tržní konkurence. Toho lze docílit zkvalitněním a zrychlením služeb či výroby. Pokud ale chcete být lepší než všichni ostatní, musíte inovovat. A investice do nových a experimentálních technologií často nejsou levné (Škabrada, 2019).

Podíváme-li se konkrétně na výrobní linky, po rozšíření robotů vzešla otázka, kam dále posouvat inovaci. Roboti totiž dokázali nahradit pouze práce, které byly dobře algoritmizovatelné a nevyžadovaly žádné hlubší rozhodování. Logicky se průmysl začal ptát, zdali by nešlo roboty opatřit jistou dávkou inteligence.

S takovou úvahou vstoupila na trh v roce 2019 společnost CLOUDCODE, která si jako oblast své expertízy vybrala kontrolu kvality. Jejím dlouhodobým cílem je vytvořit pro zákazníky řešení na míru, které dokáže predikovat, jestli je výrobek vadný (CLOUDCODE s.r.o., 2023).

2 STROJOVÉ VIDĚNÍ

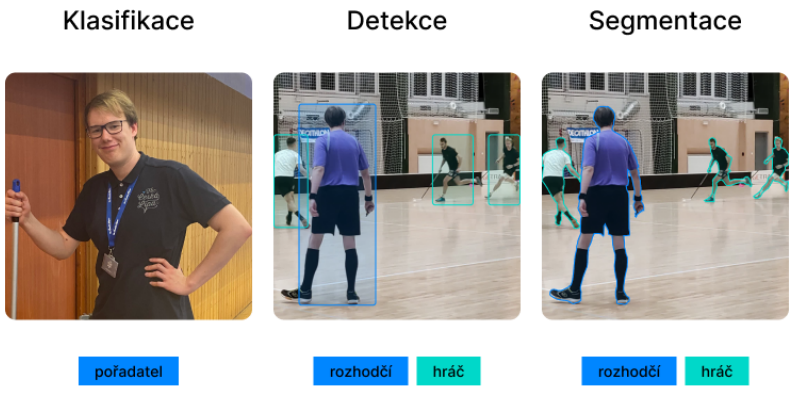
Strojové vidění je disciplína, která si klade za cíl získávat, zpracovávat a především analyzovat obsah digitálních obrázků. Cílem je tedy převést digitální obraz na užitečnou informaci (NVIDIA Corporation, 2023).

2.1 ÚLOHY STROJOVÉHO VIDĚNÍ

Informaci, kterou chceme z obrazu získat, lze obecně rozdělit do následujících kategorií:

- **Klasifikace** - Úlohou je určit, do jaké třídy obrázek patří. Například pokud je na obrázku zvíře, bude zařazeno s určitou pravděpodobností do skupiny obrázků zvířat.
- **Detekce** - Úlohou je určit přítomnost a pozici zájmových prvků v obrázku.
- **Semantická segmentace** - Úlohou je klasifikovat jednotlivé pixely do správných tříd. Výsledkem je nalezení oblastí zájmových prvků.

Informaci získánou jednou z těchto metod můžeme dále zpracovávat v kontextu konkrétní úlohy, kterou se snažíme řešit (Bandyopadhyay, 2022).



Obrázek 2.1: Úlohy strojového vidění - překleseno od Bandyopadhyay, 2022

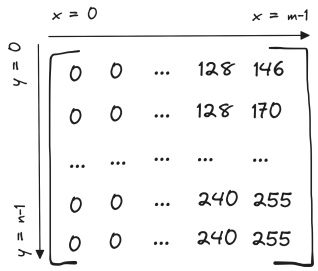
2.2 DISKRÉTNÍ OBRAZOVÁ FUNKCE

Strojové vidění pracuje s digitálními obrazy. Není cílem této práce popsat, jakým způsobem se uchovávají, ale jak s nimi pracujeme. Digitální obraz je reprezentován maticí obrazových bodů a diskrétní obrazovou funkcí (Chaloupka, 2024a). Pokud si tedy vezmeme matici bodů A o velikosti $m \times n \times k$ (šířka, výška, počet barevných kanálů) a k ní diskrétní obrazovou funkci $f(x, y, z)$, pak mezi maticí A a obrazovou funkcí $f(x, y, z)$ existuje zobrazení

$$
\begin{aligned}
f(x, y, z) &\mapsto A_{xyz} \\
\{x \in \mathbb{Z} \mid 0 \leq x < m\} \\
\{y \in \mathbb{Z} \mid 0 \leq y < n\} \\
\{z \in \mathbb{Z} \mid 0 \leq z < k\}
\end{aligned}
$$

Vzorec 2.1: Diskrétní obrazová funkce

Definiční obor diskrétní obrazové funkce $D(f)$ je omezen na velikost matice, přičemž indexování začíná na nule. To znamená, že $f(0, 0)$ je první obrazový bod v obrázku (vlevo nahoře).



Obrázek 2.2: Indexování matice

Obor hodnot diskrétní obrazové funkce $H(f)$ je složený z jasových intenzit jednotlivých barevných kanálů v obraze. Například u modelu RGB jsou v oboru hodnot čísla v intervalu $\langle 0, 255 \rangle$. Tento interval se může lišit v závislosti na bitové hloubce, která říká, kolik bitů mohu na vyjádření intenzity jednoho kanálu v konkrétním bodě použít (Krejčí, 2023; Chaloupka, 2024a).

Výhodou zavedení diskrétní obrazové funkce je, že nám to dovolí pracovat s maticí obrazových bodů jako s funkcí a skládat ji s dalšími funkcemi. Můžeme tak algoritmus strojového vidění reprezentovat jako sérii transformací.

2.3 PŘEDZPRACOVÁNÍ OBRAZU

Při pořizování obrazu se může velmi snadno stát, že bude výsledný snímek zatížen vadami. Tyto vady nám mohou ztížit jeho zpracování. Jedná se tak zejména o nežádoucí šum, optické zkreslení, nízký kontrast či nevýrazné klíčové charakteristiky, jako jsou například hrany. Tyto nedostatky se snaží řešit proces takzvaného *předzpracování*, jehož výsledkem je obraz, který je lepší pro další zpracování (Hlaváč, 2024d).

2.3.1 Změna barevného prostoru

Barevné prostory definují, jakým způsobem popisujeme barvy v obraze. Určují rozsah barev, které můžeme v digitálním obrázku reprezentovat (Kelly, 2023).

Cílem změny barevného prostoru je najít takový barevný kanál, ve kterém je nejvíce užitečné informace. Dimenze obrazu se pak redukuje o jeden rozměr, a to o hloubku. Pokud nelze vybrat žádný vhodný kanál, je možné obraz převést na šedotónový, kde je pouze kanál Y, který reprezentuje jasovou složku.

$$Y(r, g, b) = 0.3r + 0.6g + 0.1b$$

Vzorec 2.2: Převod z prostoru RGB na jasovou složku Y

Výhodou redukce na jeden kanál je to, že lze poté s obrazem pracovat jako s dvourozměrným. Na souřadnicích x a y se tak nenachází vektory, ale skaláry, se kterými pracují například algoritmy pro filtrování šumu, prahování či detekci hran.

2.3.2 Filtrace šumu

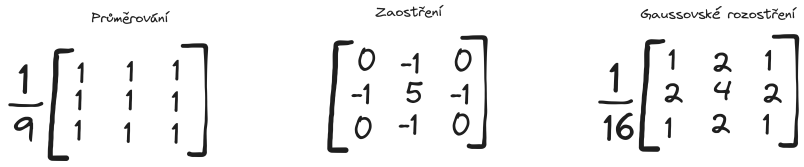
Jako šum v obraze chápeme náhodné změny v hodnotách diskrétní obrazové funkce, které nejsou námi žádané. Šum se v obraze může objevit již při jeho sejmutí světlocitlivým senzorem, při kvantizaci v AD převodníku, kompresi či dále při jeho zpracovávání, kdy se do obrazu promítne zaokrouhlovací chyba při výpočtech (Hlaváč, 2024c).

Při snaze odstranit šum v obraze spoléháme na fakt, že sousední pixely mají stejnou nebo podobnou hodnotu. Správná hodnota obrazového bodu se tak odhaduje z hodnot v jeho malém okolí. Tento odhad provádíme pomocí konvoluce, což je operace, jejímž výstupem je lineární kombinace hodnot obrazové funkce v malém okolí bodu. Koeficienty (váhy) jednotlivých bodů v okolí, neboli jak moc bod přispívá k nové hodnotě určuje konvoluční jádro $h(x, y)$.

$$g(x,y) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} f(m,n) \cdot h(x-m, y-n)$$

Vzorec 2.3: Diskrétní dvourozměrná konvoluce (Ahn, 2024)

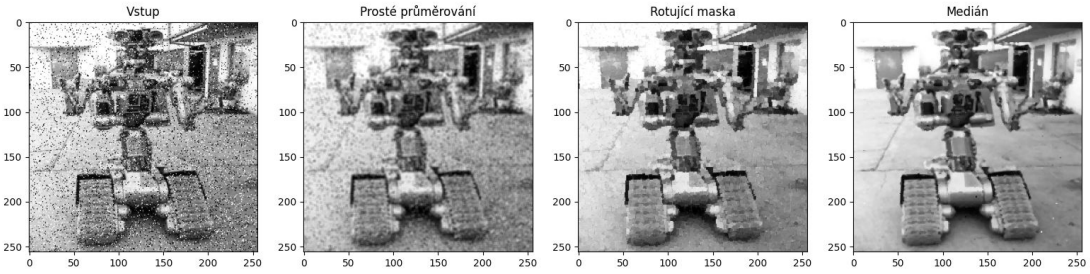
V rámci filtrování se konvoluce používá pro rozostření či zaostření obrazu, ovšem své uplatnění najde i v zvýrazňování charakteristických rysů. Tím, že vytváří lineární kombinace, měla by pro konvoluci platit pravidla aditivity a homogenity. V praxi tomu tak není, protože obor hodnot je omezen a nelze ho přesáhnout (například $\langle 0, 255 \rangle$). Nevýhodou těchto metod založených na konvoluci je to, že dochází k rozmazávání hran při náhlých změnech jasu (Hlaváč, 2024c).



Obrázek 2.3: Konvoluční jádra (Ormesher, 2020)

Nelineární filtrovací metody mají za cíl omezit rozmazávání hran. Nevyužívají tak konvoluci, ale robustnější statistiku. Mezi dvě významné metody patří filtrace pomocí rotující masky a pomocí mediánu. Princip rotující masky spočívá v tom, že se maska posouvá kolem zkoumaného bodu tak, aby byl pokaždé v jiné poloze v rámci masky. Následně se vybírá ta maska, ve které mají hodnoty jasu nejmenší rozptyl. Filtrování pomocí mediánu používá pro určení hodnoty namísto průměru druhý výběrový kvantil (který běžně známe jako medián). Výhodou mediánu je, že narozdíl od průměru není náchylný na vychýlení extrémními hodnotami (Hlaváč, 2024c).

Heuristicky pak můžeme zvolit nejvhodnější způsob pro daný obrázek. Například pokud vidíme, že je obraz zatížen šumem typu *salt and pepper*, je vhodné filtrovat pomocí mediánu z lokálního okolí obrazového bodu.

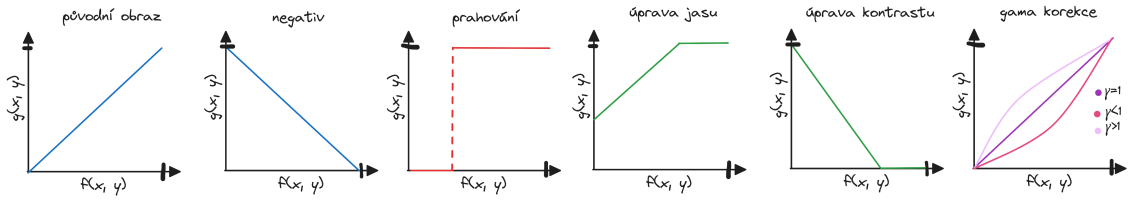


Obrázek 2.4: Filtrace šumu typu *salt and pepper*

2.3.3 Jasová korekce

Jas, neboli světelná intenzita obrazového bodu, hraje klíčovou roli ve vnímání obrazu a přímo ovlivňuje schopnost obraz interpretovat. Změny v jasu mohou dramaticky zjednodušit interpretaci či následné zpracování obrazových dat. Prostředky pro změnu jasových úrovní v obraze nazýváme jasové korekce. Podle závislosti na poloze obrazového bodu budu v této práci rozlišovat jasové korekce statické[1] (polohově nezávislé) a dynamické[2] (polohově závislé).

Obecnou statickou jasovou korekci $f_c(i)$ pro diskrétní obrazovou funkci $f(x, y)$ a výsledný obraz $g(x, y)$ lze zapsat jako $g(x, y) = (f_c \circ f)(x, y)$. Mezi statické jasové korekce patří zejména prahování (obecně pak redukce počtu úrovní jasu), ekvalizace histogramu či gama korekce. (Chaloupka, 2024b; Horák, 2019).



Obrázek 2.5: Statické jasové korekce (překresleno od Horák, 2019)

Dynamické jasové korekce se od těch statických liší tím, že používají informaci o poloze obrazového bodu ke svému výpočtu. Polohu bodu potřebuje degradační funkce $e(x, y)$, která obsahuje opravné koeficienty. Poruchy se často vyjadřují multiplikativním modelem, tudíž pro diskrétní obrazovou funkci $f(x, y)$, výsledný obraz $g(x, y)$ a degradační funkci $e(x, y)$ platí, že $g(x, y) = e(x, y) \cdot f(x, y)$ (Hlaváč, 2024d).

Obrázek 2.6: Dynamická jasová korekce

[1] V citované literatuře se označují jako transformace jasové stupnice (Chaloupka, 2024b)
[2] V citované literatuře se označují jako jasové korekce (Chaloupka, 2024b)

2.3.4 Geometrické transformace

Geometrické transformace nám dovolují ovlivňovat tvar, velikost, orientaci a polohu obrazu. Máme tak možnost eliminovat chyby v perspektivě při snímání a normalizovat tak polohu a rotaci zájmových objektů ve snímcích. Uvažujme čtyři základní transformace v kartézské soustavě souřadnic, a to posunutí T, rotaci R, změnu měřítka S a zkosení Sh. Pro každý bod diskrétní obrazové funkce $f(x, y)$ lze spočítat novou polohu pixelu pomocí následujících vztahů:

$$\begin{aligned} T(x, y, t) &= [x + t_x, y + t_y] \\ R(x, y, \theta) &= [x \cdot \cos(\theta) - y \cdot \sin\theta, x \cdot \sin\theta + y \cdot \cos\theta] \\ S(x, y, s_c) &= [x \cdot s_{cx}, y \cdot s_{cy}] \\ Sh(x, y, s_k) &= [x + sky \cdot y, s_{kx} \cdot x + y] \end{aligned}$$

Vzorec 2.4: Transformace v kartézských souřadnicích (Surynková, 2024)

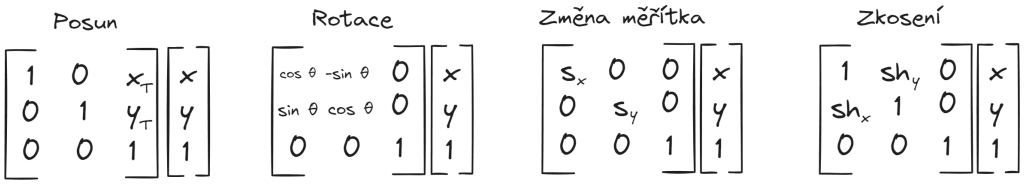
Při aplikaci geometrických transformací na diskrétní obrazovou funkci může velmi snadno dojít ke ztrátě informace. Výsledné body mohou buďto ležet mimo rastr, nebo jim nemusí být přiřazena hodnota. Ke korekci tohoto problému se používá proces interpolace, který pomáhá vyplnit mezery mezi existujícími hodnotami obrazových, čímž zajistí plynulý přechod a zachování detailů v transformovaném obraze. Mezi interpolační metody patří metoda nejbližšího souseda, bilineární interpolace a bikubická interpolace (Hlaváč, 2024b).

Problémy nenastávají pouze s výstupem. Pokud na vstupu pracujeme v kartézské soustavě, musíme každou složitější transformaci, respektive složenou transformaci, počítat postupně krok po kroku. Tedy v případě otočení, posunutí a dalšího otočení musíme vypočítat všechny tři kroky zvlášť. Kvůli tomu se v praxi namísto kartzéských souřadnic používá systém homogenních souřadnic. Uvažujme dvourozměrnou kartézskou soustavu souřadnic, poté homogenní souřadnice jsou rozšířením těchto souřadnic o jednu další dimenzi, zpravidla označovanou písmenem w. Pro každou souřadnici (x, y) v dvourozměrném kartézském systému existuje homogenní souřadnice $(x, y, 1)$ (Kaleniuk, 2017).

$$(x, y)_k = \left(\frac{x}{w}, \frac{y}{w}, w\right)_h$$

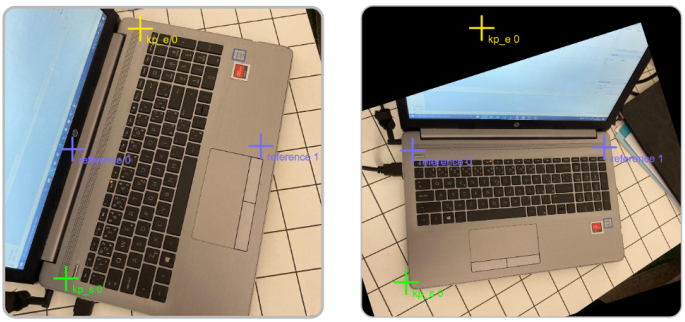
Vzorec 2.5: Převod kartézských souřadnic na homogenní souřadnice

Homogenní souřadnice umožňují vyhnout se těžkopádným vztahům z kartézských souřadnic a vyjádřit geometrické transformace pomocí čtvercových matic. Skládání transformací je poté záležitost násobení matic, které odpovídají jednotlivým operacím. Výsledkem je jedna matice, která v sobě uchovává všechny transformace na ní vykonané. Této matici říkáme projekční matice (Kaleniuk, 2017).



Obrázek 2.7: Matice transformací (Surynková, 2024)

Projekční matici pro transformaci obrazu lze buď předpočítat na začátku, nebo ji počítat pro každý obraz zvlášť. Tato volba závisí na povaze aplikace a potřebách daného úkolu. V případech, kdy nelze zaručit stálé a konzistentní zkreslení obrazu, je často lepší vypočítat projekční matici dynamicky pro každý konkrétní obrázek. V takových případech je nutné identifikovat klíčové rysy, které jsou stabilní a snadno identifikovatelné ve všech obrazech. Transformace obrazu pak probíhá na základě porovnání těchto klíčových rysů s odpovídajícími referenčními body nebo rysy.



Obrázek 2.8: Geometrická transformace - zarovnání na referenční body

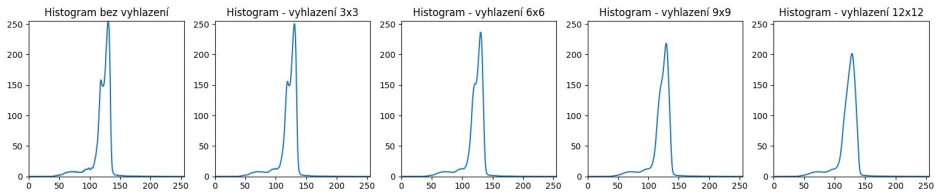
2.3.5 Zvýraznění klíčových charakteristik

Tímto krokem se zaměřujeme na identifikaci a zvýraznění důležitých rysů, jako jsou hrany, textury nebo oblasti zájmu, které jsou klíčové pro následné analýzy a úlohy zpracování obrazu.

Hrana je definována jako prudká změna jasu v lokálním okolí obrazového bodu. Pro identifikaci hran se využívá analýza změn hodnot v obrazové funkci $f(x, y)$. Analýza rychlosti změn v hodnotách funkce se provádí pomocí derivací, zejména těch parciálních, jelikož pracujeme s dvourozměrnou funkcí. Parciální derivace nám poskytují informace o rychlosti změny hodnot

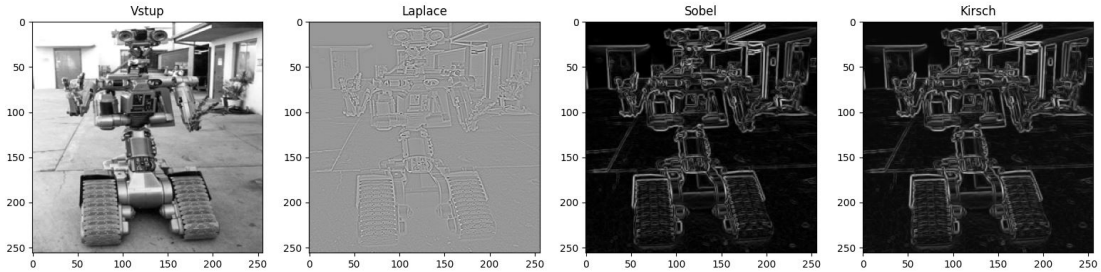
v jednotlivých osách obrazu. Ty se poté používají k vytvoření gradientu funkce, který nám poskytuje informaci o rychlosti změn a směru nejprudšího růstu v daném bodě. Z hlediska detekce hran je právě směr tou klíčovou informací (Hlaváč, 2024a).

Vzhledem k tomu, že pracujeme s diskrétní funkcí, je nutné parciální derivace aproximovat. Toho lze docílit diferencí, která aproximuje hodnotu derivace pomocí rozdílu mezi sousedními body v obraze. Při této aproximaci vzniká ve výsledné derivaci mnoho lokálních extrémů kvůli fluktuacím hodnot jasu v oblastech, kde změna jasu není tak velká a mění se postupně. Tento problém lze vyřešit buďto vyhlazením průběhu aproximované derivace, a nebo předzpracováním výsledného obrazu s nalezenými hranami (Hlaváč, 2024a).



Obrázek 2.9: Vyhlazování signálu

Pro výpočet diferencí v okamžitém bodě je možné využít konvoluci, pro kterou lze vhodně zvolit hodnoty jádra tak, aby výsledek byl aproximací derivace ve směru. Konvoluční jádra typu Prewitt, Sobel, Robinson a Kirsch jsou navržena právě pro odhad první derivace. Tyto jádra používají matici o velikosti 3×3 a analyzují okolních osm bodů. Protože výsledek se představuje aproximaci směrové derivace, počítá se pro gradient osm konvolucí pro každý směr v diskrétním obraze se čtvercovou mřížkou. Z osmi výsledků se za gradient zvolí ten, který má největší absolutní hodnotu. Konvoluční jádro se pro každý směr upravuje rotací hodnot kolem středu matice. K detekci hran lze také využít aproximaci druhé derivace, známou jako Laplacián. Ten je směrově nezávislý, což výpočet redukuje na jednu konvoluci, ale za to ztrácíme informaci o směru hrany (Hlaváč, 2024a).



Obrázek 2.10: Hranové detektory Laplace, Sobel a Kirsch

2.4 ZPRACOVÁNÍ OBRAZU

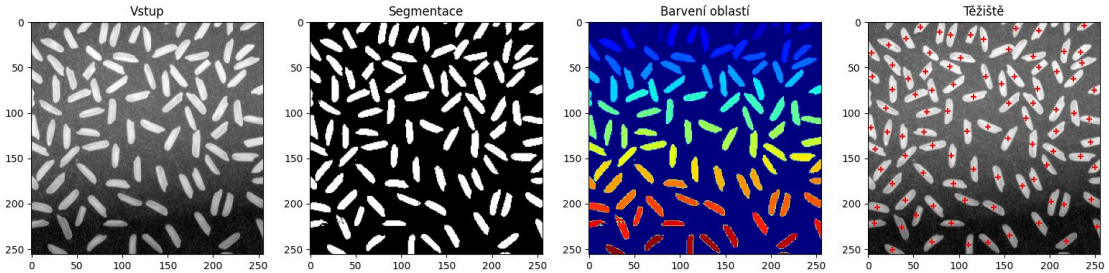
Zpracováním obrazu myslíme již samotný proces, který se zabývá analýzou obrazových dat a jejich interpretací. V této části procesu se řeší již dříve zmíněné úlohy strojového vidění, jako je klasifikace, segmentace a detekce objektů. Přístupy, jak z obrazu informaci vytěžit, jsem v této práci rozdělil do tří kategorií, které postupně představím.

2.4.1 Procedurální algoritmy

Procedurální algoritmy představují tradiční přístup k zpracování obrazu, který využívá předem definovaného souboru pravidel a operací k manipulaci s obrazovými daty. Využívá se znalosti o dané úloze a doméně k smysluplné extrakci informací. Výhodou je úplná kontrola nad procesem zpracování, nicméně mají problém se vypořádat s variabilitou ve vstupních datech. Částečně se to dá vyřešit důslednějším předzpracováním, které se bude snažit odchylky od kalibračního obrazu eliminovat.

Jednou z úloh, se kterou se můžeme setkat, je segmentace. Ta má za úkol obraz rozdělit na smysluplné segmenty, které odpovídají objektům, strukturám nebo oblastem zájmu. Nejjednoduší metodou segmentace je prahování, které přiřadí obrazovému bodu logickou jedničku, pokud je jeho hodnota nad nebo na hodnotě prahu, a logickou nulu, pokud je pod prahem. Tento přístup funguje dobře, pokud máme jasně oddělené objekty zájmu od pozadí. V složitějších případech lze využít lokálního nebo vícepásmového prahování (Hlaváč, 2019).

Pokud je na obrázku více oblastí zájmu, může být výhodné tyto oblasti po segmentaci od sebe oddělit a zpracovávat je dále samostatně. Jednou takovou technikou pro oddělení těchto oblastí je algoritmus barvení oblastí, který hledá souvislé plochy a každé přiřadí číslo od dvojky výše[3]. Výsledek barvení oblastí je úzce spjatý s kvalitou segmentace, obzvláště pokud jsou oblasti hodně blízko u sebe.



Obrázek 2.11: Příklad segmentace a barvení oblastí u rýžových zrn

[3]Od dvojky se začíná, protože nulou je reprezentováno pozadí a jedničkou segmentovaný obrazový bod, který ještě nebyl algoritmem vyhodnocen.

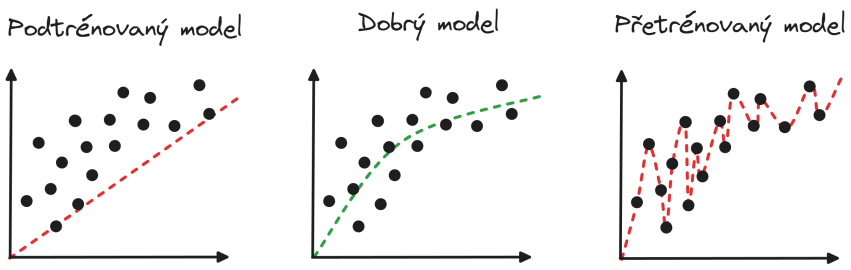
2.4.2 Strojové učení

Strojové učení umožňuje z dat vytvářet modely schopné provádět úkoly bez explicitního programování. Hlavní výhodou strojového učení je, že výsledné modely, narozdíl od procedurálních algoritmů, nejsou tolik náchylné na selhání při větší variabilitě v datech a tím dosahují vysoké přesnosti. Na druhou stranu však mohou být náchylné k přetrénování (Microsoft, 2024b).

Práce s modely strojového učení se skládá ze dvou fází, a to z tréninku a predikce. Během tréninku se model učí identifikovat vzory a vytváří si vnitřní představu o struktuře a závislostech v datech. Během tohoto procesu se optimalizují parametry modelu tak, aby co nejlépe odpovídaly trénovacím datům a aby model uměl zobecňovat nová data. Predikce je schopnost modelu vyhodnocovat nová data. V této fázi model aplikuje naučené znalosti na nové vstupy a generuje výstupy. Podle přístupu k učení modelu se dají algoritmy rozdělit do tří tříd (Microsoft, 2024a):

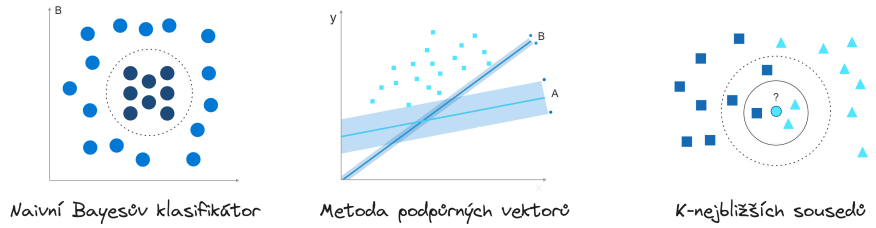
- **učení pod dohledem**, kde trénovací data mají označeny výsledky, které by z nich měli vzejít. Model se pak učí vzory, které vedou k jednotlivým výsledkům.
- **učení bez dohledu**, kde trénovací data nejsou nijak označena a model v nich hledá strukturu.
- **zpětnovazební učení**, kde se modely učí z výsledků tím, že obdrží zpětnou vazbu, zdali postupovali dobře či špatně.

Při učení modelu může nastat situace, kdy model ztratí schopnost úlohu zobecňovat a stane se příliš závislým na trénovacích datech. Tomuto jevu říkáme přetrénování modelu. Tento jev se projevuje tak, že dosahuje velmi nízké nebo dokonce nulové chyby na trénovacích datech, ale jeho výkon na nových datech je nepřiměřeně horší. Abychom tomuto problému předešli, používáme techniku rozdělení datové sady na trénovací, validační a testovací sadu. Na validační sadě experimentujeme s různými parametry modelu a vybíráme ty, které dosahují nejmenší chyby. Tím simuluje efektivitu modelu na nových datech. Pokud dosáhneme minimální chyby na validační sadě, můžeme předpokládat, že model bude úspěšný i při zpracování dalších nových dat. Testovací sada, oddělená od trénovací a validační sady, slouží k ověření konečné kvality modelu (Machine Learning College, 2020a).



Obrázek 2.12: Podtrénování a přetrénování modelu

Tradiční strojové učení stále spoléhá na manuální extrakci klíčových charakteristik z obrazu. Oproti procedurálním algoritmům je ale robustnější při analýze obrazu, kdy model můžeme trénovat na vzorcích s různými vadami. V této práci se zaměřujeme především na klasifikační algoritmy, které rozdělují obrázky do tříd. Mezi ně patří K-nejbližších sousedů, který klasifikuje obrázky na základě podobnosti klíčových charakteristik; naivní Bayesovy klasifikátory, které třídí obrázky na základě pravděpodobnosti, že mají rysy dané třídy; a metoda podpůrných vektorů, která odděluje rysy pomocí hyperroviny a klasifikuje data na základě toho, na které straně této roviny leží (Microsoft, 2024a).



Obrázek 2.13: Vybrané algoritmy strojového učení (Microsoft, 2024a)

2.4.3 Hluboké učení

Hluboké učení je podoblastí strojového učení, která se inspiruje strukturou lidského mozku a skládá se z více vrstvých neuronových sítí. Tyto sítě se učí automaticky z dat a dokáží zachytit složité vztahy mezi vstupy a výstupy. Díky tomu dosahují hluboké sítě mnohem lepších výsledků než tradiční algoritmy strojového učení (MathWorks, 2024b).

Hluboké neuronové sítě se skládají z mnoha vrstev propojených neuronů. Vrstvy mezi vstupní a výstupní vrstvou se nazývají skryté. Tyto sítě mohou být konstruovány různými architekturami, jako jsou konvoluční neuronové sítě (CNN) pro zpracování obrazových dat nebo rekurentní neuronové sítě (RNN) pro sekvenční data. Nás budou zajímat právě sítě konvoluční (MathWorks, 2024b).

Konvoluční neuronové sítě jsou složeny z několika základních vrstev. Kromě vstupní a výstupní vrsty zde ještě figurují konvoluční, aktivující (ReLU) a pooling vrstvy.

- **Konvoluční vrstvy** aplikují konvoluční filtry na vstupní obrázky, čímž se aktivují rysy
- **Aktivující vrstvy** mapují negativní hodnoty na nulu a udržují kladné hodnoty, efektivně tak odstraní neaktivované rysy
- **Pooling vrsty** snižují rozměry dat a zjednodušují výstupy

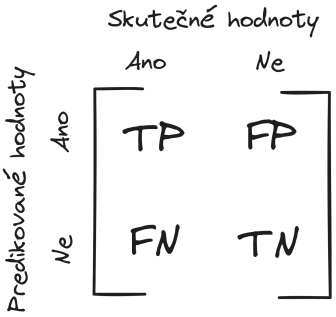
Tyto operace jsou opakovány přes desítky nebo stovky vrstev, přičemž každá vrstva se učí identifikovat různé rysy obrázku. V konvolučních sítích jsou váhy a bias sdíleny mezi všemi neurony v dané skryté vrstvě, což umožňuje síti detekovat stejné rysy v různých částech obrázku. Efektivně tak odpadá valná většina předzpracování obrazu, protože narozdíl od procedurálních algoritmů a tradičního strojového učení si konvoluční neuronové sítě umí klíčové charakterstiky z obrazu extrahovat bez zásahu člověka (MathWorks, 2024a).

Po natrénování konvoluční sítě se lze pustit do klasifikace. Předposlední vrstva je plně propojená vrstva, která vytváří vektor o K dimenzích (kde K je počet tříd, do kterých lze obraz zařadit) a obsahuje pravděpodobnosti pro každou třídu obrázku. Poslední vrstva používá výstup klasifikační vrstvy k poskytnutí konečného výsledku klasifikace (MathWorks, 2024a).

2.4.4 Evaluační metriky strojového učení

Evaluační metriky slouží k posouzení výkonnosti modelů strojového učení. Jinak řečeno, umožňují nám určit, jak dobře model plní svůj úkol. Základní metrikou je takzvaná ztráta (*loss*), neboli jak moc se predikce modelu liší od skutečných hodnot. Průběh ztráty v čase se nazývá ztrátová funkce.

Metriky pro klasifikační úlohu vycházejí z matice záměn. Matice záměň (*confusion matrix*) je matice, ve které jsou v jedné dimenzi skutečné třídy, a v druhé dimenzi jsou třídy predikované.



Obrázek 2.14: Matice záměn

Na hlavní diagonále matice záměn se nachází počet správně klasifikovaných datových bodů pro danou třídu (začínají písmenem T). Na vedlejší diagonále se nachází počty, které byly nesprávně klasifikovány do jiné třídy. V případě binárního klasifikátoru jsou tak na hlavní diagonále počty správně zařazených prvků, a na vedlejší diagonále počty špatně zařazených prvků. Z těchto hodnot můžeme dále spočítat další metriky, především nás bude zajímat správnost, přesnost, pokrytí a f-skóre.

- **Accuracy** (*správnost*) určuje, kolik hodnot klasifikátor správně zařadil. $ACC = \frac{TP+TN}{TP+TN+FP+FN}$

- **Precision** (*přesnost*) určuje, kolik pozitivních hodnot označných klasifikátorem je skutečně pozitivní. $PREC = \frac{TP}{TP+FP}$

- **Recall** (*pokrytí*) určuje, kolik pozitivních hodnot byl klasifikátor schopný odhalit. $REC = \frac{TP}{TP+FN}$

- **F1** (*f-skóre*) je harmonický průměr přenosti a pokrytí. $F1 = 2 \cdot \frac{PREC \cdot REC}{PREC+REC}$

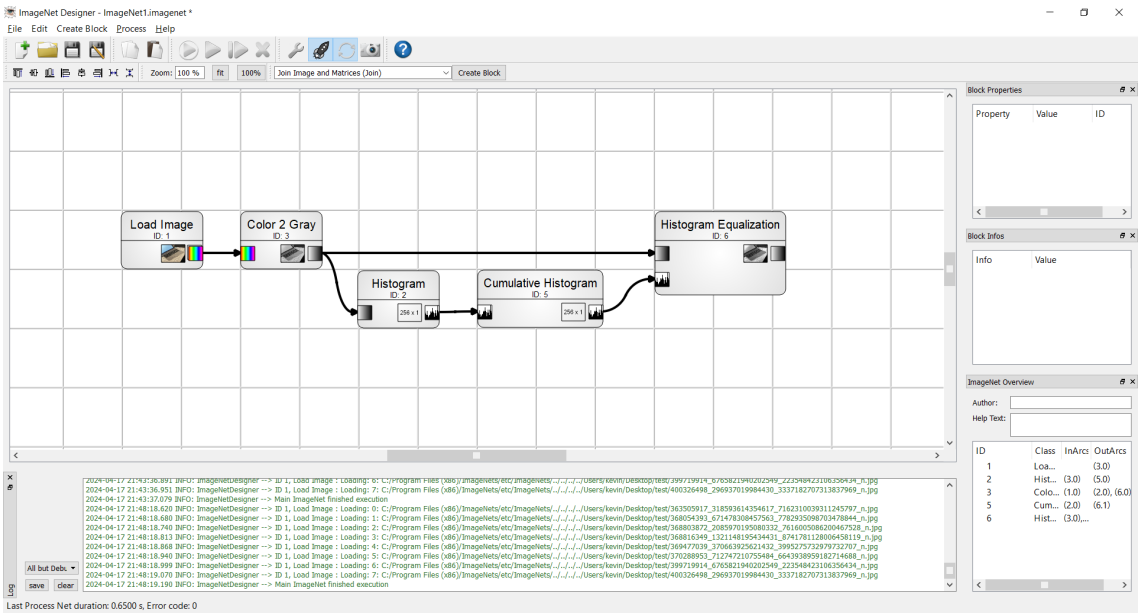
(Machine Learning College, 2020b).

2.5 NÁSTROJE PRO RAPIDNÍ VÝVOJ ALGORITMŮ STROJOVÉHO VIDĚNÍ

V této kapitole představím několik vybraných nástrojů dostupných na trhu, které umožňují vývoj algoritmů strojového vidění. Tyto nástroje byly zvažovány firmou CLOUDCODE před tím, než došli k závěru, že bude potřeba vyvinout vlastní řešení. Každý z těchto nástrojů má své výhody a nevýhody, které je třeba zvážit při rozhodování o nejvhodnějším přístupu k vývoji algoritmů strojového vidění.

2.5.1 ImageNet Designer

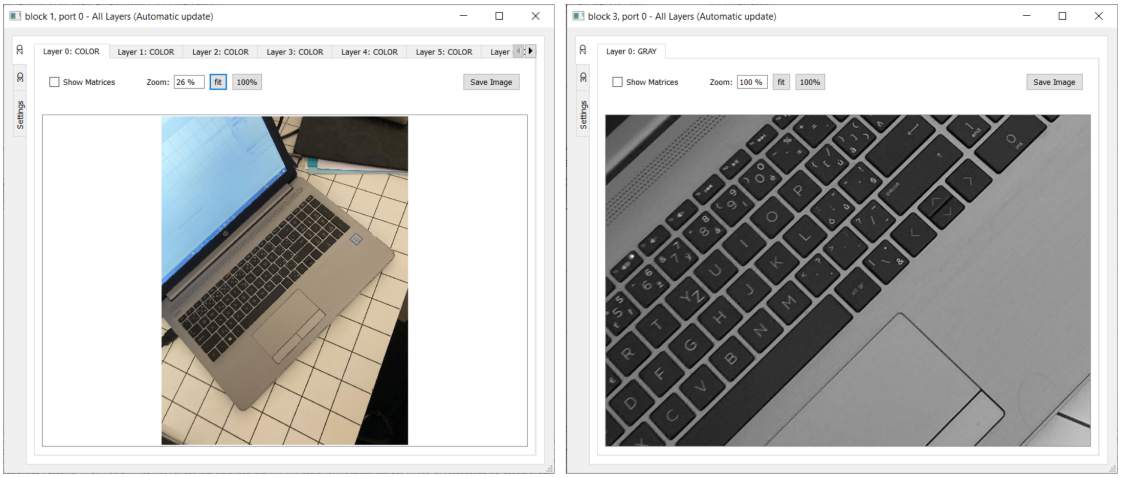
ImageNet Designer je grafické rozhraní pro rapidní prototypování kombinací funkcí strojového vidění. Vznikl v institutu automatizace německé univerzity v Brémách. Cílem bylo zkombinovat rychlost vývoje v Matlabu a rychlost vykonání instrukcí z OpenCV (Eldeep et al., 2012; Graeser, 2010).



Obrázek 2.15: ImageNet Designer

Program Imagenet Designer odstraňuje překážky spojené s vytvářením algoritmů tím, že nabízí grafické programování pomocí předdefinovaných bloků, které se skládají do orientovaného grafu. Tento program je také silně modularizovaný, což znamená, že selhání jednoho bloku nemá vliv na celkový chod programu. Uživatelé mohou přidávat vlastní bloky pomocí API v jazyce C/C++. Vytvořené algoritmy (ImageNety) jsou uloženy do souboru XML, který lze následně načíst a spustit pomocí C++ knihovny (Graeser, 2010).

Nejnovější verze programu ImageNet Designer (verze 2.3) byla vydána v roce 2012 a je volně dostupná ke stažení na platformě SourceForge. Po stažení jsem program úspěšně spustil a vyzkoušel. Na stránce programu na SourceForge je prezentován jako edukační nástroj, a dle mého názoru se tento charakter odráží i na jeho funkcích (Eldeep et al., 2012). Prvním větším problémem je, že není možné nahrát do jednoho ImageNetu více obrázků současně. Technicky je to možné, ale vícesnímkové nahrání není určeno pro zpracování více obrázků, ale pro bloky, které tento postup podporují, například kalibrační blok pomocí šachovnicové desky. ImageNet Designer vícesnímkový blok reprezentuje ve vrstvách. Na obrázku níže je vidět, jak se vrstvy vytratí v moment, co je blok nepodporuje (v tomto případě blok převodu na šedotónový obrázek).



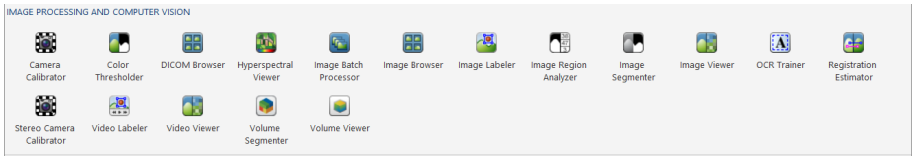
Obrázek 2.16: ImageNet Designer - Vrstvy

Každý ImageNet je jednoúčelový, což znamená, že pokud chci provádět více úloh na jednom snímku, musím buď vytvořit jeden velký ImageNet, který zahrnuje všechny potřebné operace, nebo vytvořit více menších ImageNetů a spustit je pomocí knihovny v C++. Dalším problémem je absence bloku pro vytváření anotací. I když by tento blok mohl být do programu doprogramován, existují další nedostatky, jako je nutnost znalosti operací OpenCV a základů zpracování obrazu, které z tohoto řešení činí nevhodnou volbu hlavně v případě, kdy chceme práci vývoje algoritmů delegovat na anotátory, kteří mají spíše doménové znalosti než znalosti o zpracování obrazu.

Posledním hřebíkem do rakve ImageNet Designeru je fakt, že neumí pracovat s modely strojového učení a neuronovými sítěmi. Tento bod jsem ani neočekával, vzhledem k tomu, že se jedná o edukační nástroj pro zpracování obrazu, který se zaměřuje primárně na procedurální algoritmy, ovšem je to bod, který je pro CLOUDCODE víc než důležitý.

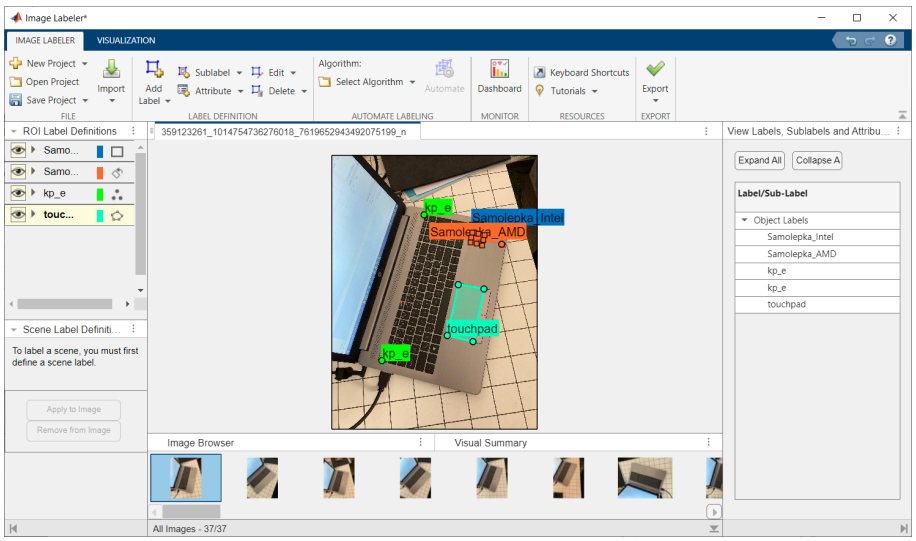
2.5.2 MATLAB

Matlab je prostředí pro technické výpočty a analýzu dat, často využívané pro svůj programovací jazyk (MathWorks, 2024c). Avšak není pouze o programování - nabízí širokou škálu rozšíření, tzv. Toolboxů, které rozšiřují jeho funkcionality. Tyto Toolboxy nejen přidávají nové funkce, ale také poskytují grafická rozhraní pro interakci s uživateli. Tímto způsobem může být Matlab okrajově užitečný i pro ty, kteří se nezabývají programováním. Budeme zkoumat pouze možnosti právě těchto grafických aplikací, protože jedním z kritérií je schopnost vytvářet algoritmy strojového vidění i uživateli bez programovacích dovedností.



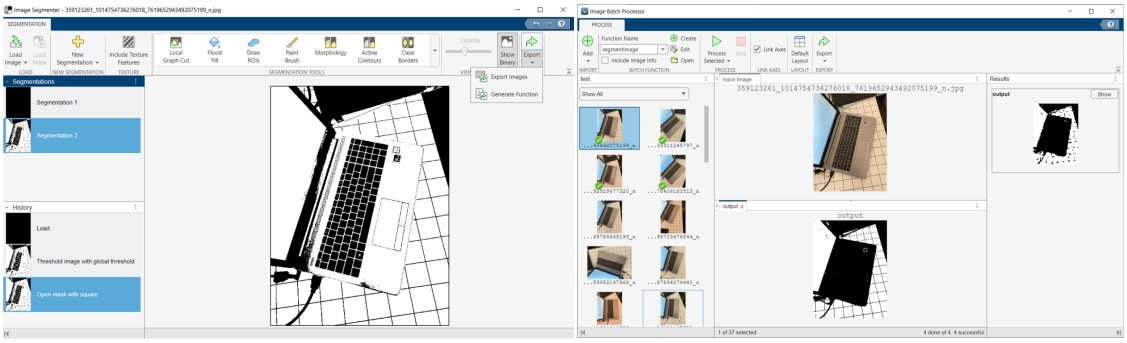
Obrázek 2.17: Matlab - Computer Vision Toolbox Aplikace

Když jsem poprvé otevřel aplikaci Image Labeler, zaujala mě možnost anotovat snímky v rámci projektů, které umožňují individuální anotování nebo spolupráci v týmu. Také mě trochu překvapilo, že jsem mohl nahrát více snímků najednou. Po otevření testovacího datasetu jsem anotoval první snímek. Proces přidávání anotací byl intuitivní a snadný. Poté jsem se pokusil exportovat anotace, a zjistil jsem, že MATLAB je ukládá do souborů MAT, které lze následně načíst v MATLAB skriptech. Pro práci v ekosystému MATLABu to není takový problém, ale soubory jsou binární, tudíž je nelze zkontrolovat v textovém editoru, nebo dokonce upravit mimo MATLAB.



Obrázek 2.18: Matlab - Image Labeler

Poté jsem zkoumal aplikaci Image Segmenter, která mi umožnila definovat několik úrovní segmentace pomocí různých operací předzpracování, jako je například prahování. Tato aplikace také umožňuje exportovat výslednou segmentaci do MATLAB funkce. Tento soubor lze následně využít v aplikaci Image Batch Processor, která umožňuje spustit skript na vybraných snímcích nebo celém datasetu.

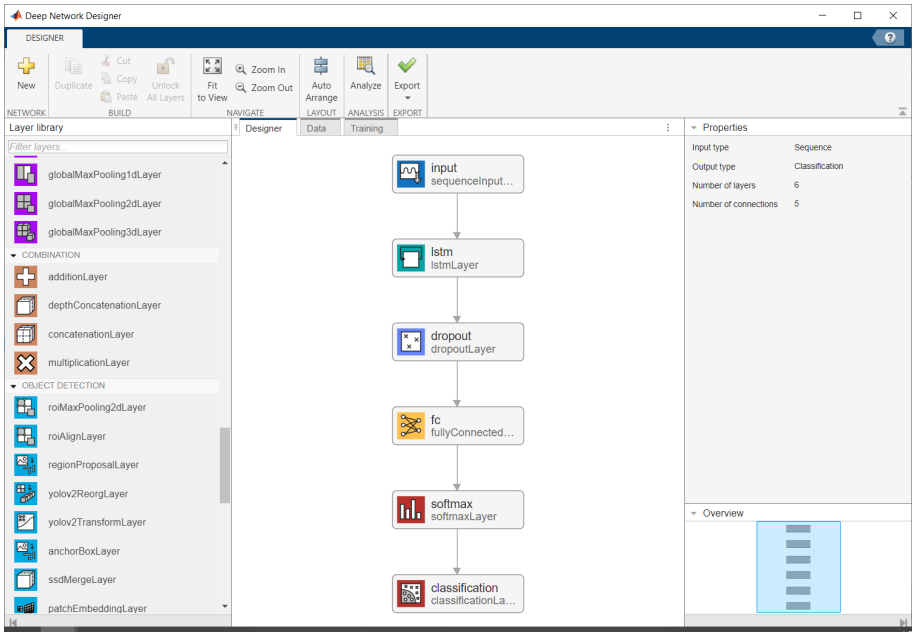


Obrázek 2.19: Matlab - Segmentace a dávkové zpracování

Zvláště aplikaci Image Segmenter považuji za nejpodstatnější součást celého Toolboxu, protože umožňuje i uživatelům bez znalosti programování v MATLABu spouštět předdefinované nástroje na celém datasetu. Pokud by se tak udržela disciplína v pojmenovávání nástrojů, může se jednat o opravdu mocnou aplikaci. Na rozdíl od ostatních nástrojů zde nejsme omezeni na procedurální algoritmy; spouštěný skript může například použít neuronovou síť k vyhodnocení snímku a vrátit obrázek s semanticky segmentovanými oblastmi.

Jedinou bariérou pro anotátora je, že MATLAB nemá žádnou možnost orchestrace těchto nástrojů zasebou mimo psaní skriptů. Jinými slovy, uživatel si buď musí pamatovat pořadí operací nebo někdo musí vytvořit skript, který tyto nástroje propojí. V takovém případě ztrácí použití MATLABu smysl, protože by anotátor měl mít k dispozici jednotlivé dílky a umět si je sám sestavit bez nutnosti opouštět grafické rozhraní.

Oproti ImageNet Designeru se zde již lze bavit o neuronových sítích. V MATLABu je Deep Learning Toolbox, který obsahuje nástroj Deep Network Designer. Tento nástroj umožňuje zkoumat, upravovat, diagnostikovat a trénovat hluboké neuronové sítě. V režimu úprav lze editovat sítě na úrovni jednotlivých vrstev pomocí orientovaného grafu, což je opravdu mocný nástroj. Nicméně z pohledu anotátorů je toto řešení nezajímavé a jedinou užitečnou funkcionalitou pro ně jsou méněcenné záložky s importem dat a trénováním.



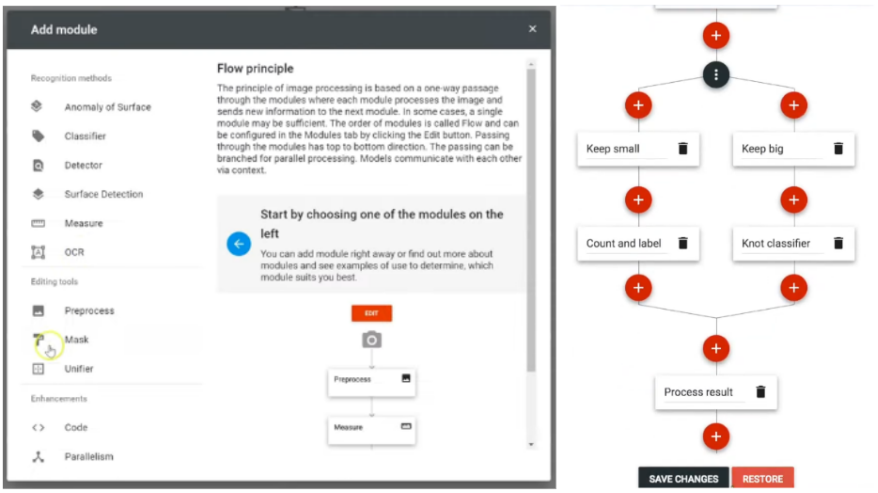
Obrázek 2.20: Matlab - Deep Network Designer

Realizace rapidního vývoje algoritmů strojového vidění v MATLABu je tedy teoreticky možná, pokud bychom věnovali čas vytvoření potřebných grafických uživatelských rozhraní a nástrojů v podobě MATLAB skriptů. Nicméně, když zvážíme všechny drobné nedostatky, připočítáme licenční poplatek ve výši 22 tisíc korun ročně a další poplatky za potřebné Toolboxy, a k tomu ještě přidáme skutečnost, že MATLAB není příliš efektivní s využitím systémových prostředků, stává se to zkrátka neekonomickým řešením. Navíc by MATLAB nemohl jednoduše spolupracovat s anotačními nástroji, které má CLOUDCODE vyvinuté, protože si své anotace ukládá v binárním formátu. Dále by se musela výsledná inspekce u zákazníka, která je napsaná v Pythonu, umět poradit s výstupy MATLABu. Všechno tohle by znamenalo investici do licencí a vývoje, který může být v důsledku bezvýsledný.

2.5.3 PEKAT VISION

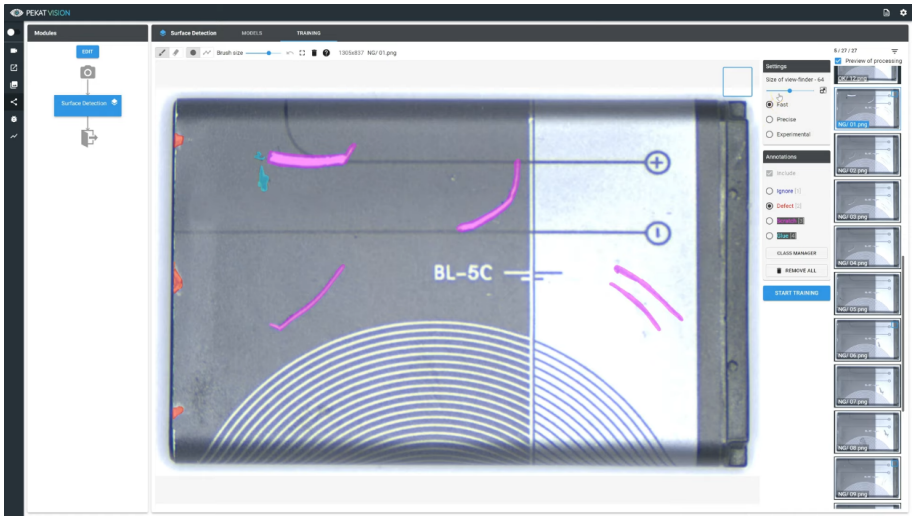
PEKAT VISION je software určený pro vizuální inspekci a kontrolu kvality, který se zaměřuje na průmyslové aplikace. Jeho hlavní síla spočívá ve využití strojového učení, konkrétně techniky nazývané *focused learning*. Tato metoda se snaží emulovat lidské oko a zaměřuje se na detaily v obrazech pomocí kombinace různých přístupů včetně hlubokého učení (PEKAT s.r.o., 2022).

Vývoj algoritmů strojového vidění v PEKAT VISION se řídí pomocí modulů, které se skládají do orientovaného grafu. Mezi tyto moduly patří detekce anomálií, detekce a klasifikace objektů, měření a vlastní bloky s kódem v programovacím jazyce Python (PEKAT s.r.o., 2024).



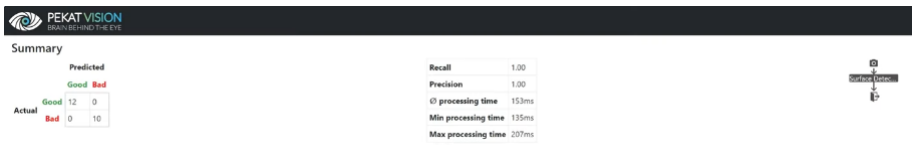
Obrázek 2.21: PEKAT VISION - Moduly (PEKAT s.r.o, 2021a)

PEKAT VISION je svojí funkcionalitou řešení, které je nejblíže tomu, který si pro své potřeby vyvinul CLOUDCODE. PEKAT datasetům říká projekty. V nich jsou uloženy snímky a algoritmus strojového vidění. Proces anotace snímků se skládá s přiřazení třídy a případného značení vad. Přiřadit třídu k jednotlivým snímkům lze buď postupným procházením, nebo hromadně dle řetězce v cestě souboru, přičemž se v takovém případě využívají regulární výrazy. Pro označení vad je k dispozici jednoduchý režim kreslení, který umožňuje označit pixely štětcem nebo nakreslit polygon odpovídající dané třídě. V CLOUDCODE se značení snímků liší. V PEKAT VISION anotátor značí snímky pro každý modul zvlášť. V CLOUDCODE Labelingu anotátor značí všechno do jednoho snímku, přičemž v případě předzpracování obrazu geometrickými transformaci má anotátor k dispozici ke značení jak původní, tak zpracovaný obraz v jednom snímku.



Obrázek 2.22: PEKAT VISION - Labeling (PEKAT s.r.o, 2021b)

Po označení vad a přiřazení tříd může anotátor dát daný modul trénovat. Pro různé moduly probíhá trénování různě. Buďto se ukončí automaticky, nebo ho musí zastavit uživatel na základě vlastního úsudku ze sledování průběhu ztrátové funkce, která je v rozhraní vykreslována. Po natrénování modelu se u daného modulu ukáže v záložce *models*. Validace modelu probíhá pomocí vyhodnocovacích kritérií, které anotátor nastaví. Těmi určí, kdy je snímek označený jako vadný a kdy ne. PEKAT VISION umožňuje vygenerovat zprávu o výsledcích modelu, ve které lze nalézt bližší statistiky jako matici záměň nebo průměrnou dobu pro vyhodnocení snímku (PEKAT s.r.o, 2021b).



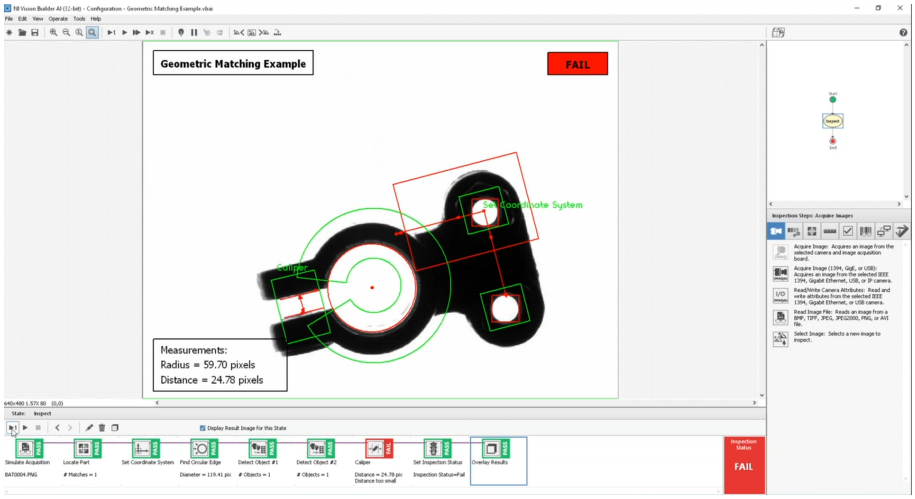
Obrázek 2.23: PEKAT VISION - Statistiky modelu (PEKAT s.r.o, 2021b)

Z hlediska anotátora v CLOUDCODE je PEKAT VISION software, který by mu vyhovoval až na pár drobností. Proč tedy CLOUDCODE nepoužívá PEKAT VISION? Jedná se spíše o technické problémy. Během testování jsme zjistili, že nedostatky v grafickém rozhraní, jako je chybějící možnost filtrování snímků, když se predikce neshoduje s anotací, jsou zanedbatelné ve srovnání s neschopností trénovat jeden model z více datasetů (projektů). CLOUDCODE se ocitá v situaci, kdy má pro jednoho zákazníka přes 20 datasetů, a ne každý potřebuje trénovat vlastní zarovnání. Jinými slovy, modely jsou používány a trénovány napříč datasety. Na druhou stranu nemůže všechno existovat v jednom datasetu, protože různé datové sady mají své vlastní inspekce a anotace.

PEKAT VISION je bezesporu skvělým nástrojem pro rapidní vývoj algoritmů strojového vidění. Bohužel však nevyhovuje specifickým potřebám firmy CLOUDCODE. Začínaje drobnými nedostatky v grafickém rozhraní, přes nedostatečnou dokumentaci a chybějící funkcionalitu, PEKAT VISION není schopen uspokojit všechny požadavky a potřeby firmy CLOUDCODE a jejich zákazníků. Navíc je třeba brát v úvahu i licenční poplatek, který by zvýšil náklady na využití tohoto nástroje.

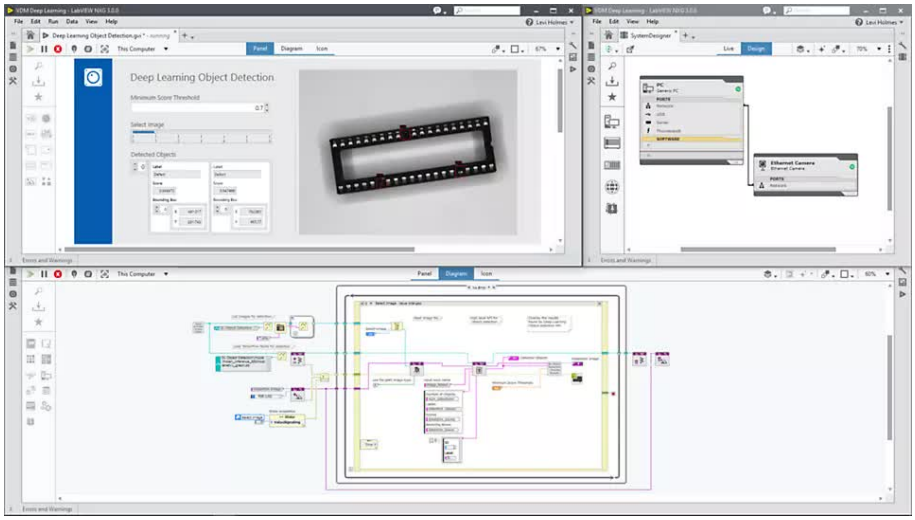
2.5.4 Vision Builder AI

Vision Builder AI od společnosti National Instruments je softwarový nástroj pro vývoj a nasazení systémů strojového vidění pro automatizovanou kontrolu. Umožňuje uživatelům bez znalostí programování konfigurovat kamery, analyzovat obrazy a generovat výsledky pro jednotlivé inspekce a jejich stavy (National Instruments Corp., 2024b).



Obrázek 2.24: Vision Builder AI (National Instruments Corp., 2024b)

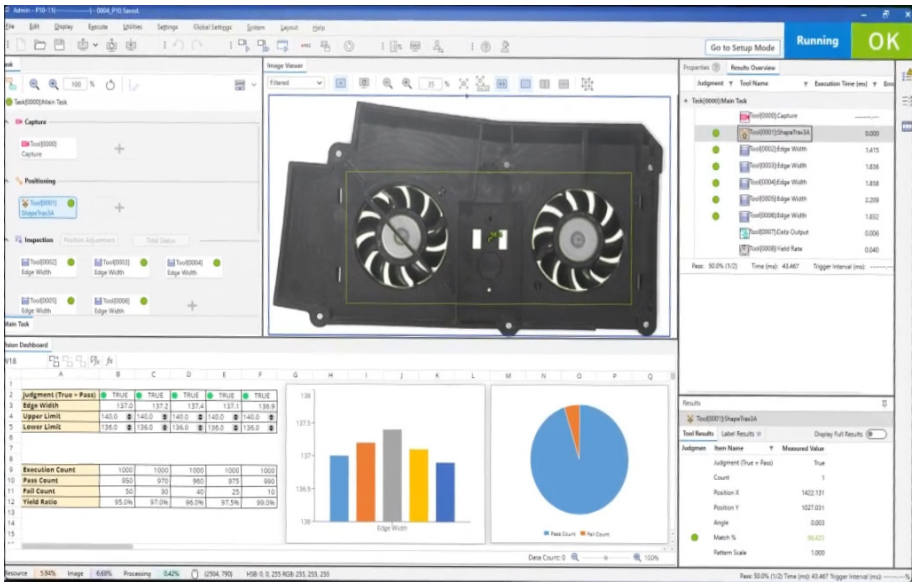
Toto řešení bylo, stejně jako ImageNet Designer, rychle v úvahách zahozeno, protože neřeší trénování modelů. Jeho primární funkce spočívá v konfiguraci kamer, kroků analýzy obrazu a parametrů inspekce bez nutnosti programování. Nabízí širokou škálu nástrojů pro analýzu obrazu, včetně detekce objektů, klasifikace, segmentace a měření. Ale trénování neuronových sítí ze snímků nenabízí. Narozdíl od ImageNet Designeru se umí Vision Builder integrovat s externími nástroji pro trénování neuronových sítí. Je tak možné využít externí nástroje, například TensorFlow, k trénování vlastních neuronových sítí a následně je nasadit do inspekce (National Instruments Corp., 2024a).



Obrázek 2.25: Vision Builder AI - Modul pro Deep Learning (National Instruments Corp., 2024a)

2.5.5 KEYENCE Vision System Creator

KEYENCE Vision System Creator je softwarový nástroj navržen speciálně pro rychlý vývoj algoritmů strojového vidění pro kamery řady VS od té samé společnosti. Tento software poskytuje uživatelům prostředí pro konfiguraci a nastavení kamer, analýzu obrazů a tvorbu inspekčních algoritmů. Jeho předností je rychlost nasazení celkového řešení pro vizuální inspekci. Uživatel tak nemusí řešit složitou kalibraci kamer či komunikaci s řídícím programem, ale řeší pouze příslušenství pro kameru a samotné programování inspekce (KEYENCE Corporation, 2024).



Obrázek 2.26: KEYENCE VS Creator
(KEYENCE Corporation, 2024)

Vision System Creator jednotlivé algoritmy strojového vidění nazývá úlohami (task). Tyto úlohy dále rozděluje do čtyř kroků:

- **Zachycení** (*Capture*), ve kterém lze například zapnout světla pro lepší osvícení zkoumaného povrchu.
- **Pozicování** (*Positioning*), ve kterém lze produkt například zarovnat pomocí referenčních bodů.
- **Inspekce** (*Inspection*), ve kterém se provádí samotná inspekce povrchu a vyhodnocení, zdali je kus vadný nebo nikoliv.
- **Komunikace** (*Communication*), ve kterém se výsledky inspekce posílají do vizualizací či externích systémů.

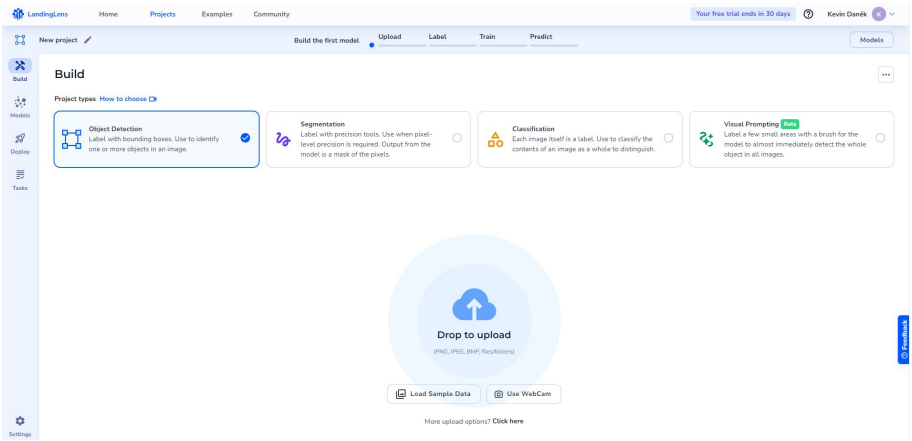
Tyto kroky jsou součástí úlohy. Úloha reprezentuje konkrétní algoritmus strojového vidění vytvořený uživatelem. Pro každý s těchto kroků nabízí

nástroje, které je možné do detailu nastavit. Tyto nástroje jsou z většiny založené na procedurálních algoritmech, ale obsahuje i nástroje, které jsou založeny na modelech umělé inteligence. Jedná se tak zejména o detekci vad nebo třídění (klasifikace) výrobků (KEYENCE Corporation, 2024).

Ačkoliv je Vision System Creator uživatelsky velmi přívětivý a je vhodný i pro pokročilé uživatele, tak podstatnou překážkou při používání Vision System Creator je jeho závislost na kamerách od společnosti KEYENCE. Tyto kamery jsou sice kvalitním hardwarem, což jsem si mohl ověřit při testování kamery VS-L500CX spolu se světlem CA-DEW10X, avšak stále se jedná o uzavřený ekosystém. Kamery od KEYENCE nabízejí kompletní řešení s integrovanou výpočetní jednotkou přímo v kameře, což pro některé zákazníky může být výhodou. Nicméně pro potřeby v CLOUDCODE, kde se používají kamery od společnosti BASLER, tato závislost znamená, že Vision System Creator není možné využít.

2.5.6 LandingLens

LandingLens je platforma od společnosti Landing AI, která si klade za cíl zpřístupnit vytváření modelů strojového vidění těm, kteří nemají žádné znalosti v oblasti umělé inteligence. Nabízí intuitivní rozhraní a nástroje pro rychlý iterativní vývoj a nasazování modelů (Landing AI, 2024).

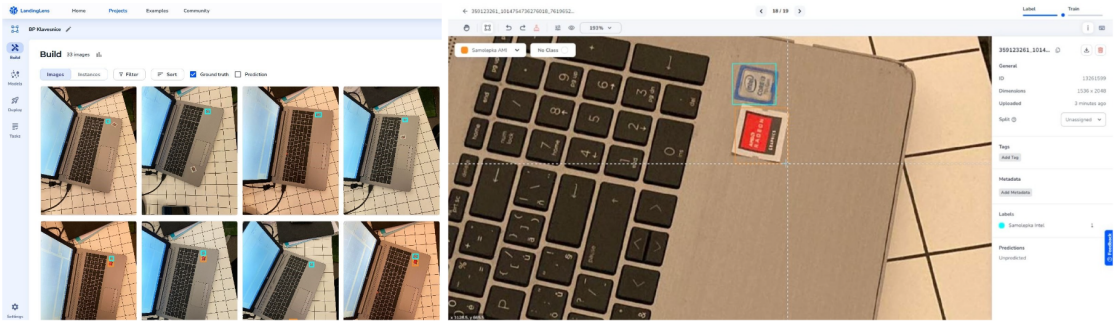


Obrázek 2.27: LandingLens - Tvorba projektu

Ve spolupráci se společností CLOUDCODE jsme se rozhodli podrobněji prozkoumat toto řešení, zejména kvůli jeho údajné jednoduchosti v procesu trénování. Pro účely testování jsem založil projekt zaměřený na detekci objektu, konkrétně samolepky, podobně jako u předchozích scénářů. Při zakládání projektu jsou do projektu nahrány první obrázky, které jsou pak zobrazeny v mřížce. Zde je možné filtrovat neoznačené a nepredikované obrázky, a také je třídit podle jednotlivých tříd. Po nahrání jsem přešel k anotaci. Protože se jedná o projekt na detekci objektů, tak jednotlivé třídy reprezentují objekty zájmu v obraze. Pro trénování modelu je tak

nutné označit jejich polohu pomocí ohraničujícího obdélníku, známého jako *bounding box*.

Anotační nástroje v LandingLens jsou jednoduché, ale fungují. Spíše než na rozsáhlé funkcionality se vývojáři zaměřili na vytvoření dobrého uživatelského prožitku (UX), který je až na pár drobností skvělý. LandingLens je prezentován jako datově orientovaný nástroj. To znamená, že místo aby se zaměřoval na úpravu parametrů modelu, se snaží zlepšovat kvalitu dat, ze kterých se model trénuje. K dosažení tohoto cíle využívá několik strategií, včetně funkce *Label Book*, která umožňuje zadavatelům objasnit definice jednotlivých tříd pro anotátory a zajistit tak jasnost ohledně toho, co by mělo být v dané třídě označeno. Další strategií je dosažení shody mezi anotacemi různých anotátorů ohledně umístění anotací, což LandingLens dosahuje aktivním vyhledáváním konsensu mezi všemi anotacemi (Landing AI, 2024).

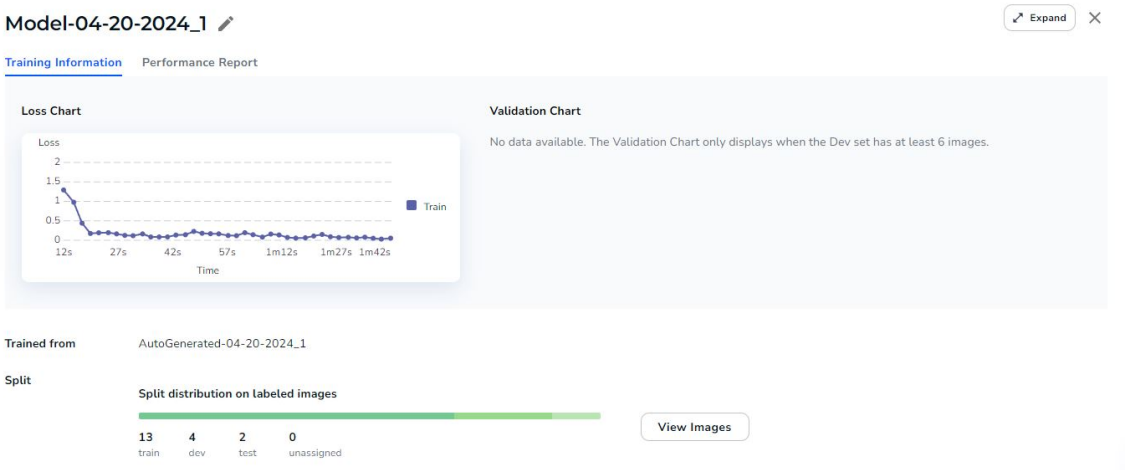


Obrázek 2.28: LandingLens - Dataset a Labeling

Po dokončení anotace prvních sedmnácti snímků jsem se přesunul k trénování první verze modelu. Existují dvě možnosti spuštění tréninku: kliknutím na tlačítko *Train*, což spustí trénink s výchozími hodnotami, nebo volbou vlastního tréninku (*Custom Training*), která umožňuje upravit tyto hodnoty. Při vlastním tréninku lze specifikovat poměr rozdělení dat mezi trénovací, validační a testovací sadu. Dále je možné nastavit parametry pro transformaci obrazu, jako je například škálování, a také augmentace, což jsou modifikované verze obrázků, které jsou přidány do datasetu pro zvýšení robustnosti modelu vůči výkyvům v osvětlení nebo natočení obrazu.

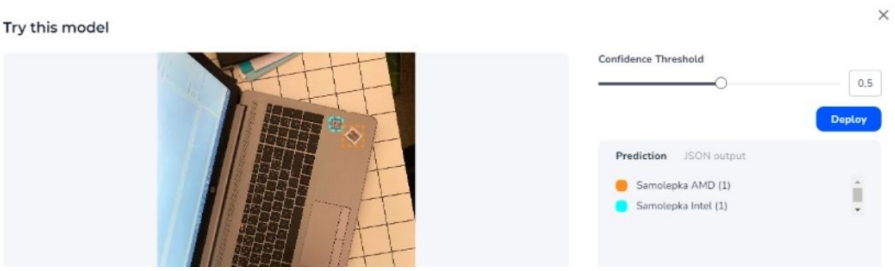
Obrázek 2.29: LandingLens - Parametry rozdělení dat

Po rozdělení dat a spuštění trénování je možné v bočním panelu sledovat průběh trénování, který zobrazuje vývoj ztrátové funkce. Trénování lze buďto přerušit předčasně, nebo nechat doběhnout do předem nastavených čtyřiceti epoch. Po dokončení trénování se statistiky nejnovější verze modelu zobrazí v bočním panelu, a je možné prohlédnout si také statistiky předchozích verzí v záložce *models*.



Obrázek 2.30: LandingLens - Statistiky modelu

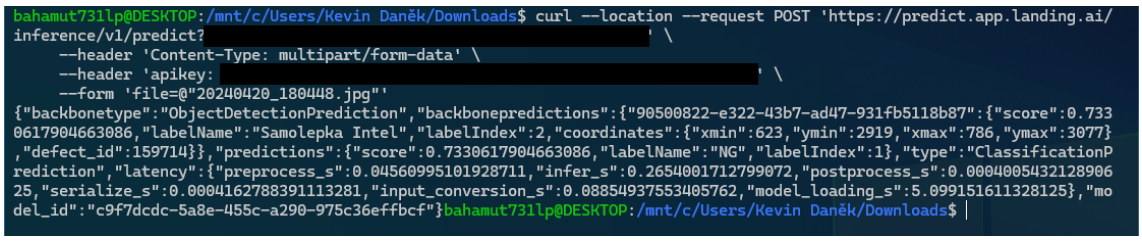
Natrénovaný model je možné okamžitě otestovat přímo v uživatelském rozhraní. Po dokončení trénování se v bočním panelu vedle statistik zobrazí tlačítka *Deploy* pro nasazení modelu a *Predict*, které umožňuje nahrát obrázky pro predikci. Přešel jsem tedy k predikci a nahrál jsem obrázky, na nichž nebyl model trénován. Zatímco modrá samolepka označující značku Intel a červená samolepka AMD Radeon byly správně identifikovány, model omylem kategorizoval třetí druh samolepky, která obsahovala logo AMD Ryzen, jako samolepku Radeon. Tímto se potvrdilo, že je nutné přidat další snímky do datasetu a natrénovat novou verzi modelu.



Obrázek 2.31: LandingLens - Chybná predikce

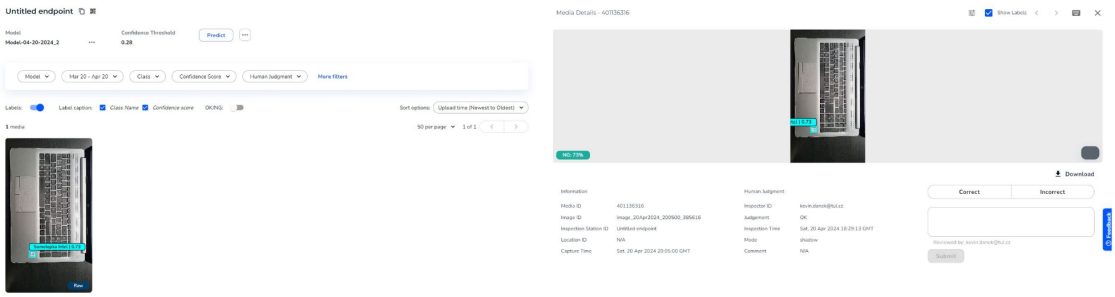
Po doplnění nových snímků, jejich anotaci a následném opětovném trénování modelu jsem připravil fotografii, kterou model ještě neviděl a která byla pořízena za jiných světelných podmínek. Pro predikci jsem se rozhodl

využít poskytované webové API. Tato platforma umožňuje nasadit model buď na koncový bod webového API v Cloudu, nebo lokálně pomocí Docker obrazu. Obě varianty využívají webového API, avšak lokální verze je vhodná v situacích, kde není možná komunikace směrem ven. Po zadání API klíče a vyplnění parametrů dotazu jsem pomocí cURL příkazu odeslal požadavek na koncový bod. Po chvíli čekání jsem obdržel odpověď ve formátu JSON.



Obrázek 2.32: LandingLens - Predikce pomocí volání webového API

Odeslaný obrázek lze nalézt i na stránce koncového bodu, na které lze kromě nasazeného modelu lze nalézt i obrázky, které lze takto vyhodnotit. Zde se taky nachází funkcionalita, která je pro CLOUDCODE obzvlášť důležitá, a to je možnost jednotlivé snímky lidským anotátorem zkontrolovat, zhodnotit, a tyto snímky v případě potřeby stáhnout pomocí pokročilých filtrů.



Obrázek 2.33: LandingLens - Endpoint a Inspekce obrázku

LandingLens mě příjemně překvapil a nabídl nečekaně pohodlný způsob vývoje modelů strojového vidění oproti konkurenci. To vede k otázce, proč není využíván v CLOUDCODE? Jednoduchost a zaměření na kvalitu dat, která jsou hlavními přednostmi LandingLens, se stávají zároveň jeho slabým místem. I když se platforma zaměřuje na kvalitu dat, existují situace, kdy je potřeba upravit konkrétní parametry modelu, což zatím LandingLens neposkytuje.

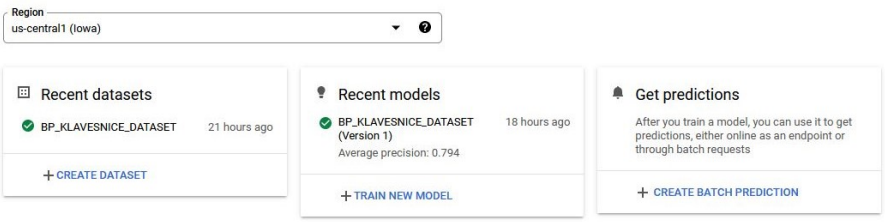
Dále chybí možnost spouštět predikci více modelů na jednom snímku. To by šlo vyřešit tak, že by se nasadil každý model na samostatný koncový bod a následně by se tyto koncové body volaly ze softwarového řešení pro řízení inspekce na výrobní lince. To přináší svá vlastní úskalí v podobě latence vyvolané navazováním TCP spojením a možnou bezpečností

politikou zákazníka, která by přístup na koncové body v cloudu zatrhla. Ač LandingLens nabízí nasazení lokálně pomocí Docker obrazů, pořád se platí daň v podobě zdrojů, které potřebuje Docker Engine.

Kromě toho není možné výsledné modely z LandingLens extrahovat ve formátu, který by byl spustitelný v jiných běhových prostředích, například onnxruntime. Také se nabízí otázka, jak by platforma zvládla dataset s tisíci snímky, což je běžná situace ve společnosti jako CLOUDCODE. Sečteno podtrženo, LandingLens by byl skvělou volbou, pokud by CLOUDCODE teprve začínal a počítalo se s ním již od začátku, ale integrovat ho do již existujícího procesu je obtížnější, než se zdá, nehledě na přidané náklady.

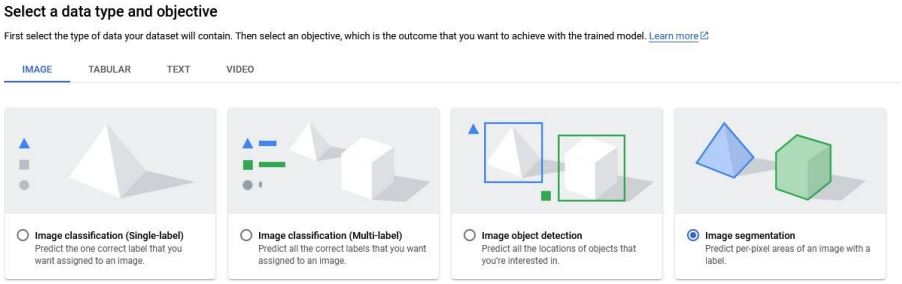
2.5.7 Vertex AI

Vertex AI je platforma od společnosti Google, která je součástí balíčku Google Cloud a umožňuje trénovat a nasazovat modely strojového učení a umělé inteligence. Vertex AI sdružuje pod jednou střechou všechny nástroje potřebné pro kompletní životní cyklus strojového učení, jako správu dat, trénování modelů, nasazení a sledování výkonu, čímž zjednodušuje a zefektivňuje celý proces. (Google, 2024).



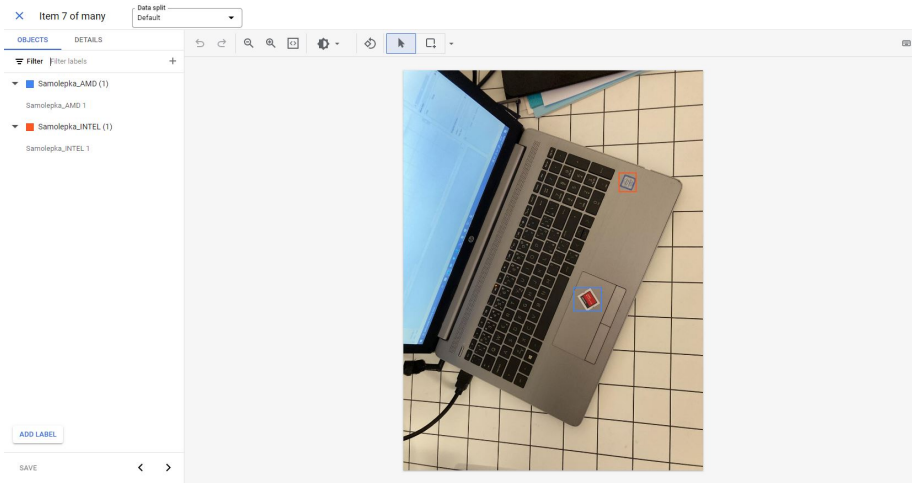
Obrázek 2.34: Vertex AI - Dashboard

Prvním krokem v práci s Vertex AI je vytvoření datasetu, který slouží jako základní úložiště pro data. Každý dataset má určený datový typ - může se jednat o obrazová, textová, tabulková nebo video data. Dataset je dále rozdělen do množin anotací, které se od sebe liší podle specifické úlohy, kterou mají řešit. Při vytváření datasetu je vždy vytvořena jedna sada anotací. Pro účely trénování modelů je nezbytné, aby dataset byl uložen v Google Cloud Storage, kde jsou fotografie určené k trénování nahrané.



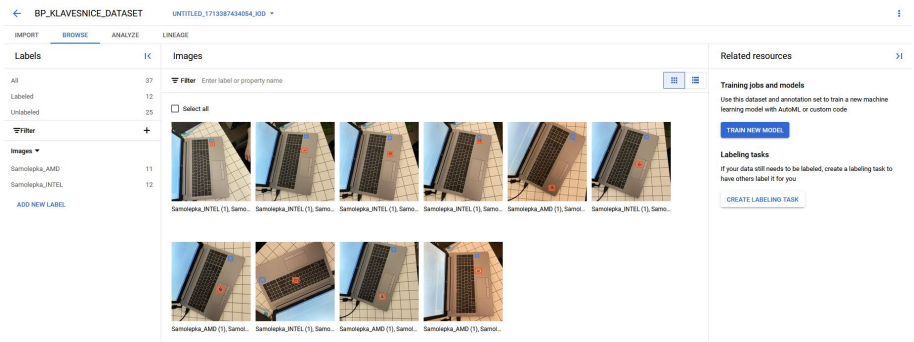
Obrázek 2.35: Vertex AI - Výběr úlohy strojového vidění

Po nahrání dat do Vertex AI může anotátor zahájit proces označování. Způsob anotace se liší podle typu úlohy, kterou má množina anotací řešit. Nejprve se definují typy tříd, do kterých se následně anotují prvky obrázku. Nástroje pro anotaci ve Vertex AI jsou relativně základní. CLOUDCODE naproti tomu disponuje pokročilejšími nástroji, které nabízí větší komfort a flexibilitu při práci s daty.



Obrázek 2.36: Vertex AI - Anotace snímku

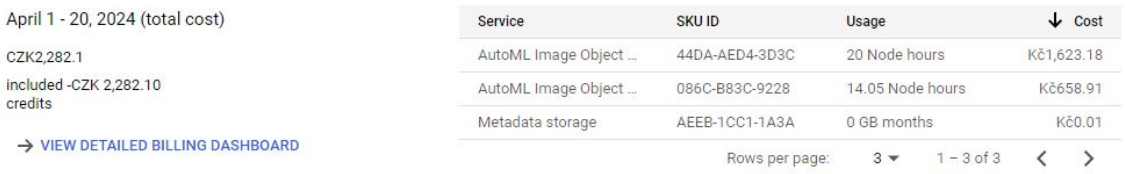
Po označení fotek jsem dal trénovat nový model. Vertex AI nabízí trénovat model buďto pomocí vlastního trénovacího skriptu, který se do Vertex AI nahraje, nebo pomocí AutoML, který nevyžaduje žádnou znalost procesu trénování. Zvolil jsem tedy tuto možnost a alokoval jsem minimální počet hodin, který byl možný, tudíž dvacet. Trénování trvalo dvě hodiny a čtyřicet tři minut a jak na jeho začátek, tak jeho konec jsem byl upozorněn e-mailem, což například LandingLens nedělá.



Obrázek 2.37: Vertex AI - Dataset

Po dokončení trénování jsem se podíval na statistiky modelu, ovšem více mě zaujala jiná statistika, a to byla cena za trénování. Za natrénování modelu jsem utratil 1 623 korun v kreditech. Vertex AI sice chce šetřit čas a zdroje

díky automatizaci a snadnému použití, ale jeho provoz může být pro menší organizace s omezeným rozpočtem nákladný. Cena se odvíjí od množství využitých výpočetních zdrojů, úložiště a dalších služeb Google Cloud Platform. Může se stát, že úspory z automatizace celého procesu budou překryty náklady na cloudové prostředky. Pokud bych si naprogramoval vlastní model, respektive použil vlastní kód pro trénování modelu a nahrál ho na Google Cloud, pravděpodobně by to bylo levnější, ale zabralo by mi to více času



Obrázek 2.38: Vertex AI - Cena za trénování AutoML modelu

Následně jsem se pokusil provést predikci s obrázkem, ale na rozdíl od LandingLens mi to nešlo. Vertex AI nabízí dvě možnosti pro predikci: buďto dávkovou predikci souborů uložených v Google Cloud Storage, kde je třeba mít k dispozici textový soubor s cestami k obrázkům, které chceme predikovat, nebo lze model nasadit na koncový bod webového API. Nicméně ani jedna z těchto možností mi nefungovala.

Vertex AI běží na cloudové infrastruktuře Google, což znamená, že uživatelé nemají plnou kontrolu nad hardwarovými prostředky. Dohody o mlčenlivosti a přísná bezpečnostní politika nejsou v automobilovém průmyslu nezvyklé, proto se zde jedná nejenom o bezpečnostní riziko, ale za prostor s daty se musí samozřejmě platit. Lze pak diskutovat, zdali je to o tolik výhodnější než spravovat vlastní řešení s robustní zálohovací politikou.

Posledním důvodem proti použití Vertex AI je, že CLOUDCODE nechce řešení pro kompletní správu procesu učení a nasazování modelů. Tyto nástroje a techniky již má zavedené a hledá pouze řešení, které umožní anotátorům bez technických znalostí trénovat modely. I když Vertex AI tuto funkcionalitu obsahuje, široká škála dalších možností, které CLOUDCODE nevyužívá, pro něj představuje zbytečné náklady a komplikuje proces.

3 REŠENÍ

V této části práce se zaměřím na hledání odpovědí. S firmou CLOUDCODE s.r.o. jsme prozkoumali, jaké překážky jim brání v používání nástrojů pro vývoj algoritmů strojového vidění, které jsou momentálně na trhu a jak by si představovali jejich řešení.

Cílem je najít způsob, jak sjednotit procedurální algoritmy a strojové učení do jednoho konceptu, který není závislý na rozlišování mezi tradičními algoritmy a neuronovými sítěmi, dovoluje jednotlivé algoritmy iterativně vylepšovat a testovat, a jehož obsluhu by měl zvládnout i pracovník, který prošel opravdu základním školením. Tímto způsobem bude moci firma rychleji dosáhnout lepších výsledků a získat tak konkurenčí výhodu.

3.1 KLÍČOVÉ PROBLÉMY

Po provedení rešerše programů a konzultaci s firmou CLOUDCODE s.r.o. jsem identifikoval několik klíčových problémů, respektive oblastí, které chtějí využít a které tuto firmu v ostatních řešeních trápí:

- Použití jednoho algoritmu na více oblastí zájmu
- Použití více algoritmů na jednom snímku
- Práce s neuronovými sítěmi (trénování, predikce, evaluace metrik)
- Práce s (proměnlivou) množinou snímků

Tyto problémy plynou hlavně ze skutečnosti, že se CLOUDCODE zaměřuje na náročnější aplikace strojového vidění, ve kterých je běžné kontrolovat desítky oblastí zájmu na jednom snímku. Firma to doposud řešila tak, že definici spouštění modelů v produkci manuálně definoval programátor, který měl projekt na starosti.

3.2 POŽADAVKY

V této kapitole představím specifikaci očekávaného řešení. Specifikace se inspiruje ze standardu IEEE 830 a jeho pozdějších revizích[1].

3.2.1 Rozsah

Cílem řešení je vytvořit rozhraní, které umožní definovat sadu nástrojů, které budou mít stejné rozhraní, ať už se jedná o algoritmus či neuronovou síť. Všechny nástroje bude možné z grafického rozhraní trénovat, vyhodnotit a poměřit na specifických metrikách.

Vývojářům je klíčové poskytnout možnost definovat tyto nástroje tak, aby nemuseli zasahovat do kódu uživatelského rozhraní. Zároveň je potřeba nabídnout jim programové rozhraní, do kterého budou moci implementovat jednotlivé metody, jako je trénování, vyhodnocování a měření. Rozhraní musí být ustálené napříč nástroji, aby bylo možné využít polymorfismu.

Anotátorům je třeba poskytnout grafické rozhraní, které umožní vytvářet a upravovat nástroje, nastavovat jejich parametry a následně provádět trénink, vyhodnocování a sběr metrik. Z minulosti jsou zvyklí využívat záložku tabulek v systému Labeling pro zobrazení výsledků měření, čehož by bylo vhodné využít.

Pro servisní personál je potřeba, aby byla data z řešení přenositelná i mimo Labeling a dala se využít pro následné nasazení do kontrolního procesu. Jinak řečeno, aby bylo možné definovat sekvenci predikcí a tu včetně konfigurací přenést k zákazníkovi.

3.2.2 Perspektiva řešení

Řešení by mělo navázat na dřívější snahu o vývoj podobného projektu nazvaného EWQ. Ten se zaměřoval zejména na vývoj procedurálních algoritmů strojového vidění v obdobném duchu jako ImageNet Designer. Projekt EWQ byl opuštěn v důsledku změny priorit a přesunu lidských zdrojů směrem k nástrojům pro neuronové sítě. Přestože nedosáhl úspěchu, poskytl cenné zkušenosti, které lze využít při tvorbě nového řešení.

Řešení musí být součástí systému CCM *(CLOUDCODE Manager)*, ve kterém je nástroj Labeling. CCM a Labeling poskytují webová a socketová API, která může řešení využít pro práci s daty ze systému Labeling, jako jsou datasety, obrázky či anotace. Je vhodné řešení zasadit právě do grafického rozhraní v systému Labeling.

[1] IEEE 830 je dokument od mezinárodní organizace IEEE definující standardy pro vytváření dokumentů specifikace softwarových požadavků(“IEEE Recommended Practice for Software Requirements Specifications”, 1998)

3.2.3 Charakteristika uživatele

Vývojáři mají plný přístup k celému zdrojovému kódu a jsou schopni se v něm orientovat. Jsou zodpovědní za implementaci nových funkcionalit a úpravy existujícího kódu. Pro ně je důležité mít možnost manipulovat s vnitřními mechanismy systému a provádět pokročilé úpravy v kódu.

Anotátoři nemají přístup k zdrojovému kódu a spoléhají pouze na grafické uživatelské rozhraní. Jejich hlavním úkolem je anotace dat a používání těchto nástrojů v jednotlivých iteracích jejich trénování. Pro ně je klíčové, aby grafické rozhraní bylo intuitivní, jednoduché a hlavně aby dostatečně komunikovalo aktuální stav věcí, například stav tréninku.

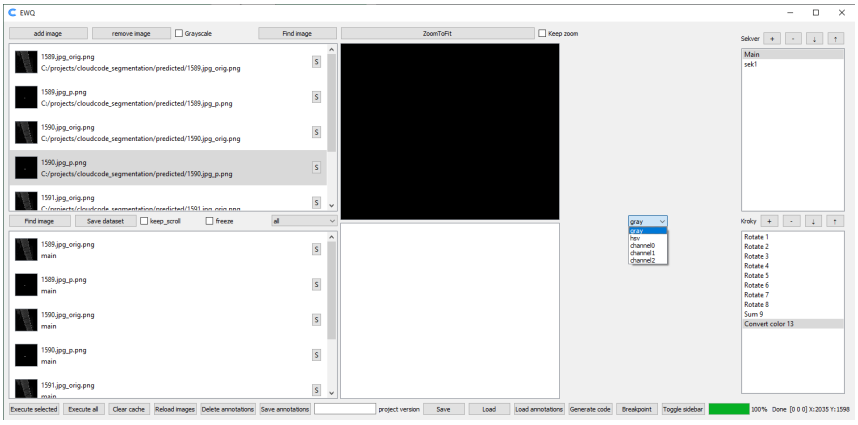
Servisní technici mají sice přístup ke zdrojovému kódu, ale mezi nimi mohou být lidé, kteří nejsou konkrétním detailům znalý, takže by jeho průchod zabral více času než metoda *pokus omyl*. Jejich práce má vizuální zpětnou vazbu ve formě logů v konzoli a vizuálního ověření v praxi na výrobní lince.

3.2.4 Rizika a omezení

Existuje několik rizik spojených s integrací řešení do systému CCM a jeho závislostí na jeho rozhraní. Jedním z hlavních rizik je, že se řešení stává závislým na CCM a jeho funkčnosti, což může vést k omezení flexibility a obtížnější údržbě. Dalším potenciálním rizikem je nedostatečná správa závislostí nebo nekompatibilní hardware, což může narušit provoz a spolehlivost systému při trénování modelů neuronových sítí. Důležité je také brát v úvahu možná budoucí rozšíření rozhraní nástrojů, což může vyžadovat úpravy v celém systému, což může být časově náročné a nákladné.

3.3 PŘEDCHOZÍ GENERACE

Tato práce není rozhodně první snahou v CLOUDCODE o vytvoření programu pro rapidní vývoj algoritmů strojového vidění. V minulosti, ještě před mým příchodem, byl vyvinut software s názvem **EWQ**, který byl napsán v Pythonu za pomocí knihovny PySide. Ta poskytuje vazby na knihovnu *Qt*, která slouží k vytváření multi-platformích grafických aplikací (Qt Group, 2023; Qt wiki, 2022).

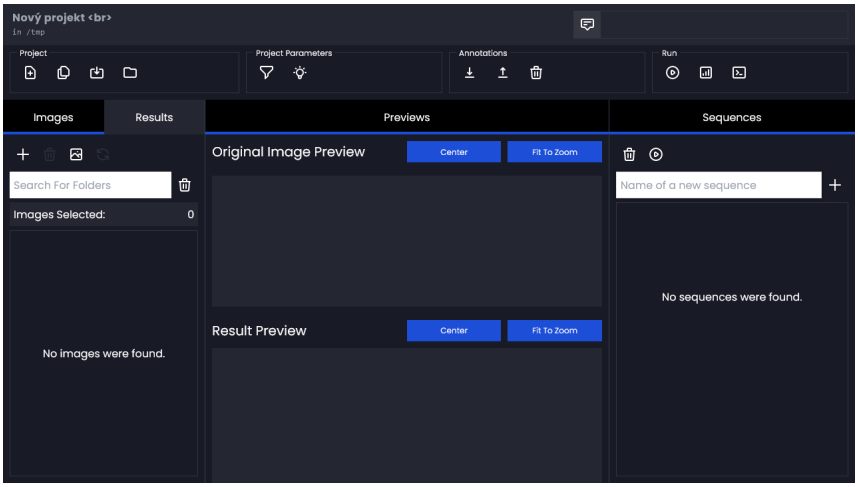


Obrázek 3.1: První verze EWQ

Časem se však ukázalo, že původní software má své nedostatky a omezení, která vyplývala nejen z jeho prvotního návrhu, ale také z použitých technologií. Proto byly zahájeny práce na nové verzi, která využila stávajícího backendu a přidala k němu vrstvu vytvořenou pomocí knihovny Flask. Zároveň došlo k úplné výměně frontendu. K propojení mezi backendem a frontendem byly využity websockety. Nový frontend byl postaven především na knihovně React a prostředí Electron.

Druhá verze EWQ se vyvíjela přibližně šest měsíců, než jsem byl přeřazen na projekt Labelingu v CCM. Toto přeřazení prakticky znamenalo konec vývoje EWQ, jelikož se veškeré kapacity a zdroje přesunuly na nástroje pro labeling. EWQ tak ztratilo prioritu a jeho další vývoj se zastavil.

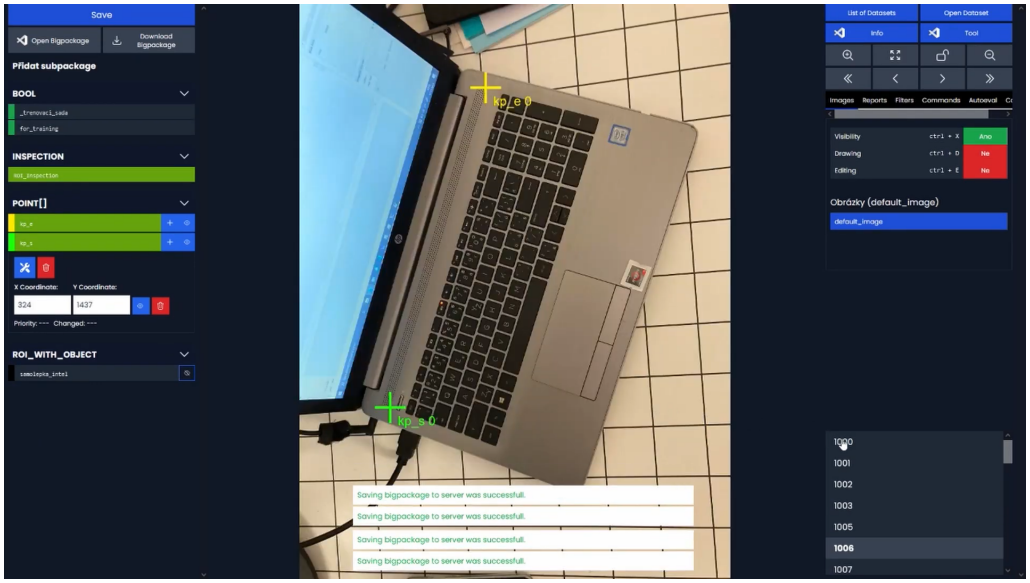
Původně byl EWQ navržen jako software umožňující snadné vytváření procedurálních algoritmů strojového vidění pomocí jednoduchých kroků, které simulují operace OpenCV, jako je například prahování, rozostření nebo konverze do jiného barevného modelu. Nicméně s postupem času se ukázalo, že mnoho těchto problémů lze efektivněji a rychleji řešit pomocí neuronových sítí. Na základě praktických zkušeností se projevila jejich vyšší tolerance vůči nepřesnostem a schopnost lépe zvládat i jednodušší úlohy. Proto se začalo investovat do nástrojů určených pro vývoj, trénování a testování těchto sítí.



Obrázek 3.2: Druhá verze EWQ

Úkolem, který CLOUDCODE často řeší, je zarovnání obrázků. To se týká situací, kdy je nutné upravit orientaci a pozici objektů na obrázcích pořízených v průmyslových procesech. Například na výrobní lince se často pořizují snímky výrobků, ale ty nemusí být vždy správně umístěny, což může narušit správné vyznačení zón, ve kterých se vyhledávají potenciální vady.

Místo manuálního upravování zón pro každý snímek, což je časově náročné a náchylné k chybám, se CLOUDCODE zaměřil na automatizaci procesu zarovnávání obrázků pomocí neuronových sítí. Toho lze dosáhnout definováním alespoň dvou kotvících bodů na obrázcích v trénovací sadě. Poté se natrénuje model, který tyto body predikuje v nových obrázcích, a na základě nich obraz zarovná. Úloha zarovnání se tak redukuje na nalezení těchto kotvících bodů.



Obrázek 3.3: Umístění kotvících bodů na obrázku z testovacího datasetu

3.4 CLOUDCODE MANAGER

CLOUDCODE Manager (označovaný jako *CCM*) je softwarový systém, který slouží jako rozhraní pro správu a obsluhu kamerové kontroly. Jeho využití však není omezeno pouze na koncové zákazníky; nabízí také nástroje pro interní využití zaměstnanci společnosti CLOUDCODE, kteří tak mohou vylepšovat nasazené neuronové sítě u zákazníka. CCM se skládá z různých systémů, které mezi sebou vzájemně interagují. Kromě systémů určených specificky pro jednotlivé zákazníky obsahuje také několik univerzálních. Mezi tyto klíčové systémy patří:

- **Kamerová kontrola** - Systém pro okamžitou kamerovou kontrolu ve výrobě.
- **Archiv** - Systém pro zpětnou kontrolu výsledků kamerové kontroly.
- **Kalibrace** - Nástroje pro kalibraci kamer a robotů.
- **Labeling** - Nástroje pro anotaci nafocených snímků.

3.4.1 Labeling

Labeling je klíčový nástroj v CLOUDCODE, který slouží k anotaci dat pro následné trénování neuronových sítí. Anotace dat spočívá v manuálním označování a popisu důležitých objektů a oblastí na snímcích, čímž se vytváří informační "mapy" pro strojové učení. Například při detekci vad se v procesu labelingu označují oblasti, které obsahují vady na produktu (CloudFactory, 2023).

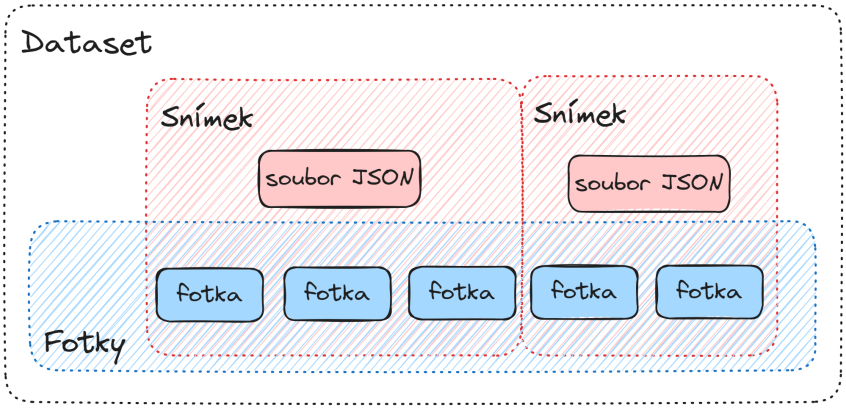
Dříve byl Labeling používán výhradně k trénování neuronových sítí zaměřených na detekci vad. V současné době je proces anotace dat pro CLOUDCODE složitější a vícefázový. V rámci tohoto procesu se vytváří několik neuronových sítí, z nichž každá řeší specifické úkoly v kontrolním procesu. Tento proces probíhá nad souborem snímků, přičemž každá síť využívá odlišné skupiny anotací a mezivýsledků v datasetu. Mezivýsledky mohou zahrnovat například snímek po zarovnání, soubory s informacemi z měření nebo indikace přítomnosti zájmových prvků ve snímku.

3.4.2 Fotky, snímky a datasety

Pro účinné využití nástroje Labeling je nezbytné mít k dispozici data. Na začátku jsou sbírána ručně, kdy jednotlivé kusy jsou snímány pracovníkem u zákazníka. Po instalaci kontrolní stanice na výrobní lince se začnou snímky pořizovat automaticky v rámci kamerové kontroly, která zpočátku pouze fotografuje. Tyto snímky však nevytvářejí žádný výstup pro další části výrobní linky, například pro odklon vadných kusů. Následně jsou tyto fotografie tříděny do složek. Proces třídění se mírně liší v závislosti na

potřebách zákazníka, ale výsledkem je vždy několik složek obsahujících snímky, které jsou připraveny k anotaci.

Z fotografií, které jsou připraveny k anotaci, vytvoří Labeling snímky. Každý snímek obsahuje jeden soubor ve formátu JSON a jednu nebo více fotografií. Tento soubor pak obsahuje podbalíčky, což jsou malé datové struktury, jež reprezentují entity v grafickém rozhraní. Nejmenší snímek obsahuje pouze jeden podbalíček s informací o výchozí fotografii. Jakmile jsou všechny fotografie zpracovány do snímku, složka se stává datasetem. Dataset kromě snímků obsahuje také soubor s vyhrazeným názvem "info.json", který obsahuje informace sdílené napříč celým datasetem, jako jsou výsledky trénování, definice tříd pro klasifikaci a další.



Obrázek 3.4: Diagram vztahů mezi fotkami, snímky a datasetem

3.4.3 Podbalíčky

Podbalíčky jsou datovým schématem ve výměnném formátu JSON, které slouží ke standardizaci struktury dat, které si jednotlivé systémy v CCM předávají. Podbalíčky vznikly původně čistě pro systém *Labeling*, ale později se díky své flexibilitě dostaly i do dalších systémů. Dnes již tak lze pomocí podbalíčku reprezentovat například označenou vadu, informaci o velikosti výrobku, zda-li je snímek připraven k trénování či jak model tuto fotku vyhodnotil.

```
1 interface Subpackage {
2     name: string,
3     type: string,
4     value: unknown
5 }
```

Obrázek 3.5: Struktura obecného podbalíčku

Jedním z typů podbalíčku jsou příkazy. Jedná se o podbalíčky, které dovolují uživateli za kontrolovaných podmínek spustit kód na serveru. Vytvářejí tak rozhraní, kdy uživatel může programu předat argumenty, které vyžaduje pro svůj běh, a následně ho spustit. Tato technika sloužila jako dočasné řešení problému, protože šlo jednotlivé kroky libovolného algoritmu rozdělit na dílčí příkazy, které byly z rozhraní vykonatelné.

Příkazy se používají jako dočasné řešení pro vykonávání jednotlivých kroků algoritmu. Nevýhodou ale je, že uživatel musí každý krok spustit samostatně, a pokud navíc došlo k chybě, musí se práce obnovit ze zálohy do předchozího stavu. Tento proces pro zhruba 10 aktivních datasetů v rozsahu od 50 až 3000 snímků není dlouhodobě udržitelný. Prvotní výhodou a i účelem tohoto kroku však byla abstrakce volání z příkazové řádky do seznamu tlačítek a seznamu v dokumentaci.

3.4.4 Kroky, nástroje a inspekce

Pro popis správného pořadí jsem chtěl využít již datových struktur, které jsou v systému CCM zavedené, a akorát je vhodně rozšířit. Z potřeb, které vzešly na základě analýzy z kapitoly porovnávající již existující řešení jsem došel ke systému, který pracuje s třemi entitami:

- **Kroky** - Určují etapu, ve kterém se dataset nachází
- **Nástroje** - Definují operace, které lze nad snímkem provést
- **Inspekce** - Instance jednotlivých nástrojů pro konkrétní zájmový prvek ve snímku

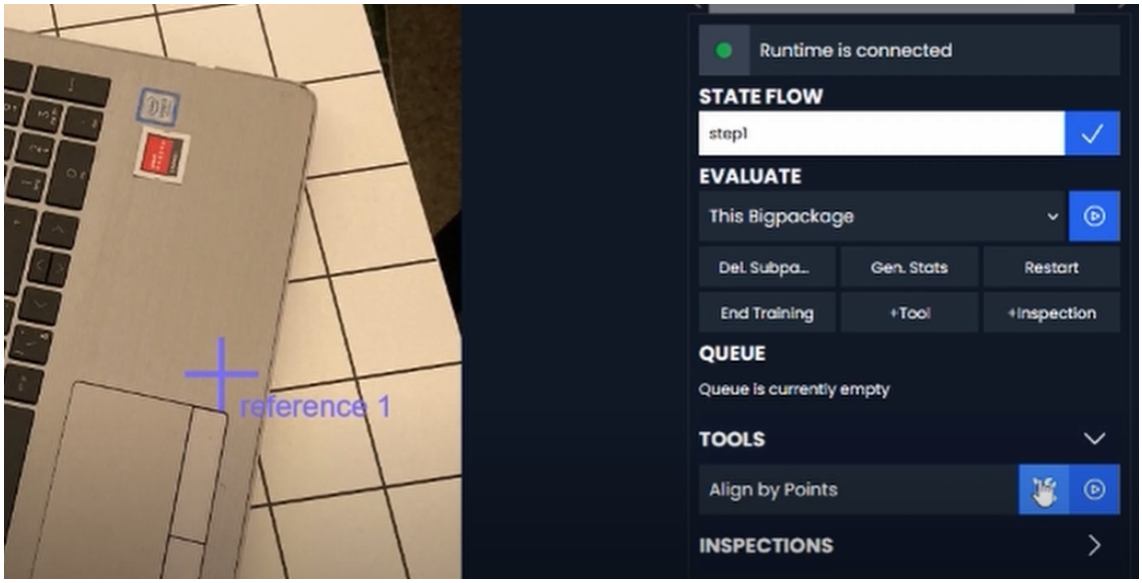
Nástroje jsou novým typem podbalíčku, který je ve svém principu evolucí příkazů. Nástroje neodkazují na konkrétní spustitelný soubor, ale referují na šablonu nástroje a jaké parametry mají použít. Příkladem může být nástroj pro klasifikaci, kde správným vyplněním parametrů můžeme vytvořit nástroj trénující či vyhodnocující model pro kontrolu přítomnosti šroubku v oblasti zájmu.

Inspekce jsou dalším typem podbalíčku, který jsem v rámci své práce přidal. Koncept inspekcí vznikl pro využití jednoho modelu na vícero oblastí zájmu. Pokud opět použiji příklad kontroly přítomnosti šroubku. Nemá smysl trénovat čtyři modely kontrolující přítomnost jednoho a toho samého šroubu, který je akorát na čtyřech různých místech. Stačí tak natrénovat jeden model, který je pak následně vyhodnocen na čtyřech místech. A právě vyhodnocení na vícero oblastech zájmu zajištují inspekce, na které díky tomu pohlížet jako na ”instance nástroje”[2].

[2]Nástroje jako takové nepotřebují inspekce k tomu, aby mohly být trénovány či vyhodnoceny. Údaje o oblasti, které jsou inspekcí poskytovány, mohou být i přímo vepsány do nástroje

Kroky jsou posledním přidaným typem, který slouží k určení pořadí vykonávání. Ke krokům lze přiřadit jednotlivé nástroje či inspekce, které se v rámci kroku vykonají najednou. Pořadí kroků je následně určeno v grafickém rozhraní *labelingu*. Je tedy třeba při návrhu algoritmu zvážit, zdali některé nástroje nezávisí na výstupu jiných a správně je rozdělit do kroků.

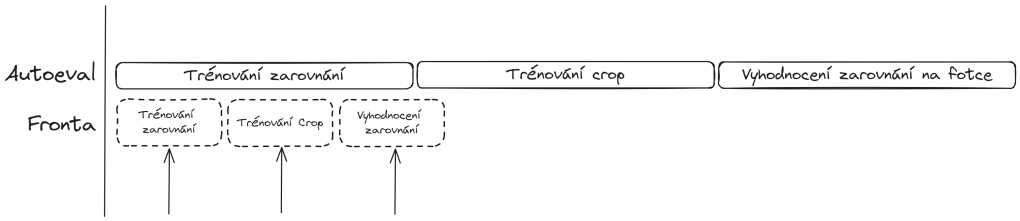
Protože se nástroje mohou opakovat v různých datasetech, ukládají se do samostatného souboru `tool.json`. Následně jsou nástroje v datasetu uloženy se stejným klíčem a informací, jestli se mají v daném datasetu vykonávat či nikoliv.



Obrázek 3.6: Panel pro správu nástrojů a inspekcí v Labelingu

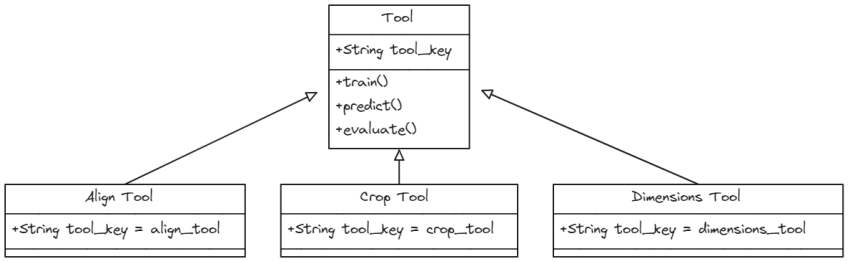
3.5 RUNTIME PRO NÁSTROJE A INSPEKCE

Trénování, vyhodnocování a měření statistik se provádí v běhovém prostředí s názvem Autoeval, které běží mimo CCM. Tohle běhové prostředí přijímá instrukce, které sekvenčně vykonáva.



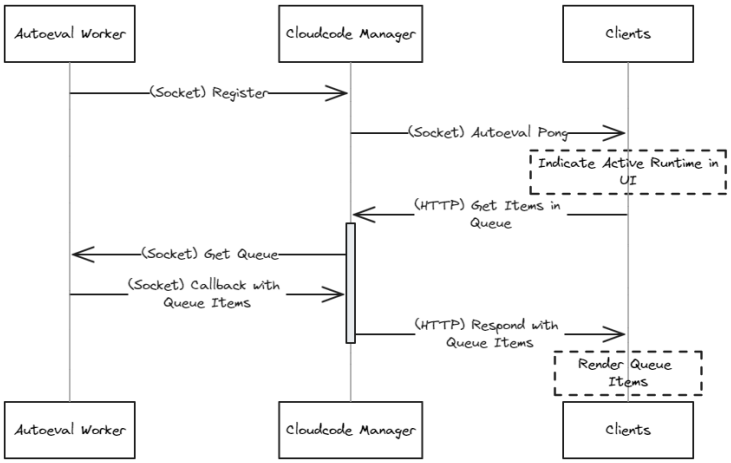
Obrázek 3.7: Autoeval fronta

Architektura Autoevalu je postavena na modulech, což umožňuje snadné přidávání a odebírání nástrojů. Centrální roli hraje *Autoeval Worker*, který do fronty přijímá požadavky na spuštění různých úkolů, jako je trénování nebo vyhodnocování. Dále máme registr nástrojů, což je seznam všech nástrojů, které můžeme použít. Všechny tyto nástroje vychází ze společné základní třídy, která definuje rozhraní, respektive signatury metod.



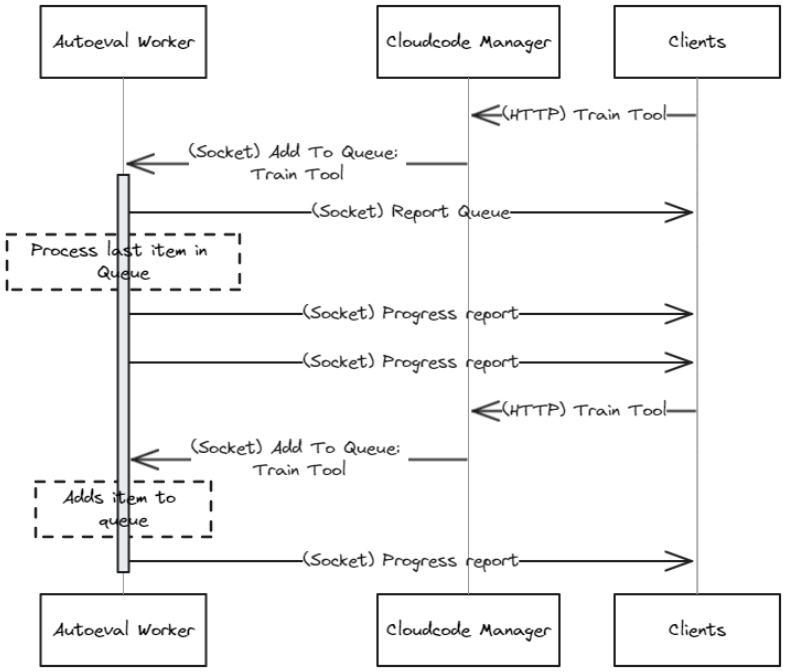
Obrázek 3.8: Nástroje odvozené od společného základu

Komunikace s Autoevalem probíhá prostřednictvím síťových socketů, kdy Autoeval funguje jako klient, který se připojuje k socketovému serveru na CCM. Klient naopak s CCM komunikuje skrz protokol HTTP. Informace jsou vyměňovány v formátu JSON. Po spuštění Autoevalu se jednotka zaregistruje v CCM jako aktivní a připravená na příjem úloh. Klientům připojeným k CCM je o této události zasláno upozornění, aby mohli aktualizovat indikátor v grafickém rozhraní. Pokud uživatel přejde na záložku Autoeval v systému Labeling, klient se dotazuje, zda je Autoeval aktivní a zdali jsou ve frontě nějaké úlohy, které by měly být zobrazeny v rozhraní spolu se svým stavem.



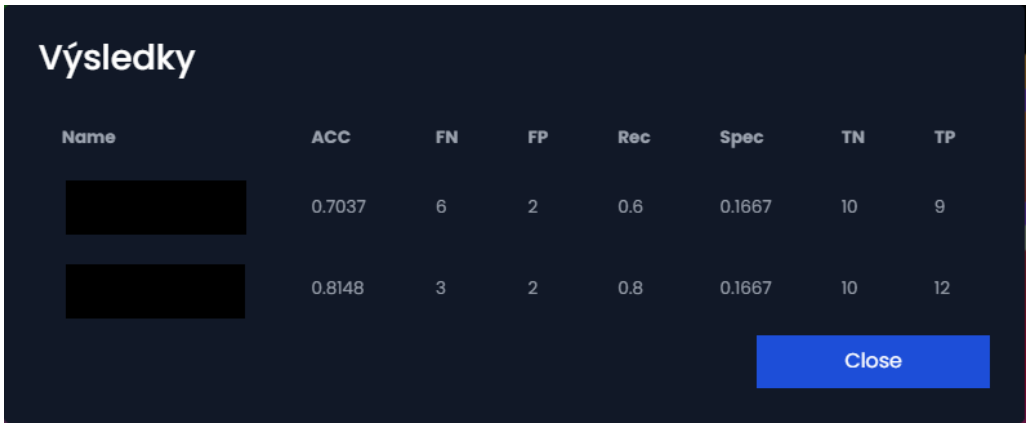
Obrázek 3.9: Komunikace Autoevalu při jeho spuštění

Při příjmu požadavku na trénování, vyhodnocení nebo měření statistik nástroje si Autoeval nejprve vyhledá typ požadované operace v příchozí zprávě a identifikuje nástroj, který má být použit. Pokud je požadovaný nástroj dostupný, respektive je zaveden v registru nástrojů, Autoeval ho instancuje. Následně Autoeval extrahuje z zprávy klíčové informace, jako jsou datasety, konfigurační parametry a specifické požadavky na operaci. Tyto informace následně předá do nástroje a operaci spustí. Každé operaci je přiděleno unikátní ID operace. Toto ID je používáno pro sledování a hlášení průběhu operace z Autoevalu. V případě trénování Autoeval hlásí procentuální postup v epochách.



Obrázek 3.10: Komunikace Autoevalu při trénování

Po dokončení operace Autoeval zapíše výsledky do relevantních souborů. Obvykle se jedná o úpravu podbalíčků ve snímku nebo úprava info.json souboru. Tímto způsobem jsou výsledky uloženy a mohou být dále použity pro další analýzu nebo vyhodnocení.

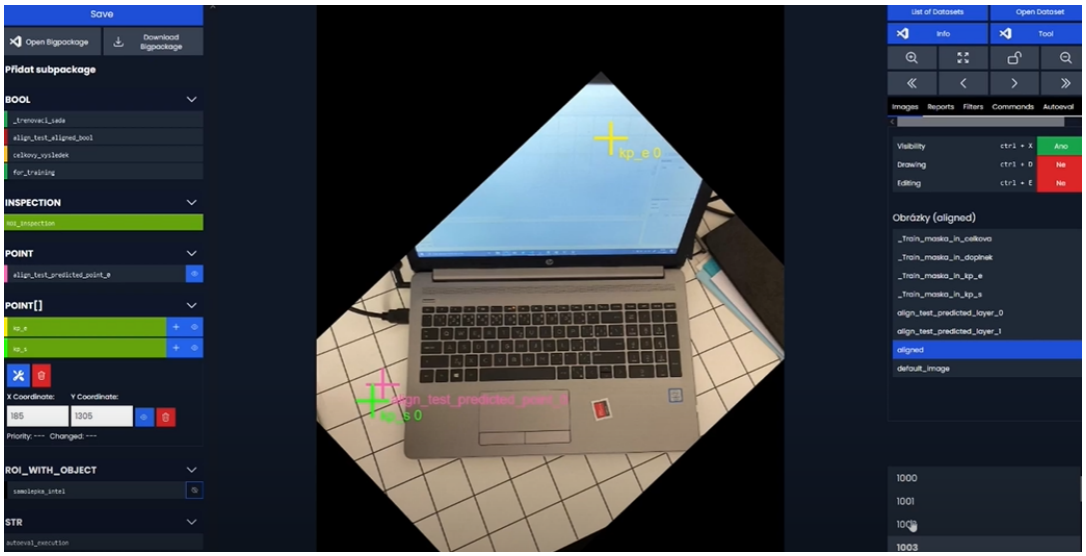


Obrázek 3.11: Metriky klasifikačního modelu po měření

Doposud jsem mluvil pouze o zpracování nástrojů v rámci Autoevalu. Jak je to s inspekcemi? Z pohledu Autoevalu jsou inspekce v podstatě také nástroje, liší se však v tom, že před jejich vykonáním dojde k průniku konfigurací inspekce a nástroje, přičemž hodnoty z inspekce mají přednost. Jediný významný rozdíl oproti nástroji je přítomnost informace, ke kterému nástroji inspekce patří. Bez toho by nebylo možné správně spárovat inspekci s nástrojem.

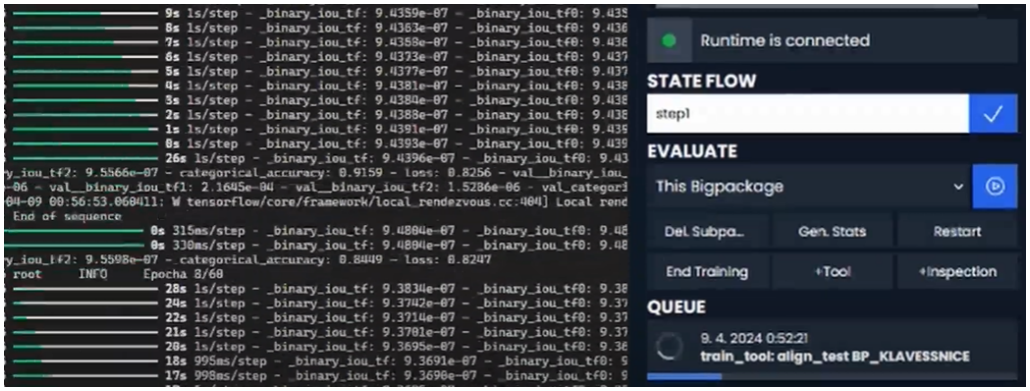
4 ZPĚTNÁ VAZBA

Řešení bylo nasazeno již od února, kdy nejprve prošlo uzavřeným testováním vývojáři, aby se odladila základní funkcionalita Autoevalu a rozhraní. Poté byli k systému postupně připuštěni anotátoři. Toto rozšíření uživatelské základny odhalilo několik předpokladů o znalosti systému, které jsem při vývoji udělal a které jsou klíčové pro správné využití. Například noví anotátoři, kteří nikdy předtím neslyšeli o Autoevalu, nevěděli, že mají hledat trénování nebo vyhodnocení, které dříve bylo v příkazech, v záložce s názvem *Autoeval*. Tento drobný zádrhel lze odstranit krátkým školením, ovšem upozornil na větší problém, a to jsou možné předpoklady znalostí, které nutně nemusí vždy existovat. Systém je cílen na proškolené pracovníky ve firmě CLOUDCODE, tudíž to naštěstí není takový problém.



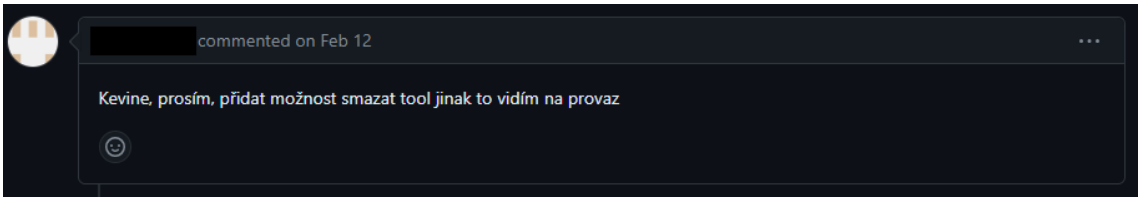
Obrázek 4.1: Labeling 3 v CCM, se záložkou Autoeval v pravém panelu

Další zpětnou vazbou bylo neexistující hlášení chyb v Autoevalu. Pokud během trénování dojde k chybě, například kvůli selhání běhového prostředí tensorflow nebo neočekávané události, položka z fronty prostě zmizí, aniž by uživatel obdržel jakékoliv hlášení o problému. Uživateli dojde, že je něco špatně poté, co neuvidí očekávané výsledky v rozhraní, ale stále je to z hlediska anotátora, který nemá ke konzoli Autoevalu přístup, velmi frustrující.



Obrázek 4.2: Autoeval Fronta s otevřenou konzolí Autoevalu

Posledním významným bodem zpětné vazby je mazání nástrojů a inspekcí. Zde měly zájmové skupiny rozdílné názory, kdy anotátoři by si tuto možnost přáli pro přehlednější rozhraní (páč nástroje jsou sdílené napříč datasety, takže seznam velmi rychle zaplní pro daný projekt nerelevantní nástroje), ale vývojáři s personálem trénující neuronové sítě by byli s touto možností velmi obezřetní. Jedná se hlavně o možnost ztrátu konfigurací, která by mohla být nepříjemná. Po diskuzi ve firmě by mohl kompromis ležet s mazáním nástrojů za heslem, respektive smazání nástroje by musel vždy ověřit heslem zaměstnanec s privilegovanými právy.



Obrázek 4.3: Zpětná vazba s prosbou o přidání mazání nástrojů

Tyto body však nebránily firmě v plném uplatnění řešení v praxi. Stále se jedná o značný posun oproti předchozímu přístupu s příkazy, kde proces fungoval ve stylu *"klikni a modli se"*. Navíc mají všechny možné operace standardizované rozhraní, což bylo doceněno i vývojáři implementující samostatné nástroje, protože jim stačí metody implementovat, nástroj zaregistrovat v Autoevalu a volitelně pro něj vytvořit šablonu do grafického rozhraní. Poté už je vše automatické.

5 ZÁVĚR

Cílem této práce bylo pro společnost CLOUDCODE s.r.o. vytvořit řešení, které umožní rapidní vývoj algoritmů strojového vidění využívající jak procedurální algoritmy, tak i neuronové sítě.

V rámci práce jsem představil tři základní úlohy strojového vidění, jaké jsou možnosti předzpracování a samotného zpracování obrazu, a softwarová řešení, které umožňují tyto poznatky aplikovat a vytvářet tak algoritmy strojového vidění. Po rešerši dostupných nástrojů na trhu jsem ve spolupráci s firmou vytyčil čtyři klíčové body, které musely být vyřešeny:

- Použití jednoho algoritmu na více oblastí zájmu.
- Použití více algoritmů na jednom snímku.
- Práce s neuronovými sítěmi (trénování, predikce, evaluace metrik).
- Práce s (proměnlivou) množinou snímků.

Jako řešení jsem zvolil koncept nástrojů, kroků a inspekcí. Tyto koncepty umožňují definovat operace algoritmu strojového vidění nezávisle na tom, jestli se jedná o procedurální algoritmus nebo neuronovou síť. Každou operaci lze trénovat, vyhodnotit a měřit její statistiky. Vývojáři mohou nové operace definovat pomocí jednotného předpisu, a uživatelé s nimi mohou pracovat v grafickém rozhraní, které je součástí anotačního systému Labeling v rámci CCM.

Řešení už je aktivně nasazeno v produkčním prostředí a stalo se nedílnou součástí každodenního pracovního cyklu anotátorů. Díky tomu jsem získal cennou zpětnou vazbu, kterou budu dále využívat k vylepšení celkové kvality řešení. Sběr zpětné vazby nebyl nijak strukturovaný a probíhal pouze ústně či formou připomínek v systému Github Issues, protože je výsledný systém používán pouze úzkým okruhem lidí ze společnosti CLOUDCODE.

Kromě toho mám v plánu funkcionalitu modifikovat tak, aby řešení bylo distribuované nebo alespoň vícevláknové. Existují také oblasti, nad kterými lze diskutovat, jako jsou datové struktury - konkrétně možnost reprezentovat kroky jako samostatné podbalíčky. Plánuji také kontejnerizaci běhového prostředí, což by eliminovalo riziko chybějících závislostí, ale zároveň zavede Docker Engine, což zvedne nároky na systémové zdroje. To by mohlo být částečně kompenzováno jeho širší adopcí ve firmě, aby neběžel pouze pro ”jednu věc”.

REFERENCE

AHN, Song Ho, 2024. *Example of 2D Convolution* [online]. [cit. 12. 04. 2024]. Dostupné z: https://www.songho.ca/dsp/convolution/convolution2d_example.html.

BANDYOPADHYAY, Hmrishav, 2022. *What is Computer Vision? [Basic Tasks and Techniques]* [online]. 9. 6. 2022. [cit. 01. 01. 2024]. Dostupné z: https://www.v7labs.com/blog/what-is-computer-vision.

CLOUDCODE S.R.O., 2023. *"CLOUDCODE.cz"* [online]. [cit. 06. 12. 2023]. Dostupné z: https://cloudcode.cz/.

CLOUDFACTORY, 2023. *Image Annotation for Computer Vision* [online]. 5. 11. 2023. [cit. 05. 11. 2023]. Dostupné z: https://www.cloudfactory.com/image-annotation-guide.

ELDEEP, Ahmed; LANGE, Uwe, 2012. *ImageNets Download* [online]. 28. 9. 2012. [cit. 17. 11. 2023]. Dostupné z: https://sourceforge.net/projects/imagenets/.

GOOGLE, 2024. *Introduction to Vertex AI* [online]. 16. 4. 2024. [cit. 18. 04. 2024]. Dostupné z: https://cloud.google.com/vertex-ai/docs/start/introduction-unified-platform.

GRAESER, Axel, 2010. *Digital Image Processing with ImageNet Designer* [online]. 8. 12. 2010. [cit. 17. 04. 2024]. Dostupné z: https://sourceforge.net/projects/imagenets/files/Lecture%20Material/03b-ImageNetDesigner-WS10-11-Lange.ppt/download.

HLAVÁČ, Václav, 2019. *Image segmentation* [online]. 14. 11. 2019. [cit. 14. 04. 2024]. Dostupné z: https://people.ciirc.cvut.cz/~hlavac/TeachPresEn/15ImageAnalysis/32-02SegmentationTaxon+Thresh.pdf.

HLAVÁČ, Václav, 2024a. *Hrany, hranové body a ostření obrazu* [online]. [cit. 14. 04. 2024]. Dostupné z: https://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/22EdgesInImagesCz.pdf.

HLAVÁČ, Václav, 2024b. *Jasové a geometrické transformace* [online]. [cit. 12. 04. 2024]. Dostupné z: https://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/18BrightGeomTxCz.pdf.

HLAVÁČ, Václav, 2024c. *Předzpracování obrazů v prostoru obrazů, operace v lokálním sousedství* [online]. České vysoké učení technické v Praze. [cit. 09. 04. 2024]. Dostupné z: https://people.ciirc.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/21ImagPreprocCz.pdf.

HLAVÁČ, Václav, 2024d. *Předzpracování v prostoru obrazů* [online]. České vysoké učení technické v Praze. [cit. 05. 04. 2024]. Dostupné z: https://dspace.tul.cz/server/api/core/bitstreams/f47b0f1f-b87a-4511-908c-f2c14db49210/content.

HORÁK, Karel, 2019. *Jasové transformace* [online]. [cit. 12. 04. 2024]. Dostupné z: http://vision.uamt.feec.vutbr.cz/ZVS/lectures/05_Jasove_transformace.pdf.

CHALOUPKA, Josef, 2024a. *Úvod do zpracování obrazů - Prezentace přednášky č. 1* [online]. 20. 2. 2024. [cit. 04. 04. 2024]. Dostupné z: https://elearning.tul.cz/course/view.php?id=14732.

CHALOUPKA, Josef, 2024b. *Úvod do zpracování obrazů - Prezentace přednášky č. 4* [online]. 20. 2. 2024. [cit. 04. 04. 2024]. Dostupné z: https://elearning.tul.cz/course/view.php?id=14732.

IEEE Recommended Practice for Software Requirements Specifications, 1998. *IEEE Std 830-1998*, s. 1–40. Dostupné z DOI: 10.1109/IEEESTD.1998.88286.

KALENIUK, Oleksandr, 2017. *Programmer's guide to homogeneous coordinates* [online]. 9. 5. 2017. [cit. 12. 04. 2024]. Dostupné z: https://hackernoon.com/programmers-guide-to-homogeneous-coordinates-73cbfd2bcc65.

KELLY, Cullen, 2023. *The Essential Guide to Color Spaces* [online]. 18. 12. 2023. [cit. 12. 04. 2024]. Dostupné z: https://blog.frame.io/2020/02/03/color-spaces-101/.

KEYENCE CORPORATION, 2024. *Vision System with Built-in AI* [online]. [cit. 20. 04. 2024]. Dostupné z: https://www.keyence.eu/cscz/products/vision/vision-sys/vs/.

KREJČÍ, Martin, 2023. *Bitová hloubka* [online]. 3. 11. 2023. [cit. 04. 04. 2024]. Dostupné z: https://web.archive.org/web/20231003212310/https://www.onlinefotoskola.cz/pomucky/databaze-fotografickych-pojmu/bitov%C3%A1+hloubka.html.

LANDING AI, 2024. *AI Platform for Deep Learning-Based Computer Vision: LandingLens* [online]. [cit. 20. 04. 2024]. Dostupné z: https://landing.ai/platform/.

MACHINE LEARNING COLLEGE, 2020a. *4 Problém přetrénování* [online]. 20. 4. 2020. [cit. 15. 04. 2024]. Dostupné z: https://www.youtube.com/watch?v=49L0BFNeyHQ.

MACHINE LEARNING COLLEGE, 2020b. *5 Evaluační metriky* [online]. 3. 6. 2020. [cit. 20. 04. 2024]. Dostupné z: https://www.youtube.com/watch?v=q8Ru8nz6iXA.

MATHWORKS, 2024a. *What Is a Convolutional Neural Network?* [online]. [cit. 16. 04. 2024]. Dostupné z: https://www.mathworks.com/discovery/convolutional-neural-network.html.

MATHWORKS, 2024b. *What Is Deep Learning?* [online]. [cit. 15. 04. 2024]. Dostupné z: https://www.mathworks.com/discovery/deep-learning.html.

MATHWORKS, 2024c. *What is MATLAB?* [online]. [cit. 17. 04. 2024]. Dostupné z: https://www.mathworks.com/discovery/what-is-matlab.html.

MICROSOFT, 2024a. *Algoritmy strojového učení* [online]. [cit. 15. 04. 2024]. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-are-machine-learning-algorithms.

MICROSOFT, 2024b. *Co je machine learning?* [online]. [cit. 15. 04. 2024]. Dostupné z: https://azure.microsoft.com/cs-cz/resources/cloud-computing-dictionary/what-is-machine-learning-platform.

NATIONAL INSTRUMENTS CORP., 2024a. *How Do I Use the Vision Development Module to Deploy a Deep Learning Model to a Machine Vision System?* [online]. [cit. 20. 04. 2024]. Dostupné z: https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/how-do-i-deploy-a-deep-learning-model-to-a-machine-vision-system.html.

NATIONAL INSTRUMENTS CORP., 2024b. *What Is Vision Builder for Automated Inspection?* [online]. [cit. 20. 04. 2024]. Dostupné z: https://www.ni.com/en/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-vision-builder-for-automated-inspection.html.

NVIDIA CORPORATION, 2023. *Computer vision – What Is It and Why Does It Matter?* [online]. 27. 2. 2023. [cit. 01. 01. 2024]. Dostupné z: https://www.nvidia.com/en-us/glossary/data-science/computer-vision/.

ORMESHER, Ian, 2020. *Convolution Filters* [online]. 9. 5. 2020. [cit. 12. 04. 2024]. Dostupné z: https://medium.com/@ianormy/convolution-filters-4971820e851f.

PEKAT S.R.O, 2021a. *PEKAT VIDEO - Code and Flow* [online]. 25. 3. 2021. [cit. 19. 04. 2024]. Dostupné z: https://www.youtube.com/watch?v=XRGJudSi6HQ.

PEKAT S.R.O, 2021b. *Surface Detection Tutorial Video - Batteries Inspection (version 3.10.4)* [online]. 24. 3. 2021. [cit. 19. 04. 2024]. Dostupné z: https://www.youtube.com/watch?v=THGQB04dFrU.

PEKAT S.R.O., 2022. *pekat-vision-examples* [online]. 1. 9. 2022. [cit. 18. 04. 2024]. Dostupné z: https://github.com/pekat-vision/pekat-vision-examples.

PEKAT S.R.O., 2024. *Funkce* [online]. [cit. 18. 04. 2024]. Dostupné z: https://www.pekatvision.com/cs/features/.

QT GROUP, 2023. *Design GUI with Python: Python Bindings for Qt* [online]. [cit. 12. 12. 2023]. Dostupné z: https://www.qt.io/qt-for-python.

QT WIKI, 2022. *About Qt* [online]. 18. 7. 2022. [cit. 12. 12. 2023]. Dostupné z: https://wiki.qt.io/Qt_for_Beginners.

ROSER, Max, 2022. The brief history of artificial intelligence: The world has changed fast – what might be next? *Our World in Data* [online] [cit. 27. 10. 2023]. Dostupné z: https://ourworldindata.org/brief-history-of-ai.

ROSER, Max, 2023. Artificial intelligence has advanced despite having few resources dedicated to its development – now investments have increased substantially. *Our World in Data* [online] [cit. 27. 10. 2023]. Dostupné z: https://ourworldindata.org/ai-investments.

SURYNKOVÁ, Petra, 2024. *Geometrické transformace* [online]. [cit. 12. 04. 2024]. Dostupné z: http://surynkova.info/dokumenty/mff/PG/Prednasky/prednaska_4.pdf.

ŠKABRADA, Jan, 2019. *Inovační strategie podniku* [online]. [cit. 27. 10. 2023]. Diplomová práce. Masarykova univerzita, Ekonomicko-správní fakulta.