Žilinská univerzita v Žiline

Elektrotechnická fakulta

Katedra mechatroniky a elektroniky

28260620221005

REALIZÁCIA UMELEJ NEURÓNOVEJ SIETE NA PLATFORME RASPBERRY PI

2022

Jakub Krško

ŽILINSKÁ UNIVERZITA V ŽILINE
ELEKTROTECHNICKÁ FAKULTA
KATEDRA MECHATRONIKY A ELEKTRONIKY

Realizácia umelej neurónovej siete na platforme Raspberry Pi

BAKALÁRSKA PRÁCA

Študijný program:	Výkonové elektronické systémy
Študijný odbor:	Elektrotechnika
Školiace pracovisko:	Žilinská univerzita v Žiline, Elektrotechnická fakulta, Katedra mechatroniky a elektroniky
Školiteľ:	Ing. Peter Klčo, PhD.
Konzultant:	doc. Ing. Dušan Koniar, PhD.

2022
Jakub Krško

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta elektrotechniky
a informačných technológií

Katedra mechatroniky
a elektroniky

ZADANIE BAKALÁRSKEJ PRÁCE

Meno: **Jakub Krško**

Študijný program: Elektrotechnika

Názov bakalárskej práce:

Realizácia umelej neurónovej siete na platforme Raspberry Pi

Pokyny pre vypracovanie bakalárskej práce:

1. Využitie strojového učenia a neurónových sietí v diagnostike elektronických systémov
2. Analýza vybraných defektov elektronických dosiek a možnosť ich detekcie obrazovými algoritmami
3. Zostavenie alebo adaptácia datasetu potrebného k natrénovaniu vybranej konvolučnej siete
4. Realizácia konvolučnej NN a verifikácia presnosti na definovanej úlohe

Predpokladaný rozsah práce - počet strán textu: max. 40
počet strán grafických príloh: max. 10

Školiteľ bakalárskej práce: Ing. Peter Klčo, PhD.

Konzultant bakalárskej práce : doc. Ing. Dušan Koniar, PhD.

Dátum zadania bakalárskej práce: **12. 11. 2021**

Dátum odovzdania bakalárskej práce: **16. 5. 2022**

prof. Ing. Michal Frivaldský, PhD.
vedúci katedry

Abstrakt

Bakalárska práca sa zaoberá použitím neurónových sietí na detekciu kvapiek cínu na doskách plošných spojov vo výrobe výkonových polovodičových modulov. Hlavným cieľom práce bola tvorba vlastného datasetu a následne trénovanie nami vytvoreného modelu na plnenie zadanej úlohy. Na trénovanie bola použitá sieť založená na sieti typu YOLO, ktorá poskytuje pred trénované modely pre uľahčenie trénovania a je možné jednoducho vytvoriť konfiguračné súbory pre trénovanie. Detekcia s vytvoreným modelom je riešená na Raspberry Pi. Pre detekciu sme vytvorili Python aplikáciu, ktorá deteguje obrázky zo súboru s využitím knižnice Torch a natrénovaného modelu.

Kľúčové slová: Neurónová sieť, YOLO, Raspberry Pi, počítačové videnie

Abstrakt v angličtine:

The bachelor thesis deals with the use of neural networks for the detection of solder drops on printed circuit boards in the production of power semiconductor modules. The main goal of the work was to create our own dataset and then train the model we created to perform the task. A YOLO based network was used for training, which provides pre-trained models to help ease the training process and configuration files can be easily created for training. Detection with the created model is handled on a Raspberry Pi. For detection, we created a Python application that detects images from a file using the Torch library and the trained model.

Keywords: Neural network, YOLO, Raspberry Pi, computer vision

ŽILINSKÁ UNIVERZITA V ŽILINE, ELEKTROTECHNICKÁ FAKULTA
KATEDRA MECHATRONIKY A ELEKTRONIKY

ANOTAČNÝ ZÁZNAM – BAKALÁRSKA PRÁCA

Meno a priezvisko: Jakub Krško **Akademický rok:** 2021/2022
Názov práce: Realizácia umelej neurónovej siete na platforme Raspberry Pi

Počet strán:	**33**	**Počet obrázkov:**	**12**	**Počet tabuliek:**	**3**
Počet grafov:	**6**	**Počet príloh:**	**1**	**Počet použ. Lit.:**	**15**

Anotácia v slovenskom jazyku: diplomová práca

Bakalárska práca sa zaoberá použitím neurónových sietí na detekciu kvapiek cínu na doskách plošných spojov vo výrobe. Hlavným cieľom práce je tvorba vlastného datasetu a následne trénovanie modelu na plnenie zadanej úlohy. Detekcia s vytvoreným modelom je riešená na Raspberry Pi. Súčasťou práce bola aj tvorba Python aplikácie na detekciu obrázkov zo súboru.

Anotácia v anglickom jazyku:

Batchelor thesis is about usage of neural networks for detection of solder splashes on printed circuit boards in manufacturing. Main aim of the work is construction of custom dataset and training own model for this task. For image detection with custom trained model was used a Raspberry Pi board. In addition a Python application for image detection from file was created.

Kľúčové slová: Neurónová sieť, Raspberry Pi, počítačové videnie, YOLO
Vedúci diplomovej práce: Ing. Peter Klčo, PhD.
Konzultant: doc. Ing. Dušan Koniar, PhD.
Recenzent: _____
Dátum odovzdania práce: 16.05.2022

Obsah

Zoznam obrázkov a tabuliek

Zoznam skratiek

Skratka	Anglický význam	Slovenský význam
2D	Twodimensional	Dvojrozmerný
3D	Threedimensional	Trojrozmerný (priestorový)
CNN	Convolutional neural network	Konvolučná neurónová sieť
DPS	-	Doska plošných spojov
FLOP	Floating point operations per second	Operácie s desatinnou čiarkou za sekundu
GPU	Graphics processing unit	Grafický procesor (grafická karta)
GPIO	General purpose input output	vstupný alebo výstupný port pre periférie
GUI	Grafical user interface	Grafické rozhranie pre užívateľa
HDMI	High definition media interface	Rozhranie na prenášanie obrazu vo vysokej kvalite
ID	Identifier	Identifikácia unikátneho objektu
NN	Neural network	Neurónová sieť
OS	Operating system	Operačný systém
RAM	Random access memory	Pamäť s náhodným prístupom
RGB	Red green blue	červená zelená modrá (farebné kanály obrazu)
RPi	Raspberry Pi	skrátené označenie pre značku
SoC	System on chip	čip na ktorom je integrovaných väčšina komponentov počítača
USB	Universal serial bus	Univerzálna sériová zbernica

Slovník termínov

Termín	Význam termínu
Diagram	Symbolická reprezentácia informácii podľa zobrazovacej techniky.
Fully connected layer	Anglický názov pre plne prepojenú vrstvu v neurónovej sieti. Slúži na klasifikáciu vlastností z predchádzajúcich vrstiev.
Linux	Open source operačný systém vytvorený v roku 1991, je základom pre OS ako Ubuntu a dalšie systémy.
Neural network	Sieť pozostávajúca z uzlov alebo neurónov, spojenia medzi ktorými je definované pomocou váh (priorít). Hlavné časti sú vstupná vrstva, ktorá slúži na predprípravu dát a výstupná vrstva, ktorá poskytuje výsledok analýzy dát.
Open-source	Akákoľvek dostupná informácia verejnosti za podmienok slobodného šírenia pri zachovaní nezmenného obsahu. Filozofiu open-source založil Eric Raymond.
Platforma	Pracovne prostredie, po hardvérovej stránke, tak aj softvérovej. Softvérová platforma určuje použitý operačný systém, knižnicu, ale taktiež použitý programovací jazyk alebo kompletný framework.
Pooling layer	Označenie združovacej vrstvy v konvolučnej sieti. Využíva sa na znižovanie zložitosti neurónovej siete.
Sigmoida	Matematická funkcia, ktorá má priebeh v tvare písmena S. Je to ohraničená, diferencovateľná, reálna funkcia, ktorá je definovaná pre všetky reálne vstupné hodnoty a má v každom bode nezápornú deriváciu a presne jeden inflexný bod.
UNIX	Operačný systém vytvorený hlavne pre programátorov v roku 1963, podporuje viac užívateľov naraz. Spočiatku bol napísaný v assembleri, dnes je písaný v jazyku C.

Poďakovanie

Týmto by som chcel poďakovať konzultantovi mojej diplomovej práce Ing. Petrovi Klčovi, PhD. za cenné rady a poskytnuté materiály, ktoré som využil pri realizácii mojej práce. Taktiež sa chcem poďakovať doc. Ing. Dušanovi Koniarovi, PhD., za zabezpečenie návštevy firmy SEMIKRON za účelom zberu dát na tvorbu datasetu potrebnému k učeniu modelu neurónovej siete.

ÚVOD

V súčasnej dobe nastáva rýchly vývoj v oblasti elektroniky, hlavne čo sa týka veľkosti, materiálov a spôsobu výroby súčiastok a modulov. Preto je dôležité vytvárať aj nové prístupy na kontrolu kvality produktu počas výroby. Jednou z takýchto možností je využívanie neurónovej siete na detekciu nežiadúcich chýb na doskách vo výrobe. Týmto je možné zamedziť a analyzovať, kde tieto chyby často nastávajú a vylúči sa tým aj ľudský faktor, keďže sa môže stať, že človek kontrolujúci takéto dosky sa môže raz za čas pomýliť, čo sa pri aplikáciách v počítačoch stáva menej často. Nespočetný počet firiem v dnešnej dobe používa tieto siete, či už na predpovedanie, alebo analýzu numerických parametrov, alebo obrazových dát v prípade výrobnej linky.

Preto hlavným cieľom tejto bakalárskej práce je použitie a aplikácia neurónových sietí na detekciu kvapiek cínu na doskách plošných spojov s pomocou platformy Raspberry Pi. Neurónové siete patria ku umelej inteligencii a strojovému učeniu a úzko súvisia s počítačovým videním, pomocou ktorého počítače interpretujú obraz napríklad z kamier. Žiadaným výsledkom práce je vytvorený model, ktorý je schopný tieto kvapky detegovať s dostatočnou presnosťou a má nízke číslo nesprávnych detekcií na určitom množstve analyzovaných obrázkov.

V tejto práci sa budeme venovať tvorbe datasetu potrebného na trénovanie modelu konvolučnej siete s jej následnou verifikáciou presnosti a využití v praxi na plnenie požadovanej úlohy v priemysle. Taktiež zanalyzujeme presnosť modelu v závislosti od veľkosti vstupných dát, parametrov detekcie a rýchlosti detekcie na natrénovanom modeli.

1 SÚČASNÝ STAV RIEŠENEJ PROBLEMATIKY DOMA A V ZAHRANIČÍ

Dosky plošných spojov sú v súčasnej dobe základným komponentom väčšiny elektronických zariadení. Počas ich vývoja boli upravené pre firemné aj jednoduchšie aplikácie, ktoré nemusia spĺňať prísne parametre, ako je tomu pri použití v priemysle. Dosky plošných spojov sú tenká vrstva laminovaného materiálu ako napr. sklolaminát alebo epoxid, ktorý tvorí základ dosky, na ktoré sú následne pridané vodivé cesty, najčastejšie vytvorené z medi. Na DPS sú uložené komponenty a integrované obvody.

Tu prichádzajú na rad rôzne metódy na kontrolu vyrobených DPS, ktoré sa neustále musia zlepšovať, kvôli zmenšujúcej sa veľkosti súčiastok a ich hustejšieho uloženia na doske. Na detekciu takýchto dosiek sa používa buď automatická detekcia, po ktorej je nutné, aby ešte školený pracovník skontroloval dosku, keďže veľa dosiek po tomto type detekcie označených ako chybné, sú častokrát v poriadku. Závislosť na ľudskom faktore pri kontrole vyžaduje veľa prostriedkov a dostatočného zaškolenia. Aj skúsení kontrolóri sa môžu počas detekcie chýb na doskách pomýliť, takže tu nastupujú presnejšie systémy na kontrolu. Tieto systémy používajú *deep learning* siete na detekciu, sú rýchlejšie a presnejšie ako ľudia. Hlavným cieľom v dnešnej dobe je návrh systému na detekciu chýb na DPS, ktorý zníži počet nesprávne detegovaných dosiek a zvýši počet vyrobených kusov.

Výskumníci používajú rôzne metódy na dosiahnutie daného cieľa. Jednou z najpoužívanejších je sieť typu YOLO (*You Only Look Once* – pozrieť sa iba raz), ktorá slúži na detekciu objektov v obrázku, pričom naraz dokáže zistiť viac hľadaných objektov. Model YOLO je založený na konvolučnej neurónovej sieti, ktorá sa používa na detekciu v jednom obrázku rozdelenom na regióny, pričom celý obrázok je skontrolovaný len raz. Vykoná sa len jeden prechod dopredu v neurónovej sieti na generovanie predpovedí o objektoch na obrázku. Nájdené objekty sú vyznačené pomocou súradníc ohraničujúceho rámčeka (*bounding box*) a každý má priradenú pravdepodobnosť zhody s hľadaným objektom. Tento algoritmus používa celé obrázky na trénovanie, takže sa učí všeobecné reprezentácie objektov počas trénovania.

Na druhej strane má aj nevýhody, ako napríklad viac chýb pri lokalizácií ako pri použití rýchlej konvolučnej neurónovej siete. Okrem toho má nižší *recall* ako iné metódy čo tiež používajú detekciu na závislosti od regiónov. Vynikajúce výsledky sa dajú dosiahnuť s použitím veľkých sietí a rôznych metód. YOLO sa uprednostňuje hlavne kvôli svojej presnosti detekcie v obmedzenom časovom rámci a jej výkonnosť sa neustále zlepšuje postupne s časom.

Väčšina moderných algoritmov na detekciu alebo klasifikáciu objektov používa VGG-16 ako základný extraktor funkcií pretože dosahuje vysokú presnosť a je robustný pre rôzne použitia. Na druhej strane, tento algoritmus je veľmi zložitý a vyžaduje 30,69 miliárd operácií za sekundu (FLOPs) na dosiahnutie rozlíšenia obrázka 224x224. Preto sa väčšinou používa vlastná sieť založená na google Net architektúre pre YOLO štruktúry. Model je tým pádom rýchlejší, ale má o niečo menšiu presnosť ako s použitím VGG-16 a požaduje len 8,52 miliardy FLOP na jeden prechod cez obrázok. S použitím modelu YOLO je možné dosiahnuť presnosť pri detekcii až do 99 % [1].

2 CIELE PRÁCE

Cieľom tejto práce bolo využitie strojového učenia na detekciu a diagnostiku chýb na doskách plošných spojov s využitím neurónových sietí. Práca vychádzala z projektu pre firmu SEMIKRON na výskum metód pre zlepšenie kvality a životnosti hybridných výkonových polovodičových modulov, ktorý sa zameriava na poskytovanie výrobkov pre zákazníkov, s najlepšou kvalitou a znižovaním počtu defektných kusov v procese výroby.

V práci sú využité neurónové siete na dosiahnutie hlavného cieľa, ktorým je detekcia defektov, najčastejšie v podobe kvapiek cínu na miestach, kde v neskorších krokoch výroby môžu spôsobiť problémy napríklad pri ultrazvukovom zváraní, kde následne dochádza ku poškodeniu hrotu zváračky v prípade, že narazí na kvapku cínu v mieste, kde má privariť drôt, ktorý slúži na prepojenie čipov v module, alebo na pripojenie na externé kontakty modulu. Tento cieľ sa zameriava na tvorbu datasetu, s ktorým následne bude daná neurónová sieť natrénovaná pre účely detekcie kvapiek cínu. Dataset obsahuje rôzne modely skúmaných modulov, na zaručenie rôznorodosti fotiek a tým zvýšenie schopnosti modelu pre detekciu na rôznych miestach a taktiež zníženie počtu nesprávnych detekcií.

Posledným cieľom tejto práce je verifikácia presnosti natrénovaného modelu neurónovej siete pomocou vytvoreného datasetu a vytvorenie aplikácie pre využitie modelu na detekciu obrázkov alebo automatizovanú detekciu s pomocou kamery realizovanú na platforme Raspberry Pi.

3 TEORETICKÝ ÚVOD

Problematika tejto bakalárskej práce sa odvíja hlavne okolo strojového učenia a neurónových sietí, takže v tejto časti bude spomenutých veľa anglických výrazov, ktoré budú vysvetlené v slovenčine a potom budú používané v angličtine na zjednodušenie textu, keďže ku väčšine neexistuje slovenský ekvivalent.

3.1 NEURÓNOVÁ SIEŤ

Neurónové siete sú podkategória strojového učenia a sú založené na *deep learning* (strojové učenie s využitím umelých neurónových sietí s viacerými vrstvami – hĺbkové učenie) algoritmoch. Názov a štruktúra sú inšpirované neurofyzikou ľudského mozgu, pretože napodobňujú spôsob, akým neuróny v mozgu prenášajú signály medzi sebou. Neurónové siete sa skladajú z vrstiev uzlov, ktoré sa ďalej delia na vrstvy vstupné, výstupné a skryté.

Každý uzol alebo neurón je spojený s ďalšími a má pridelenú váhu a prahovú hodnotu. Keď je výstup z uzla nad danou prahovou hodnotou, tak dochádza k jeho aktivácií a posiela dáta do ďalšej vrstvy siete. V opačnom prípade nedochádza k prenosu dát k ďalšej vrstve [2].

3.1.1 Typy neurónových sietí

Perceptrón

Predstavuje najjednoduchší a najstarší typ neurónových sietí. Skladá sa len z jedného neurónu, ktorý spracuje vstup a využije aktivačnú funkciu na vygenerovanie binárneho výstupu. Tento model neurónovej siete neobsahuje skryté vrstvy, používa sa len pri klasifikácii s binárnymi dátami [3].

Sieť typu Feed Forward

Siete typu *Feed Forward* (posúvanie dopredu) sa skladajú z mnohých neurónov a skrytých vrstiev, ktoré sú navzájom prepojené. Názov siete je odvodený z toho,

že tieto údaje prúdia len dopredu a nedochádza k spätnému šíreniu. V závislosti od použitia skryté vrstvy môžu, ale nemusia byť prítomné v sieti.

Pri viacerých úrovniach je možné prispôsobiť viac váh. Toto má za následok zlepšenie schopnosti siete učiť sa. Aktivačná funkcia prijíma výstup násobenia váh so vstupmi, čo funguje ako prahová hodnota. Medzi hlavné oblasti využitia tohto typu neurónovej siete patria: Rozpoznávanie hlasu, tvárí a vzorov [3].

Rekurentná neurónová sieť

Je založená na myšlienke zachovania výstupu a jeho vrátenia na vstup, aby pomohol predpovedať výsledok danej vrstvy. Prvá vrstva je vytvorená rovnako ako vstupná vrstva v sieti typu *Feed forward*. Po spracovaní vstupu prebehne rekurentný proces, teda každý neurón si z jedného kroku do druhého zapamätá informácie z predchádzajúceho kroku. V dôsledku tohto každý neurón vykoná výpočty, ako keby bol pamäťovou bunkou. Sieť pokračuje v šírení dát do ďalších vrstiev a pri nesprávnej predpovedi sa použije chyba na vykonanie drobných zmien, aby sa pomocou rekurzie dopracovala sieť na vytvorenie správnej predpovede [3].

Konvolučná neurónová sieť

Konvolučné siete majú namiesto dvojdimenzionálneho poľa trojdimenzionálne pole skladajúce sa z neurónov. Prvá vrstva je konvolučná. Každý neurón v tejto vrstve analyzuje dáta len z určitej časti z vizuálneho poľa. Vstupné funkcie sú načítavané po dávkach. Sieť dekóduje obrázky po častiach a vykonáva tieto operácie viackrát na dokončenie spracovania celého obrázka. Obrázok je počas spracovania konvertovaný do odtieňov šedej. To uľahčuje detekciu hrán a hodnôt pixelov a pomáha pri zaraďovaní obrázkov do niekoľkých kategórií.

Tento typ sa najčastejšie používa na klasifikáciu obrázkov s využitím počítačového videnia. Predchodcom takéhoto rozpoznávania boli manuálne a časovo náročné metódy na extrakciu rysov z obrázka. Konvolučné siete v súčasnosti poskytujú škálovateľnejší prístup k úlohám na klasifikáciu obrazu a rozpoznávanie objektov, pričom na detekciu vzorov v obraze využívajú lineárnu algebru, konkrétne násobenie matíc. Aj napriek

tomuto môžu byť náročné na výpočtový výkon, hlavne počas trénovania je nutné využitie grafických procesorov (GPU) [3].

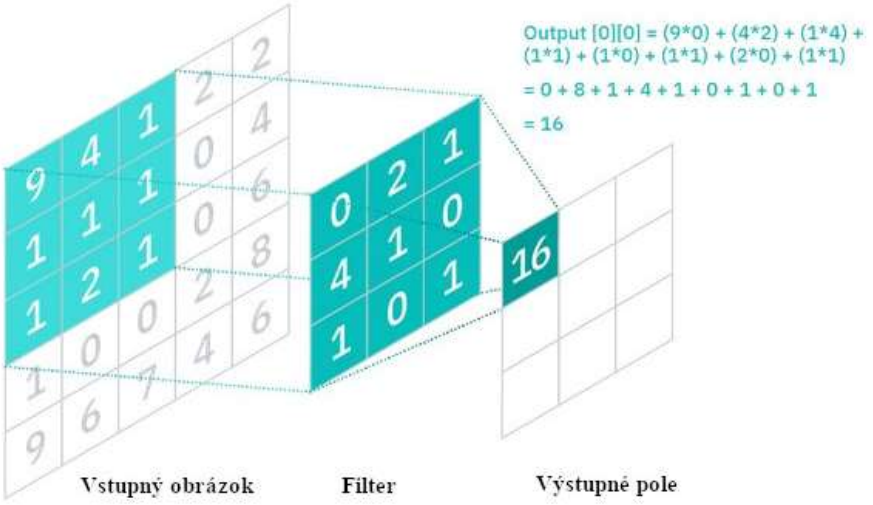
3.1.2 Princíp funkcie Konvolučnej neurónovej siete

Konvolučné siete sa od ostatných líšia vynikajúcim výkonom pri vstupoch obrazového alebo zvukového signálu. Používajú tri hlavné typy vrstiev. Konvolučná vrstva je prvá a môže byť nasledovaná ďalšími konvolučnými alebo združovacími vrstvami, pričom plne prepojená vrstva (*fully connected layer*) slúži ako posledná vrstva. S každou ďalšou vrstvou narastá komplexita CNN, ale zároveň aj jej schopnosť identifikovať väčšie časti z obrázka. Nižšie vrstvy sa zameriavajú na jednoduché vlastnosti, akými sú farby a hrany. Postupne ako vstupné obrazové dáta prechádzajú cez sieť, tak začína rozpoznávať väčšie elementy alebo tvary objektu pokiaľ sa úplne neidentifikuje daný objekt [4].

Konvolučná vrstva

Konvolučná vrstva je základným prvkom CNN a prebieha v nej väčšina výpočtov. Vyžaduje vstupné dáta, filter a mapa príznakov (vlastností). Farebný obrázok na vstupe je 3D matica pixelov, ktorej šírka, výška a hĺbka reprezentujú farebné kanály RGB. Tiež sa v tejto vrstve nachádza detektor vzorov, nazýva sa aj *kernel*, ktorý sa pohybuje po obrázku a kontroluje zhodu s hľadaným vzorom. Tento proces je známky ako konvolúcia.

Detektor vzorov je 2D pole váh, ktoré predstavujú časť obrazu. Síce je ich veľkosť rôzna, veľkosť tohto detektora je zvyčajne matica 3x3, čo určuje veľkosť receptívneho poľa. Detektor sa aplikuje na oblasť obrazu a počíta sa vektorový súčin medzi vstupnými pixelmi a filtrom. Daný súčin sa prenesie do výstupnej matice a filter sa posunie o krok ďalej. Proces sa opakuje, pokiaľ detektor neprejde celý obraz. Výsledný výstup zo série vektorových súčinov je mapa príznakov alebo konvolúcia príznakov.



Obr. 3.1 Proces konvolúcie [4]

Z obr. 3.1 vyplýva, že mapa príznakov sa nemusí spájať s každým pixelom vo vstupnom obrázku. Výstup musí byť hlavne spojený s daným poľom pixelov, na ktorom sa filter použil. Kvôli tomuto sa konvolučné vrstvy označujú aj čiastočne prepojené vrstvy (*partially connected*) [4].

Pooling layer

Združovacia vrstva, alebo vrstva so znižovaním vzorkovania (*downsampling*) vykonáva redukciu parametrov na vstupe. Podobne ako pri konvolučnej vrstve, tak aj táto vrstva prechádza s filtrom po vstupe, ale s rozdielom, že v tomto prípade filter nemá žiadne váhy. Namiesto toho *kernel* využije agregáciu na hodnoty v recepčnom poli, čím sa naplní výstupné pole. Existujú dva hlavné typy združovania:

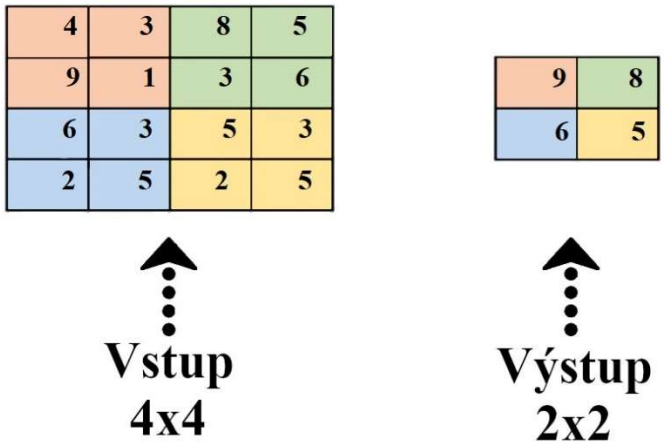
a) Maximálne združovanie: Pri pohybe filtra cez vstup vyberie pixely s maximálnou hodnotou a uloží ich do výstupného poľa. Tento spôsob sa v porovnaní s priemerným združovaním používa častejšie.

b) Priemerné združovanie: Filter pri pohybe po vstupe vypočíta priemernú hodnotu v rámci recepčného poľa, ktorá sa uloží do výstupného poľa.

Síce sa v *pooling layer* stráca veľa informácií, má táto vrstva pre CNN aj niekoľko výhod. Pomáha znižovať zložitosť, zlepšuje efektívnosť a obmedzuje riziko nadmerného prispôsobenia [4].

Max pooling (maximálne združovanie)

Táto operácia sa bežne pridáva za konvolučné vrstvy na zníženie rozmerov obrázkov redukovaním počtu pixelov vo výstupe z predchádzajúcej vrstvy. Keď filter konvoluje vstup, tak na výstupe je matica pixelov s hodnotami, ktoré boli vypočítané počas konvolúcií na danom obrázku, toto nazývame výstupné kanály.



Obr. 3.2 Princíp max pooling [5]

Max pooling funguje nasledovne: Ako prvé si zadefinujeme oblasť o veľkosti n x n ako zodpovedajúci filter pre operáciu *max pooling*. Následne si zadefinujeme krok, o koľko pixelov sa filter posunie pri posúvaní po obrázku. Na konvolučnom výstupe vezmeme prvú oblasť o veľkosti filtra a vypočítame maximálnu hodnotu z daných hodnôt. Táto hodnota sa uloží do výstupného kanála a filter sa posunie o krok ďalej. Celý postup sa opakuje, pokiaľ sa nedostaneme úplne doprava v obrázku, kedy sa filter posunie o krok dole a začne znovu prechádzať vstupnú vrstvu. Tento proces sa opakuje pre celý obrázok a na výstupe dostaneme novú reprezentáciu obrázka, čo sa taktiež označuje ako výstupný kanál. Na obrázku hore je tento proces graficky znázornený [5].

Dropout layer (vylučovacia vrstva)

Tento typ vrstvy sa používa hlavne na odstránenie šumu alebo duplicitných údajov z neurónovej siete na zlepšenie detekcie a času potrebného na získanie výsledkov. Výzvou pre neurónové siete je nájsť spôsob, ako znížiť šum vo veľkom počte navzájom komunikujúcich neurónových uzlov, aby sa neprekročili možnosti spracovania siete.

Na tento účel sieť eliminuje všetku komunikáciu, ktorú prenášajú neurónové uzly a ktorá priamo nesúvisí s problémom alebo trénovaním, na ktorom sieť pracuje.

Podobne ako neuróny v ľudskom mozgu, aj jednotky neurónovej siete náhodne spracúvajú nespočetné množstvo vstupov a potom v danom čase vypúšťajú nespočetné množstvo výstupov. Proces a výstupy každej jednotky môžu byť medziproduktmi, ktoré sa odovzdávajú inej jednotke na ďalšie spracovanie, dlho pred tým, ako vznikne konečný výstup alebo záver. Niektoré z týchto spracovaní končia ako šum, ktorý je medziproduktom spracovateľských činností, ale nie je konečným výstupom [6].

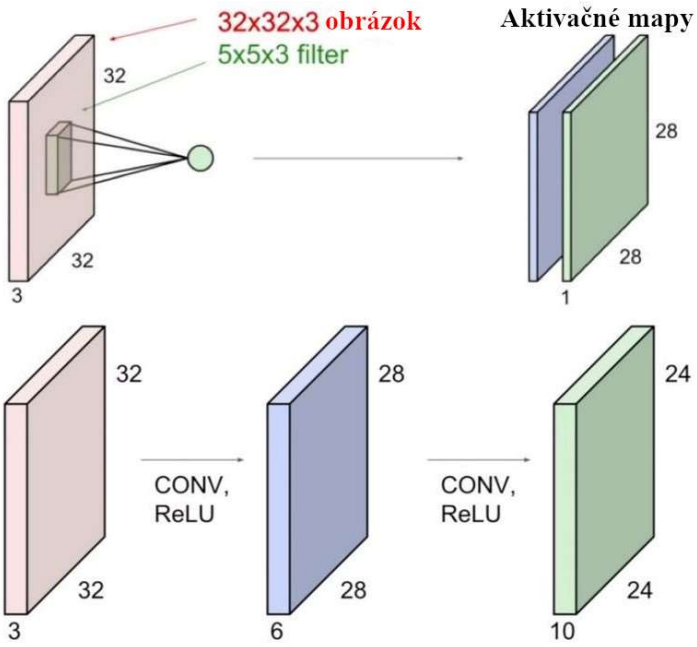
Fully connected layer

Hodnoty pixelov nie sú priamo spojené s výstupnou vrstvou v *pooling layer*. V tejto vrstve je každý uzol nachádzajúci sa vo výstupnej vrstve priamo spojený s uzlom z predchádzajúcej vrstvy.

Táto vrstva zabezpečuje klasifikáciu založenú na vlastnostiach z predchádzajúcich vrstiev a ich rozličných filtrov. Konvolučná a *pooling layer* používajú ReLu funkcie, ale táto využíva *softmax* aktivačnú funkciu na priamu klasifikáciu vstupov, ktorá vygeneruje hodnotu pravdepodobnosti od 0 do 1 [4].

ReLu (Rektifikovaná lineárna jednotka)

Konvolučné filtre začínajú na ľavom hornom rohu vstupného obrázku a postupne sa posúvajú doprava. Na každej pozícii je následne vygenerovaný skalárny súčin, ktorý tvorí výstup, ktorý sa tiež nazýva aktivačná mapa a tá slúži ako vstup do aktivačnej funkcie tejto vrstvy. Keď si zoberieme jeden filter a budeme ho posúvať po všetkých priestorových umiestneniach v obrázku, tak dostaneme aktivačnú mapu, ktorá obsahuje hodnoty zo všetkých lokácií v danom obrázku.



Obr. 3.3 Grafické znázornenie ReLu funkcie [7]

Výstup z vrstvy je zároveň vstupom do ďalšej konvolučnej vrstvy. V tejto vrstve sú dáta vynásobené váhou vrstvy a je použitá aktivačná funkcia pre nelinearitu. ReLu vrstva je vkladaná medzi konvolučné vrstvy, v skratke ReLu sa používa na filtrovanie informácií, ktoré sa šíria ďalej sieťou.

Na vstupe sa vykonáva operácia s prvkami a v prípade ak je vstup záporný, tak sa vo funkcii zmení na nulu, ak je kladný, tak sa jeho hodnota na výstupe nemení. Toto avšak predstavuje aj nevýhodu, lebo sigmoida nie je centrovaná okolo nuly, takže sa stáva, že keďže ReLu ignoruje záporné vstupy, tak je možné aby vznikli tzv. „mŕtve neuróny“, ktoré sa nikdy neaktivujú [7].

3.2 RASPBERRY PI

Raspberry Pi je lacný jednodoskový počítač veľkosti kreditnej karty, ktorý je možné pripojiť k monitoru a využíva štandardnú klávesnicu a myš. Nízka cena umožňuje ľudom jednoduchší prístup k počítačom a programovaniu v jazykoch ako napríklad Python. Taktiež je ho možné použiť na prehliadanie webu, prehrávanie videí alebo ľahkú kancelársku prácu [8].

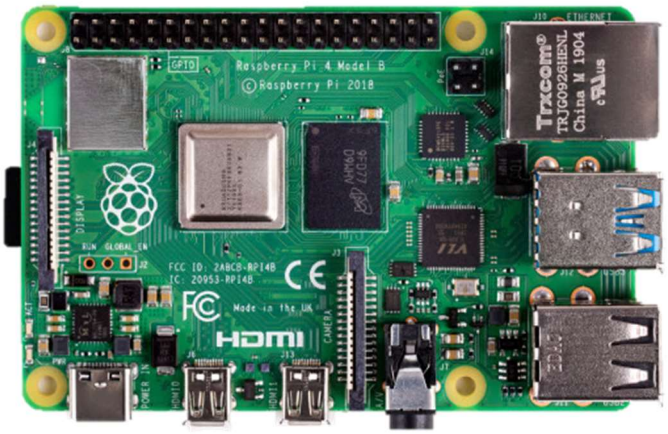
3.2.1 Hardvér

Raspberry Pi je založené na architektúre typu ARM, ktorá sa vyznačuje redukovanou sadou inštrukcií, nižšou spotrebou a je hlavne určená pre zariadenia, ktoré sú napájané z batérie. Na doske sa nachádza pamäť RAM, procesor, ktorý má vstavanú GPU, eternetový port, USB porty, HDMI na pripojenie obrazovky a GPIO na pripojenie externých periférií. Na rozdiel od počítačov Raspberry Pi načítava operačný systém z SD karty alebo z USB disku. Staršie modely mali 256 až 512 MB RAM, dnešné modely využívajú od 2 do 8 GB RAM. Procesor v najnovšom modeli beží na 1,5 GHz, avšak je ho možné pretaktovať až na 2,1 GHz.

Rapberry Pi má dva typy modelov: Model A a Model B. Líšia sa tým, že Model A má len jeden USB port s eternetovým konektorom. Model B má zvyčajne 4 USB porty a eternetový konektor. Tieto dva modely sa v starších verziách líšili aj veľkosťou pamäte RAM [9].

3.2.2 Raspberry Pi 4 Model B

Ide o prvý model, ktorý používa iný typ procesora ako predchádzajúce modely jednodoskových počítačov od nadácie Raspberry Pi. Tento model používa procesor od firmy Broadcom BCM2711 SoC s výkonnejšími jadrami, vylepšenou GPU a podporou pre väčšiu kapacitu pamäte RAM. Na rozdiel od prechádzajúcich modelov, tento má aj porty typu USB 3.0, ktoré poskytujú väčšiu prenosovú rýchlosť. Nechýba tiež podpora pre pripojenie dvoch monitorov až do rozlíšenia 4K s 30 snímkami za sekundu.



Obr. 3.4 Pohľad zhora na Raspberry Pi [10]

Na doske je tiež 40 pinový GPIO *header*, ktorý sa dá použiť pomocou knižnice GPIO Zero na ovládanie externých periférií ako tlačidlá, senzory alebo LED diódy. Raspberry Pi 4 má vyššiu spotrebu, kvôli výkonnejšiemu procesoru a perifériám, takže je potrebné použiť zdroj s väčším prúdom [10].

3.3 PYTHON

Python je programovací jazyk, ktorý má podporu veľkého množstva knižníc, čo má za následok jeho jednoduchosť a spoľahlivosť spolu s poskytovaním veľkého množstva súpravy funkcií ako samostatných modulov. Má ľahko a rýchlo pochopiteľnú syntax, dôležitejšia je čitateľnosť nad rýchlosťou exekúcie programu. Základy Pythonu sú implementované v jazyku C a sú založené na rozsiahlych, dobre zrozumiteľných a prenosných knižniciach jazyka C. Bez problémov je ho možné použiť na platformách ako UNIX a Linux [11].

3.3.1 Vznik

Python bol vytvorený v roku 1991 holandským programátorom menom Guido van Rossum na základe poznatkov z hodín o operačných systémoch a jazykoch a je založený na ABC jazyku (príkazový programovací jazyk vyvinutý v Holandsku, náhrada BASIC a je určený na výučbu alebo prototypovanie) a Modula-3. Vývoj Pythonu je *open-source* a je centralizovaný na SourceForge [11].

3.3.2 Vlastnosti jazyka Python

Je to objektovo orientovaný programovací jazyk s dôrazom na jednoduchosť pri vývoji programov. To, že je interpretovaný jazyk znamená, že inštrukcie sa vykonávajú postupne a nie je nutná kompilácia pri spúšťaní programu ako je tomu napríklad pri jazykoch z rodiny C.

Výhodou jazyka je to, že nemá GOTO príkaz, takže odpadá potreba exciplitne spravovať pointre a pamäť počas behu programu. Syntax príkazov sa čo najviac podobá angličtine, takže program je ľahko čitateľný a používa len zopár pravidiel pre syntax programu, ako napríklad odsadenie pre sprehľadnenie celého programu [11].

3.3.3 Odlišnosť od iných programovacích jazykov

Python sa od ostatných programovacích jazykov líši hlavne tým, ako bolo spomenuté v predchádzajúcej podkapitole, že je to implementovaný jazyk, takže pri spúšťaní programu nie je nutná kompilácia, čo mu umožňuje jednoduchšiu prenosnosť, keďže je potrebný len jeden súbor na spustenie programu a úpravu. Pre porovnanie v jazyku C je súbor .c, ktorý obsahuje zdrojový kód a potom skompilovaný program napr. s príponou .exe. Prenosnosť tiež umožňuje použitie rovnakého kódu na rôznych platformách [11].

3.4 YOLO MODEL

Skratka YOLO znamená *You Only Look Once* – pozrieť sa iba raz, čo v praxi označuje typ modelu konvolučnej siete pre rozpoznávanie obrazových dát, ktorý prejde cez každý obrázok filtrom len raz a dokáže klasifikovať viac ako jeden objekt v danom obrázku. V algoritme na detekciu je využitá jedna neurónová sieť na detekciu obrázka, ktorý je rozdelený na viac oblastí. Presnosť každej oblasti je predpovedaná následne, táto hodnota určuje aj pozíciu ohraničujúcich polí.

Tento model je populárny vďaka jeho vysokej presnosti a efektivite. Model používa celé obrázky na proces trénovania a optimizuje počas trénovania aj výkonnosť detekcie, aj počas detekcie sa používa celý obrázok, kvôli rýchlosti modelu.

Na druhej strane YOLO model má aj nevýhody. Má nižšiu hodnotu *recall* a lokalizáciu so zachovaním vysokej presnosti klasifikácie. V súčasnej dobre sú *deep networks* (siete využívajúce hĺbkové učenie) preferované pre počítačové videnie. Vynikajúce výsledky sa dajú dosiahnuť použitím väčších sietí a rôznych metód. YOLO je preferovaným modelom, keďže má presnú detekciu v obmedzenom časovom rámci. Jeho výkonnosť sa postupne s novými verziami zlepšuje [1].

3.4.1 Parametre modelu YOLO

Model YOLOv5 má zopár parametrov, ktoré určujú presnosť siete počas trénovania. Prvý parametrom je *recall*, ktorý označuje percento pomeru správne označených objektov *(true positive)* ku súčtu *true positive* a nedetegovaných objektov *(false negative)*, výpočet tohto parametru je vo vzorci 1:

$$recall = \frac{tp}{tp+} \tag{1}$$

Presnosť je percento *true positives* ku všetkým získaným výsledkom, je ho možné vypočítať pomocou vzorca 2, kde n je súčet *true positives* a *false positives* (nesprávne označené objekty za objekty záujmu), napr. bol detegovaný *solder blob*, ale v skutočnosti detegovaný objekt nie je *solder blob*.

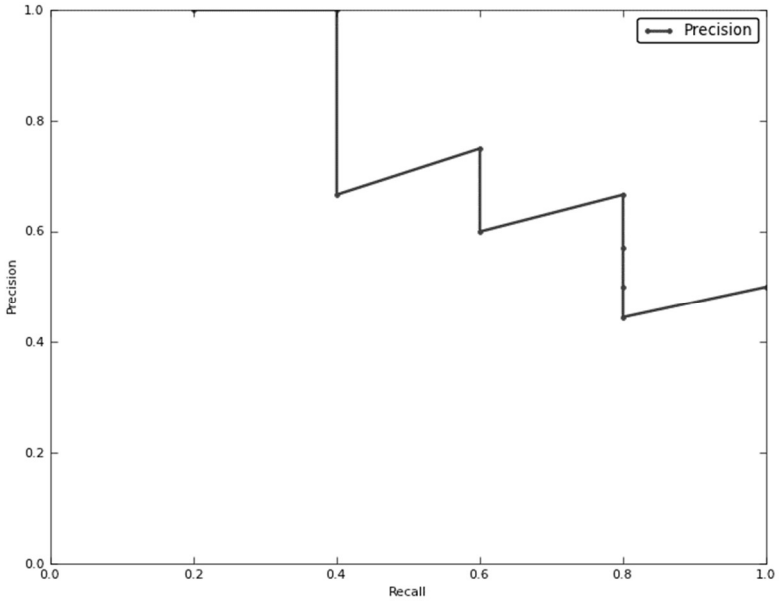
$$presnosť = \frac{tp}{tp+} = \frac{tp}{n} \tag{2}$$

Napríklad keď je presnosť 0,75, to znamená že v 75% objektov, ktoré boli označené ako hľadaný objekt boli skutočne hľadanými objektmi, zvyšné objekty boli nesprávne označené ako objekty záujmu. *Recall* 60% znamená, že zo všetkých objektov v množine model správne označil za objekt záujmu 60% a zvyšné nenašiel [12].

$$F_1 = 2 \cdot \frac{presnosť \cdot recall}{presnosť + recall} \tag{3}$$

Ďalej sa dá tiež presnosť siete ohodnotiť jedným parametrom – F1. Tento parameter je užitočný na zistenie optimálnej spoľahlivosti, ktorá je v rovnováhe s presnosťou a *recall* hodnotami pre daný model, avšak táto hodnota je pre hodnoty istoty od 0 do 1. Zo F1 pre daný model je možné odvodiť metriku hodnotenia, ktorá môže byť dobrým ukazovateľom celkovej výkonnosti modelu [13].

Dobrým spôsobom, ako charakterizovať výkon klasifikátora je vykreslenie zmeny presnosti a *recall* pri zmene prahu (*threshold*). Dobrý klasifikátor zaradí žiadané objekty na začiatok zoznamu a získa veľa objektov pred získaním iných, čo znamená že presnosť zostane veľká pri zvyšovaní *recall*. Slabý klasifikátor bude musieť výrazne znížiť presnosť, aby dosiahol vyšší *recall*. Táto závislosť za väčšinou zobrazuje grafom na Obr. 3.5.



Obr. 3.5 Graf závislosti precision od recall [12]

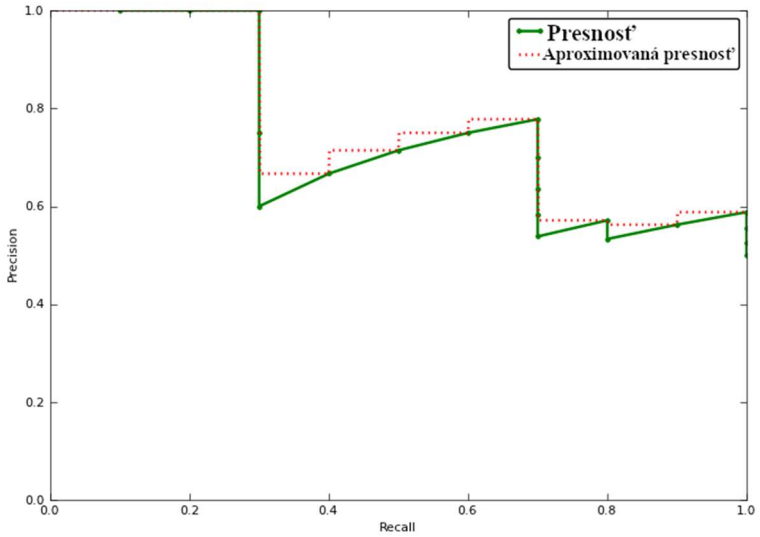
Namiesto kriviek je užitočné mať taktiež jedno číslo, ktoré jednoznačne charakterizuje presnosť klasifikátora. Týmto označením je priemerná presnosť (*average precision*). Presnejšie povedané, priemerná presnosť je presnosť vypočítaná pre všetky hodnoty *recall* medzi 0 a 1 cez integrál ako je písané vo vzorci 4:

$$priemerná\ presnosť = \int_0^1 p(r)dr \tag{4}$$

Vo vzorci je p funkcia presnosti a r je hodnota *recall*. Toto sa rovná ploche pod krivkou, v praxi sa tento integrál presne aproximuje súčtom presností pri každej možnej prahovej hodnote vynásobeným zmenou *recall*:

$$\sum_{k=1}^{N} P(k)\Delta r(k) \tag{5}$$

Kde N je celkový počet obrázkov v datasete, P(k) je presnosť hraničnej hodnoty z k obrázkov a Δr(k) je zmena *recall*, ku ktorej došlo medzi hraničnou hodnotou k-1 a hodnotou k.

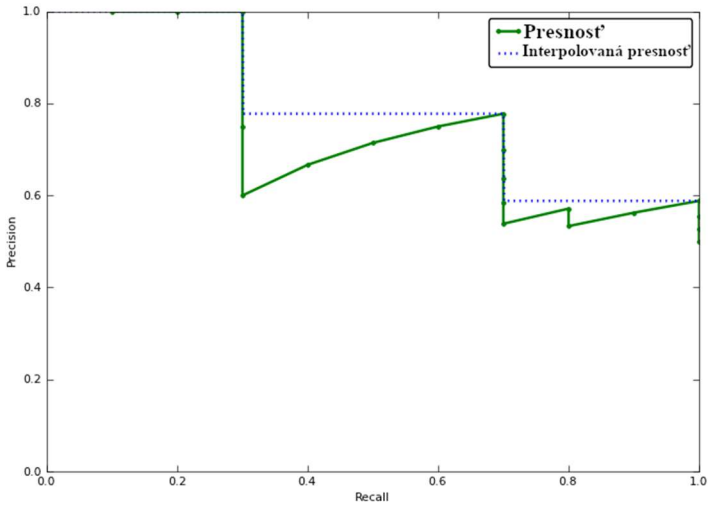


Obr. 3.6 Graf presnosti a aproximovanej presnosti [12]

Spolu s priemernou presnosťou sa tiež používa druhý typ presnosti označovaný ako interpolovaná presnosť, aj keď ju tiež často označujú ako priemerná presnosť. Namiesto použitia P(k) presnosti pri hranici k obrázkov sa používa presnosť ako vo vzorci 5:

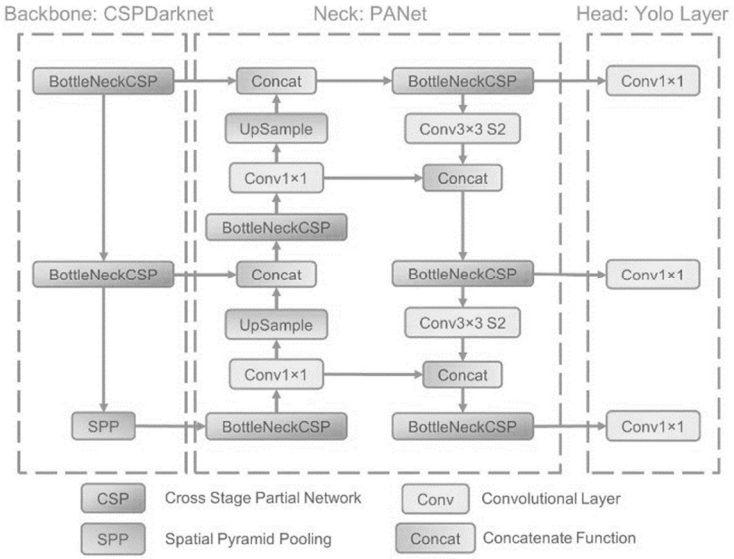
$$\sum_{k=1}^{N} max_{\bar{k} \geq k} P(\bar{k}) \Delta r(k) \tag{5}$$

Inak povedané, tak namiesto presnosti, ktorá bola pozorovaná pri hraničnej hodnote k , sa pri interpolovanej priemernej presnosti používa maximálna presnosť pozorovaná pri všetkých hraničných hodnotách s vyšším *recall*. Na obr. 1.6 je porovnanie aproximovanej presnosti a interpolovanej presnosti [12].



Obr. 3.7 Graf presnosti a interpolovanej presnosti [12]

3.4.2 Architektúra YOLOv5



Obr. 3.8 Architektúra YOLOv5 siete [14]

Siete YOLO používajú *DarkNet* ako svoj základ, čo rieši problém s opakujúcimi sa gradientmi a integruje tieto zmeny pomocou mapy rysov, čo má za následok zníženie parametru *FLOPs* modelu, takže výsledný model je menej náročný na výpočtový výkon. Toto priamo tiež zlepšuje rýchlosť a presnosť a zároveň redukuje veľkosť modelu.

Ďalej použitím *PANet* (*Path aggregation network* – agregačná sieť ciest), ktorá zvyšuje tok informácií. PANet má novú funkciu pyramídová štruktúra siete (FPN – *Feature pyramid network*), čo má za úlohu vylepšiť šírenie prvkov na nižších úrovniach. Zároveň používa adaptívne združovanie, ktoré spája sieť funkcií (*feature grid*) na vytvorenie užitočných informácií na každej úrovni, čo sa šíri do nasledujúcej podsiete. PANet zlepšuje využitie presnejších lokalizačných signálov v nižších vrstvách, čo sa podieľa na presnosti lokalizácie.

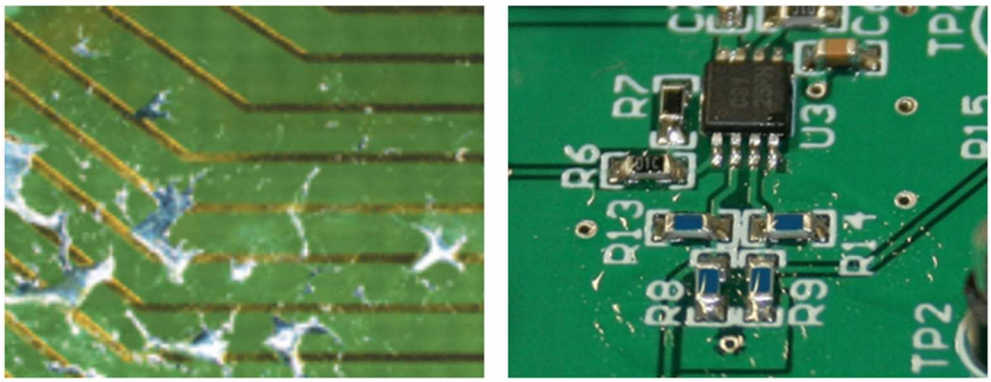
Ako posledná je tu časť Yolov5 generuje 3 rôzne veľkosti (18x18, 36x36, 72x72) máp vlastností na dosiahnutie predikcie vo viacerých mierkach, čo umožňuje modelu zvládnuť malé, stredné a veľké objekty [14].

3.5 SOLDER BLOB (KVAPKA CÍNU)

Spájka pod čipmi obsahuje okrem cínu aj tavivo alebo spájkovaciu pastu. Spájka pod DCB (*Direct Copper Bonded* – priame spájanie medi) neobsahuje tavivo, pri použití metódy dvoj krokového spájkovania výkonových čipov, kedy sa spájka pod čipmi taví ešte raz, kedy dochádza k vytekaniu taviva spopod hrán čipu. Ak sa tam nachádza zábrana, alebo vplyvom mechanického odporu taviva dôjde ku zvýšeniu vnútorných síl, čo sa priamo podieľa na vzniku kvapiek cínu. Tieto kvapky sa môžu nachádzať na rôznych produkčných linkách využívajúcich iné zariadenia a zliatiny materiálov na spájkovanie.

Vo výrobe sú definované regióny, kde sa nemôžu nachádzať kvapky cínu, lebo by mohlo dôjsť k zníženiu kvality ďalších krokov, tak preto je potrebné ich z tohto miesta odstrániť. Najpoužívanejšia metóda na ich odstraňovanie je frézovanie alebo využitie ultrazvukového čistenia. Kvapky najčastejšie robia problémy pri mikro-spájkovaní, bondovaní (spájanie dvoch materiálov) alebo pri ultrazvukovom zváraní. Väčšie kvapky môžu spôsobiť poškodenie hlavy spomínaných zariadení.

Ďalší problém nastáva, keď sa tieto kvapky nachádzajú na identifikačnom kóde danej dosky. Toto taktiež spôsobí poškodenie tohto kódu a obmedzí sledovanie kusu počas výrobného procesu [15].



Obr. 3.9 Kvapky cínu na cestách DPS [15]

4 PRAKTICKÁ ČASŤ

Realizácia bakalárskej práce pozostávala z viacerých menších celkov. Tie najrozsiahlejšie boli príprava datasetu a samotné trénovanie modelu, čo zabralo najviac času. Následne nasledovala analýza presnosti natrénovaného modelu a vytvorenie skriptu alebo programu na detekciu pomocou natrénovaného modelu na Raspberry Pi pomocou externej kamery. Počas realizácie bakalárskej práce sa vyskytlo zopár problémov, avšak väčšinu sa nám podarilo vyriešiť alebo nájsť alternatívne riešenie.

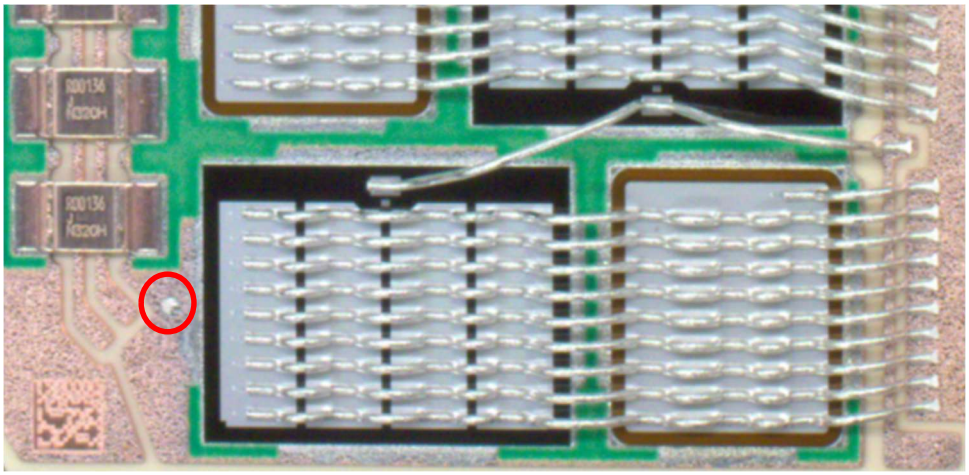
4.1 VÝBER TYPU SIETE A MODELU

Prvým krokom pri realizácií našej bakalárskej práce bola voľba typu neurónovej siete akú použijeme a potom to, že akým spôsobom budeme model trénovať. Na tento účel sme si zvolili konvolučnú neurónovú sieť, hlavne kvôli tomu, že tento typ siete sa najčastejšie používa na rozpoznávanie obrázkov. Základ pre tvorbu modelu sme zvolili architektúru modelov YOLO, ktorá poskytuje predtrénované s väčším počtom tried na rôzne použitie, napríklad medzi tieto triedy patria typy objektov ako auto, autobus, ľudia a podobne. My sme potrebovali natrénovať model, ktorý bude mať len jednu triedu na rozpoznávanie kvapiek cínu.

4.2 TVORBA DATASETU

Pred trénovaním modelu sme potrebovali vytvoriť zbierku obrázkov, tzv. dataset, ktorý obsahuje objekty ktoré chceme detegovať. V našom prípade to je kvapka cínu *(solder blob)*, ktorý sa nachádza na danej doske na mieste, kde by v neskorších krokoch výroby mohla spôsobiť problémy. Na obr. 4.1 je v krúžku vyznačená kvapka cínu, ktorá tam vznikla počas spájkovania čipov na medený základ pri ohrievaní cínu.

Najdôležitejšie vlastnosti dobrého datasetu sú, že musí obsahovať čo najviac fotiek, ktoré obsahujú hľadaný objekt a ešte lepšie je, keď je hľadaný objekt – kvapka cínu umiestnený na rôznych doskách a v rôznych tvaroch. Toto najviac pomáha pri učení modelu, takže je potom schopný rozoznať väčšiu variáciu kvapiek cínu a môže správne označiť aj také, ktoré v datasete pri učení nemal.



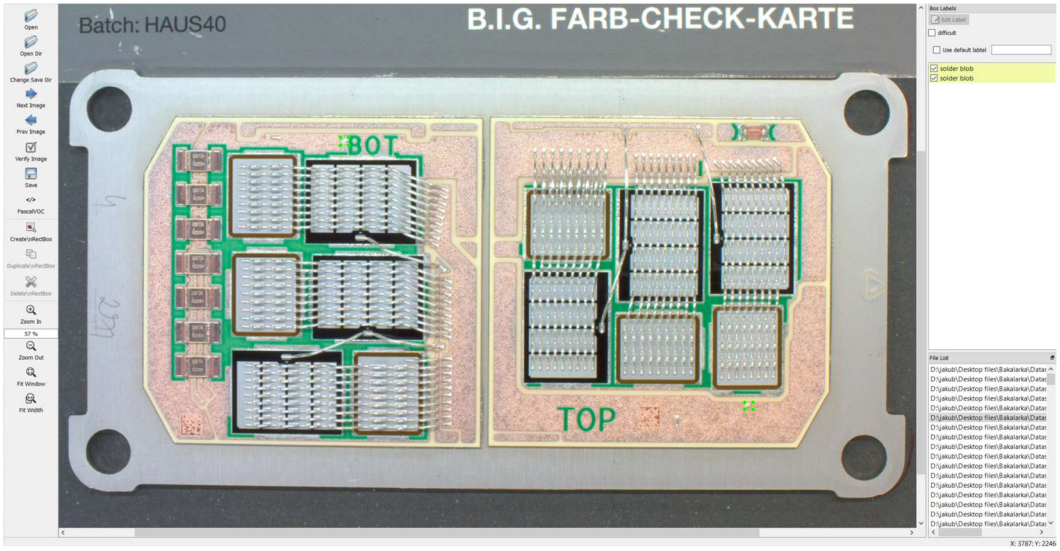
Obr. 4.1 DPS s kvapkou cínu

Počas písania tejto bakalárskej práce sme boli vo firme SEMIKRON, kde sme nafotili a označili 200 fotiek dosiek obsahujúcich kvapky cínu, ktoré boli neskôr použité na trénovanie modelu neurónovej siete. Nad dosku sme umiestnili farebnú kartu, pre prípadnú potrebu korekcie bielej, aby bolo zabezpečené, že všetky fotky budú mať rovnaké farebné odtiene. Nakoniec sme zistili, že to nebolo potrebné, keďže v hale, kde sme dané snímky vyhotovovali bolo dostatočne stabilné osvetlenie s nízkym podielom iných farieb, takže farby medzi snímkami sú konzistentné.

Fotky v datasete boli fotené pomocou USB 3.0 kamery Basler s rozlíšením senzora 15 MPx a s využitím programu Pylon na zaznamenávanie a ukladanie fotiek, ktorý umožňuje upraviť hodnoty bielej na fotke a poskytuje taktiež aj možnosť záznamu videa pomocou danej kamery.

4.2.1 Anotácia obrázkov

Aby bolo možné použiť vytvorený dataset na trénovanie modelu, tak je potrebné na každom obrázku vyznačiť, kde sa nachádza hľadaný objekt, ktorý chceme aby vedel model následne rozoznať na iných obrázkoch. Na anotáciu sme použili *open-source* program LabelImg, v ktorom je možné vytvárať anotácie, či už pre YOLO siete alebo vo formáte PascalVOC.



Obr. 4.2 Rozhranie programu LabelImg

Tieto dva typy anotácií sa líšia tým, že YOLO formát určuje súradnice *bounding box* (rámček, kde sa nachádza hľadaný objekt) cez pomery šírky a výšky rámčeka a veľkosti celého obrázku, tieto hodnoty sú vyjadrené desatinným číslo od 0 po 1. Tento formát zahŕňa len základné informácie o objekte. Z PascalVOC formátu je možné zistiť ku akej triede patrí daný objekt, súradnice *bounding box*, veľkosť obrázka a taktiež kde je možné na disku nájsť daný obrázok, ku ktorému sa daná anotácia viaže.

4.3 TRÉNOVANIE MODELU NEURÓNOVEJ SIETE

Pri trénovaní sme pôvodne chceli použiť nástroj Darknet, čo je *open-source* neurónová sieť napísaná v C, ktorá používa ako jadro YOLO sieť a hlavne sa používa pre trénovanie modelov pomocou GPU Nvidia s podporou CUDA jadier. Nakoniec sme použili spomínanú sieť typu YOLO, ale bez použitia Darknet, hlavne kvôli tomu, že poskytuje pred trénované modely, ktoré je možné následne znovu natrénovať so žiadaným datasetom a taktiež má jednoduchšie inštalovanie potrebných knižníc. Existuje viac druhov týchto modelov, líšia sa veľkosťou modelu a jeho presnosťou, zároveň výber modelu záleží na tom, aké výkonné zariadenie a architektúra bude použitá na detekciu.

Z modelov, ktoré poskytuje YOLO, sme zvolili model YOLO v5 medium, ktorý má strednú presnosť detekcie v porovnaní s jeho rýchlosťou. Veľkosť súboru s váhami

nášho modelu je 40 MB, modely typu *large* môžu mať aj 400 MB, v porovnaní s nami vybraným modelom poskytujú väčšiu presnosť s vyššou rýchlosťou, čo záleží na použitom zariadení a veľkú rolu hrá aj to, či je použitá GPU na zvýšenie rýchlosti procesu detekcie.

4.3.1 Proces trénovania

Pred začatím trénovania je potrebné vytvorenie konfiguračných súborov pre trénovanie, ktoré obsahujú cesty ku priečinkom, ktoré obsahujú obrázky pre trénovanie a validáciu a súbory kde sú anotácie objektov, ktoré chceme aby vedela neurónová sieť detegovať. V tomto súbore sú taktiež informácie o počte tried pre trénovanie a ich názvy.

```
path: D:\jakub\Desktop files\Bakalarka\Datasets\Semikron-15-02-2022/
train: images/train
val: images/val

# number of classes
nc: 1

# class names
names: ['solder blob']
```

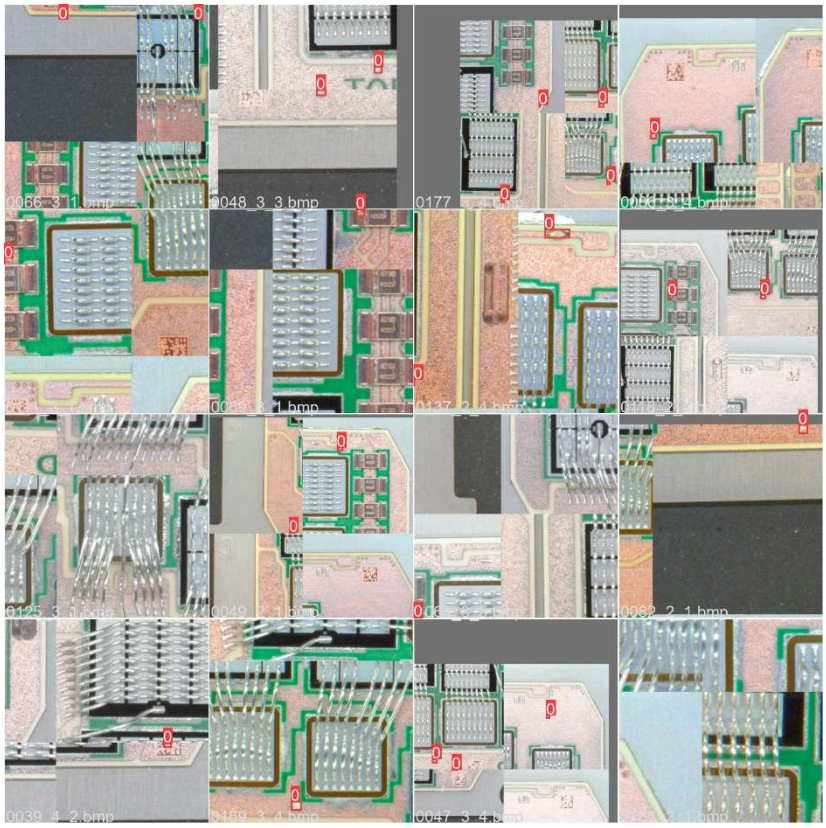
Obr. 4.3 Vzor konfiguračného súboru

Na obr. 4.3 je príklad použitého konfiguračného súboru, parameter *path* určuje, kde sa nachádza dataset v počítači a následne pomocou parametrov *train* a val sú upresnené priečinky, kde sa nachádzajú obrázky použité na trénovanie a verifikáciu modelu počas trénovania. Parametre *nc* a *names* určujú počet tried a ich názvy, pre ktoré chceme model natrénovať. V tomto súbore sa môžu nachádzať aj pokročilé parametre, ktoré následne upravujú architektúru modelu. Patria medzi nich napríklad hĺbka kernelov modelu alebo rýchlosť učenia. Tieto parametre sme sa rozhodli nepoužiť, nakoľko sa nám podarilo dosiahnuť dobrú presnosť modelu so základnými nastaveniami učiaceho algoritmu.

Prvým ktorom pred začatím trénovania je určenie, či chceme trénovať model od začiatku, alebo použijeme model, ktorý má už natrénované váhy a budeme ďalej pokračovať s jeho trénovaním. My sme si zvolili druhú možnosť a na trénovanie sme použili model, ktorý je predtrénovaný pre použitie s 80 triedami. Trénovanie

s modelom, ktorý má už vytvorené váhy je rýchlejšie, presnosť natrénovaného modelu je aj kvôli tomu lepšia a tým je zaručené dosiahnutie dobrých výsledkov, pretože pri trénovaní s neinicializovanými váhami výsledný model nemusí mať požadovanú presnosť a podľa autora YOLO modelov sa táto metóda neodporúča.

Dôležitými parametrami pri trénovaní je tiež nastavenie veľkosti mozaiky pri trénovaní, počet obrázkov v dávke (*batch size*) a počet epoch trénovania. Trénovanie prebieha tak, že algoritmus ide po obrázkoch nachádzajúcich sa v zložke *train* a rozpoznáva označené objekty na daných fotkách.



Obr. 4.4 Mozaika obrázkov z trénovania

Veľkosť batch size taktiež udáva rozmery mozaiky obrázkov, ktorá sa tvorí počas trénovania. V našom prípade bol tento parameter 16, takže výsledná mozaika sa skladá z mriežky obrázkov 4 x 4. Na obrázku 4.4 je možné vidieť ako sú orezané obrázky z datasetu a následne kombinované z viacerých obrázkov do jednotlivých políčok spôsobom, aby sa v každom nachádzal aspoň jeden hľadaný objekt. Pri tomto kroku sa

nám niekedy stalo, že obrázky boli orezané tak, že kvapka cínu bola na kraji obrázku, alebo bolo z nej vidno len polovicu.

Túto chybu sme nakoniec vyriešili pomocou využitia skriptu, ktorý zabezpečuje to, aby sa hľadaný objekt pri orezávaní datasetu nachádzal v určitej vzdialenosti od okrajov, takže nedochádza k tomu, že na mozaikách je len polovica daného *bounding box*.

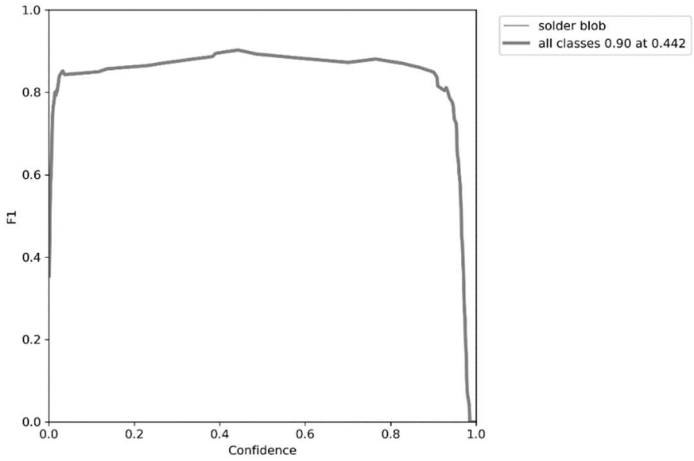
Trénovanie modelu bolo nastavené na 1000 epoch, samotné trénovanie zabralo približne tri hodiny, pričom bolo ukončené po 500 epochách, lebo hodnota *patience* (trpezlivosť) sa dostala nad jej maximálnu hodnotu z dôvodu, že posledných pár epoch sa presnosť modelu pri verifikácií na konci každej epochy nezlepšovala.

4.3.2 Problémy pri trénovaní

Jedným z problémov, ktorý sa vyskytol pri trénovaní bol veľký nárok na pamäť grafickej karty, pôvodne sme chceli použiť kartu s 6 GB pamäte, ale to nepostačovalo na trénovanie s vytvoreným datasetom pri vyhovujúcej veľkosti obrázkov, lebo bolo možné použiť najväčšiu veľkosť obrázka 640 x 640 s počtom 16 v jednej dávke. Nakoniec sme na trénovanie použili grafickú kartu Nvidia RTX 3060, ktorá má 12 GB pamäte, čo bolo dostatočné na natrénovanie modelu s priemernou presnosťou 95 %. Pri použití tejto GPU karty sme dokázali trénovať s veľkosťou obrázkov 2048 x 2048 pri rovnakej veľkosti dávky ako pri menšej veľkosti.

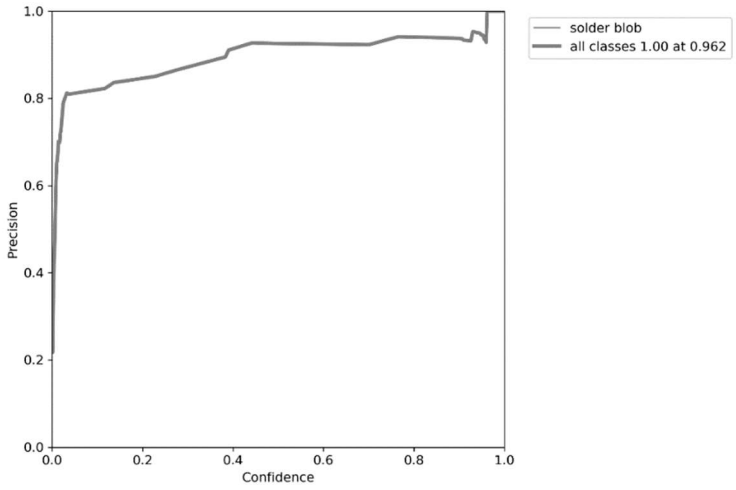
4.3.3 Verifikácia presnosti natrénovaného modelu

Verifikácia modelu bola riešená na Raspberry Pi, kde rýchlosť detekcie záleží na nastavení veľkosti obrázku v príkaze na detekciu. Tento čas sa pohybuje od 5 sekúnd až do 2 minút na obrázok. Výsledok detekcie sa uloží ako fotka s anotovanými objektami do zložky, kde je uložený repozitár yolov5. Každý objekt má tiež priradenú hodnotu istoty, s akou ho sieť detegovala, čím je toto číslo vyššie, tým si sieť bola viac istá že ide o objekt, na ktorý bola trénovaná.



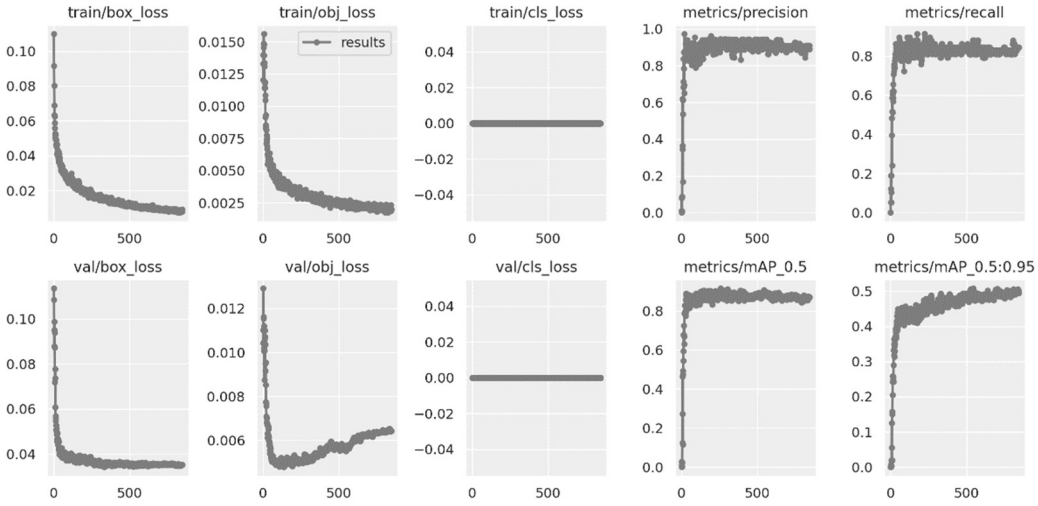
Obr. 4.5 Graf presnosti modelu

Graf na obr. 4.5 udáva rovnováhu medzi presnosťou a *recall*, ďalej sa dá z neho vyčítať najvyššia hodnota presnosti, pri zvolenej istote na hodnote 0,8, ktorá je v tomto prípade 0,85, čo je blízko maximálnej hodnote F1 0,91. Táto krivka určuje použiteľnú presnosť modelu, teda ak by sa istota zvýšila nad 0,8, tak by hodnota *recall* poklesla a tým by sa znížila aj presnosť detekcie nami natrénovaného modelu.



Obr. 4.6 Graf závislosti presnosti a istoty modelu

Presnosť modelu a jeho istota spolu súvisia, pričom ako vidno na obr. 4.6, tak možno pozorovať na grafe optimálny dizajnový bod modelu, tento graf je viazaný ku grafu istota – F1, ktorý obsahuje optimálnu hodnotu istoty detekcie.



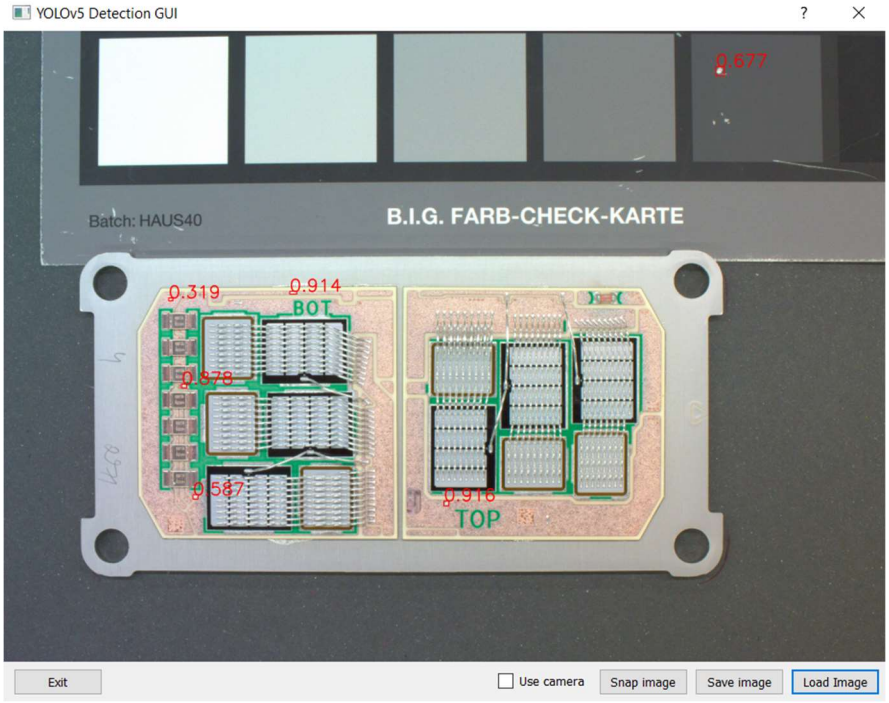
Obr. 4.7 Graf chýb a metrík modelu

Na grafe z obr. 4.7 je vidno priebeh a zmeny parametrov počas jeho trénovania. Vľavo hore a dole je priebeh chýb na *bounding box* počas jednotlivých epoch trénovania a validácie táto hodnota postupne klesala, až po hodnotu cca 0,01, kedy bolo trénovanie ukončené. Ďalej je tu vykreslený parameter mAP (*mean Average precision* – priemerná stredná presnosť), kde vidno, ako hodnota stúpala počas testovania, táto hodnota priamo súvisí s hodnotou *recall* a presnosti. Parameter mAP je definovaný, tak že v prípade modelov s viacerými triedami ak je viac ako 50% pixelov v danom *bounding box* patriacich do danej kategórie, tak je daný región priradený do tej triedy. Tento parameter nemá na náš model žiadny vplyv, keďže my sme trénovali model len s jednou triedou.

Natrénovaný model má 290 vrstiev a spolu má 20 852 934 parametrov, pre porovnanie základný model, z ktorého sme tento model vytvárali má 213 vrstiev a 7 225 885 parametrov.

4.4 GUI ROZHRANIE PRE DETEKCIU

Detekcia pomocou natrénovaného modelu je realizovaná v jazyku Python. Grafické rozhranie sme urobili pomocou knižnice PyQt5, vstup na detekciu je buď načítanie obrázku zo súboru, alebo využitie Rasbperry Pi kamery.



Obr. 4.8 GUI rozhranie

Načítanie obrázku do súboru je riešené pomocou kontextového okna, kde užívateľ zvolí žiadaný súbor, ktorý je následne načítaný do programu. Prvým ktorom pri detekcii je zlúčenie súboru váh so základným modelom, na ktorom následne prebehne detekcia so zvoleným obrázkom. Tu sa naskytá jedna nevýhoda tohto programu, lebo nie je možné s pomocou knižnice Torch načítať len váhy modelu, nakoľko je potrebné aby si knižnica stiahla z internetu základný model a následne natrénované váhy modelu zlúčila so stiahnutým základným modelom. Keďže k tejto operácií je potrebný prístup k internetu, tak nie je možné aby detekcia bežala *offline*. Následne sa v okne aplikácie zobrazí zdetegovaný obrázok a každému objektu je priradená hodnota presnosti detekcie. Aplikácia tiež poskytuje možnosť uloženia anotovaného obrázku do súboru.

Druhou možnosťou realizácie detekcie pomocou aplikácie je využitím kamery pripojenej k Raspberry Pi. V tomto móde sa zobrazuje v okne obraz, ktorý sníma kamera a po stlačení tlačidla *Snap Image* dôjde k zaznamenaniu fotky, na ktorej následne prebehne detekcia. Avšak s touto možnosťou sa vyskytli problémy, hlavne kvôli tomu, že bolo treba použiť inú verziu operačného systému, pretože kamera nefungovala na staršej verzii, ale v tejto staršej verzii bol problém s knižnicou Torch.

4.4.1 Problémy pri použití Raspberry Pi kamery

V najnovšej verzii Raspberry OS s označením Bullseye ešte nie je implementované použitie tejto kamery pomocou novej knižnice libcamera a python skriptov. Kvôli tomuto sme museli použiť staršiu verziu systému s označením Buster, ktorý používa kameru pomocou pôvodnej knižnice picamera, ktorá funguje bez problémov. Oficiálna verzia tohto systému je len v 32-bitovej architektúre, dajú sa nájsť aj verzie so 64-bitovou, ale je ich málo a môžu sa na nich objaviť iné problémy. Avšak v tejto verzii bol problém s inštaláciou knižnice Torch potrebnej na detekciu obrázkov s využitím nášho natrénovaného modelu. Nakoniec sme sa rozhodli len pre detekciu zo súboru, keďže sa nám nepodarilo vyriešiť daný problém s kamerou, hlavne kvôli chýbajúcim knižniciam a podpory zo strany operačného systému.

4.5 AUTOMATIZOVANÁ DETEKCIA

Ďalším krokom v bakalárskej práci je návrh využitia detekcie na pracovisku vo výrobe. Toto sa dá realizovať spôsobom, že budú manuálne snímané fotky dosiek a tie budú načítané do programu na detekciu, ale využitie funkcie YOLO modelov na rozpoznávanie videa. V tomto prípade by detekcia bežala stále a v prípade nájdenia kvapiek cínu na doske, by sa táto informácia zobrazila priamo v obraze z kamery.

Iným spôsobom použitia automatizovanej detekcie by bolo využitie tlačidla, pripojeného na GPIO *pin*, ktoré by po stlačení vytvorilo snímku dosky a následne by spustilo detekciu a výsledky by sa zobrazili na obrazovke, poprípade by sa uložili do databázy spolu s identifikačným číslom dosky.

Taktiež by táto detekcia mohla obsahovať počítadlo nájdených kvapiek cínu, z čoho by bolo možné neskôr spraviť štatistiku počtu chybných kusov voči tým čo neobsahovali defekt a taktiež koľko kvapiek v priemere sa na týchto chybných doskách nachádzalo.

5 VÝSLEDKY PRÁCE

Výsledkom tejto práce je model schopný rozpoznávať kvapky cínu na doskách plošných spojov. Model sme trénovali pomocou platformy YOLO na stolnom počítači, testovanie presnosti natrénovaného modelu sme realizovali na Raspberry Pi. Na tabuľke 5-1 je prehľad časov detekcie od rôznych nastavení veľkosti vstupného obrázka.

Tabuľka 5-1 Presnosť detekcie v závislosti od veľkosti obrázka

Veľkosť vstupného obrázka	**Rýchlosť detekcie**	**Počet detegovaných kvapiek**
Originál (4608x3288)	-	6
600	3,1 s	0/6
1024	5,2 s	2/6
2048	21,9 s	4/6
3072	53,1 s	5/6
4096	1 min 29,2 s	5/6

Keď porovnáme rýchlosť a presnosť detekcie pri rôznych časoch, tak z toho vyplýva, že síce detekcia prebehla rýchlejšie, ale zas pri nej boli niektoré kvapky cínu na obrázku nerozoznané a naopak, keď je parameter veľkosti obrázka väčší, tak detekcia trvá skoro dve minúty, ale zas jej výsledok je správny. Na detekciu bol použitý príkaz, ktorý je súčasťou yolov5 repozitára, všetky parametre okrem súboru s váhami a veľkosti obrázku boli ponechané v ich predvolených hodnotách. Z tabuľky taktiež vyplýva, že nad určitou hodnotou veľkosti vstupného obrázka už nedosiahneme lepší výsledok detekcie. V tomto prípade je to kvôli tomu, že na testovacom obrázku je šiesta kvapka veľmi malá a ešte k tomu sa nachádza na rezistore určenom na meranie prúdu, takže má model problém ju na tomto mieste identifikovať. Skúšali sme aj vyššiu hodnoty veľkosti ako je veľkosť originálneho obrázka, ale spôsobí to nedostatok pamäte a výpočtového výkonu v Raspberry Pi. Jednou z najťažších úloh bolo nájsť najlepší pomer medzi rýchlosťou a presnosťou detekcie.

Zostava na využitie tejto bakalárskej práce v praxi by vyšla podľa hodnôt uvedených v tabuľke (*tabuľka 5-2*) na 222,49 €. V prehľade je rátané, že na pracovisku by bolo použité Raspberry Pi s originálnym 7“ displejom, pre kompaktné prevedenie zariadenia. Napríklad by sa ešte ku kamere dalo pridať osvetlenie na zabezpečenie konzistentných farieb snímok.

Tabuľka 5-2 Prehľad zostavy pracoviska na detekciu

Produkt	**Cena**	**Poznámka**
Raspberry Pi 4	66,67 € / 90,86 €	4 alebo 8 GB RAM model
Pi kamera v2	27,23 €	verzia s alebo bez infračerveného filtra
Napájací zdroj USB-C	9,98 €	5 V a min. 3A
Krabička pre Raspberry	12,08 €	pasívne alebo aktívne chladenie
Displej	74,70 €	použiteľný originálny RPi displej alebo HDMI monitor
Myš	13,29 €	
Klávesnica	18,54 €	

Na snímanie je možné použiť aj kameru pripojenú cez USB, ale tu sa môžu naskytnúť problémy s kompatibilitou určitých modelov kamier s Raspberry OS. Plné využitie kamery bude však možné až po vydaní plnej podpory knižníc pre najnovšiu verziu systému s názvom Bullseye, keďže v dobe písania tejto práce ešte neexistovala knižnica pre Python pre komunikáciu s kamerou, takže sme boli nútení použiť verziu Buster, kde bol problém s knižnicou Torch, takže sa nám nepodarilo zrealizovať detekciu s použitím Rasperry Pi kamery.

Spotreba nášho riešenia je oveľa nižšia, ako pri použití klasického počítača, aj keď naše riešenie je limitované výpočtovým výkonom, ale na druhej strane táto alternatíva zaberá menej miesta, keďže je použitý jednodoskový počítač.

Obr. 5.1 RPi s kamerou

Raspberry Pi je napájané s pomocou USB-C adaptéra, ktorý má na výstupe 5 V, v tabuľke 5-3 je prehľad prúdov a výkonov počas rôznych činností. Hodnoty boli merané pomocou USB testera, ktorý ukazuje okamžité hodnoty prúdu a napätia a je možné z neho odčítať aj spotrebovanú energiu v mAh. Operačný systém je uložený na USB kľúči hlavne kvôli tomu, že je možné z neho čítať rýchlejšie ako z SD karty, ktoré sú síce lacnejšie, ale majú kratšiu životnosť a nedokážu zvládnuť veľký počet zápisov.

Tabuľka 5-3 Spotreba RPi

Činnosť	**Prúd dodávaný zdrojom [A]**	**Výkon [W]**
Spustenie systému	1,1	5,5
Chod naprázdno	0,69	3,42
Detekcia	1,5	7,5

Siete YOLO sa úspešne používajú napríklad na detekciu rôznych predmetov, vozidiel, alebo je možné na internete nájsť aj model, ktorý je natrénovaný na rozpoznávanie výtlkov na cestách, čo pomáha pri analýze stavu ciest.

ZÁVER

Cieľom práce bolo vytvorenie modelu neurónovej siete na rozpoznávanie kvapiek cínu na doskách plošných spojov. Vytvorili sme dataset, ktorý sa použil na trénovanie modelu siete, s ktorým sme následne zisťovali jeho presnosť detekciou rôznych fotiek. Následne sme na túto detekciu vytvorili grafickú aplikáciu, pomocou ktorej je možné jednoducho analyzovať obrázky na Raspberry Pi či už zo súboru, alebo pomocou externej kamery. Jednou z výziev bolo nájdenie dobrého pomeru medzi rýchlosťou a presnosťou detekcie vzhľadom na obmedzený výpočtový výkon dosky.

Presnosť nami natrénovaného modelu je možné ešte vylepšiť tým, že bude vytvorený väčší dataset, ako s ktorým sme trénovali náš model, takže model bude vedieť rozoznať viac typov a veľkostí kvapiek cínu na rôznych miestach a zníži sa tiež tým percento nesprávnych detekcií na fotkách. Taktiež pre vytvorenie dobrého modelu je podstatné, aby kvapky cínu v datasete mali rôzne tvary a boli umiestnené aj na miestach, kde sa často v realite nevyskytujú. Následne presnosť sa dá zlepšiť aj použitím fotiek z datasetu, ktoré majú väčšie rozlíšenie k čomu následne nadväzuje využitie väčšieho modelu YOLO, ktorý by poskytoval väčšiu presnosť pri detekcii.

Rýchlosť detekcie modelu sa dá vylepšiť použitím iného typu základného modelu, alebo realizáciou detekcie na inej platforme, kde je možné použiť GPU akceleráciu na dosiahnutie nižšieho času potrebného na spracovanie fotky pomocou natrénovaného modelu. Na platforme kde je k dispozícií GPU akcelerácia je možné detekciu robiť pomocou obrázkov s väčším rozlíšením, keďže limitácie tejto platformy sú oveľa nižšie, ako je tomu na Raspberry Pi, kde presnosť detekcie je značne obmedzená.

Detekcia pomôže pri kontrole ľudského faktoru, keďže chybovosť modelu je určite menšia, ale môžu sa vyskytnúť aj prípady kedy má natrénovaný model problém rozoznať určité kvapky, hlavne keď sa nachádzajú medzi kontaktmi, kde ich je ťažšie vidieť aj voľným okom.

ZOZNAM POUŽITEJ LITERATÚRY

[1] Defect Detection in Printed Circuit Boards Using You-Only-Look-Once Convolutional Neural Networks. [online]. Venkat Anil Adihatla a kol.: MDPI. 2020. https://www.mdpi.com/2079-9292/9/9/1547

[2] Neural Networks. [online] IBM Cloud Education: IBM. 2020. https://www.ibm.com/cloud/learn/neural-networks

[3] 8 Types of Neural Networks. [online] Utsav Mishra. 2022. https://www.analyticssteps.com/blogs/types-neural-networks

[4] Convolutional Neural Networks. [online] IBM Cloud Education:IBM. 2020. https://www.ibm.com/cloud/learn/convolutional-neural-networks

[5] Max Pooling In Convolutional Neural Networks Explained. [online] Deep Lizard. 2018. https://deeplizard.com/learn/video/ZjM_XQa5s6s

[6] What is dropout in deep neural networks. [online] Mary E. Shacklett. 2021. https://www.techtarget.com/searchenterpriseai/definition/dropout

[7] How ReLU works in convolutional neural network. [online] Knowledge Transfer. 2020. https://androidkt.com/how-relu-works-in-convolutional-neural-network/

[8] What is a Raspberry Pi. [online] Raspberry Pi Foundation. 2020. https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/

[9] Know all about Raspberry Pi Board Technology. [online] WatElectronics. 2019. https://www.watelectronics.com/know-all-about-raspberry-pi-board-technology/

[10] Raspberry Pi 4 specs and benchmarks. [online] Rob Zwetsloot. 2019. https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks

[11] Python Programming. [online] Steven F. Lott. 2008. https://www.linuxtopia.org/online_books/programming_books/python_programming/python_ch01.htm

[12] Average precision. [online] Sancho McCann. 2011. https://sanchom.wordpress.com/tag/average-precision/

[13] A Single Number Metric for Evaluating Object Detection Models. [online] Peter Lebiedzinski. 2021. https://towardsdatascience.com/a-single-number-metric-for-evaluating-object-detection-models-c97f4a98616d

[14] A Forest Fire Detection System Based on Ensemble Learning. [online] Renjie Xu a kol.:MDPI. 2021. https://www.mdpi.com/1999-4907/12/2/217

[15] 13 Common PCB Soldering Problems to Avoid. [online] Helen. 2021. https://www.seeedstudio.com/blog/2021/06/18/13-common-pcb-soldering-problems-to-avoid

ČESTNÉ VYHLÁSENIE

Vyhlasujem, že som zadanú diplomovú prácu vypracoval samostatne, pod odborným vedením konzultanta diplomovej práce Ing. Petra Klča, PhD. a vedúceho doc. Ing. Dušana Koniara, PhD. a používal som len literatúru uvedenú v práci.

Súhlasím so zapožičiavaním bakalárskej práce.

V Žiline dňa 16.05.2022

_____

podpis

Prílohová časť

Zoznam príloh

Príloha A: Obsah repozitára so zdrojovými kódmi

- Zdrojový kód GUI aplikácie
- Váhy natrénovaného modelu neurónovej siete
- Popis na použitie modelu
- Ukážka datasetu

URL adresa: https://github.com/Jakuko99/realizacia_NN_na_Rasbperry_Pi