

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

**Automatická analýza fotovoltaických panelů ze
snímků pořízených dronem**

**František Hamrla
Jihomoravský kraj**

Brno, 2025

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 18: Informatika

**Automatická analýza fotovoltaických panelů ze
snímků pořízených dronem**

**Automatic analysis of photovoltaic panels from
images taken by drone**

Jméno: František Hamrla

Škola: Gymnázium Brno, Vídeňská, příspěvková organizace,
Vídeňská 55/47, 639 00 Brno

Kraj: Jihomoravský kraj

Konzultant: Doc. Ing. Vítězslav Beran, Ph.D.

Prohlášení

Prohlašuji, že jsem svou práci SOČ vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Brně dne

František Hamrla



jihomoravský kraj

Poděkování

Chtěl bych poděkovat svému školiteli, Doc. Ing. Vítězslavu Beranovi, Ph.D., za přínosné konzultace a za jeho přístup, kterým mě vedl k výsledku. Choval se vždy laskavě a i v nejhorších situacích, kdy se vše zdálo ztracené, mi pomohl vymyslet správné řešení problému. Taktéž bych chtěl poděkovat své rodině, která mě v psaní práce motivovala a pomohla mi tím k dosažení kýženého výsledku. A nakonec chci poděkovat kamarádům z kapely, díky nimž se zdálo psaní práce o něco lehčí.

Tato práce byla vypracována za finanční podpory JMK.

Anotace

Ve své práci SOČ se zabývám automatickou analýzou fotovoltaických panelů ze snímků pořízených dronem. Výstupem je program, který je schopen detektovat jednotlivé fotovoltaické panely ze snímků. K dosažení tohoto cíle byly použity metody strojového učení a následně metody zpracování obrazu. Práce taktéž vysvětuje pojmy spojené právě s problematikou strojového učení a zpracování obrazu.

Klíčová slova

Strojové učení; počítačové vidění; zpracování obrazu; segmentace obrazu; fotovoltaické panely

Abstract

In this thesis, I deal with the problem of automatic analysis of photovoltaic panels from images taken by a drone. My goal was to create a program that would be able to detect individual photovoltaic panels from images. To achieve this goal, I used methods of machine learning, followed by image processing methods. The thesis also explains concepts related to machine learning and image processing.

Keywords

Machine learning; computer vision; image processing; image segmentation; photovoltaic panels

Obsah

1	Úvod	1
2	Teoretický úvod	2
2.1	Monitorování fotovoltaických elektráren drony	2
2.2	Zpracování obrazu	4
2.3	Strojové učení	6
2.4	Segmentace obrazu	12
2.5	Existující nástroje a knihovny	13
3	Detekce FVP v obrazu z dronu	15
3.1	Model CNN vs. Zpracování obrazu?	15
3.2	Segmentace FVP pomocí CNN	16
3.3	Detekce FVP pomocí metod zpracování obrazu	18
4	SW nástroj pro detekci FVP	20
4.1	Dataset	20
4.2	Model strojového učení	24
4.3	Detekce jednotlivých FVP	29
4.4	Diskuze výsledků a návrhy na zlepšení	36
5	Závěr	38

1 Úvod

V dnešní době se lidstvo čím dál více ubírá k obnovitelným zdrojům energie. Jedním z těchto zdrojů je energie solární, která je nejčastěji zachycována fotovoltaickými panely [3]. Tyto panely mohou být instalovány na střechy domů, vedle silnic, či na pole. Vznikají tak fotovoltaické elektrárny, které však potřebují komplexní údržbu. Jedním z těchto úkolů je detekce vadných panelů a jejich případná výměna. Tento proces je v dnešní době velmi nákladný a šel by automatizovat. Na začátku procesu se musí nejdříve panely nafotit. Následně se panely zdlouhavě ručně zaznačují do snímků, což je právě krok, který chci v této práci zautomatizovat.

Cílem této práce bylo vytvořit a zdokumentovat program, který bude schopen úspěšně detektovat jednotlivé fotovoltaické panely ze snímku, který byl pořízen dronem. K detekci byly použity technologie zpracování obrazu a postupy strojového učení – konkrétně hluboké učení.

Práce obsahuje návrh řešení výše zmíněného problému a následnou praktickou implementaci tohoto návrhu. V první části této práce se zabývám teoretickými pojmy, se kterými je nutné se seznámit, pro bližší pochopení této problematiky. Součástí tohoto vyšvětlení je také popis nástrojů, které jsou pro implementaci řešení vhodné. Ve finální části práce vytvořený program zhodnocuji a popisuji případné návrhy na zlepšení.

Hlavním rozhodnutím, proč se zabývat touto problematikou, byla moje fascinace strojovým učením. Tato záliba přišla s nárůstem popularity umělé inteligence u běžných lidí. Chtěl jsem tématu více porozumět a vyzkoušet si práci s ním v praxi. Celková problematika mi přišla zajímavá a prospěšná, a proto jsem se jí chtěl věnovat více.

2 Teoretický úvod

Obor práce spojující umělou inteligenci a zpracovávání obrazu se nazývá počítačové vidění [6]. V programech, zabývajících se oborem, je vstupní hodnotou obraz. Tento vstup se většinou metodami zpracování obrazu předzpracuje a poté se pomocí modelu umělé inteligence analyzuje. Model následně učiní predikci a tato predikce se požaduje jako výstup. Počítačové vidění se využívá pro automatizaci lidských úloh, např. v medicíně, v analýze obrazu či u autonomních vozidel a robotů. Podoborem počítačového vidění je segmentace obrazu, kterou se v práci zabývám.

2.1 Monitorování fotovoltaických elektráren drony

Fotovoltaické panely (FV panely) [3, 2] se skládají z fotovoltaických článků, které jsou schopné generovat elektrický proud ze slunečního záření. Tento proces získávání energie je udržitelný a ekologický. Pole FV panelů, používané ve FV elektrárnách (viz obr. 2.1), vznikne spojením více FV panelů buďto paralelně, či sériově. Zkombinováním těchto dvou typů zapojení lze dovršit nejvyšší účinnosti v získání solární energie.

FV panely mají omezenou životnost a často dochází k různým vadám [9]. Tyto vady mohou být způsobeny buďto interními, nebo externími faktory. Rozlišují se zde také permanentní a dočasné vady. Permanentní vady se považují za dlouhodobé, patří mezi ně například porucha elektronické součástky. Dočasné vady bývají přítomny jen na krátkou dobu. Patří mezi ně například sníh či prach na části panelu, nebo zakrytí panelu stínem z mraku, či z blízkých budov a objektů. Tyto vady je nutno odstranit pro správný a efektivní chod elektrárny. Panely s vadou bývají odstraňovány specializovanými techniky. Proces hledání vady a následná výměna panelu je časově i finančně nákladná.



Obrázek 2.1: Fotovoltaická elektrárna – Brno Tuřany, Autor: Petr Opletal – Vlastní dílo,

CC BY-SA 3.0,

<https://commons.wikimedia.org/w/index.php?curid=25750540>

Ve FV elektrárnách velkých rozměrů se k inspekci jednotlivých panelů většinou používají **drony** [3]. Na tyto malé bezpilotní letouny (UAV) se připevní kamera s termovizním snímačem (viz obr. 2.2). Následně se pečlivým skenováním nafotí letecké snímky celé elektrárny, které se potom dále zpracují.



Obrázek 2.2: Dron s termovizním snímačem, Zdroj: <https://hp-drones.com/en/drones-with-thermal-cameras-what-are-their-advantages/>

Obrazy však nemusí mít pouze podobu termosnímků, běžně se lze setkat i se snímkami ze satelitů, či dronů bez termovizního snímače (viz obr. 2.3). Tyto snímkы se liší vzdálostí (výškou) snímacího letounu od snímaného objektu, kvalitou snímacího zařízení, či podmínkami v okolí snímané elektrárny (např. počasí).



Obrázek 2.3: Snímek FV elektrárny pořízený dronem, Zdroj:

<https://www.kaggle.com/datasets/salihammaidi07/solar-panel-detection-and-indentification>

2.2 Zpracování obrazu

Obraz v počítačové formě, neboli **digitální obraz** [4] je zobrazení obrazu pomocí pixelů na výstupní zařízení. Lze rozlišit dva typy digitálního obrazu, vektorový a rastrový. Vektorový vykresluje snímek pomocí geometrických objektů, zatímco rastrový pomocí jednotlivých pixelů. Pixel je tedy nejmenší možná část obrazu, uchovávající informace o barvě, kterou právě nese. Počítač s obrazem pracuje jako s maticí malých celých čísel.

Podle hodnot, které jednotlivé pixely nabývají, lze rozlišit typ digitálního obrazu podle funkce, ke které počítač obraz využívá¹. Ne vždy je totiž potřeba, aby snímek nabýval všech barevných hodnot. Pokud pixel dosahuje hodnoty 0 nebo 1, jedná se o binární obraz, ten se často používá k vytvoření masky obrazu při segmentaci obrazu. Obraz s pixely obsahujícími pouze jeden barevný kanál se nazývá obraz ve stupních šedi (šedotónový obraz), používá se často v úlohách se zpracováním obrazu, kdy barevné kanály nejsou zapotřebí. Jestliže pixely obsahují hodnoty pro 3 barevné kanály, jedná se o běžný barevný RGB obraz. Ještě existuje RGBA obraz obsahující 4 kanály. Kromě základních barevných kanálů se zde vyskytuje ještě kanál pro průhlednost.

¹<https://www.v7labs.com/blog/image-processing-guide>

V jazyce Python existuje více knihoven na práci s obrazem, mezi největší a nejznámější představitele patří OpenCV a Pillow. Obě knihovny mají rozsáhlou dokumentaci, a proto je jednoduché s nimi začít.

S obrazem je možno pracovat pomocí **digitálního zpracování obrazu** [5]. Jedná se o využití digitálního počítače k zpracování obrazu pomocí algoritmů. Je to esenciální krok při tvorbě efektivního programu při práci s detekcí objektů, detekcí obličejů, či kompresí obrazu. Často se používá k pozvednutí důležitých detailů obrazu, což může vysoce zvýšit efektivitu modelu strojového učení. Shrnu zde metody, které ve výsledném návrhu řešení používám.

Obecně se za **oříznutí** považuje odstranění nevhodných částí obrazu. Dochází tak ke změně jeho rozlišení či aspektu. Taktéž lze oříznutím rozdělit větší obraz na vícero menších částí a zachytit tak větší množství detailů. Toho lze dobře využít, když je málo vzorků v datasetu. Jednoduše každý vzorek rozdělíme a vznikne tím více vzorků k trénování modelu.

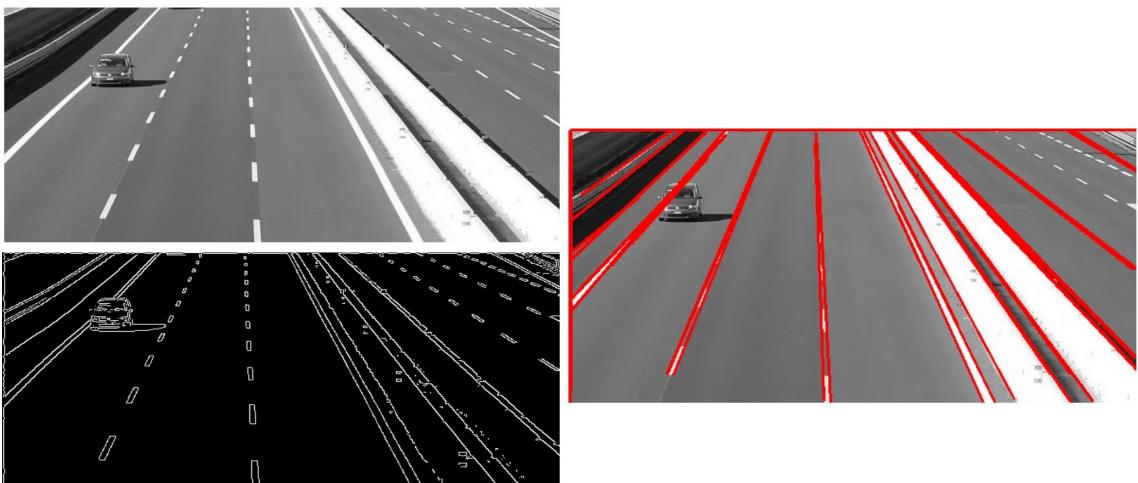
Obraz se může **škálováním** zvětšit či zmenšit. Je zde také rozdíl mezi škálováním rastrových a vektorových snímků. U vektorových se vektory snímku jednoduše přeškálují a není zde ztráta kvality. Rastrové obrazy se skládají z určitého počtu pixelů, který se musí škálováním změnit, dojde proto ke změně kvality snímku. Existuje také více typů škálování, at' už pomocí matematických filtrů, či různých algoritmů.

V úlohách se strojovým učením přistupuje stroj k obrazům jakožto k maticím. Jednotlivé hodnoty v této matici odpovídají hodnotám pixelů. Převod snímku na matici se nazývá **konverze**. Tato operace taktéž může proběhnout opačně a pole se konvertuje na snímek.

Detekce hran vezme obrázek a pomocí rozdílů v jasnosti a barvě pixelů detekuje hrany. Tento algoritmus zvýrazní všechny příslušné oblasti, nehledě na konečný tvar detekované hrany, na rozdíl od detekce čar, kdy finální detekovaný objekt nabývá tvaru pravidelné křivky (rozdíl lze lépe pozorovat na obr.2.4). Tyto zvýrazněné hrany lze dále použít v algoritmu detekce čar.

Houghova transformace² se nejčastěji používá k detekci pravidelných křivek, v tomto případě čar. Výhoda této metody je, že je relativně tolerantní vůči šumu v obrazu a mezerám v hranicích prvků. Často se před aplikováním této operace použije filtr (operátor) k detekci hran. Houghova transformace poté funguje přesněji. V dnešní době operace pro detekci hran a čar pracují na principu predikce díky strojovému učení.

²<https://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>



Obrázek 2.4: Viditelný rozdíl mezi detekcí hran (vlevo) a detekcí čar (vpravo). Zdroj:
<https://rubikscode.net/2022/06/13/thresholding-edge-detection-and-line-detection-with-opencv/>

2.3 Strojové učení

Pro bližší pochopení následující pasáže zde popíší pojmy související se strojovým učením [1]. **Model** strojového učení lze pochopit jako výsledný program, který je schopen zpracovat vstup a vydedukovat z něj výstup. Tato dedukce probíhá díky procesu učení, ke kterému dochází před nasazením modelu.

Dataset (někdy také *datová sada*) je množina dat, se kterými model interaguje. Tato data pak slouží k trénování a validaci modelu. Dataset může být také anotovaný, což znamená, že data jdou v páru s ideálním výstupem, kterého by měl model v nejlepším případě predikcí dosáhnout.

Učení modelu (někdy také *trénování modelu*) je proces, kterým si model prochází v procesu jeho tvorby. V procesu učení s učitelem se modelu zpřístupní data, model na nich učiní predikci a nakonec je seznámen se správnými výsledky. Model si následně sám upravuje vnitřní váhy neuronů a proces opakuje. Učení je pak rozděleno na několik epoch a je možné model zpětně dotrénovat. Pozor se však musí dát na tzv. **přeúčení** (*overfitting*). Tento jev se objevuje, když je model trénován na příliš velké množství epoch. Model se tímto učí pouze na datech, která mu jsou zpřístupněna, ale ztrácí schopnost činit precizní predikce na datech jiných. Opakem přeúčení je tzv. **nedostatečné učení** (*underfitting*), kdy model učením ještě nedosáhl nejlepšího možného výsledku. Musí se tedy dávat velký důraz na správnou délku učení modelu pro jeho nejlepší úspěšnost.

Aktivační funkce je matematická funkce, která vypočítává výstup neuronu na základě zadaných vstupů a vah. Existuje několik typů aktivačních funkcí a každá se hodí pro jinou úlohu. Často se využívá k evaluaci procesu učení.

Maticy³ je dvourozměrné pole hodnot či objektů. Pokud se pracuje s obrazem v digitální formě, většinou má formu právě matice. Pole matic se pak nazývá **tenzor** (viz obr. 2.5). Tyto tenzory se v modelu strojového učení vyskytují právě při přerozdělení obrazu na menší části v jednotlivých vrstvách.

Matici	Tenzor
$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\left[\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \right]$

Obrázek 2.5: Vizualizovaný rozdíl mezi maticí a tenzorem.

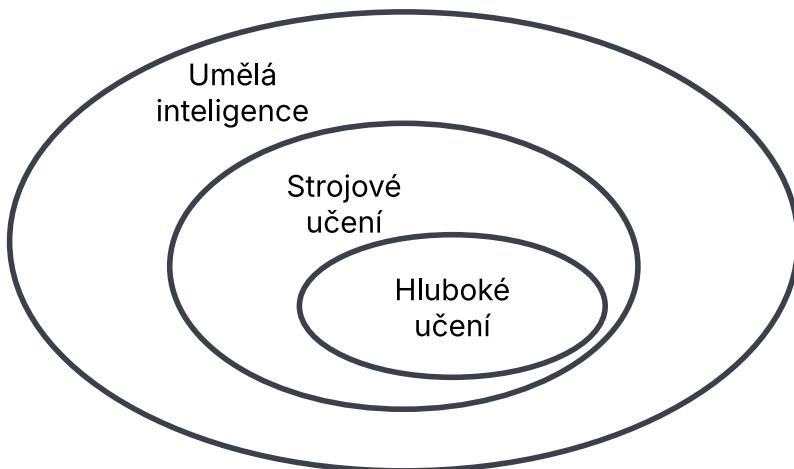
Kernel⁴ modelu se dá chápat jako jádro modelu. Používá se v algoritmech pro hledání vzorů v datech. Samotný kernel je matematická funkce, která převádí nelineární vztahy mezi daty do lineární podoby. Toto se hodí pro simplifikaci již zmíněných vztahů do podoby, se kterou mohou jiné funkce pracovat. Kernelsy se také využívají ve funkcích zpracování obrazu. Tyto funkce často fungují na principu modelu strojového učení.

Konvoluce je v matematice operace, která po aplikování na dvě funkce produkuje funkci třetí. Nejčastěji se používá v kernelech jednotlivých vrstev modelu strojového učení. Taktéž má své využití v algoritmech pro detekci hran.

³https://www.w3schools.com/ai/ai_algebra.asp

⁴<https://dida.do/blog/what-is-kernel-in-machine-learning>

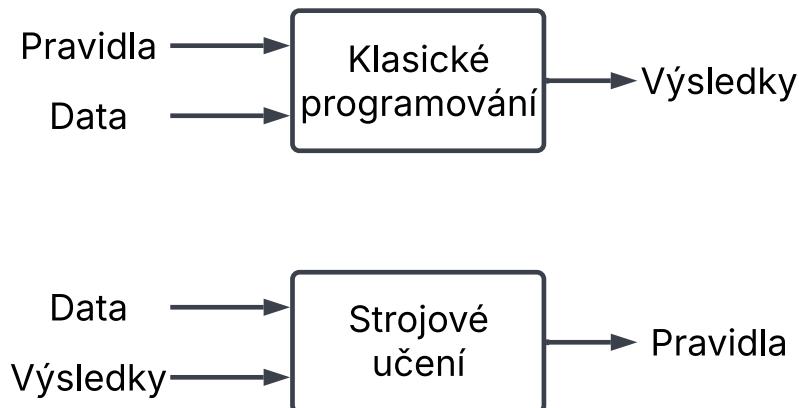
Umělá inteligence (*Artificial Intelligence – AI*) [1] se dá chápat jako automatizace intelektuálních úloh, které jsou běžně vykonávány člověkem. Jedná se o velký obor, pod který spadá třeba strojové učení (viz obr. 2.6). Učení však není to jediné, čím se obor zabývá. Například jedny z prvních implementací AI byly algoritmy pro hru šachy, které využívaly široké škály pravidel ručně načíslovaných programátorem. Tento postup definovaní pravidel se nazývá symbolické AI. Stal se však neefektivním při řešení komplexnějších problémů, kupříkladu klasifikace objektů z obrazu, kdy je potřeba opravdu rozsáhlé řady pravidel. Většina problémů je proto spíše řešena pomocí strojového učení.



Obrázek 2.6: Obrázek ukazující podobory umělé inteligence.

Běžný postup při tvorbě počítačového programu je zadání pravidel a dat počítači, který z nich poté vytvoří odpovědi (výsledky). **Strojové učení** (*machine learning – ML*)⁵ tento postup převrací, počítači se zadají data a odpovědi a on vytvoří pravidla, podle kterých k odpovědím z dat došel (vizualizace tohoto principu na obr.2.7). Model strojového učení je tedy natrénován, spíše než explicitně naprogramován. Obor úzce souvisí s matematickou statistikou, avšak jedná se o plně samostatnou disciplínu. Úspěšnost modelu strojového učení nezávisí pouze na implementaci, ale také na dostupných datech a efektivitě hardwaru (jedná se o výpočetně složitou disciplínu). Obor je poměrně mladý, začal se objevovat teprve v 90. letech 20. století. Navzdory tomu se však jedná o nejpopulárnější a nejúspěšnější podobor AI.

⁵<https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/>



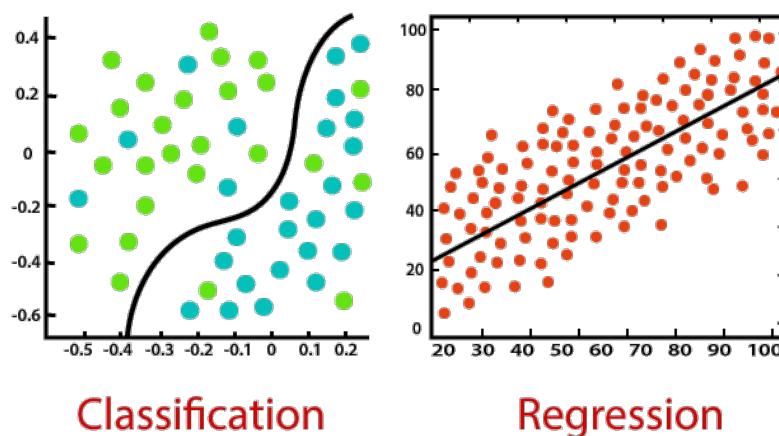
Obrázek 2.7: Obrázek ukazující princip strojového učení, oproti klasickému způsobu programování.

Existuje několik typů strojového učení, a každý z nich se hodí pro jiný typ úlohy. Pokusím se je tedy pro bližší pochopení popsat.

Učení s učitelem (*supervised learning*)

Stroje jsou trénovány na anotovaných datasetech. Data a příslušící odpovědi jsou pevně zadány, stroj tedy musí „jenom“ vytvořit pravidla pro práci s nimi. Jelikož ve svém řešení užívám anotovaného datasetu, lze říci, že mnou vytvořený model je trénovaný na principu učení s učitelem. Tento postup se dále dělí na dva typy, regresi a klasifikaci.

Regresivní algoritmy zvládají problémy, kde vstupní data mají s výstupními odpověďmi lineární vztah. Program, který bude předpovídат počasí, tedy využije postup regrese. Pod klasifikaci lze poté zařadit algoritmy, které řeší klasifikační úlohy. Stroj rozhoduje, do jaké třídy zařadit vstupní data (viz obr. 2.8). Příkladem je například úloha segmentace, o které tato práce pojednává. Stroj rozhoduje, kde v obrazu se nachází daný objekt a kde ne.



Obrázek 2.8: Grafy popisující rozdíl mezi klasifikací (vlevo) a regresí (vpravo), Zdroj: <http://www.javatpoint.com/regression-vs-classification-machine-learning>

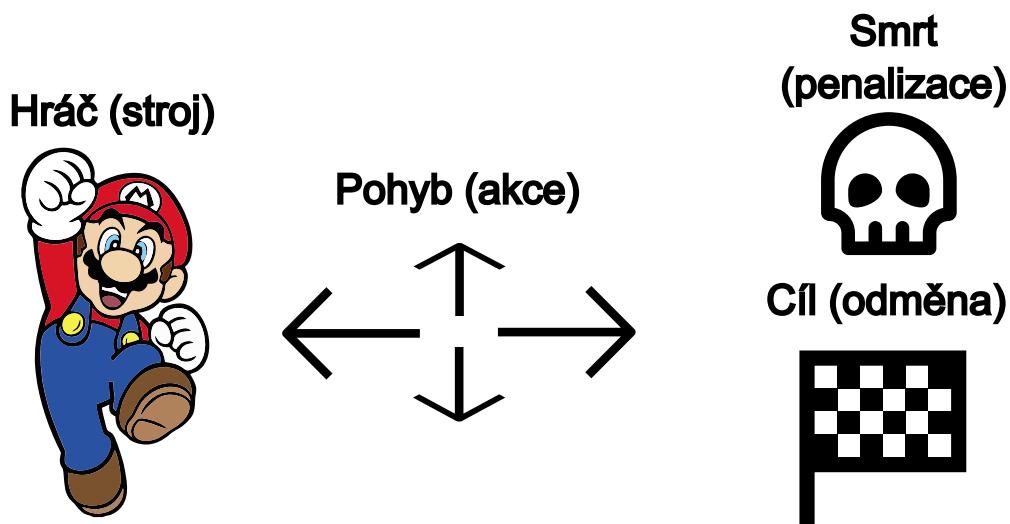
Učení bez učitele (*unsupervised learning*)

Stroje jsou trénovány na datech, která nejsou anotována. Stroj tedy sám najde vzory podobností na datech a vyvodí z nich výstup. Zase jde zde rozlišit dva typy, shlukování a asociaci.

Princip shlukování je, že stroj spojí objekty do shluků podle jejich parametrů, jako jsou jejich podobnosti či rozdíly. Asociace se týká vztahů mezi proměnnými u velkého souboru dat. Určuje závislost různých datových položek a mapuje související proměnné.

Zpětnovazební učení (*reinforced learning*)

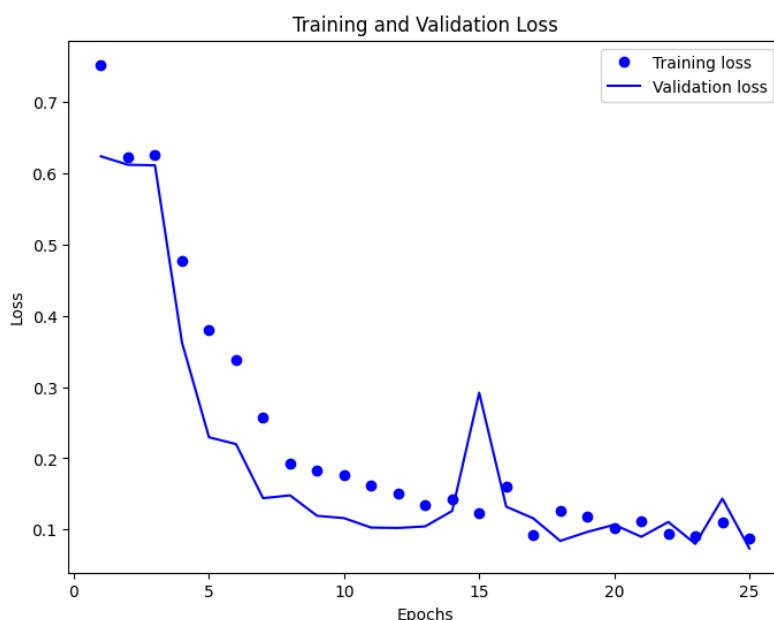
Poslední typ strojového učení využívá principu zpětné vazby. Stroj se učí postupem pokus-omyl. Důležité u tohoto postupu je definování odměny, která se stroji poskytne za dobré výsledky, a definování penalizace, která se stroji udělí za špatné výsledky. Stroj se tedy neučí z anotovaných dat, ale z vlastních zkušeností. Skvělým příkladem využití tohoto principu jsou projekty, snažící se AI naučit hrát videohry (viz obr. 2.9).



Obrázek 2.9: Příklad definování akcí, trestů a odměn pro hru Super Mario Bros.

Je důležité před použitím ověřit úspěšnost modelu. Toto ověření může dělat člověk osobně, ale také je možné, aby se model kontroloval sám. Existují *evaluační techniky*, které lze provést již při přípravě datasetu, ale také techniky, které se provádějí až ve finále, po naučení modelu. Většinou se hodnotí přesnost, preciznost či senzitivita modelu.

Přesnost modelu lze vyhodnotit **ztrátovou (loss) funkcí** (viz obr. 2.10). Funkce spočítá, jak správně model za danou epochu učení učinil predikce. Model může pak pomocí této funkce upravovat své váhy při procesu učení. Graf této funkce vyobrazuje, jak se měnila přesnost modelu v průběhu trénování. Pomocí tohoto grafu můžeme poznat, zda se model nedostává do fáze tzv. přeúčení (overfitting). Graf ztrátové funkce napomáhá hlídat tyto možné nedostatky.



Obrázek 2.10: Snímek grafu ztrátové funkce mého modelu

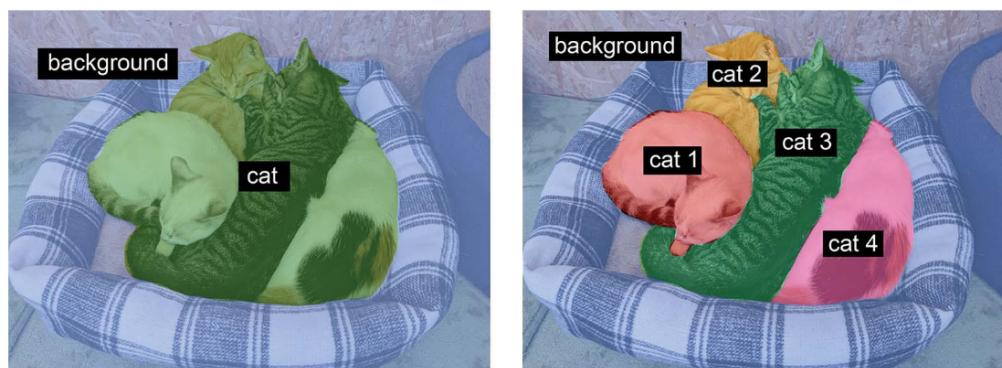
Hluboké učení [1, 7] je podobor strojového učení. Model se zde skládá z několika vrstev (neuronů), kde každá vrstva provádí odlišnou funkci. Některé vrstvy pouze zpracovávají vstup do lepší podoby, zatímco jiné upravují podobu výstupu. Takhle jsou vrstvy postupně skládány za sebe, dokud nevznikne hotový model. Počet vrstev se nazývá hloubka – odtud název hluboké učení. Vrstvy lze také někdy nazvat neurony a celý model pak nese název neuronová síť. Nejjednodušší model neuronové sítě je *Perceptron*⁶. Tento model byl navržen v roce 1957 a sestává z jediného neuronu. Ve svém programu pak využívám architekturu *konvoluční neuronové sítě (CNN)*, hodí se totiž k analýze obrazových dat.

⁶<https://cs.wikipedia.org/w/index.php?title=Perceptron&oldid=23103258>

2.4 Segmentace obrazu

Ve svém projektu se zabývám primárně segmentací obrazu [1]. Segmentace rozděluje každý pixel snímku do tříd, většinou na popředí, pozadí a přechod mezi nimi. Výstupem segmentace je maska obrazu, čili snímek, kde pixely nabývají hodnoty korespondující klasifikátoru třídy.

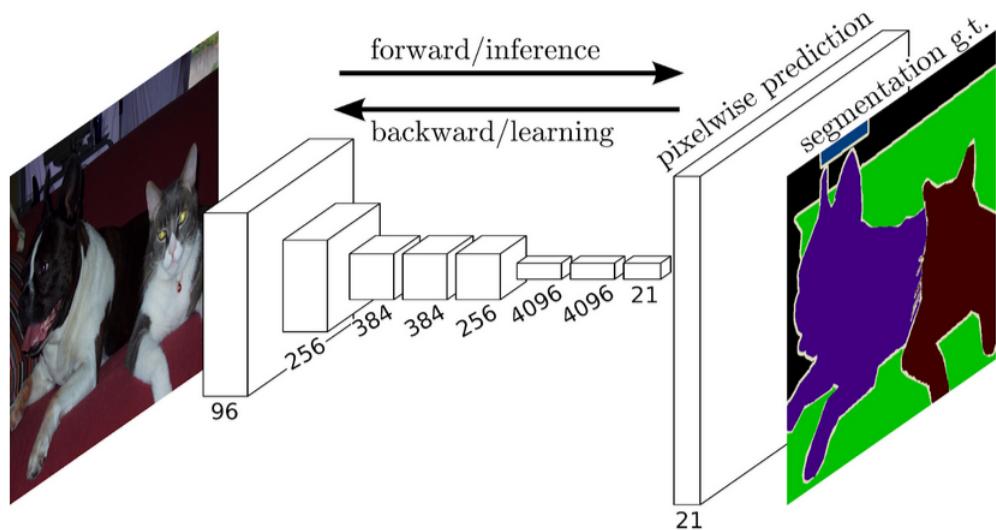
Segmentaci dále dělíme na segmentaci sémantickou a segmentaci instancí (viz obr. 2.11). Sémantická segmentace neřeší, kolik odlišných objektů stejné třídy je na snímku, vše řadí jako stejnou třídu. Segmentace instancí naopak pracuje s detekováním odlišných objektů stejné třídy. Na finálním výstupu tedy vyjde maska s více třídami, i když se jedná o objekt stejného typu.



*Obrázek 2.11: Příklad sémantické segmentace (vlevo) a segmentace instancí (vpravo).
Zdroj: [1]*

Segmentaci je možné implementovat pomocí metod zpracování obrazu, nebo metod strojového učení⁷. Obě metody mají své výhody i nevýhody a je nutné rozhodnout se, jaká metoda bude pro daný problém vhodnější. Metody zpracování obrazu produkují výsledky analýzou inherentních vlastností pixelů, jako je barva či intenzita. Mezi známé techniky patří prahování, detekce hran, či použití histogramů. Metody strojového učení naopak využívají neuronové sítě pro hledání sofistikovaných vzorů v obrazu. Mezi běžně používané architektury modelů se řadí *Plně konvoluční sítě* (viz obr. 2.12), architektura *U-Net*, či architektura *Mask R-CNNs*.

⁷<https://www.ibm.com/think/topics/image-segmentation>



Obrázek 2.12: Vizualizace architektury plně konvoluční sítě. Zdroj:
<https://paperswithcode.com/method/fcn>

2.5 Existující nástroje a knihovny

K napsání programu je použit jazyk **Python** [8], konkrétně verze Python 3. Jedná se o vysokoúrovňový interpretovaný programovací jazyk navržený v roce 1991. Podporuje dynamickou kontrolu datových typů, nebo třeba objektově orientované programování. Python je také uživatelsky přívětivý a má poměrně jednoduchou syntaxi. Zvolil jsem si jej, jelikož je hojně zastoupen v problémech strojového učení a zpracování obrazu. Je také hodně rozšířený a součástí je spousta uživateli vytvořených knihoven pro usnadnění práce. Jako instalátor balíčků knihoven používá pip. Zdrojový kód Pythonu je volně dostupný.

Jelikož je trénování modelu pro detekci objektů poněkud náročné na výpočetní techniku, celý program je tvořen v cloudovém prostředí *google colaboratory*⁸. Toto prostředí dovoluje použití externí techniky k výpočetním operacím. Zdroje, které lze využít, jsou však omezené a pro jejich větší škálu je nutné pořízení prémiové verze aplikace.

Knihovny v Pythonu jsou většinou napsány v kompliovaném jazyce C. Tento jazyk je rychlý a lehký a Python slouží pouze ke spouštění funkcí z knihoven. Knihovny slouží jako soubory funkcí, metod, objektů, či tříd, které se používají k usnadnění práce uživatele. Pro přehlednost zde jmenuji hlavní knihovny, které ve svém programu používám.

⁸<https://colab.google/>

NumPy⁹ je knihovna nejčastěji používaná k vědeckým výpočtům. Slouží k definici n-rozměrných polí a práci s nimi (výpočty, logické operace, tvarové manipulace, ...). Ve svém programu ji používám k snadnějšímu přístupu k obrazu ve formě matice a úpravě některých hodnot obrazu.

OpenCV, neboli „open computer vision“, je největší knihovna k zpracování počítacového vidění. Používám ji ke čtení a následné práci s obrazem.

Scikit-learn slouží k řešení úloh s využitím strojového učení. Z knihovny používám metodu k načtení a rozdelení dat do finálních datasetů, které používám k trénování a evaluaci modelu. Proces implementace strojového učení pak implementují za pomocí knihovny TensorFlow.

MatPlotLib je rozšířená knihovna, používaná k vizualizaci dat. Lze pomocí ní zobrazit data v různých typech grafů a tabulek. Ve svém programu ji používám k zobrazení grafu ztrátové funkce za epochy trénování modelu strojového učení.

Knihovna **TensorFlow**¹⁰ běžně slouží k vytváření modelů pro strojové učení. Součástí je spousta již předpřipravených modelů, datasetů a příkladů pro práci s nimi. V programu tuto knihovnu využívám k operacím s modelem strojového učení, konkrétně definici, komplikaci a trénování modelu strojového učení. Taktéž pomocí vestavěných funkcí transformuji obrázek do formy pole. Obraz v této formě je totiž model schopen zpracovat.

⁹<https://numpy.org/>

¹⁰<https://www.tensorflow.org/>

3 Detekce FVP v obrazu z dronu

V této části nejdříve srovnám metody, pomocí kterých se dá problém řešit. Poté navrhuji řešení, které by mohlo být použito k vyřešení daného problému. Nejedná se však o doslovný popis tvorby programu z mé praxe, tomu se věnuji v následující kapitole.

3.1 Model CNN vs. Zpracování obrazu?

Volba metody využívající **strojové učení** je závislá na datasetu. Pokud je k dispozici dataset, který obsahuje spoustu dat, na kterých lze model natrénoval, lze zaručit, že model bude hodně precizní. Taktéž lze změnou některých parametrů v architektuře modelu model vylepšit a lze mít několik verzí modelu celkem rychle připravených, to se pak hodí u testování a ladění programu. Modely také lze dle potřeby dotrénovávat. Strojové učení se taktéž hodí k detekci objektů nepravidelných tvarů. Stroj si sám najde určitý vzor, podle kterého pak postupuje a programátor nemusí řešit, dle jakých parametrů se detekce děje. Dobrý příklad je detekce lidí či zvířat v pohybu.

Nevýhodou strojového učení, se kterou jsem se potýkal asi nejvíce, je příprava/volba vhodného datasetu. Příprava vlastního datasetu je velmi časově náročná a jedná se o tzv. „myší práci“. Na internetu lze naštěstí najít spousty datasetů již připravených¹. V problematice detekování jednotlivých FVP jsem však nebyl schopen najít jeden vhodný dataset, což se ve finále odrazilo na výsledku. Trénování modelu je taktéž náročné na výpočetní techniku. V dnešní době naštěstí existují cloudové služby, které dovolují si výkonnější techniku na dálku „vypůjčit“ a použít. Tyto cloudové služby však mají omezený plán použití zdarma, v případě trénování pokročilejšího modelu tyto omezené zdroje nemusí stačit. Několikrát jsem se potýkal s nedostatkem paměti RAM, které je k trénování modelu potřeba vysoké množství.

Výhodou tradiční metody **zpracování obrazu** je široká škála různých operací, metod a filtrů, které lze použít. Pokud tedy nebude jedna metoda vhodná, je možné, že jiná metoda, která bude fungovat na podobném principu, dosáhne ve finále lepších výsledků. Spousta těchto metod je již implementována v různých knihovnách na práci s obrazem.

¹<https://www.kaggle.com/>

Tyto operace také mohou být lehce modifikovány a výsledky testování mohou být rychle a snadno zhodnoceny.

Stejně jako strojové učení je zpracování obrazu poněkud náročné na výpočetní techniku. Jelikož se pracuje s každým pixelem v obrazu, velikost a rozlišení snímku hrají velkou roli v časové náročnosti programu. Stále je však metoda zpracování obrazu většinou méně náročná na výpočetní techniku, než postup využívající strojové učení. Metody zpracování obrazu se také nemusí hodit pro každý typ úlohy, pokud je úkolem detektovat např. nepravidelné tvary, které se mění z obrázku na obrázek, je lepší využít strojové učení.

Oba přístupy se tedy hodí na jiný typ úloh. Řešený problém je detektovat jednotlivé FVP, které jsou na snímku seřazeny. Díky seřazení lze využít operace s obrazem a detektovat hrany panelů. Tento postup je méně náročný na výpočetní techniku a není nutná příprava vhodného datasetu. Pokud však je k dispozici vhodný dataset, je možné problém řešit i pomocí strojového učení. Zde je výhodou, že stroj je schopen detektovat i různé anomálie, které by se mohly na snímku vyskytnout (větší úspěšnost modelu). Oba postupy jsou tedy validní a lze je použít pro řešený problém. Ve svém finálním řešení spojuji tyto dvě metody do jedné. Metody zpracování obrazu a metody strojového učení jdou ruku v ruce a společně vytvářejí obor zvaný počítačové vidění.

3.2 Segmentace FVP pomocí CNN

Před líčením návrhu na přípravu celého programu je nutno zmínit, že tento postup je navržen na konkrétní dataset, který je v práci používán. V tomto datasetu jsou panely zarovnány, čemuž tak většinou na originálních snímcích není. Toto zarovnání se většinou dělá dále v předzpracování dat. Pokud by se tedy tento návrh měl aplikovat na jiná data, je zde vysoká šance, že bez úprav nebude fungovat.

Dataset

Pro funkční program je potřeba funkčního datasetu. Ten se používá jak při učení modelu strojového učení, tak při výsledném zhodnocení výsledků. Vhodný dataset by měl obsahovat data ve vysoké kvalitě a k nim odpovídající anotace. Data by také měla mít různé podoby, aby se model přizpůsobil určitým anomáliím, ke kterým by mohlo dojít. Taktéž je vhodné použít opravdu obsáhlý dataset, pokud to používaná technika umožňuje.

Obecně se doporučuje dataset vyhledat, tvorba datasetu vlastního je totiž velmi časově nákladná. Na internetu existuje spoustu portálů, kde je možné vhodný dataset vyhledat.

V mém případě právě využiji dataset již vytvořený. Data v tomto datasetu by měla splňovat podmínky, které jsem zmínil výše.

Před začátkem trénování modelu se dataset musí předzpracovat. Data je nutné rozdělit na *trénovací, validační a testovací sady*. Pomocí trénovacích dat bude model natrénovan a poté pomocí validačních ohodnocen. Nakonec se úspěšnost modelu ověří podle testovacích dat, se kterými model za celý čas trénování nepřišel do styku. Obvykle se data rozdělí v poměru 70:20:10.

Pokud je k dispozici málo dat, nebo jsou snímky moc velké, můžou se jednotlivá data ořezem rozdělit na menší segmenty. Tento princip se nazývá *augmentace datasetu*². Jelikož se data rovnoměrně oříznou, vznikne více segmentů jednoho obrazu se všemi detaily zachovanými. Nesmí se však zapomenout na oříznutí dat i jejich anotací a správné uložení těchto souborů.

Taktéž lze k získání většího množství detailů aplikovat různé filtry na snímek. Díky tomu může z obrazu vyniknout větší množství detailů, které by jinak zůstaly skryty. Musí se však dát pozor, aby se obraz netransformoval moc. Mohl by vzniknout nechtěný šum a důležité detaily by mohly zaniknout.

Nakonec se musí snímky v datasetu převést do formy, kterou může model využít k trénování. Obrazy se tedy převedou na dvojrozměrná pole hodnot (matice). V tomto kroku také může dojít k normalizaci masek. Některé masky totiž mohou nabývat hodnot mimo hodnotící škálu modelu a cílem programátora je, aby tyto hodnoty zase do škály vrátil.

Model

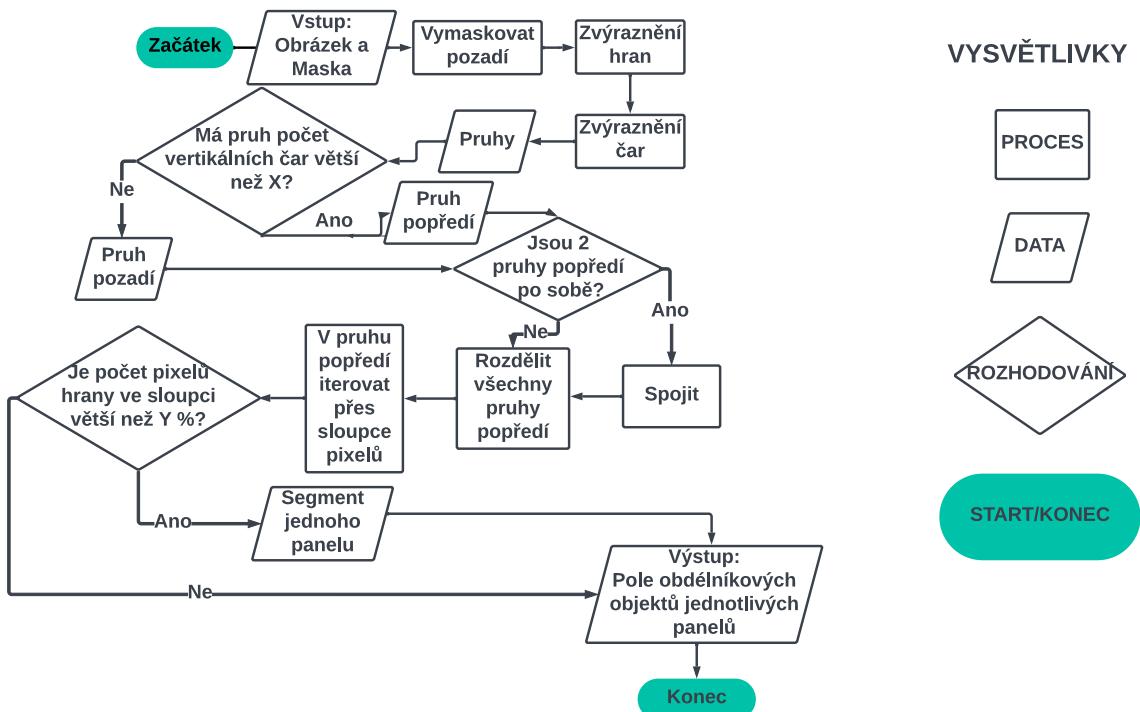
Princip *trénování modelu* spočívá v tom, že probíhají tzv. epochy, kdy se model sám trénuje na datech, která mu byla dána k dispozici. V každé epoše se model zlepšuje, ale musí se dohlédnout, aby nedošlo k přeúčení. Model by pak měl skvělé výsledky na trénovacích datech, ale v praxi by výsledky byly neadekvátní. Na tento problém lze dohlédnout pomocí grafu ztrátové funkce modelu.

Posledním krokem je *vyhodnocení modelu*. Model se zhodnotí pomocí testovacích dat a vyhodnotí se jeho úspěšnost. Taktéž je vhodné zobrazit graf ztrátové funkce pro každou epochu a zjistit, zda někde nedochází k přeúčení nebo k nedostatečnému učení. Pokud bude model naučen nedostatečně, lze jej ještě doučit. Taktéž se doporučuje pozměnit některé parametry modelu a zkoumat získat lepší výsledek tímto způsobem.

²<https://medium.com/@pouyahallaj/data-augmentation-benefits-and-disadvantages-38d8201aeaf>

3.3 Detekce FVP pomocí metod zpracování obrazu

V předešlém kroku se v ideálním případě vytváří snímek s vymaskovanou oblastí, ve které se nachází FVP. Tento snímek se dále používá v tomto kroku jako vstup. Následuje implementace řešení (viz obr. 3.1) a výstupem je snímek, na kterém lze pozorovat jednotlivé vyznačené panely.



Obrázek 3.1: Vývojový diagram procesu

Detekce pruhů panelů

Pokud je k dispozici maska, která ukazuje, kde přesně se na snímku nacházejí oblasti s panely, lze ji využít v předzpracování. Aplikací masky na snímek zaniknou zbytečné detaily v pozadí a lze tak dosáhnout lepších výsledků při detekci hran, což je použito v následujícím kroku. Pozadí však občas obsahuje detaily důležité, proto je nutné otestovat, zda program dosáhne lepších výsledků s maskováním či bez.

Jelikož se na snímku vyskytují oblasti panelů i pozadí, musí se dále zjistit, kde se nachází oblast pozadí a kde oblast popředí (panely). Stále je tu však šance, že predikce nebude precizní. Lze tedy využít detekce hran k rozdelení pruhů popředí a pozadí.

Nejdříve se na obrázek aplikují filtry pro zjednodušení detekce čar. Tyto filtry by měly redukovat přebytečný šum a zvýraznit hrany, které udávají hranice pruhů. Poté lze využít

filtru pro detekci horizontálních čar a výsledkem jsou hranice pruhů panelů a pozadí. Odlišit je od sebe se dá pomocí filtru pro detekci vertikálních čar. Jelikož jsou panely seřazeny, bude v pruhu s panely více vertikálních čar než v pruhu s pozadím.

Je možné, že středy některých pruhů panelů se detekují jako hranice pruhu a některé ne, pro správné fungování programu se tomuto musí předejít. Nejdříve se výše zmíněnou detekcí zjistí, zda následují dva pruhy popředí po sobě, pokud ano, spojí se. Takto se bude iterovat přes všechny pruhy. Následně se všechny pruhy popředí rozdělí ve středu na dva, jeden pruh tedy bude obsahovat pouze jeden pás panelů a ne dva pásy, jak tomu mohlo dojít před touto implementací.

Detekce jednotlivých panelů

Nyní jsou k dispozici pouze pruhy popředí. Pro detekci jednotlivých panelů se stačí po pruhu posouvat po horizontální ose a v každé části zjistit, kolik pixelů má vertikální hrana (pokud se zde nachází). Pokud to bude více než určitý zlomek z celkového počtu pixelů v této části, vyznačí se hranice segmentu. Takhle je možné se posouvat po horizontální ose a na snímku tak vykreslit hranice jednotlivých segmentů. Tyto segmenty pak představují jednotlivé panely.

Je však možné, že v pruhu segmentů panelů se nachází i segment pozadí. Pro naprostou jistotu, že k tomu nedojde, se použije jednoduchý model pro klasifikaci popředí a pozadí. Nejdříve se ručně sepíší hodnoty některých segmentů popředí a segmentů pozadí. Z těchto segmentů se poté vytvoří histogram, který se anotuje, vznikne tak dataset. Nakonec se vytvoří model, který se natrénuje na těchto anotovaných datech a následně se použije na celý snímek. Výsledkem bude obraz, na kterém budou jednotlivé fotovoltaické panely precizně vyznačeny.

4 SW nástroj pro detekci FVP

Tato kapitola pojednává o tom, jak přesně probíhala tvorba programu, na jaké problémy jsem narazil a popřípadě na jaké nové poznatky jsem tímto praktickým bádáním přišel.

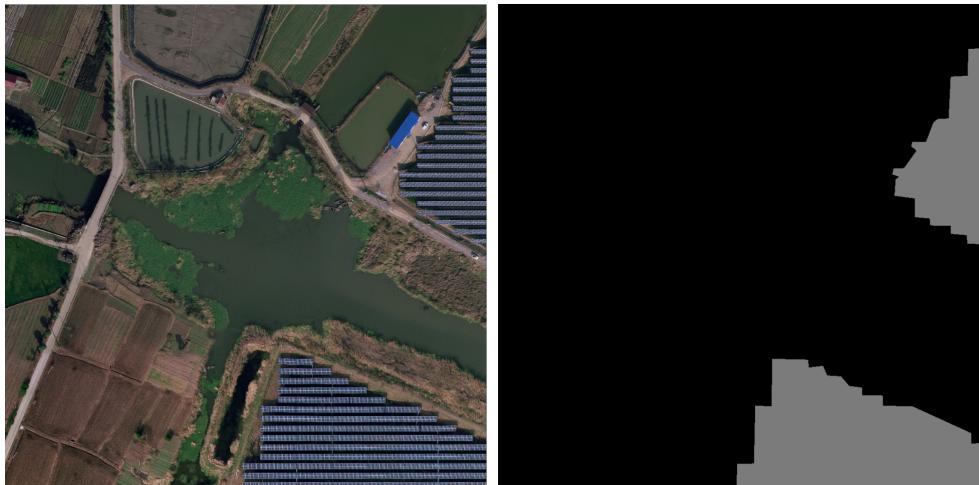
4.1 Dataset

Volba datasetu

Dataset jsem se pokusil najít na internetu, avšak vhodný exemplář se mi nalézt nepodařilo. Hledal jsem primárně na stránce Kaggle¹, která slouží jako databáze různých datasetů a příkladů jejich použití. Nejvhodnější dataset, který jsem našel, anotuje panely do jedné velké skupiny. Model tedy nedetektuje jednotlivé kusy, ale oblasti, kde se na snímku panely nachází. Kvůli neschopnosti nalézt vhodný dataset proto musím v druhé části programu použít jinou metodu než strojové učení. Pro přesnou detekci jednotlivých panelů je možné vytvořit dataset vlastní, jedná se však o náročný a zdlouhavý proces, který jsem vyhodnotil jako nevhodný.

Struktura datasetu obnáší několik adresářů, ve kterých se nachází snímky a jejich anotace (viz obr. 4.1). Každý adresář obsahuje snímky panelů s různými typy pozadí. Celkově se v datasetu nachází něco málo přes 4600 souborů a má velikost 5,22 GB ve formě zazipovaného archivu. Snímky a jejich masky mají velikost 1024 na 1024 pixelů a obrazový formát „.bmp“, kvalita je tedy dostatečná. Počet snímků s anotacemi je také vysoký a dataset lze tedy použít k natrénování modelu.

¹<https://www.kaggle.com/datasets/salimhammadi07/solar-panel-detection-and-identification>



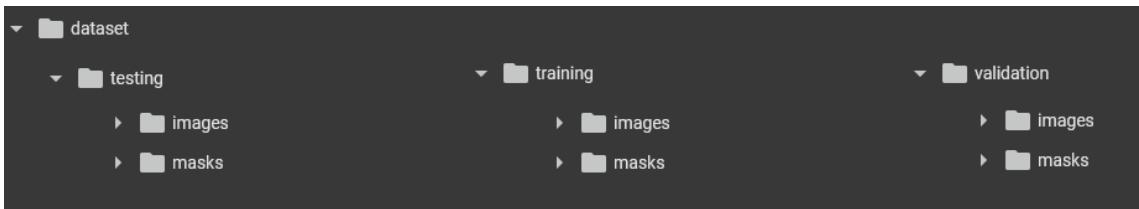
Obrázek 4.1: Příklad snímku a masky z datasetu (v tomto pořadí)

Rozdělení dat do adresářů

Data v datasetu zatím nejsou rozdělena přesně tak, jak to mnou používaný model vyžaduje. Je nutné je proto předzpracovat. Dataset nejdříve nahrávám na osobní cloud a poté do prostředí *google colaboratory*, ve kterém probíhá tvorba celého programu. Toto prostředí dovoluje použití externího hardwaru, což se při trénování modelu hodí, jedná se totiž o proces velmi náročný na techniku. V programu nejdříve dataset rozbaluji a nahrávám jeho obsah na disk prostředí *google colaboratory*. Tím se zajistí, že data jsou dostupná po celý běh programu a lze s nimi proto pracovat.

Nahraná data poté filtruji na soubory snímků a soubory anotací (masek snímků). Masky od snímků se lze odlišit jednoduše, všechny soubory masek mají jméno, které se řídí podle vzoru „*názevSouboru-label.bmp*“, soubory se snímků pouze „*názevSouboru.bmp*“. Názvy těchto souborů následně ukládám do dvou polí, která jsou poté abecedně seřazena. Zaručí se tak, že snímků s indexem *i* bude odpovídat správná maska.

Takto rozdělená data se ještě rozdělují na sadu trénovací, validační a testovací. Pro tyto sady vytvářím nové adresáře, do kterých data následně přerozděluji. Využívám k tomu funkci `train_test_split()` z knihovny *sklearn*. Funkce vezme data a náhodně je rozdělí v zadaném poměru. Tuto funkci používám celkově dvakrát, nejdříve pro rozdělení všech dat na sadu trénovací a validační v poměru 80:20 a poté na rozdělení trénovacích dat na sadu trénovací a testovací v poměru 88:12. Snímků a jejich masky pak nahrávám do jejich nových adresářů. Výsledkem jsou složky, obsahující data a masky rozdělené na trénovací, validační a testovací sady (viz obr. 4.2). Tyto sady lze následně využít k učení a zhodnocení modelu.



Obrázek 4.2: Hierarchie nově vytvořených adresářů

Oříznutí dat

Architektura modelu, který jsem si pro práci zvolil, přijímá data pouze ve formátu 200 na 200 pixelů. Jelikož jsou snímky o dost větší, oříznu je a k práci s modelem použiji vzniklé ořezy. Oříznutí používám z důvodu zachování detailů, jestliže bych snímky pouze zmenšil pomocí škálování, důležité detaily by zanikly a model by nebyl tak přesný. Pokud snímek 1024 na 1024 ořezem rozdělím na 4 stejně velké části, vzniknou mi snímky velikosti 512 na 512 pixelů. Pokusil jsem se také data oříznout na velikost 256 na 256 pixelů (což jsou hodnoty mnohem bližší k požadované velikosti), avšak setkal jsem se s problémem nedostatku zdrojů v prostředí Colab. Při trénování modelu se proces nečekaně ukončí s chybovou hláškou „nedostatek paměti RAM“. V důsledku nedostatku zdrojů proto výsledné ořezy mají velikost 512 na 512 pixelů, i když výsledný snímek musím před trénováním modelu více než dvakrát zmenšit. Taktéž se nesmí zapomínat na oříznutí jak snímků, tak k nim korespondujících masek.

Oříznutí funguje na jednoduchém principu. Nejdříve iteruji přes všechny snímky a jejich masky v příslušném adresáři. Pomocí knihovny *OpenCV* si obrazové soubory otevím, aby byly připraveny ke zpracování pomocí této knihovny. Oříznutí pak funguje zjištěním dimenze obrazu. Tyto souřadnice následně rozděluji v půlce a pomocí těchto hodnot definují dimenze nových snímků (viz obr. 4.3). Jelikož stroj přistupuje k obrazu jako k matici, stačí mi vytvořit nové pole formy matice velikosti nově vzniklých dimenzí a hodnotami pixelů stejnými jako v originálním snímku. Obrazy pak pomocí *OpenCV* přepisují do adresáře, který používám čistě pro ořezaná data. Tento postup aplikuji na všechna data. Vznikají tak adresáře s oříznutými daty, se kterými následně pracuji při učení modelu.



Obrázek 4.3: Vizualizovaný princip oříznutí pro lepší představu (vlevo originální snímek, vpravo vzniklé ořezy)

V kroku rozdelení dat do adresářů originální snímky stejně jako ořezy ukládám. Oříznuté snímky využívám při učení modelu a originální snímky slouží k pozdější detekci jednotlivých FVP pomocí metody zpracování obrazu. Je proto potřeba vytvoření obou adresářů, i když tato data zabírají hodně místa na disku. Pokud by se při učení modelu dalo využít pouze dat v originálním měřítku, výsledný program by nevyžadoval takto mnoho místa na disku.

Konverze snímků na matice

Nyní jsou připraveny adresáře s oříznutými daty, stále v obrazovém formátu „.bmp“. Šlo by s nimi již od začátku pracovat jako s maticemi, jistě by to ušetřilo místo na disku. Jelikož však s nimi v grafické podobě později ještě pracuji, konverze na matice probíhá jako poslední bod procesu před trénováním modelu. Hodnoty masky se také musí normalizovat na hodnoty, které model správně zpracuje.

Konverze proběhne pomocí funkcí `img_to_array()`, `load_img()` knihovny *tensorflow*. Data se nejdříve načítají a škálují na požadovaných 200 na 200 pixelů pomocí funkce `load_img()`. U masek se také aplikuje konverze na typ „`uint8`“ (malá celá čísla). Následná normalizace masky přepíše hodnoty pixelů masky velikosti 127 na hodnotu 1 a hodnoty velikosti 256 se mapují na hodnotu 2. Normalizaci provádí, jelikož masky z používaného datasetu nenabývají stejných hodnot jako model. Validace procesu učení by tak probíhala špatně a úspěšnost modelu by byla katastrofální. Po normalizaci masky konvertuji snímky na matice, která je uložena do pole. Na finální pole je ještě aplikována funkce `np.array()` z knihovny *Numpy*. Převede se tím na formát, se kterým může model pohodlněji pracovat. Výsledkem jsou tedy pole matic, která symbolizují obrazy a masky. Tato pole použijí k trénování modelu.

4.2 Model strojového učení

K řešení tohoto problému nejdříve pracuji se strojovým učením. Strojové učení je však velmi složitá disciplína, proto jsem se ještě před začátkem psaní práce seznámil se základy segmentace, což je přístup vhodný pro detekci objektů na snímku. Zvolil jsem si knihu *Deep learning in python* [1], která popisuje, jak začít s hlubokým učením. V knize je popsán příklad segmentace na datasetu s domácími mazlíčky, příklad jsem si proto pro procvičení udělal. Model, jenž ve svém finálním programu používám, je dost podobný modelu v tomto příkladu. Oba modely se totiž věnují sémantické segmentaci a přijde mi proto vhodné tuto architekturu modelu použít.

Definice a komplikace modelu

Dataset je úspěšně připraven, nyní stačí zajistit implementaci modelu. Jako první krok je třeba model definovat a zkompilovat. K práci s modelem využívám širokou knihovnu *Tensorflow*, konkrétně její aplikační rozhraní *Keras*². Tyto nástroje byly navrženy pro práci se strojovým učením a je k nim dostupná rozsáhlá dokumentace.

Jako vstupní vrstvu modelu používám funkci `keras.Input()`. Díky ní se vytváří instance tenzoru `keras`, který reprezentuje vícerozměrná data. Následně převádí hodnoty do vhodnějšího rozsahu vrstvou `keras.layers.Rescaling()`. Následují tři instance páru konvolučních vrstev `keras.layers.Conv2D()`. Tyto vrstvy se liší parametry, ale všechny používají aktivační funkci *relu* a velikost kernelu 3. Funkce těchto vrstev je „rozložit“ snímek na menší části a najít v těchto částech vzory vhodné pro detekci panelů. Snímek se poté pomocí tří párů vrstev `keras.layers.Conv2DTranspose()` „složí“ zpět. Jako finální výstupní vrstvu jsem zvolil `keras.layers.Conv2D()`, využívající aktivační funkci *softmax* (viz obr. 4.4). Tato aktivační funkce se hodí pro úlohy segmentace s výstupem obsahujícím 3 hodnoty (popředí, hranice, pozadí). Zkoušel jsem také konverzi hodnot masky na pouze dvě hodnoty, avšak výsledky byly horší. V případě pouze dvou hodnot masky je také nutné použít jiné aktivační funkce.

²https://www.tensorflow.org/api_docs/python/tf/keras

Finální model vytvořím vložením popsané architektury vrstev modelu jako parametr funkce keras.Model(). Ve finále vzniká nenaučený model, který je po natrénování schopen predikce masky obsahující hodnoty popředí, hranice a pozadí.

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 200, 200, 3)	0
rescaling (Rescaling)	(None, 200, 200, 3)	0
conv2d (Conv2D)	(None, 100, 100, 64)	1,792
conv2d_1 (Conv2D)	(None, 100, 100, 64)	36,928
conv2d_2 (Conv2D)	(None, 50, 50, 128)	73,856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147,584
conv2d_4 (Conv2D)	(None, 25, 25, 256)	295,168
conv2d_5 (Conv2D)	(None, 25, 25, 256)	590,880
conv2d_transpose (Conv2DTranspose)	(None, 25, 25, 256)	590,880
conv2d_transpose_1 (Conv2DTranspose)	(None, 50, 50, 256)	590,880
conv2d_transpose_2 (Conv2DTranspose)	(None, 50, 50, 128)	295,840
conv2d_transpose_3 (Conv2DTranspose)	(None, 100, 100, 128)	147,584
conv2d_transpose_4 (Conv2DTranspose)	(None, 100, 100, 64)	73,792
conv2d_transpose_5 (Conv2DTranspose)	(None, 200, 200, 64)	36,928
conv2d_6 (Conv2D)	(None, 200, 200, 3)	1,731
Total params: 2,880,643 (10.99 MB)		
Trainable params: 2,880,643 (10.99 MB)		
Non-trainable params: 0 (0.00 B)		

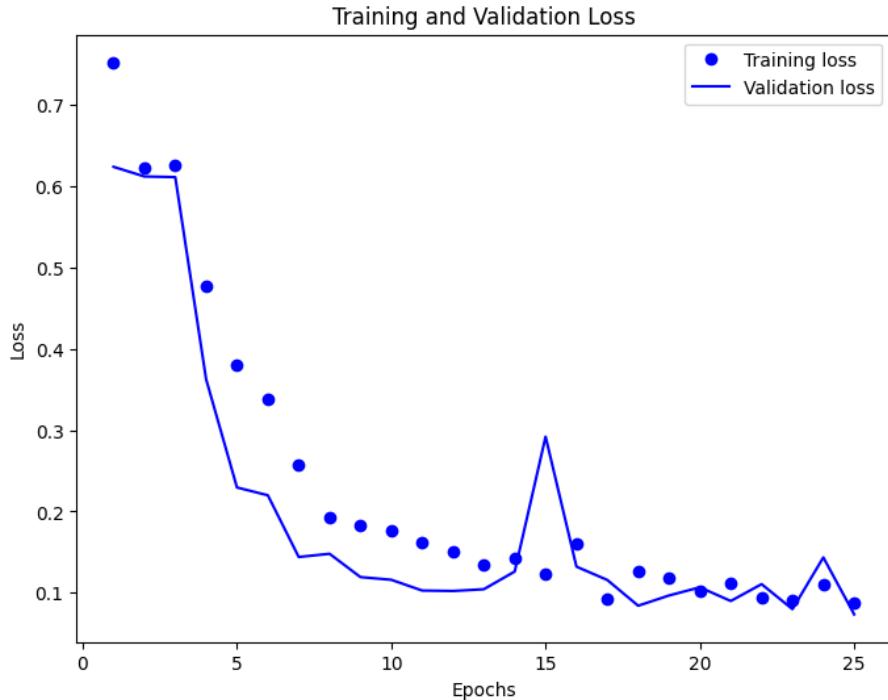
Obrázek 4.4: Architektura modelu

Model je ještě potřeba zkompilovat. Ke kompliaci modelu používám optimizátor *rmsprop* a ztrátovou funkci *sparse_categorical_crossentropy*. Používaný typ optimizátoru zlepšuje výkon a rychlosť učení modelu, zatímco ztrátová funkce počítá odchylku (ztrátu) mezi maskami a předpověďmi.

Trénování modelu

Učení modelu probíhá ve funkci `model.fit()`. Model je trénován po dobu 25 epoch s velikostí jedné šarže 64. Počet epoch je adekvátní, na grafu ztrátové funkce poslední epochy již lze zaznamenat lehký nástup přeúčení. Průběh učení modelu ovlivňuje funkci keras.callbacks.ModelCheckpoint(). Definuji v ní ukládání modelu na disk a parametr *save_best_only*, který zamezuje ukládání horších verzí modelu než je verze předchozí. Výsledný natrénovaný model ukládám jako soubor s příponou „.keras“. Verzí

modelu jsem vytvořil několik a po zhodnocení jejich úspěšnosti dále používám jen tu nejlepší. Po natrénování taktéž zobrazuji ztrátovou funkci učení (viz obr. 4.5), pro předběžnou evaluaci úspěšnosti modelu.

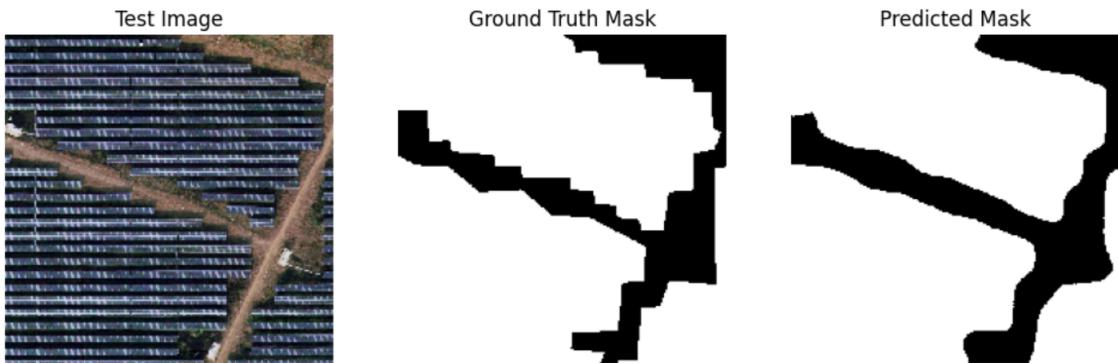


Obrázek 4.5: Ztrátová funkce modelu

Také chci zmínit, že při tvorbě modelu jsem narazil na problém, kdy graf ztrátové funkce nabýval podobu lineární křivky, avšak finální predikce obsahovala jen a pouze hodnoty pozadí. Tento problém může nastat, pokud snímky z datasetu obsahují více pixelů pozadí než popředí. Model se díky tomu naučí, že pixelů pozadí je více, a automaticky pak predikuje celý snímek jako pozadí. Problém se dá vyřešit pozměněním určitých parametrů modelu. Radikálnějším řešením je poté změna celé architektury modelu či celého datasetu.

Zobrazení výsledků

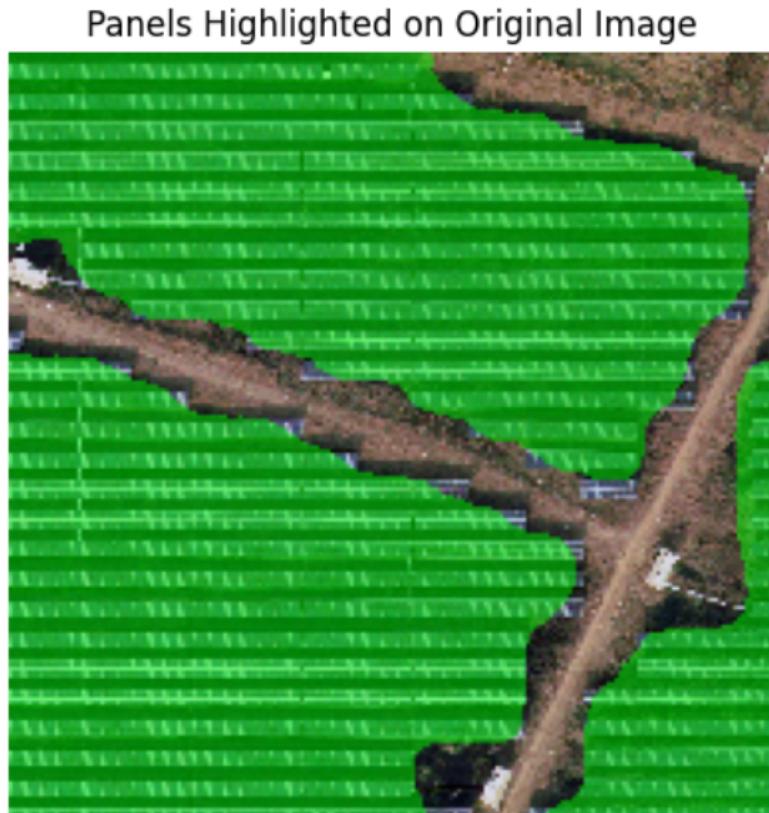
Důležité je také model použít a zhodnotit na testovací sadě dat, se kterou model v procesu učení nepřišel do styku. Tím lze zjistit úspěšnost modelu v praxi. Taktéž se hodí zobrazit výsledky a zhodnotit je osobně, jestli model pracuje správně. Zobrazuji proto počáteční zmenšený snímek, jeho správnou masku a poté modelem predikovanou masku. K zobrazení snímků používám knihovnu *Matplotlib*. Stejnou knihovnu taktéž využívám k vykreslení grafu ztrátové funkce. K predikování masky pak slouží funkce *keras.predict()*, která pomocí zvoleného modelu udělá predikci na zvolených datech. Na predikovanou masku se ještě aplikuje funkce *numpy.argmax()* knihovny *Numpy*. Tím se hodnoty v masce normalizují na hodnoty 0 nebo 1 a masku je možné zobrazit (viz obr. 4.6).



Obrázek 4.6: Vizuální porovnání obrázku, pravé masky a predikované masky (v tomto pořadí)

Dobrou volbou je také zhodnocení křivek úspěšnosti. Těchto grafů existuje více typů. Různé typy grafů se hodí pro různé typy úloh a je jen na programátorovi, jaký přístup si zvolí. Většinou se však používá graf ztrátové funkce modelu, který jsem již v práci popisoval, nebo graf úspěšnosti, který zobrazí procentuální úspěšnost modelu v predikci výstupu.

Pro získání finálního výstupu pak zvýrazňuji oblasti, ve kterých se vyskytují foto-voltaické panely. Nejdříve se provádí predikce masky na snímku. V oblastech originálního snímku, ve kterých se vyskytuje maska, se následně upraví hodnoty pixelů. Hodnoty upravuji zvednutím hodnot zeleného barevného kanálu pixelu. Výstupním obrazem je pak snímek, ve kterém jsou oblasti s panely zvýrazněny zelenou barvou (viz obr. 4.7).



Obrázek 4.7: Obrázek se zvýrazněnými oblastmi s panely (zeleně)

Vyhodnocení modelu

Pro vyhodnocení úspěšnosti modelu se běžně používá sada testovacích dat, se kterou model při procesu trénování nepřišel do styku. Lze tak otestovat úspěšnost modelu, i zda nedošlo k přeucení. Pro vypočítání úspěšnosti modelu využívám iterace přes všechna testovací data. Následně je provedena predikce masky ke každému snímku a vyhodnocení úspěšnosti porovnáním hodnot všech pixelů s pixely pravdivé masky. Ve finále úspěšnost modelu činí 71,95 %. Pro bližší porovnání výsledků modelu je také vhodné zobrazení matice záměn modelu, to však ve svém řešení nedělám. V části výše také popisují, jak vizuálně zobrazit výsledky modelu. Zhodnocením těchto výsledků lze zjistit, že predikce masky funguje, avšak ne úplně precizně. Maska definitivně zobrazuje oblasti s panely, avšak hrany těchto oblastí nejsou přesné. Stává se, že kus panelu je mimo detekovanou oblast, či že se kus pozadí nachází v detekované oblasti s panely.

4.3 Detekce jednotlivých FVP

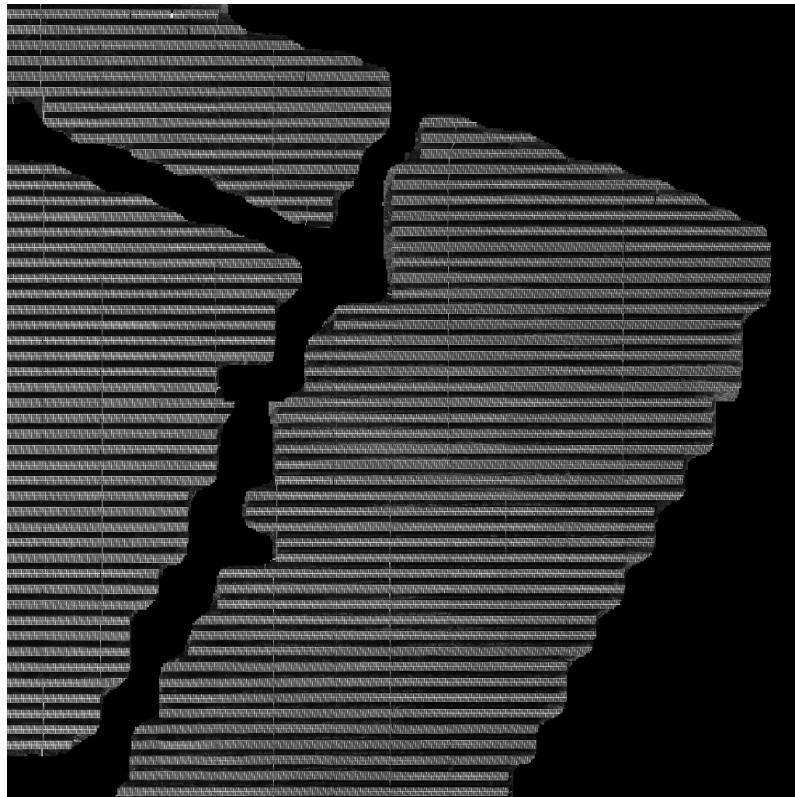
Detekce oblastí, na kterých se nacházejí panely, je hotova, nyní je úlohou z tohoto výstupu detektovat jednotlivé panely. Jelikož jsou panely zarovnány v řadě, je možné využít metod zpracování obrazu a pomocí detekce čar tyto panely vyznačit.

Vymaskování pozadí snímku

Detekce hran prováděná v následujícím kroku běžně detekuje hrany na celém snímku. To však není nutné, stačí, když se hrany zvýrazní pouze v oblastech, kde jsou panely. Proto se nejdříve na snímek aplikuje maska, která byla výstupem programu na segmentaci. Hrany na pozadí proto zaniknou a detekce se zvýrazněním proběhne pouze v oblastech, ve kterých je to potřeba. Tento krok však závisí na přesnosti masek predikovaných modelem. Pokud nejsou masky přesné, nebo nejsou k dispozici, tento krok je možné přeskočit.

Nejdříve se provede predikce na snímku pomocí modelu, díky čemuž lze získat predikovanou masku. Jedna maska nabývá velikosti 200 na 200 pixelů a originální snímek má velikost 1024 na 1024 pixelů, masku je proto nutné transformací zvětšit. Dalším problémem je, že jednotlivé snímky se před predikcí modelem ořezávají na 4 stejně velké výřezy. Pro použití masky na originálním snímku je proto nutné nejprve provést predikci na 4 výřezech a následně získané masky zvětšit a spojit. Zvětšením maska neztrácí důležité detaily, proto je tento postup validní. Výstupem je tedy maska, kterou lze aplikovat na originální snímek pro vyznačení oblastí s panely.

Jelikož v dalších krocích není pozadí potřeba, snímek se kompletně zbaví oblastí, kde bylo detektováno pozadí. Pole formy matice originálního snímku a pole predikované masky se zobrazí přes sebe, ale zobrazí se jen ty pixely, které hodnotou odpovídají oblasti, kde byly maskou detektovány panely (viz obr. 4.8). Výsledný snímek bez pozadí se dále využije.

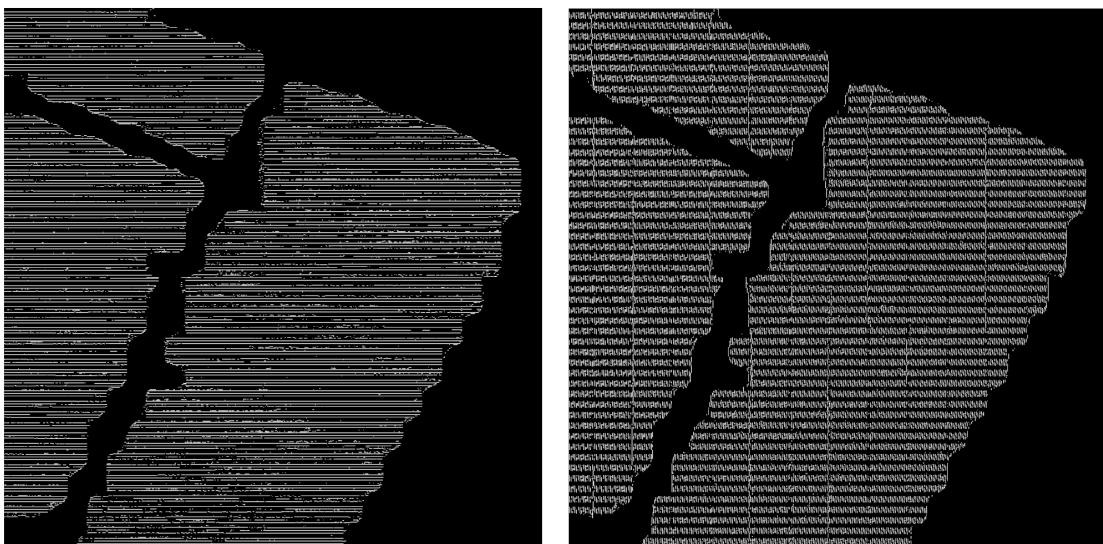


Obrázek 4.8: Šedotónový snímek s vymaskovaným pozadím

Detekce horizontálních pruhů na snímku

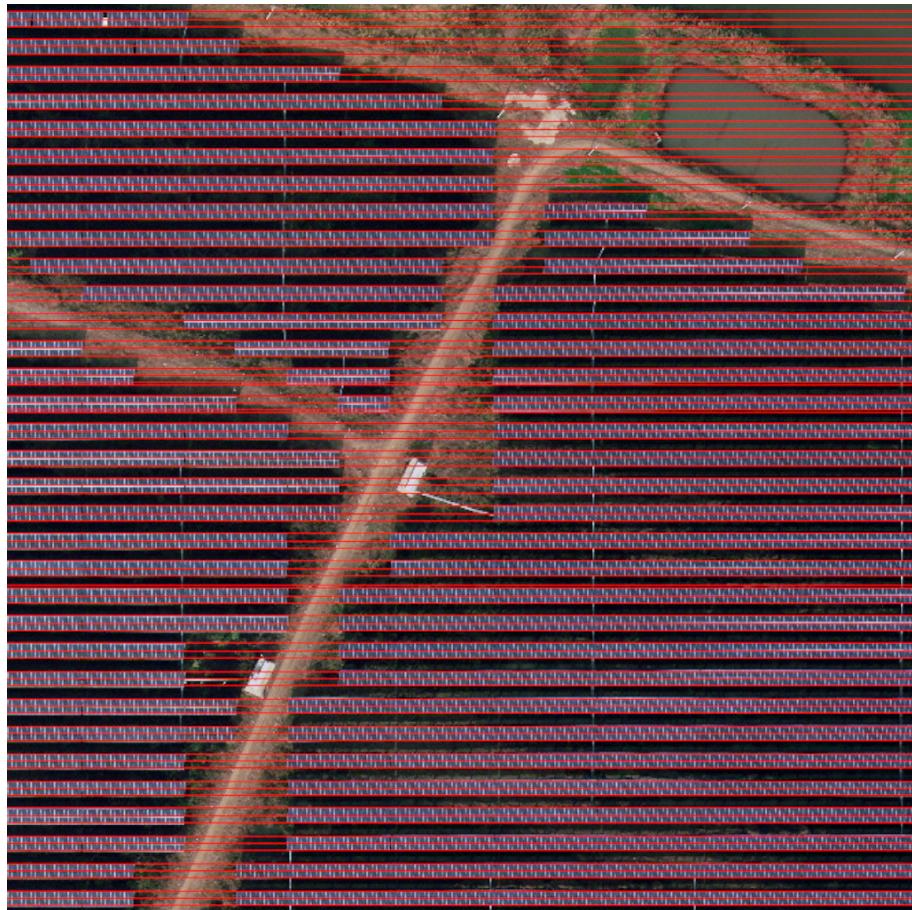
Snímek, na kterém se bude provádět detekce jednotlivých panelů, se nejdříve předzpracovává. Prvním krokem tohoto předzpracování je detekce a zvýraznění hran. Jelikož při detekci hran nejsou potřeba všechny barevné kanály pixelu, převede se snímek do šedotónové podoby. Na snímku tím pádem zůstává pouze jeden barevný kanál a následná práce se snímkem není tolik náročná na techniku. Pro výsledné zobrazení výsledku se však používá barevná verze snímků, je to pro lidské zhodnocení přirozenější.

Při detekci horizontálních hran se na šedotónový obraz aplikuje *konvoluční filtr*, který využívá ručně definovaného kernelu pro detekci horizontálních hran. Následně se snímek pomocí prahování převádí do binární formy. Vzniká snímek, na kterém jsou zvýrazněny detekované hrany. Aby se se snímkem dalo následně pracovat, převádí se hodnoty pixelů do rozsahu 0 až 255. Detekce hran vertikálních, které je potřeba v dalších fázích programu, funguje na stejném principu. Jediná změna je v kernelu, ten se před použitím v konvolučním filtrovi transponuje. Prohodí se tak jeho řádky a sloupce, takže detekuje vertikální hrany místo horizontálních. Výsledky lze pozorovat na obr.4.9.



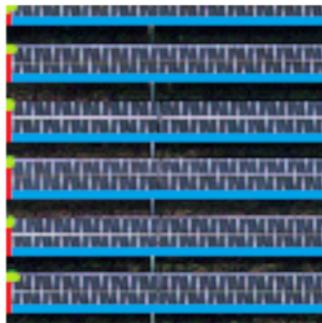
Obrázek 4.9: Snímky s detekovanými hranami, vlevo horizontálními, vpravo vertikálními

Cílem dalšího kroku je najít pruhy, ve kterých se může vyskytovat řada panelů. Detekce hran v předchozím kroku vyznačuje hrany. Z těchto hran lze aplikací *Houghova operátoru* získat čáry, které vyznačují hranice pruhů panelů (viz obr. 4.10). Houghova transformace v programu toleruje čáry, které se od úhlu hrany s horizontální osou liší maximálně o 5 stupňů. Po aplikování Houghova operátoru se nalezené horizontální čáry ukládají do pole pro další zpracování.



Obrázek 4.10: Snímek s vyznačenými čarami

Z nalezených horizontálních čar se dále vytváří horizontální pruhy. Tyto pruhy se tvoří pomocí souřadnic detekovaných čar. Z dvou po sobě jdoucích čar se vezmou souřadnice a pomocí výpočtu vznikají hodnoty pro výšku a šířku pruhu. Pruhy pak mají formu obdélníku a ukládají se ve formě *n-tice*. Tyto *n-tice* obsahují údaje souřadnic začátku pruhu na horizontální a vertikální ose a hodnoty pro výšku a šířku pruhu. Ukládá se ještě hodnota pro klasifikační třídu, která je využita v následujícím kroku (pro lepší pochopení viz obr. 4.11). Je důležité si uvědomit, jak struktura pruhu vypadá a o jaké hodnoty se jedná. Při práci s pruhy je totiž snadné se splést a uvést špatnou hodnotu. Ve finále tedy nabývá pruh formy *n-tice* o 5 hodnotách. Tyto objekty se ukládají do pole pruhů.

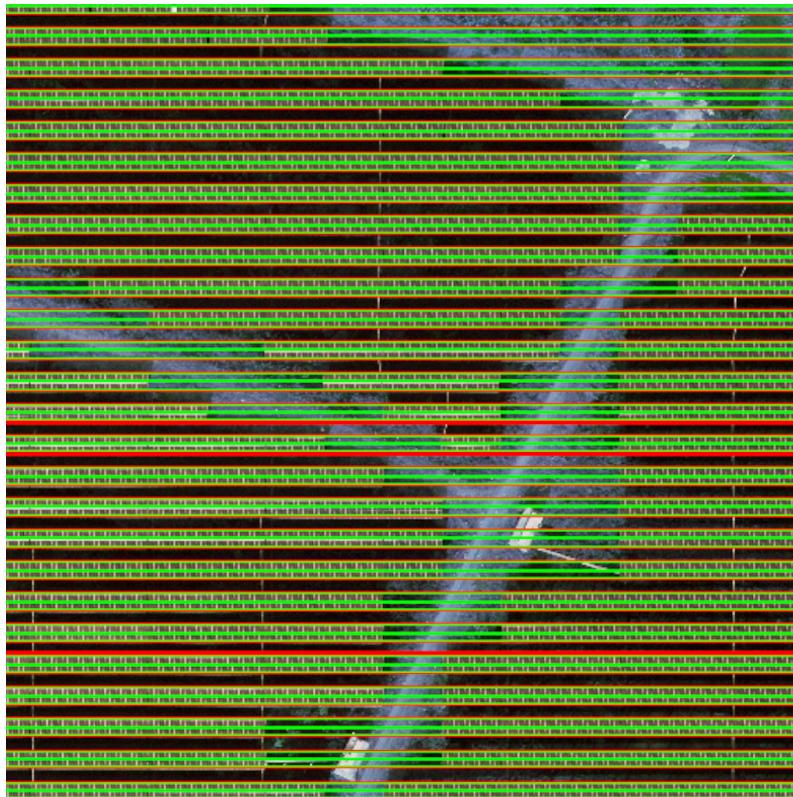


Obrázek 4.11: Vizualizovaná struktura objektu pruhu

Klasifikace jednotlivých pruhů

Výstupem předešlého kroku je pole se souřadnicemi jednotlivých pruhů. Nyní je nutné podrobit nalezené pruhy klasifikaci. Touto klasifikací se zjišťuje, který pruh obsahuje panely a který pouze pozadí. K tomu je využit již vytvořený snímek detekovaných vertikálních hran. Pruh s panely totiž díky jednotlivým segmentům panelů obsahují více vertikálních hran, na rozdíl od pruhů pozadí.

Jednotlivé pruhy se nejdříve konvertují na formát seznamu. To zaručí, že uložené parametry jsou proměnlivé a lze je změnit. Následně se iteruje přes všechny uložené pruhy. Ze snímku s vyznačenými vertikálními hranami se vyřízne část, jejíž dimenze jsou stejné jako dimenze pruhu. Na tento výřez je poté aplikována funkce, která spočítá počet pixelů hrany. Jelikož je snímek v binární podobě, pixely hrany lze jednoduše odlišit a spočítat. Počet pixelů se dále používá ve funkci, která počítá poměr pixelů hran vůči pixelům v celém pruhu. Jestliže je toto číslo větší než zvolený práh, pruh je klasifikován jako pruh panelů. Tento klasifikátor se doplní jako poslední hodnota *n-tice* symbolující jeden pruh. Výstupem je klasifikační třída všech pruhů (vizualizována na obr. 4.12).



Obrázek 4.12: Snímek s klasifikovanými pruhy (panely zelené, pozadí červené)

Spojení pruhů pozadí a jejich rozdělení

Jelikož jsou panely v jednom pruhu v řadách po dvojici, dochází k rozdělení některých pruhů panelů v půlce. To je samozřejmě správně, jenže u některých pruhů tomu tak není. Řešením je spojit všechny pruhы panelů, které následují těsně po sobě. Následně se rozdělením všech pruhů panelů v půlce dosáhne kýženého výsledku.

Spojení panelů probíhá v cyklu. Nejdříve se kontrolují dva po sobě jdoucí pruhы. Pokud jsou oba tyto pruhы klasifikovány jako panel, dojde ke spojení. Nová instance pruhu má zase formu n -tice. Hodnoty souřadnic i šířka zůstávají stejné jako u prvního pruhu v pořadí, jediné, co se mění, je výška. Nový pruh se přidává do nově vytvořeného pole pro budoucí použití. Cyklus probíhá, dokud nejsou všechny pruhы zpracovány. Na konci cyklu se ještě musí zpracovat poslední pruh, pokud tomu tak v cyklu nedošlo.

Nyní je možné všechny pruhы panelů rozdělit v půlce bez obav, že by dělení proběhlo nesprávně. Rozdělení probíhá zase v cyklu, kdy se iteruje přes pole spojených pruhů. Pokud aktuální pruh spadá do kategorie panelů, provede se rozdělení. Přesně v polovině se pruh rozdělí na dva nové, které se uloží do nově vytvořeného pole, ted' už správně vyznačených pruhů. Jestliže se jedná o pruh pozadí, uloží se bez žádných dalších augmentací.

Detekce jednotlivých panelů

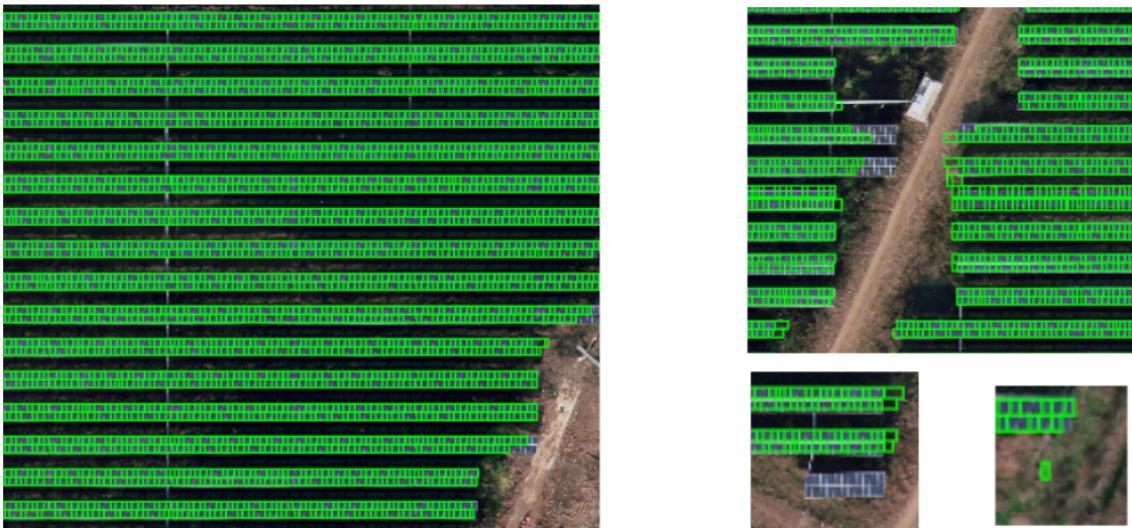
Jednotlivé pruhy jsou předzpracovány a zbývá už jen provést samotnou detekci jednotlivých segmentů panelů. K tomu je zase využit snímek se zvýrazněnými vertikálními hranami. K uložení souřadnic detekovaných panelů je zase využito pole. Celková detekce funguje na podobném principu, na kterém fungovala klasifikace pruhů.

Program iteruje přes pole pruhů. Pokud je aktuální pruh klasifikovaný jako panely, ze snímku s hranami je vyříznuta část odpovídající dimenzím pruhu. Poté se spouští druhý cyklus, který iteruje přes jednotlivé vertikální sloupce pixelů výřezu. Pro každý sloupec je spočítán poměr pixelů hrany k celkovému počtu pixelů ve sloupci. Pokud toto číslo překračuje určený práh, znamená to, že se jedná o hranici jednoho panelu. Jeden panel je uložen ve formě *n-tice*, obsahující souřadnice jeho levého horního a pravého spodního rohu. Panel je tedy uložen ve formě obdélníku. Ve finále vzniká pole, ve kterém se vyskytují souřadnice obdélníků, symbolizujících jednotlivé panely.

Vyhodnocení detekce FVP

Jelikož není k dispozici žádná sada snímků s jednotlivými panely anotovány, není možné přesně zhodnotit úspěšnost programu. Co se však teoreticky dá zhodnotit, je počet správně detekovaných panelů z celkového počtu. Evaluace probíhá ve funkci, která nejdříveypočítá medián pro výšku a šířku panelů z jednoho snímku. Pro všechny panely, které splňují tyto parametry dimenzí, se zvýší hodnota počítadla správně detekovaných panelů. Při spuštění této funkce na všech snímcích lze získat celkový počet panelů a počet správně detekovaných panelů. Toto číslo pro mnou použitou sadu testovacích dat je zhruba 65%. Z toho vypovídá, že některé instance panelů jsou větší či menší, než by měly být. Chyba nejspíše bude v kroku detekce vertikálních hran či čar, kdy program nepovažuje hranu za dostatečně výraznou a nevyznačí na tomto místě čáru, či hranici panelu.

Zhodnocení výsledku lze také provést osobně, při zobrazení výsledků vizuálně (viz obr. 4.13). Při zobrazení výsledků lze pozorovat, že největší problém je v nepřesnosti predikované masky. Pokud je část panelu mimo masku, program tento panel jednoduše nedetekuje. Zbytek problémů v preciznosti pak vězí v nepřesném detekování jednotlivých panelů. Panely mají různé velikosti, což má stejný počátek problému, jako popisují v odstavci výše.



Obrázek 4.13: Vlevo příklad správné detekce, vpravo tři příklady špatné detekce

4.4 Diskuze výsledků a návrhy na zlepšení

Jak jsem již zmínil, výsledky nejsou úplně ideální. Program funguje pouze do určitých mezí. Je pravda, že detekuje oblasti s panely a následně v těchto oblastech jednotlivé panely vyznačí, avšak výsledek není zcela precizní. Lze tedy říct, že program funguje a plní cíl práce, avšak je potřeba na něm ještě zapracovat. Spoustu parametrů a detailů je potřeba vyladit k získání úspěšného a precizního výstupu programu.

Ostatní práce, které se zabývají detekcí FVP, řeší problém buďto jen pomocí neuronových sítí³, nebo pouze pomocí klasických metod zpracování obrazu⁴. Skoro všechny práce, které jsem našel, však panely nedetekovaly jako jednotlivce. V pracích jsou detekovány pouze oblasti, ve kterých se panely vyskytují. Jediná práce⁵, která detekuje FVP jako jednotlivce, řeší problém pomocí vytváření 3D modelu snímaných panelů pomocí termovizní i klasické kamery a následné detekce. Tato metoda je oproti mnou navržené metodě náročnější na výpočetní techniku.

³<https://www.sciencedirect.com/science/article/abs/pii/S0045790624000715>

⁴<https://www.mdpi.com/2227-7080/11/6/174>

⁵<https://www.sciencedirect.com/science/article/pii/S0306261924006251>

Zlepšení části segmentace oblastí s FVP

Asi největší problém s touto metodou je detekce všech panelů jako celek a ne jako jednotlivce. Pokud by model prováděl segmentaci instancí, druhá část programu (detekce jednotlivých FVP) by nebyla potřeba. Tento problém se dá odstranit volbou lepšího datasetu. Pokud by existoval dataset, kde je každý panel vyznačen jednotlivě, model by se jednoduše přetrénoval a výsledek by přesně splňoval zadání práce. Jak již ale bylo zmíněno, tento dataset jsem při své přípravě nenašel. Je zde pořád možnost tvorby datasetu vlastního, avšak tato metoda je časově náročná a pro časový termín práce nevhodná.

Preciznost modelu taktéž není zcela perfektní. Zde se může využít hned několik metod pro zlepšení. Jako první je možnost trénovat model na více datech, která se od sebe také více liší. Model je totiž natrénován pouze na limitovaném počtu dat s jedním typem pozadí. Pokud bude rozvrstvení dat širší, model bude schopen dosáhnout lepších výsledků u různých typů snímků fotovoltaických panelů. Tento postup jsem si však díky omezenosti ze strany výpočetní techniky nemohl dovolit. Ze stejného důvodu taktéž nebylo možné ořezáním vytvořit více dat. Snímky i masky mají velikost větší než je model schopen přijmout, dochází proto k škálování (zmenšení) dat. Zanikají zde pak některé detaily, které jsou důležité při učení.

Další možností je zvýšit počet epoch trénování. Trénování by pak probíhalo déle a je zde šance, že by se přesnost modelu zlepšila. Musí se však dát pozor na jev přeучení. Poslední možností je úprava parametrů modelu. Nynější architektura a parametry modelu jistě nejsou perfektní. Jistě by je šlo upravit k dosáhnutí lepších výsledků programu.

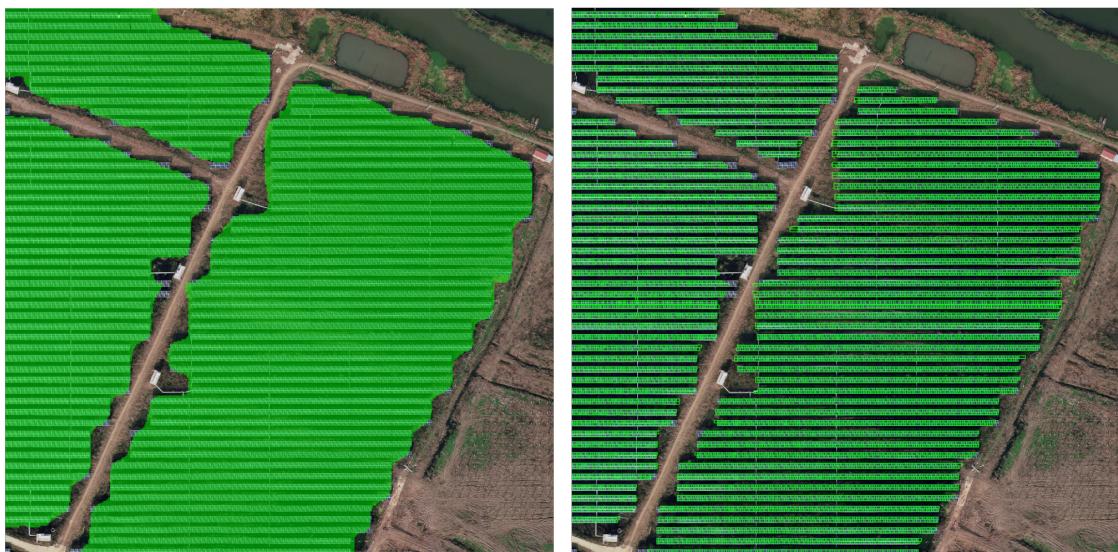
Zlepšení části detekce jednotlivých FVP

Část, ve které používám metody zpracování obrazu, lze také zlepšit. Mým návrhem je doladit část, kde se detekují buďto vertikální hrany, nebo vertikální čáry. Tím lze zlepšit detekci jednotlivých panelů a momentálně různorodé dimenze jednotlivých panelů by se tímto ustálily. Tyto úpravy mohou být pouze malé, stačí pouze vyladit některé parametry. Toto ladění je však časově náročnější a již jsem je nestihl implementovat.

S použitím těchto úprav by měl vzniknout program, který úspěšně a precizně detekuje jednotlivé fotovoltaické panely na snímku, který byl pořízen bezpilotním letounem a na kterém jsou jednotlivé řady panelů zarovnány. Program by se tak dal využít třeba v dalších pracích, zaměřených na detekci vad fotovoltaických panelů, či bližší analýze jednotlivých panelů.

5 Závěr

Cílem práce bylo vytvořit program, který rozpozná jednotlivé fotovoltaické panely ze snímku, jenž byl pořízen dronem. I přes limitace ze strany techniky se mi podařilo dosáhnout řešení tohoto problému. Nejdříve jsem vytvořil program, který za pomoci technologie strojového učení dokáže najít pole fotovoltaických panelů. Tento model strojového učení dosáhl na testovacích datech úspěšnosti 71,95 %. Program však vinou nedostupnosti vhodného datasetu nedetektuje panely jednotlivě. Druhý krok řešení využívá zpracování obrazu. Výstupem je snímek, na kterém jsou vyznačeny jednotlivé fotovoltaické panely. Úspěšnost správně detekovaných jednotlivých panelů je pak 65,20 %. Výsledky lze pozorovat na obrázku 5.1.



Obrázek 5.1: Snímek oblastí s panely (vlevo) a snímek s jednotlivě detekovanými panely (vpravo)

Důležité je však zmínit, že program funguje pouze v případě, že jsou fotovoltaické panely řazeny do řad, jak to nejčastěji bývá u fotovoltaických elektráren. Pokud budou panely „volně poházené“, řešení nebude funkční a program bude muset být upraven. Odhaduji, že je to tak kvůli vzorům, které si model strojového učení hledá v obrazech. Tyto distinkтивní vzory budou u osamocených a otočených panelů jiné, než u panelů zarovnaných v řadě. Postupy zpracování obrazu, které v práci používám, také detekují pouze čáry pod určitým úhlem, bude-li tedy panel pootočen, program jej jednoduše vynechá.

Cíl, který jsem si dal na začátku, tedy byl splněn – vytvořil jsem program schopný detektovat jednotlivé fotovoltaické panely. Jak již bylo zmíněno v úvodu, výsledky této práce by se daly využít v energetickém průmyslu, konkrétně v problematice údržby fotovoltaických elektráren. Na vytvořeném programu hodlám dále pracovat, jelikož jej lze v mnoha ohledech zlepšit (návrhy na zlepšení rozebírám v kapitole Diskuze výsledků a návrhy na zlepšení). Taktéž mě zajímá, jak by výsledky ovlivnilo použití metody učení bez učitele při vývoji modelu. Doufám, že mé poznatky v práci pomohou některým lidem porozumět této problematice a popřípadě jim napomohou k vytvoření ještě efektivnějšího programu. Finální program je volně dostupný na stránce *Github*¹.

¹<https://github.com/FanaIsDead/SOC-detekce-FVP>

Literatura

- [1] F. Chollet. *Deep Learning with Python, Second Edition*. <https://sourestdeeds.github.io/pdf/Deep%20Learning%20with%20Python.pdf>, 2021.
- [2] Herraiz Á. H., Marugán A. P., and Márquez F. P. G. Photovoltaic plant condition monitoring using thermal images analysis by convolutional neural network-based structure. <https://doi.org/10.1016/j.renene.2020.01.148>, 2020. cit. 11. 12. 2024.
- [3] Vega Díaz JJ., Vlaminck M., Lefkaditis D., Orjuela Vargas SA., and Luong H. Solar panel detection within complex backgrounds using thermal images acquired by uavs. <https://doi.org/10.3390/s20216219>, 2020. cit. 11. 12. 2024.
- [4] Wikipedia. Digital image. https://en.wikipedia.org/w/index.php?title=Digital_image&oldid=1260235774, 2024. cit. 29. 11. 2024.
- [5] Wikipedia. Digital image processing. https://en.wikipedia.org/w/index.php?title=Digital_image_processing&oldid=1261936517, 2024. cit. 10. 12. 2024.
- [6] Wikipedie. Počítacové vidění — wikipedie: Otevřená encyklopédie. https://cs.wikipedia.org/w/index.php?title=Po%C4%8D%C3%ADta%C4%8Dov%C3%A9_v%C4%9Bn%C3%AD%AD&oldid=23421688, 2023. cit. 28. 12. 2024.
- [7] Wikipedie. Hluboké učení. https://cs.wikipedia.org/w/index.php?title=Hlubok%C3%A9_u%C4%8Den%C3%AD&oldid=24462836, 2024. cit. 17. 12. 2024.
- [8] Wikipedie. Python. <https://cs.wikipedia.org/w/index.php?title=Python&oldid=24444708>, 2024. cit. 2. 12. 2024.
- [9] Ying-Yi Hong and Rolando A. Pula. Methods of photovoltaic fault detection and classification: A review. <https://doi.org/10.1016/j.egyr.2022.04.043>, 2022. cit. 11. 12. 2024.

Seznam obrázků

2.1	Fotovoltaická elektrárna – Brno Tuřany, Autor: Petr Opletal – Vlastní dílo, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=25750540	3
2.2	Dron s termovizním snímačem, Zdroj: https://hp-drones.com/en/drones-with-thermal-cameras-what-are-their-advantages/	3
2.3	Snímek FV elektrárny pořízený dronem, Zdroj: https://www.kaggle.com/datasets/salimhammadi07/solar-panel-detection-and-identification	4
2.4	Viditelný rozdíl mezi detekcí hran (vlevo) a detekcí čar (vpravo). Zdroj: https://rubikscode.net/2022/06/13/thresholding-edge-contour-and-line-detection-with-opencv/	6
2.5	Vizualizovaný rozdíl mezi maticí a tenzorem.	7
2.6	Obrázek ukazující podobory umělé inteligence.	8
2.7	Obrázek ukazující princip strojového učení, oproti klasickému způsobu programování.	9
2.8	Grafy popisující rozdíl mezi klasifikací (vlevo) a regresí (vpravo), Zdroj: https://www.javatpoint.com/regression-vs-classification-in-machine-learning	9
2.9	Příklad definování akcí, trestů a odměn pro hru Super Mario Bros.	10
2.10	Snímek grafu ztrátové funkce mého modelu	11
2.11	Příklad sémantické segmentace (vlevo) a segmentace instancí (vpravo). Zdroj: [1]	12
2.12	Vizualizace architektury plně konvoluční sítě. Zdroj: https://paperswithcode.com/method/fcn	13
3.1	Vývojový diagram procesu	18
4.1	Příklad snímku a masky z datasetu (v tomto pořadí)	21
4.2	Hierarchie nově vytvořených adresářů	22
4.3	Vizualizovaný princip oříznutí pro lepší představu (vlevo originální snímek, vpravo vzniklé ořezy)	23
4.4	Architektura modelu	25
4.5	Ztrátová funkce modelu	26

4.6	Vizuální porovnání obrázku, pravé masky a predikované masky (v tomto pořadí)	27
4.7	Obrázek se zvýrazněnými oblastmi s panely (zeleně)	28
4.8	Šedotónový snímek s vymaskovaným pozadím	30
4.9	Snímky s detekovanými hranami, vlevo horizontálními, vpravo vertikálními	31
4.10	Snímek s vyznačenými čarami	32
4.11	Vizualizovaná struktura objektu pruhu	33
4.12	Snímek s klasifikovanými pruhy (panely zeleně, pozadí červeně)	34
4.13	Vlevo příklad správné detekce, vpravo tři příklady špatné detekce	36
5.1	Snímek oblastí s panely (vlevo) a snímek s jednotlivě detekovanými panely (vpravo)	38