Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Segmentace včel na obrázcích plástů

Bakalářská práce

Sergei Filitov

Školitel: Ing. Miroslav Skrbek, Ph.D.

Konzultant: RNDr. Alena Bruce, Ph.D.

České Budějovice 2024

Bibliografické údaje:

Filitov, S., 2024: Segmentace včel na obrázcích plástů. [Bee segmentation on honeycomb images. Bc. Thesis, in Czech.] – 75 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace:

Cílem práce je provést rešerši přístupů pro segmentaci obrazu a klasifikaci textur neuronovými sítěmi, případně v kombinaci s klasickými přístupy. Vybrat vhodný přístup pro segmentaci včel na fotografiích plástů s důrazem na metody, které umožní považovat včelu za texturu s určitou orientací. Využít existující implementace nebo upravenou verzi frameworku. Zvážit možnosti eliminace vlivu kvality obrázku v různých jasových podmínkách. Provést sérii experimentů a zhodnotit výsledky z hlediska přesnosti stanovení obrysu včely.

Klíčová slova:
Python, Segmentace, Včela, YOLO, UNet

Annotation:

The goal of this work is to conduct research on approaches for image segmentation and texture classification using neural networks, possibly in combination with classical approaches. To select a suitable approach for segmenting bees in photographs of honeycomb frames, with an emphasis on methods that consider a bee as a texture with a certain orientation. To utilize existing implementations or modified versions of frameworks. To consider options for mitigating the influence of image quality under different lighting conditions. To conduct a series of experiments and evaluate the results in terms of the accuracy of bee contour determination.

Keywords:
Python, Segmentation, Bee, YOLO, UNet

Prohlašuji, že jsem autorem této kvalifikační práce a že jsem ji vypracoval pouze s použitím pramenů a literatury uvedených v seznamu použitých zdrojů.

V Českých Budějovicích dne: _____ Podpis autora: _____

Poděkování:

Rád bych touto cestou poděkoval Ing. Miroslavu Skrbkovi, Ph.D. za jeho odborné vedení a konzultace ohledně implementace a RNDr. Aleně Bruce, Ph.D. za její cenné rady v problematice včel.

Samozřejmě bych rád poděkoval své rodině za podporu ve studiu.

Obsah

1 Úvod

Medonosné včely jsou v zemědělství zásadní a hrají klíčovou roli v životním cyklu rostlin jako jeden z nejdůležitějších opylovačů. Více než 75 % ze 115 nejpopulárnějších druhů plodin na celém světě je závislých na opylování hmyzem, nebo z něj alespoň těží. Včely jsou zodpovědné za opylení přibližně 70 % všech druhů plodin na celém světě. Především z tohoto důvodu nastávají vážné ekonomické důsledky v případech, kdy včely bojují s patogeny, parazity, škůdci a dalšími faktory ovlivňujícími přežití včelstva. [1]

Existují nevysvětlené případy poruchy zhroucení včelstev (**colony collapse disorder** nebo **CCD**). Ztráty včelstev jsou považovány za multifaktoriální problém kombinující vliv biotických a abiotických stresorů, např. výživy, pesticidů a změny klimatu. Znalost fyziologického a imunologického stavu včelstev je proto klíčová při pomoci včelařům čelit potenciálním problémům a ochraně včel. [1]

Většina CCD se vyskytuje v zimě, což je pro včelstva nejrizikovější období. Proto je nanejvýš důležité studovat změny v organismu včely medonosné v zimním období. [1]

Automatizovaná segmentace včel na obrázcích plástů hraje významnou roli při monitorování zdraví včelových kolonií. Tato segmentace umožňuje identifikovat jednotlivé včely na obrázcích plástů, což je klíčovým prvkem pro sledování změn v chování, zdraví a populace včel. Výzkumníci a včelaři zjištěním známek onemocnění, stresu nebo environmentálních faktorů, které ovlivňují včely, mohou včas zasáhnout a zmírnit potenciální hrozby pro zdraví úlů.

1.1 Členění práce

Kapitola *3* se zabývá rozborem úlohy segmentace. Tato část práce čerpá informace z odborné literatury a poskytuje potřebné znalosti pro pochopení problematiky sémantické segmentace.

Kapitola *4* obsahuje rešerši segmentačních neuronových sítí, přehled vybraných architektur a formátů jejich anotací. Tato část práce čerpá informace z odborné literatury a technické dokumentace pro pochopení principů sémantické segmentace každé vybrané sítě.

Kapitola *5* popisuje implementaci vybraných architektur.

Kapitola *6* popisuje provedené experimenty s hlubokými neuronovými sítěmi.

Kapitola *7* poskytuje čtenáři diskusi nad výsledky experimentů.

2 Cíle práce

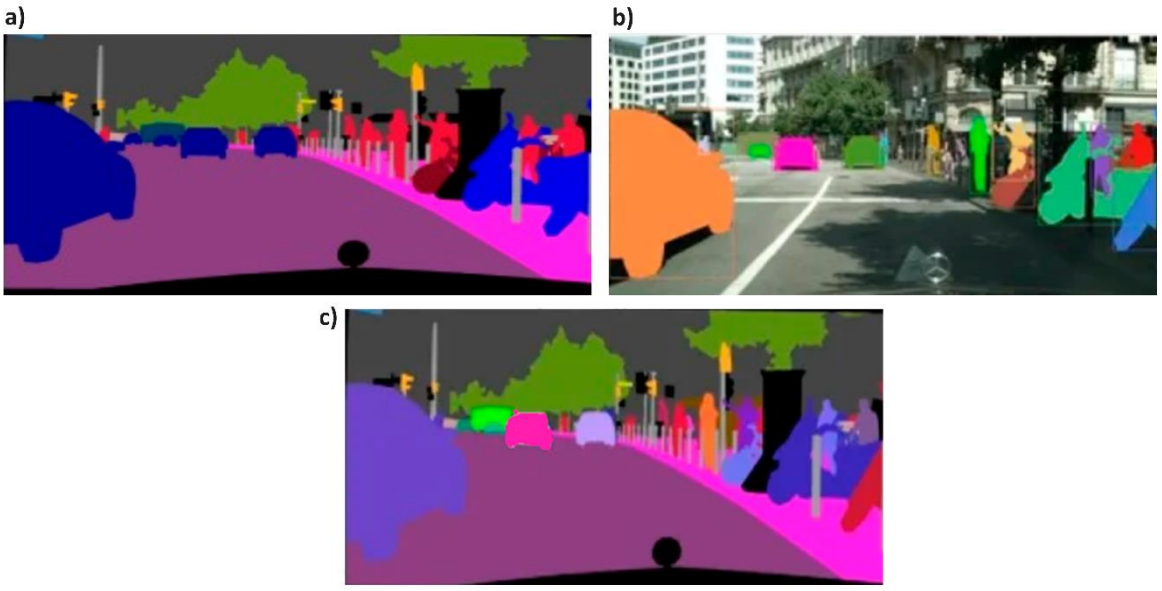
1. Provést rešerši přístupů pro segmentaci obrazu a klasifikaci textur neuronovými sítěmi, případně v kombinaci s klasickými přístupy.
2. Vybrat vhodný přístup pro segmentaci včel na fotografiích plástů. Orientovat se na metody, kdy včela bude moci být považována za texturu s určitou orientací.
3. Využít existující implementace nebo upravené existující implementace, případně přímo implementovat ve známých frameworcích, například Tensorflow.
4. Zvážit možnosti eliminace vlivu kvality obrázku v různých jasových podmínkách.
5. Provést sérii experimentů a výsledky zhodnotit z hlediska přesnosti stanovení obrysu včely.

3 Teorie

3.1 Sémantická segmentace

Segmentace obrazu hraje zásadní roli v široké škále systémů vizuálního porozumění. Jejím hlavním cílem je generovat hustou predikci pro daný obrázek, tj. přiřadit každému pixelu předem definovanou třídu (sémantická segmentace), nebo spojovat každý pixel s instancí objektu (instanční segmentace), či kombinovat oba přístupy (panoptická segmentace), což usnadňuje organizaci pixelů s podobnou sémantikou do smysluplných konceptů na vyšší úrovni. [2]

Obrázek 1 ukazuje druhy segmentace. 1.a) sémantická segmentace – označuje objekty stejné třídy jako jednu kategorii bez rozlišení mezi nimi. 1.b) instanční segmentace – rozlišuje objekty stejné třídy jako samostatné instance. 1.c) panoptická segmentace – kombinuje sémantickou segmentaci a instanční segmentaci pro komplexní porozumění scény na sémantické a instanční úrovni.



Obrázek 1 – Druhy segmentace [3]

Úkolem segmentace je rozdělení obrazu na části, které odpovídají objektům v obraze zachyceným za účelem vyčlenění zájmových oblastí od zbytku obrazu. V závislosti na míře úspěchu je možno hovořit o částečné nebo úplné segmentaci. [4]

Sémantická segmentace přiřazuje každému pixelu a každému objektu v obraze konkrétní třídu. Detekce instancí identifikuje a lokalizuje jednotlivé objekty v obraze a případně poskytuje ohraničující box kolem každé instance.

Tato úloha je výzvou kvůli potřebě přesného rozlišení mezi instancemi objektů, zejména při překrývání objektů nebo při podobných tvarech a barvách. Velkou roli hraje kvalita obrazu zejména při rozpoznávání detailů a oddělování objektů na snímcích s nižším rozlišením či nedostatečné expozici a jasových podmínkách, které mohou ovlivnit viditelnost objektů a přesnost detekce a zhoršit úplnost segmentace.

3.2 Segmentace tradičními metodami

Tradiční metody segmentace obrázků využívají techniky zpracování obrazu založené na matematických a statistických principech a těží z informace o prostorových příznacích objektů.

Tyto metody mohou být stále účinné v řadě aplikací zejména tam, kde není k dispozici dostatečné množství dat pro trénování složitějších modelů hlubokého učení.

SIFT (Scale Invariant Feature Transform) [5] je metoda segmentací obrázků využívající tzv. charakteristické body a lokální orientovaný histogram směrů gradientů.

Charakteristický bod má různé vlastnosti, které ho odlišují od okolních bodů, a je invariantní vůči transformacím. V SIFT jsou charakteristické body nalezeny jako lokální extrémy v různých měřítkách prostoru a jsou identifikovány na základě kontrastu a ostrosti okolních oblastí. Tyto body slouží jako referenční body, na kterých se provádí výpočet pomocí lokálních histogramů orientovaných gradientů.

Charakteristický bod je rozdělen do čtyř čtvercových sektorů, kde se v každém pixelu vypočítá gradient obrázku, jeho směr a velikost. Velikosti gradientů jsou váženy exponenciálně klesající vzdáleností od charakteristického bodu, což zabraňuje prudkým změnám deskriptoru při malých pozicích okna. Výsledný SIFT deskriptor je vektor s 128 komponentami, normalizovaný pro stabilitu vůči změnám jasu. SIFT se stal standardem v počítačovém vidění a vychází z myšlenky lokálních histogramů gradientů jasu.

HOG (Histogram of Oriented Gradients) [6] je metoda využívající speciálních bodů využívaných pro rozpoznávání objektů. Tato metoda počítá počet směrů gradientu v lokálních oblastech obrázku. Podobně jako SIFT vytváří histogramy gradientů, ale v husté síti rovnoměrně rozložených buněk.

Síť je mřížka s definovaným počtem řádků a sloupců, která rozděluje obrázek na pravidelné menší části, které se nazývají buňky, sloužící k analýze.

Buňky jsou jednotlivé části obrazu, do kterých je obrázek rozdělen pomocí sítě. Každá buňka obsahuje určitý počet pixelů a slouží k výpočtu lokálních histogramů orientovaných gradientů (HOG). Gradienty jsou vypočítávány uvnitř každé buňky a jsou použity k popisu směru a intenzity změn v intenzitě obrazu v dané oblasti. Počet buněk a jejich velikost jsou parametry algoritmu, které mohou ovlivnit výsledný výkon a přesnost analýzy.

Algoritmus HOG používá normalizaci překrývajícího se lokálního kontrastu pro zvýšení přesnosti. HOG se zakládá na předpokladu, že vzhled a forma objektu mohou být popsány směrem hran.

SVN (Support Vector Networks) [7] je metoda učení s učitelem používaná pro klasifikaci a regresní analýzu. Převádí původní vektory do prostoru vyšší dimenze a hledá separační hyperplochy s maximálním odstupem mezi třídami. Tímto způsobem se snaží minimalizovat průměrnou chybu klasifikátoru. SVN patří do rodiny lineárních klasifikátorů a je efektivní pro rozdělení tříd v prostoru s maximálním odstupem.

SIFT a HOG jsou metody zaměřené na extrakci příznaků, zatímco SVN slouží k jejich klasifikaci. Tyto metody jsou účinné pro jednodušší scény.

Segmentace včel na obrazcích plástů je komplexní úloha, protože zahrnuje variabilitu ve velikosti, tvaru, postavení a překrývání včel, a to v různých světelných podmínkách a prostředích. Tradiční metody, jako jsou SIFT a HOG, mohou být při takové úloze omezeny v úspěšnosti, protože jsou navrženy pro zpracování jednodušších scén s dobře definovanými objekty a hranicemi. Segmentace včel na obrazcích plástů je prostředím, kde tradiční metody dosahují svých limitů.

Proto se tato práce zaměřuje na neuronové sítě založené na konvolučních vrstvách. Takové sítě mají schopnost se adaptovat a učit složité vzory a hierarchie příznaků vstupních dat.

3.3 Segmentace neuronovými sítěmi

Neuronové sítě na segmentaci se učí na základě dat, kterými jsou obrázky spolu s nákresy, které označují, jaké části obrázku patří k jednotlivým segmentům. Síť se snaží naučit se rozpoznávat různé vzory a vlastnosti v obrazech, které odpovídají jednotlivým segmentům.

Segmentační síť pracuje se vstupním obrazem s velikostí, které jsou definované konfigurací modelu, např. 1024×1024. Obraz prochází enkodérem specializovaným na extrakci příznaků různých úrovní abstrakce. Enkodér identifikuje základní vizuální prvky, jako jsou hrany a textury, a postupně se dostává k složitějším strukturám.

Extrahované příznaky jsou dekódovány do prostorové mapy sémantických segmentů pomocí dekodéru. Konvoluční vrstvy zpracovávají dekódované mapy a zachovávají klíčové příznaky obrazu, jako jsou hrany, tvary a vzory, a zároveň zlepšují schopnost sítě rozpoznat a interpretovat tyto příznaky.

Segmentační masky představují přiřazení konkrétní třídy pro každý pixel v obraze. Existuje několik druhů segmentačních masek:

- Binární maska – je 2D matice, kde každý pixel je přiřazen hodnotě 0 nebo 1. Používá se pro binární segmentační úkoly, kde každý pixel patří buď k objektu, nebo k pozadí.
- Barevná maska – přiřazuje jedinečné RGB hodnoty různým třídám nebo segmentům. Používá se při segmentacích s více třídami, každou reprezentovanou odlišnou barvou.
- Indexovaná maska – používá obrázek, kde každá hodnota pixelu reprezentuje označení třídy nebo ID segmentu. Umožňuje uložení segmentačních masek s nižším počtem bitů na pixel než barevné masky.
- Run-Length Encoding (RLE) maska – je efektivnější způsob reprezentace segmentační masky. Nahrazuje sekvence stejných pixelů jejich hodnotou a délkou sekvence. Efektivní pro ukládání řídkých masek s dlouhými sekvencemi identických hodnot.
- Polygonální maska – specifikuje vrcholy polygonu, který ohraničuje segmentovanou oblast. Užitečné pro nepravidelně tvarované objekty nebo situace, kde polygon poskytuje lepší reprezentaci než obdélníková maska.
- Instanční maska – variace segmentační masky, která přiřazuje jedinečný identifikátor každé instanci objektu. Podstatné pro instanční segmentaci, kde je klíčové rozlišovat mezi jednotlivými objekty stejné třídy.

Následné operace převzorkování (**upsampling)** přispívají k obnově detailů a zvětšení rozlišení mapy příznaků, což umožňuje sítí lépe zachytit detaily v obraze při provádění segmentace instancí.

Aktivační funkce, jako je ReLU (Rectified Linear Unit), SoftMax apod., jsou aplikovány na výstupy konvolučních vrstev. Aktivační funkce přidávají nelinearitu do výstupů, což umožňuje síti zachytit složité vzory v datech a reprezentovat nelineární vztahy.

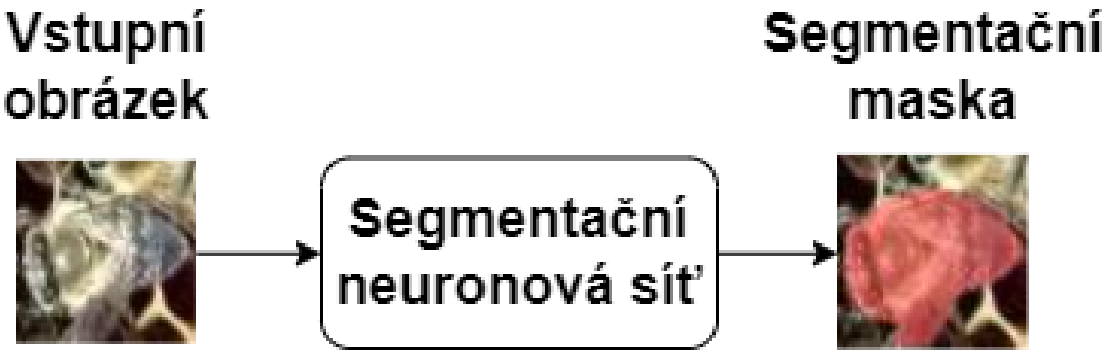
Optimalizátory, jako například SGD, Adam nebo RMSprop, aktualizují váhy sítě tak, aby minimalizovaly ztrátovou funkci (loss function). Adaptují síť na vzory v trénovací množině s cílem dosažení optima.

Výsledná segmentační mapa obsahuje přiřazení konkrétní třídy nebo kategorie pro každý pixel v obraze. Během trénování je síť učena na anotovaných datech, kde jsou známy správné segmentace instancí, a učící algoritmus se snaží nalézt optimální konfiguraci vah pro danou trénovací množinu, takovou, aby minimalizovala rozdíl mezi predikovanými a skutečnými segmentacemi.

Po ukončení učení může segmentační síť na neanotovaných obrazech provádět inferenci, což znamená, že dokáže předpovídat a označit oblasti obrázků na základě svých naučených znalostí.

Výsledkem inference je informace o detekovaných objektech, jejich třídách, konfidencích, souřadnicích bounding boxů a segmentačních maskách. Segmentační masky poskytují informace o vnitřní struktuře detekovaných objektů, identifikují pixely, které k objektům patří, a umožňují přesně vymezit hranice a obrys detekovaných objektů.

Obrázek 2 zobrazuje princip segmentace neuronovými sítěmi. Na vstup neuronové sítě přichází obrázek, na který je aplikována inference. Výsledkem inference je segmentační maska.



Obrázek 2 – Princip fungování segmentační neuronové sítě, [vlastní zdroj]

3.4 Hodnocení kvality segmentace

Segmentační neuronová síť vytváří predikce, generuje segmentační masky a bounding boxy (nepovinné), které model označuje jako součást objektu.

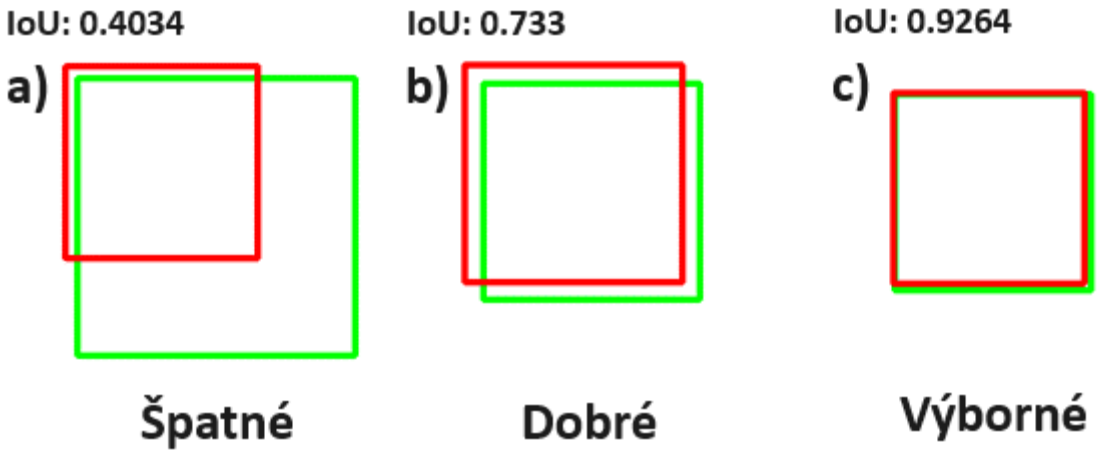
Kvalitu segmentace lze vyhodnotit vizuálně i pomocí číselných metrik. Následující metriky poskytují numerický pohled na kvalitu segmentace.

Intersection Over Union (IoU) neboli **průnik ku sjednocení** je metrika, která měří plochu překrývání predikovaného bounding boxu nebo segmentační masky s příslušnou anotací (**ground truth**). Metrika se definuje jako:

$$IoU = \frac{area(prediction \cap ground\_truth)}{area(prediction \cup ground\_truth)} \quad (1)$$

Nabývá hodnoty mezi 0 (žádný průnik) a 1 (plný průnik).

Obrázek 3 znázorňuje 3 příklady IoU pro bounding boxy: 1.a) 0.4 – malý průnik, 1.b) 0.7 – dobrý průnik, 1.c) 0.9 – výborný průnik.



Obrázek 3 – IoU bounding boxů vizualizace, [vlastní zdroj]

Vyšší hodnoty IoU naznačují lepší shodu mezi predikcí a skutečnými objekty v obraze.

Confidence Score neboli **práh konfidence** je hodnota používaná modelem k určení, zda má klasifikovat predikci jako pozitivní, nebo negativní. Pokud výstup modelu (IoU) je vyšší než tato hodnota, pak je klasifikován jako pozitivní. Tento práh stanoví modelu, jak jistý musí být, aby považoval predikci za pozitivní.

True Positive (TP) neboli **skutečně pozitivní odpověď** nastává, když model předpovídá pozitivní výsledek a skutečný výsledek je pozitivní. $IoU \geq práh\ konfidence$

False Positive (FP) neboli **falešně pozitivní odpověď** nastává, když model předpovídá pozitivní výsledek, ale skutečný výsledek je negativní. $IoU < práh\ konfidence$

False Negative (FN) neboli **falešně negativní odpověď** nastává, když model předpovídá negativní výsledek, ale skutečný výsledek je pozitivní.

True Negative (TN) neboli **skutečně negativní odpověď** nastává, když model předpovídá negativní výsledek a skutečný výsledek je negativní. Nepoužívá se.

Precision neboli **přesnost** je poměr skutečných pozitivních predikcí k celkovému počtu předpokládaných pozitivních odpovědí: skutečně pozitivních (TP) a falešně pozitivních (FP). Měří přesnost pozitivních predikcí provedených modelem. Vyšší precision znamená menší počet falešně pozitivních odpovědí.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\_detections} \tag{2}$$

Recall neboli **úplnost** je poměr skutečně pozitivních predikcí k celkovému počtu skutečně pozitivních (TP) a falešně negativních (FN). Měří schopnost modelu identifikovat všechny skutečně pozitivní případy. Vyšší recall naznačuje, že model lépe identifikuje existující objekty v obraze.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\_ground\_truths} \tag{3}$$

F1 skóre je harmonický průměr precision a recall. Pro výpočet se nejprve vytvoří binární masky pro predikované segmenty a skutečné segmenty. Poté se vypočítají precision a recall pro každý pixel v obraze. Precision vyjadřuje, jak správně model identifikoval pozitivní pixely (segmenty), zatímco recall vyjadřuje, jak mnoho ze skutečných pozitivních pixelů bylo správně identifikováno modelovými predikcemi.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{4}$$

Vyšší hodnota naznačuje lepší vyvážení mezi přesností a úplností v predikcích segmentačního modelu.

Precision-Recall křivka je grafickým vyjádřením vztahu mezi precision a recall pro různé hodnoty prahů uvažované při vyhodnocení výsledků modelu. Pro výpočet křivky jsou predikcemi modelu seřazeny podle jejich konfidencí, a následně je aplikován práh, který určí, jaké predikce jsou považovány za pozitivní nebo negativní.

Average Precision (AP) neboli **střední přesnost** je plocha pod touto křivkou, která ukazuje kvalitu učení modelu pro jednotlivé třídy z datové sady. Vyšší hodnota naznačuje lepší výkon v celém rozsahu prahů.

Výpočet plochy pod precision-recall křivkou lze provést pomocí interpolace mezi n body grafu (predikcemi modelu) a vyjádřit následujícím způsobem:

$$AP = \int_0^1 p(r)dr \tag{5}$$

kde $p(r)$ představuje interpolovanou hodnotu přesnosti pro danou hodnotu úplnosti r.

Mean Average Precision (mAP) neboli **střední průměrná přesnost** je průměr hodnot AP pro celkový počet jednotlivých tříd.

$$mAP = \frac{1}{num\_classes} \times \sum_{i=0}^{num\_classes} (AP)_i \tag{6}$$

Tato metrika poskytuje pohled na účinnost modelu při detekci a segmentaci objektů. Vyšší mAP indikuje lepší schopnost modelu správně identifikovat objekty při různých okolnostech.

Základem pro výpočet mAP je práh pro IoU. Standardně se mAP vypočítává pro prahovou hodnotu 0,5 (mAP50) a pro prahové hodnoty od 0,5 do 0,95 s krokem 0,05 (mAP50-95). Čím nižší je práh, tím vyšší je hodnota metriky.

Střední průměrná přesnost pro boxy a masky se vypočítává podobným způsobem, jediným rozdílem je výpočet IoU. V případě boxu se použijí 4 jeho souřadnice a pro masku se použije celá maska.

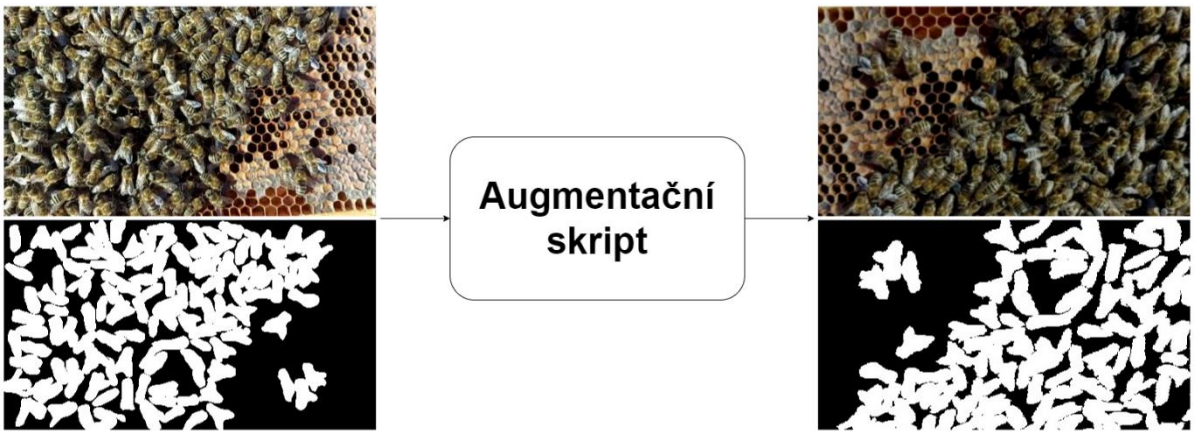
3.5 Eliminace vlivu kvality obrazu na kvalitu segmentace

Kvalita obrazu má zásadní vliv na přesnost sémantické segmentace. Ovlivňuje to, jak model rozpoznává objekty a jejich hranice. Nízký kontrast, šum nebo nesprávné osvětlení mohou způsobit chyby v segmentaci. Nepřesné hrany, zakrytí objektů, stíny a deformace v důsledku geometrického zkreslení jsou dalšími problémy spojenými s nízkou kvalitou obrazu.

Tato omezení mohou vést k chybným identifikacím objektů a snížit spolehlivost segmentačního modelu. Použití tzv. **augmentace** pomáhá modelu lépe zvládat různé podmínky a minimalizuje negativní dopady nízké kvality obrazu.

Augmentace dat vytváří syntetická data a aplikuje různé změny na existující trénovací data. Tyto změny se aplikují na obrázky a jejich anotace. Trénovací sada se rozšiřuje, model se učí rozlišovat objekty v obrazech za různých podmínek, a tím se zlepšuje jeho schopnost generalizace.

Obrázek 4 ukazuje princip augmentace obrázku a jeho anotace, na které bylo aplikováno náhodné překlopení a úprava jasu.



Obrázek 4 – Princip augmentace obrázku a anotace, [vlastní zdroj]

Existuje řada typických augmentací:

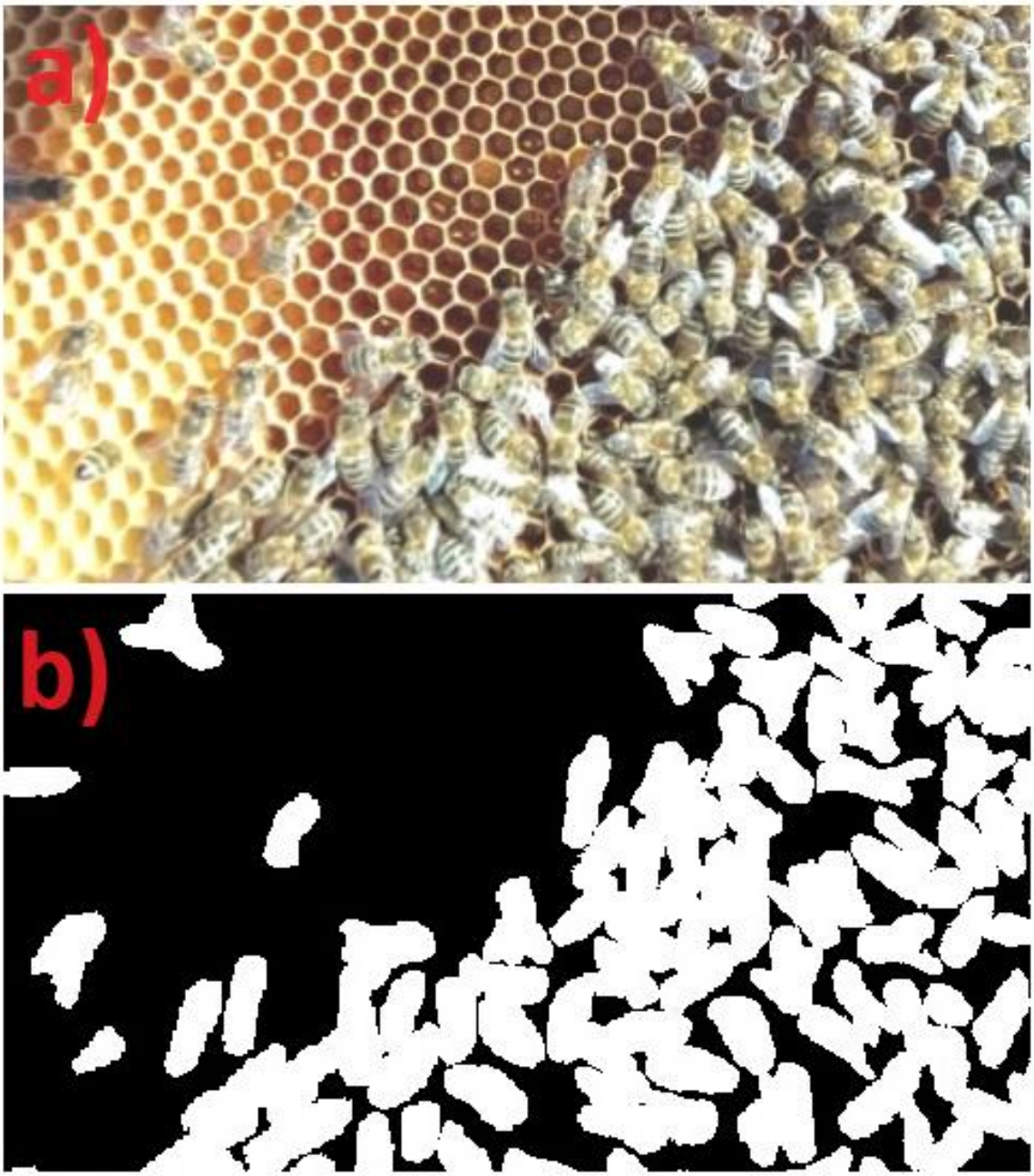
1. Náhodné horizontální/vertikální překlopení – náhodně překlopí obrázek horizontálně/vertikálně s určitou pravděpodobností. Toto pomáhá modelu se naučit příznakům bez ohledu na orientaci objektu.
2. Náhodné otočení/přiblížení/oddálení – náhodně otočí/přiblíží/oddálí obrázek. Toto zavádí variabilitu v orientaci a velikosti objektů a pomáhá modelu generalizovat různé pohledy a zvládat variace.
3. Náhodné nastavení jasu, kontrastu a saturace – náhodně upravuje jas a kontrast obrázku. Toto pomáhá modelu stát se méně citlivým na variace osvětlení.
4. Gaussovské rozostření – používá se k rozmazání hran a snížení vysokofrekvenčního šumu v datech. Výsledkem je, že ostré hrany jsou rozmazány, což vede k hladším přechodům mezi různými oblastmi dat.
5. Gaussovský šum – přidává náhodný šum k datům. Gaussovský šum je náhodný šum, který má normální rozdělení. Přidání takového šumu může změnit intenzitu jednotlivých bodů dat. Gaussovský šum pomáhá modelu se lépe adaptovat na data, která obsahují určitou míru šumu.
6. Náhodná zkosení – náhodně aplikuje zkosení obrázku, která zkreslují jeho tvar. To pomáhá modelu naučit se zpracovávat nepružné deformace tvarů objektů.
7. Náhodný výřez – náhodně vyřízne část obrázku. Tímto se model učí zaměřovat se na konkrétní oblasti a zlepšuje svou schopnost zpracovávat částečné instance objektů.

8. Barevné fluktuace – náhodně upravuje hodnoty barev pixelů. Tato variabilita v barvě pomáhá modelu naučit se segmentovat objekty za různých barevných podmínek.
9. Náhodná perspektivní transformace – náhodně aplikuje perspektivní transformaci na obrázek, simuluje změny v pohledu. To je užitečné pro modely, které zpracovávají objekty viděné z různých úhlů.
10. Cutout neboli náhodné vymazání – náhodně odstraní obdélníkové části obrázku. Tím se model povzbuzuje k zaměření se na různé části objektu a pomáhá předcházet přeučení.
11. Mozaiková augmentace – dělí obrázek na menší části, které se následně náhodně promíchají nebo přeuspořádají. Tato metoda umožňuje modelu zkoumat různé části obrázku v různých kontextech, což posiluje jeho schopnost porozumění celkové struktuře obrazu a adaptaci na různé kombinace detailů a umístění objektů v obraze.

Obrázek 5 uvádí příklad originálního obrázku 5.a) a jeho anotace 5.b).

Obrázek 5 – Originální obrázek a anotace, [vlastní zdroj]

Obrázek 6 uvádí příklad augmentovaného obrázku 6.a) a jeho augmentované anotace, obrázek 6.b), na který bylo aplikováno náhodné překlopení a úprava jasu.



Obrázek 6 – Augmentovaný obrázek a anotace, [vlastní zdroj]

4 Rešerše segmentačních neuronových sítí

Existuje řada segmentačních neuronových sítí, založených na odlišných principech:

1. YOLO (You Only Look Once) [8] je založeno na myšlence, že síť by měla provádět detekci objektů a klasifikaci v jednom průchodu. YOLO rozděluje obrázek na mřížku a pro každou buňku této mřížky predikuje bounding box a pravděpodobnosti klasifikace. Díky tomu je schopno detekovat objekty v reálném čase s vysokou přesností.
2. UNet [9] je architektura navržená pro sémantickou segmentaci. Má symetrickou strukturu s tzv. „skip connections“, které umožňují přenášení informací z nízkých úrovní (kontext) do vyšších úrovní (lokalita). To umožňuje zachytit detaily a kontext současně, což je klíčové pro sémantickou segmentaci.
3. Mask R-CNN (Region-based Convolutional Neural Network) [10] je rozšířením R-CNN pro sémantickou segmentaci. Detekuje objekty pomocí region proposal network (RPN) a následně aplikuje tzv. ROI (Region of Interest) align k extrakci příznaků pro každý detekovaný region. Kromě detekce objektů také generuje masky pixelů pro přesnou sémantickou segmentaci.
4. DeepLab [11] je založeno na konvolučních neuronových sítích a využívá tzv. dilated konvoluce pro zachování globálního kontextu při zachování prostorové rozlišovací schopnosti. Model používá tzv. atrous konvoluce pro segmentaci objektů v obrazu.
5. PSPNet (Pyramid Scene Parsing Network) [12] využívá pyramidálního přístupu k segmentaci scény. Používá tzv. pyramidální pooling, který shromažďuje informace na různých úrovních detailu, což umožňuje segmentaci objektů v různých měřítkách.

V rámci rešerše nebyla nalezena žádná z dostupných segmentačních sítí, která považuje segmentovaný objekt za texturu. Existující přístupy segmentace textur [13, 14, 15] založené na tradičních metodách nebo neuronových sítích neprovádějí požadovanou sémantickou segmentaci.

Pro sémantickou segmentaci objektu jako textury by bylo potřeba navrhnout vlastní řešení, které by kombinovalo vlastnosti obou metod. Po dohodě s vedoucím práce bylo od tohoto požadavku upuštěno, protože segmentační neuronové sítě mohou i bez zaměření na textury poskytovat dobré výsledky.

Z dostupných segmentačních neuronových sítí byly vybrané YOLO a UNet jako nejvhodnější sítě pro naši úlohu:

- YOLO [8] vyniká vysokou mírou generalizace, což umožňuje efektivní segmentaci instancí za různých okolnostních podmínek: jejich poloha/velikost nebo světelné podmínky. Tato vlastnost přispívá k přesné segmentaci včel.
- UNet [9] zachycuje jemné detaily v obrazu a umožňuje dosáhnout detailní definované segmentace instancí.

Mask R-CNN, DeepLab a PSPNet vykazují některé nedostatky:

- Mask R-CNN [10] vykazuje omezenou schopnost efektivního zacházení s překrývajícími se objekty, což může být problematické při segmentaci včel na plástech, kde časté překryvy mohou snižovat přesnost segmentace.
- DeepLab [11] projevuje citlivost na kontrast a jas, což může vést k nižší odolnosti v proměnlivých světelných podmínkách při segmentaci malých objektů, například včel na plástech. Tato citlivost může ovlivnit přesnost segmentace v dynamickém prostředí s různými světelnými podmínkami.
- PSPNet [12] s pyramidálním přístupem může vykazovat nižší odolnost v nejednotných světelných podmínkách, což může ovlivnit přesnost segmentace a způsobit nižší schopnost zachytit detaily v různých úhlech osvětlení a stínových efektech.

4.1 YOLO

YOLO framework používá konvoluční vrstvy k predikci bounding boxu a tříd pravděpodobnosti všech objektů zobrazených na obrázku. Protože YOLO je jednorázový detektor, zpracovává obraz pouze jednou. YOLO vypočítá konfidence pro každý bounding box vynásobením pravděpodobnosti, že objekt bude v podkladové mřížce obsahovat průnik ku sjednocení (**IoU**) mezi anotovaným a předpokládaným bounding boxem. Následně nemaximální potlačení (**Non-Maximum Suppression** neboli **NMS**) odstraní překrývající se bounding boxy obklopující detekovaný objekt výběrem boxu s nejvyšší konfidencí. [16]

Vstupem do YOLO je obrázek. Obrázek je transformován do formy tenzoru. Tenzor je multidimenzionální pole, které reprezentuje obraz v číselné podobě.

Tento tenzor je předáván konvolučním vrstvám, které extrahují různé příznaky a informace.

Výstupy z konvolučních vrstev jsou zpracované dalšími vrstvami, které provádějí detekci objektů a segmentaci. Segmentační vrstvy zpracovávají výstupy z konvolučních vrstev a transformují je do binárních masek. Tyto masky identifikují pixely patřící k jednotlivým instancím objektů v obraze.

Segmentační masky se tvoří pomocí speciálních konvolučních operací a aktivací, které jsou optimalizovány pro segmentační úkoly. Segmentační vrstvy používají informace o objektech získané z předchozích vrstev k přesnému určení hranic a struktury jednotlivých objektů.

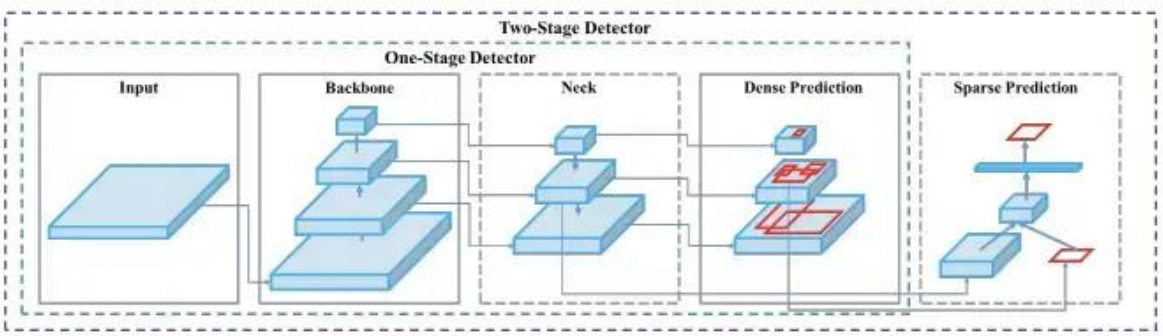
Binární masky jsou integrovány do výstupního obrazu, kde jsou vykresleny jako přesné segmentační masky, které identifikují objekty a vymezují jejich hranice.

Architektura YOLO se skládá z: páteře (backbone), krku (neck) a hlavy (head).

V páteři se nachází předem natrénovaná konvoluční neuronová síť (CNN), která analyzuje vstupní obrázek a identifikuje příznaky nízké, střední a vysoké úrovně abstrakce. Identifikované příznaky jsou propojeny v krku pomocí bloků agregace cest, například pomocí sítě Feature Pyramid Network (FPN). Propojené informace jsou předány hlavě, která se zabývá klasifikací objektů a predikcí bounding boxů.

Hlava může mít podobu jednostupňových nebo hustých predikčních modelů, jako je například Single-shot Detector (SSD) nebo predikční algoritmy [17].

Obrázek 7 ukazuje architekturu YOLO detektoru.



Obrázek 7 – YOLO architektura detektoru [17]

Segmentace v rámci YOLO architektury je integrována do páteře. Páteř YOLO architektury obsahuje segmentační síť, která extrahuje detailní informace o struktuře obrazu a objektů během jednoho detekčního procesu. Propojení mezi páteří a krkem architektury umožňuje přenos segmentačních informací do dalších fází detekce.

Segmentace v rámci YOLO architektury je integrována do páteře, která extrahuje detailní informace o struktuře obrazu a objektů během jednoho detekčního procesu. Informace pro segmentační masky jsou získávány z hlubokých vrstev konvoluční neuronové sítě, které jsou součástí páteře. Tyto vrstvy extrahují detailní příznaky a strukturální informace zobrazeného obsahu, které jsou následně využity k tvorbě segmentačních masek.

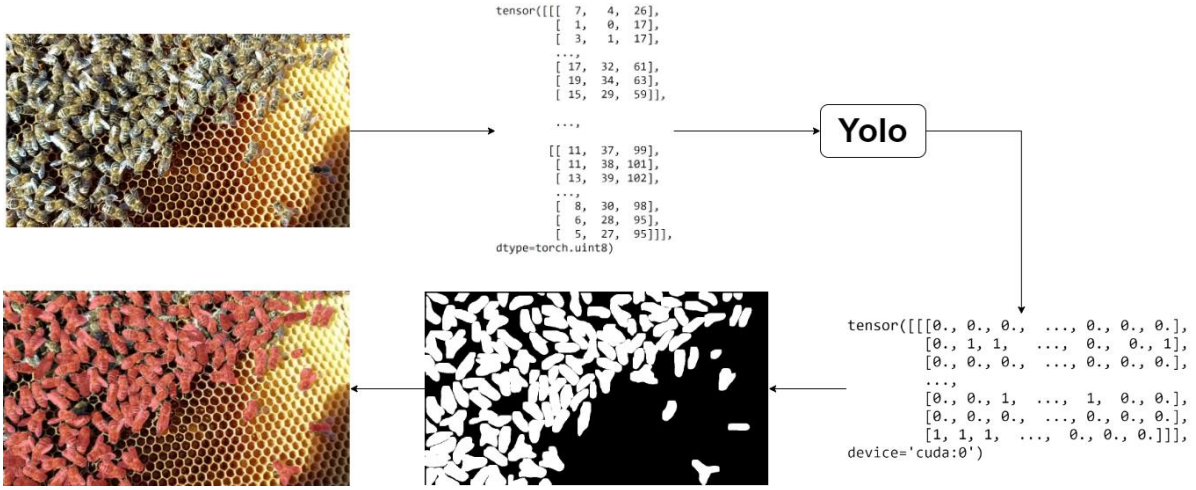
Bloky agregace cest jsou využívány pro integraci segmentačních map s detekčními informacemi.

Hlava architektury, kromě klasifikace objektů a predikce bounding boxů, obsahuje detailní segmentační masky.

YOLO modely jsou trénovány na datasetech ve stejném formátu. Poskytují vysoce přesné predikce ve třídách, ve kterých jsou naučené. Rychle se trénují a produkují výsledky vysoké přesnosti s menšími velikostmi modelů.

Tato architektura nabízí efektivní řešení pro úlohy segmentace a detekce včel na plástech. Je představována velikostmi: nano, small, medium, large a extra-large a je vybavena řadou optimalizátorů.

Obrázek 8 zobrazuje princip segmentace sítí YOLO. Vstupní obrázek se převádí na tzv. tenzor, který se zpracovává konvolučními vrstvami. Výsledkem zpracování jsou segmentační masky, které se ukládají do výstupního obrazu.



Obrázek 8 – Princip segmentace YOLO, [vlastní zdroj]

YOLO existuje ve více verzích a v následujících sekcích budou rozebrány nejpopulárnější implementace jejich základní vlastnosti.

4.1.1 YOLOv5

YOLOv5 [18] je jednou z nejpopulárnějších verzí YOLO architektury, zveřejněná v květnu 2020. YOLOv5 dědí všechny vlastnosti po svých vydaných předchůdcích, jako:

- Koncept Anchor Boxu – předdefinovaná oblast obrazu, která ilustruje idealizovanou polohu detekovaného objektu.
- Multi-Scale Training – náhodná změna velikosti obrazu v průběhu učení.
- Feature Pyramid Network (FPN) – je extraktor příznaků různého typu/formy/velikosti pro jeden obrázek. Zřetězí všechny příznaky, aby se model mohl naučit jak těm lokálním, tak i globálním.
- Bag of Freebies (BOF):
    - Techniky augmentace: Cutmix (vyříznutí a smíchání více obrázků obsahujících detekované objekty), Mixup (náhodné míchání obrázků), Cutout, Mozaiková augmentace (spojení čtyř obrázků dohromady).
    - Nové typy regresních ztrátových funkcí pro bounding boxy: MSE (Mean Squared Error), CIoU (Complete Intersection over Union), DIoU (Distance Intersection over Union).
    - Techniky regularizace: Dropout, DropPath, Spatial dropout, DropBlock.
    - Normalizace: cross-dávková a iterační-dávková normalizace, GPU normalizace.
- Bag of Specials (BOS):
    - Spatial Attention Modules (SAM) – generují mapy příznaků s využitím vztahu mezi prostorovými příznaky.
    - Non-Max Suppression (NMS) – eliminuje bounding boxy s konfidencí nižší definovaného prahu konfidence a spojí se překrývající v případě seskupených objektů.
    - Nelineární aktivační funkce: ReLU, SELU, Leaky, Swish, Mish.
    - Skip-Connections (reziduální propojení), Cross-Stage Partial Connections.

YOLOv5 přináší:

- Lepší augmentace:
    - Copy-Paste augmentace – kopíruje náhodné části z obrazu a vloží je do jiného náhodně vybraného obrazu, čímž generuje nový trénovací vzorek.
    - Random affine transformace – náhodné otáčení, změnu měřítka, posouvání a stříhání obrazu.
    - HSV augmentace – náhodné změny odstínu, sytosti a hodnoty obrazu.
- Auto učení Anchor Boxů (není potřeba přidávat manuálně).
- Nový způsob počítání ztrátové funkce – kombinace chyb pro třídy, detekci, zda je objekt přítomen v dané mřížce, a lokalizaci objektu. $Loss = \lambda_1 L_{cls} + \lambda_2 L_{obj} + \lambda_3 L_{loc}$
- Tři optimalizátory:
    - SGD (Stochastic Gradient Descent) – aktualizuje váhy sítě na základě gradientu chyby na náhodné podmnožině dat.
    - Adam – kombinuje výhody SGD a momentum metody, udržuje adaptivní rychlost učení a zachycuje historii gradientů.
    - AdamW – je varianta Adam s přidaným váhovým úpadkovým (**decay**) členem, který řeší problémy s váhovou degenerací a stabilizuje váhy v průběhu učení.

Vybranou implementaci rozšíříme o další optimalizátory z PyTorch [19] frameworku: Adamax, NAdam, RAdam a RMSProp.

4.1.2 YOLOv8

YOLOv8 [20] je nejnovější model YOLO (leden 2023). Stejně jako YOLOv5 přebírá všechny vlastnosti a schopnosti svých předků, tedy verzí 5, 6 a 7.

- Anchor-Free detekce – umožní detekcí objektů tím, že předpovídá bounding boxy na základě středových bodů objektu, čímž eliminuje potřebu předem definovaných anchor boxů. Toto zlepšuje adaptabilitu na různé tvary a velikosti objektů a zjednodušuje proces učení.
- Vylepšení procesu učení a inference – Extended Efficient Layer Aggregation Network (E-ELAN) v páteři architektury pro efektivnější výpočty.

- Oddělení hlavy od modelu – hlava již provádí klasifikaci a regresi samostatně, což zvyšuje výkon modelu.

YOLOv8 přináší:

- Nové konvoluční vrstvy v krku – umožní kanály různé velikosti na mapách příznaků, což zlepší efektivitu učení.
- Mozaiková augmentace na konci učení – na posledních 10 epochách učení se aplikuje mozaiková augmentace. Empiricky se ukázalo, že tato augmentace snižuje výkon, pokud je prováděna během celého učení.
- Čtyři optimalizátory
    - Adamax – je varianta Adam, která přizpůsobuje adaptivní rychlost učení pomocí nekonečné normy (**infinity norm**) namísto kvadratického průměru gradientů a jejich čtverců.
    - NAdam (Nesterov-accelerated Adaptive Moment Estimation) – je kombinací Nesterov Accelerated Gradient (NAG) a Adam optimalizátoru. Tento optimalizátor integruje NAG pro lepší konvergenci a využívá adaptivní rychlost učení.
    - RAdam (Rectified Adam) – je varianta Adam, která řeší nekonzistence ve výpočtu adaptivní rychlosti učení na začátku učení. Používá stabilizující prvek, který předchází divergentnímu chování.
    - RMSProp (Root Mean Square Propagation) – je optimalizátor, který reguluje adaptivní rychlost učení na základě exponenciálního klouzavého průměru čtverců gradientů. Tím zabraňuje příliš velkým aktualizacím vah v průběhu učení.

4.2 UNet

UNet se specializuje na přesné vymezení hranic jednotlivých objektů v obraze oproti YOLO, který se zaměřuje na detekci objektů a bounding boxů. Architektura je navržena tak, aby dokázala zachytit detailní informace o struktuře obrazu a objektů.

Přijímá obrázky k segmentaci na vstupu a vrací segmentační masky na výstupu ve formátu PIL.

Model pracuje s konvolučními vrstvami, ale jejich účel je odlišný. Namísto detekce a klasifikace objektů se konvoluční vrstvy zaměřují na extrakci detailních příznaků a informací, které jsou důležité pro segmentaci. Tyto informace jsou pak zpracovány dalšími vrstvami sítě, které se starají o přesné vymezení hranic jednotlivých objektů v obraze.

Segmentační masky v UNet jsou výsledkem zpracování obrazu konvolučními vrstvami, optimalizovanými pro segmentační úkoly. Masky přesně identifikují pixely patřící k jednotlivým objektům v obraze, a vytvářejí tak velmi přesné segmentace.

Masky označují každý pixel v obraze odpovídající třídou. Existuje několik běžných formátů masek:

- Binární masky – přiřazují binární hodnoty pixelům pro označení tříd, jsou snadno vytvořitelné, ale mohou být náročné na paměť. V binární segmentaci 0 představuje pozadí a 1 popředí.
- Barevné masky – používají specifické barvy k reprezentaci tříd, jsou náročné na paměť.
- Vektorové masky – ukládají anotace jako polygony nebo souřadnice, vhodné pro složité tvary a potenciálně úspornější na paměť, ale mohou vyžadovat předzpracování.
- Run-Length Encoded masky – poskytují anotace, jako délky běhu podél řádku v obraze, mohou být úsporné na paměť, ale vyžadují dekódování.

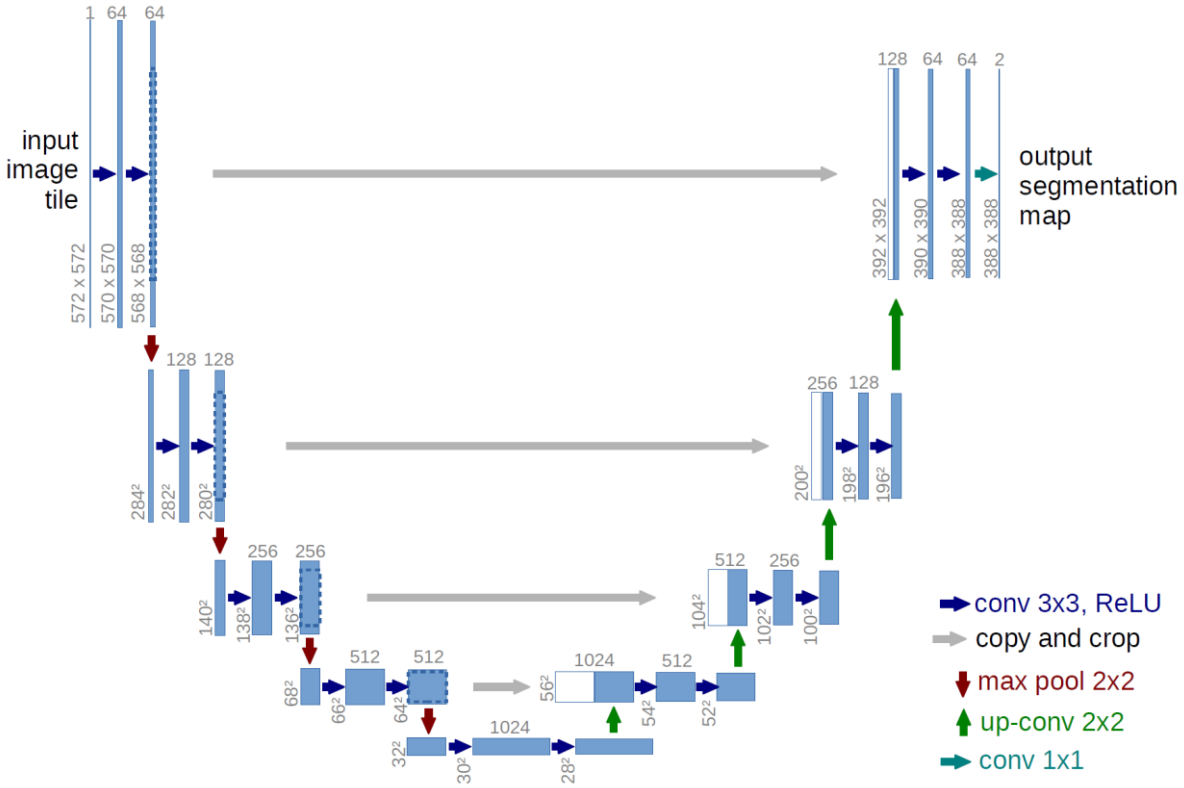
Formáty nemají žádný vliv na kvalitu segmentace, protože poskytují stejné informace o labelech tříd pro každý pixel.

Pro vizualizaci výsledků se výsledné segmentační masky aplikují na vstupní obrázek mimo síť pomocí vizualizačního skriptu.

Architektura UNet je znázorněna na obrázku 7. Skládá se z kontrahující cesty (vlevo) a expanzní cesty (vpravo). Kontrahující cesta následuje typickou architekturu konvoluční sítě. Skládá se z opakované aplikace dvou 3x3 konvolucí (neobalených konvolucí), každé následované jednotkou ReLU a operací 2x2 max pooling s krokem 2 pro převzorkování (**downsampling**). Při každém kroku downsamplingu zdvojnásobujeme počet příznakových kanálů. Každý krok v expanzní cestě se skládá z převzorkování (**upsampling**) mapy příznaků následovanou 2x2 konvolucí (**up-convolution**), která zmenšuje o polovinu počet

kanálů příznaků, konkatenací s odpovídající ořezanou mapou příznaků z kontrahující cesty a dvěma 3x3 konvolucemi, každou následovanou ReLU. Ořezání je nutné kvůli ztrátě okrajových pixelů při každé konvoluci. V poslední vrstvě se používá 1x1 konvoluce k mapování každého 64 komponentního vektoru příznaků na požadovaný počet tříd. Celkem má síť 23 konvolučních vrstev. [9]

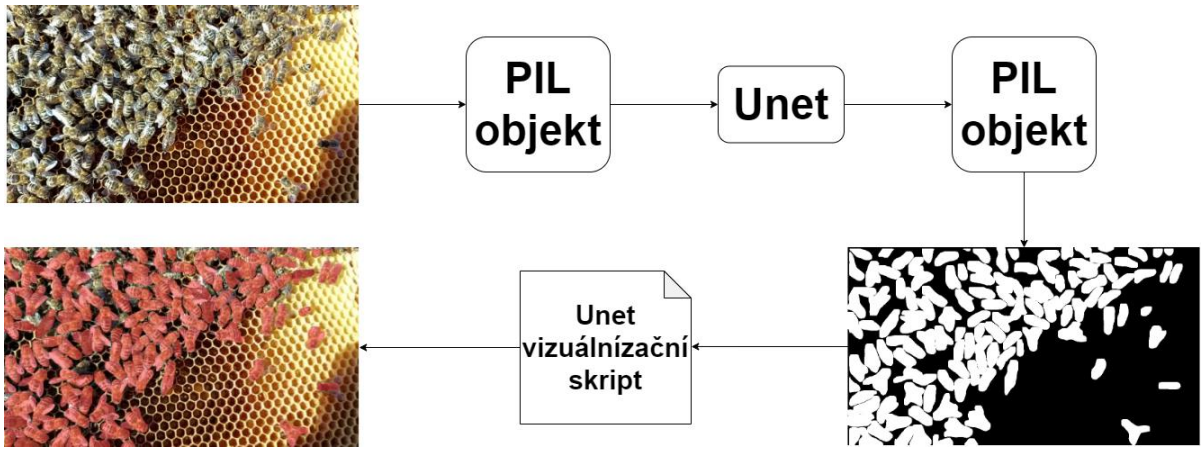
Obrázek 9 znázorňuje architekturu UNet.



Obrázek 9 – UNet architektura [9]

Tato architektura nabízí efektivní řešení pro segmentaci obrazu a je široce využívána v oblastech jako medicína nebo biologie.

Obrázek 10 znázorňuje princip sémantické segmentace síti UNet.



Obrázek 10 – UNet princip segmentace, [vlastní zdroj]

Pro naši úlohu zvolíme jednu z nejpopulárnějších implementací této architektury – Pytorch-UNet [21]. V závislosti na implementaci může mít síť různé parametry. Jeden z takovýchto parametrů je optimalizátor.

Vybraná implementace obsahuje podporu pouze pro jeden optimalizátor – RMSProp. Rozšíříme vybranou implementaci o další možné optimalizátory z PyTorch [19] frameworku: Adam, Adamax, AdamW, NAdam, RAdam a SGD.

Vybraná implementace nemá konfigurační parametr na nastavení velikostí vstupního obrazu a pouze škáluje vstupní obraz na konstantní koeficient. Velikost vstupního obrazu může mít vliv na kvalitu segmentace. Jako součást práce upravíme tuto implementaci, aby mohla přijímat velikost vstupního obrazu jako parametr, a provedeme učení na stejné velikosti s YOLO – 1024×1024.

4.3 Hodnocení kvality segmentace pro YOLO a UNet

Provedeme srovnání kvality segmentace vizuálně a na základě kvalitativních metrik modelů, viz kapitola 3.4.

YOLO architektura je určena primárně pro detekci objektů v obraze. Při detekci jsou výstupem bounding boxy detekovaných objektů a jejich konfidence, že objekty patří do příslušných tříd.

Segmentace v rámci YOLO je nadstavbou nad detekčním modelem. Segmentační část sítě využívá informaci o konfidencích z detekční části modelu.

YOLO buduje křivku **precision–recall** na základě konfidencí pro jednotlivé detekované objekty a počítá plochu pod ní, což počítá metriku AP a následně mAP.

UNet je navržen rovnou pro segmentaci obrazu. Jeho výstupem je obrázek segmentační masky, kde jsou jednotlivé segmenty či objekty odděleny a označeny. Každý pixel v tomto obrázku může být přiřazen do jedné z několika tříd.

Není možné porovnat tyto architektury z hlediska průměrné a střední průměrné přesnosti. UNet nepočítá žádnou konfidenci, tudíž není možnost vypočíst metriky AP a mAP.

Provedeme porovnání podle metrik, které jsou vypočitatelné u obou modelů:

1. Precision,
2. Recall,
3. F1 skóre,
4. IoU.

4.4 Dataset, anotace, augmentace

Použitá datová sada obsahuje 65 fotografií včel na plástech. 50 z nich je určeno pro trénink a 15 pro validaci. Vytvoříme anotace ve formátu příslušném pro architekturu. Počet jednotlivých instancí včel je 5269 pro trénink a 1302 pro validaci.

Obrázek 11 ukazuje příklad obrazu z datové sady.

Obrázek 11 – Včely na plástu z datové sady, [vlastní zdroj]

Anotace se vytvářejí nástroji na labelování obrazu, jako CVAT [22] nebo VoTT [23].

Některé implementace architektur neuronových sítí obsahují interní mechanismy pro generaci augmentačních obrázků, např. YOLOv5 [18] a YOLOv8 [19].

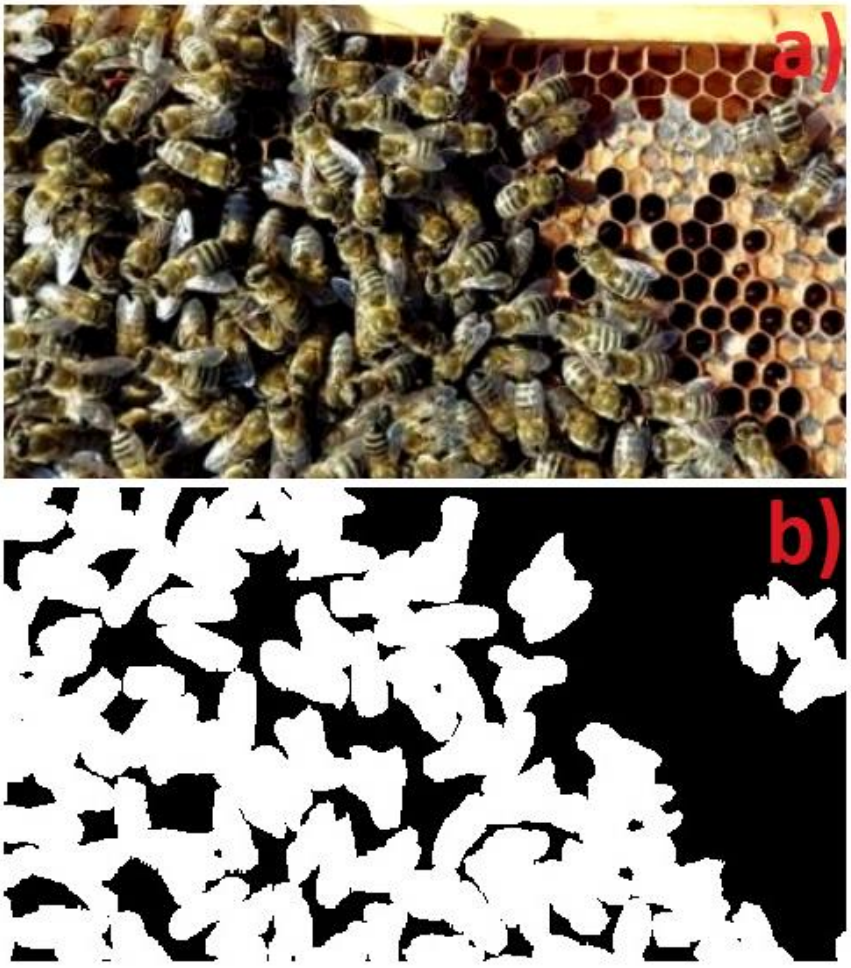
Pro implementace architektur bez takového mechanismu je třeba vytvořit augmentace pomocí knihoven, jako Imaug [24] nebo Albumentations [25].

Pro obě vybrané architektury vytvoříme následující augmentace:

1. Náhodné horizontální/vertikální překlopení,
2. náhodné otočení,
3. náhodné nastavení jasu, kontrastu a saturace.

Tyto augmentace simulují různé úhly pohledu na plást a mají podobu nekvalitní fotografie, když snímek je zasvícený/zatmavený.

Obrázek 12 ukazuje příklad augmentovaného obrázku 12.a) a jeho segmentační masky 12.b).



Obrázek 12 – Aplikované augmentace – příklad, [vlastní zdroj]

4.4.1 YOLO datový formát

YOLO je textový formát anotace obrazů, kde každý textový soubor odpovídá obrázku v datasetu a každý řádek v textovém souboru představuje anotovanou instanci objektu ve formátu:

```
<class-index> <x1> <y1> <x2> <y2> ... <xn> <yn>
```

Zdrojový kód 1 – YOLO segmentační anotace – formát

V tomto formátu `<class-index>` je indexem třídy objektu a `<x1> <y1> <x2> <y2> ... <xn> <yn>` jsou ohraničující souřadnice segmentační masky objektu. Souřadnice jsou odděleny mezerami.

```
0 0.747321 0.436494 0.555293 0.035147 ... 0.839905 0.063866
```

Zdrojový kód 2 – YOLO segmentační anotace – příklad

YOLO používá normalizované souřadnice, díky tomu je tento formát nezávislý na velikosti obrazu.

Obrázek 13 vizualizuje YOLO anotaci.

Obrázek 13 – YOLO anotace – vizualizace, [vlastní zdroj]

Obrázky a textové anotační soubory se spolu uloží do požadované složkové struktury.

YOLO očekává **kořenovou složku** datové sady se dvěma dalšími složkami **train** a **val**, kde každá obsahuje složky **images** a **labels** s obrázky a anotacemi.

```
data
|
|__train
|   |
|   |__images
|   |   |
|   |   |__1.png
|   |   |__2.png
|   |      ...
|   |__labels
|      |
|      |__1.txt
|      |__2.txt
|         ...
|__val
   |
   |__images
   |   |
   |   |__126.png
   |   |__127.png
   |      ...
   |__labels
      |
      |__126.txt
      |__127.txt
         ...
```

Zdrojový kód 3 – YOLO struktura datové sady

Velikost obrázků je definována pomocí parametru **imgsz** při spouštění učení.

K datové sadě je potřeba přidat konfigurační soubor s příponou **.yaml**. Cesta k tomuto souboru je parametrem při učení sítě. Samostatný soubor definuje cesty k trénovacím a validačním složkám datové sady, slovník tříd pro indexy z anotace a hyperparametry pro učení, např. rychlost učení nebo augmentace.

Na základě uvedených hyperparametrů je YOLO schopné generovat augmentační obrazy k datové sadě.

Hyperparametry jsou stejné pro obě verze a vkládají se do konfiguračního souboru do libovolného místa.

```
lr0: 0.01  # (float) initial learning rate
lrf: 0.01  # (float) final learning rate (lr0 * lrf)
momentum: 0.937  # (float) SGD momentum/Adam beta1
weight_decay: 0.0005  # (float) optimizer weight decay 5e-4
warmup_epochs: 3.0  # (float) warmup epochs (fractions ok)
warmup_momentum: 0.8  # (float) warmup initial momentum
warmup_bias_lr: 0.1  # (float) warmup initial bias lr
box: 7.5  # (float) box loss gain
cls: 0.5  # (float) cls loss gain (scale with pixels)
dfl: 1.5  # (float) dfl loss gain
```

```
pose: 12.0  # (float) pose loss gain
kobj: 1.0  # (float) keypoint obj loss gain
label_smoothing: 0.0  # (float) label smoothing (fraction)
nbs: 64  # (int) nominal batch size
hsv_h: 0.015  # (float) image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # (float) image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # (float) image HSV-Value augmentation (fraction)
degrees: 0.0  # (float) image rotation (+/- deg)
translate: 0.1  # (float) image translation (+/- fraction)
scale: 0.5  # (float) image scale (+/- gain)
shear: 0.0  # (float) image shear (+/- deg)
perspective: 0.0  # (float) image perspective (+/- fraction), range
0-0.001
flipud: 0.0  # (float) image flip up-down (probability)
fliplr: 0.5  # (float) image flip left-right (probability)
mosaic: 1.0  # (float) image mosaic (probability)
mixup: 0.0  # (float) image mixup (probability)
copy_paste: 0.0  # (float) segment copy-paste (probability)
auto_augment: randaugment  # (str) auto augmentation policy for
clas-sification (randaugment, autoaugment, augmix)
erasing: 0.4  # (float) probability of random erasing during
classi-fication training (0-1)
crop_fraction: 1.0  # (float) image crop fraction for
classification evaluation/inference (0-1)
```

Zdrojový kód 4 – YOLO hyperparametry konfigurace

Konfigurační soubory obou verzí architektur jsou velmi podobné, s rozdílem v definici slovníku tříd.

U **YOLOv5** se uvádí celkový počet tříd v datasetu a výčet jejich názvu. Sled výčtu musí korespondovat s jeho indexem v anotaci.

```
train: data/train
val: data/val
nc: 1 # number of classes
names: ['bee'] # class names
```

Zdrojový kód 5 – YOLOv5 – konfigurace datové sady

U **YOLOv8** se uvádí slovník, kde index třídy koresponduje s jejím názvem napřímo.

```
path: data
train: train
val: val
names:
  0: bee
```

Zdrojový kód 6 – YOLOv8 – konfigurace datové sady

Textové anotační formáty mají obtíže při reprezentaci složitých anotačních struktur. Z těchto důvodů se široce používá alternativní formát COCO.

COCO (Common Objects in Context) je **JSON-based** anotační formát, který poskytuje informace o instancích objektů v obrázku, jejich bounding boxech, kategoriích a segmentačních maskách. Anotace zahrnuje:

- **Images** – informace o obrazu:
  - ID obrazu,
  - šířka obrazu,
  - výška obrazu,
  - jméno souboru.
- **Annotations** – informace o instancích tříd objektů:
  - ID instance (**id**),
  - ID kategorie (**category_id**),
  - souřadnice bounding boxu (**bbox**),
  - segmentační maska (**segmentation**).
- **Categories** – definice kategorií:
  - ID kategorie (**id**),
  - jméno kategorie.


```
{
  "images": [
    {"id": 1, "width": 608, "height": 608, "file_name": "19.jpg"},
    ...,
    {"id": 15, "width": 416, "height": 416, "file_name":
"zelizone.jpg"}
  ],
  "annotations": [
    {
      "id": 1, "image_id": 1, "category_id": 1,
      "segmentation": [[46.54, 275.76, ..., 277.25]],
      "area": 252.0,
      "bbox": [32.7, 275.76, 16.14, 23.33],
      "iscrowd": 0,
      "attributes": {"occluded": false}
    },
    ...,
    {
      "id": 1302, "image_id": 15, "category_id": 1,
      "segmentation": [[386.4, 130.05, ..., 130.05]],
      "area": 1197.0,
```

```
      "bbox": [371.99, 51.15, 44.01, 78.9],
      "iscrowd": 0,
      "attributes": {"occluded": false}
    }
  ],
  "categories": [{"id": 1, "name": "bee", "supercategory": ""}]
}
```

Zdrojový kód 7 – COCO anotace – příklad

Souřadnice bounding boxů mohou být zadané v normalizovaných nebo absolutních hodnotách.

Jako součást práce vytvoříme anotace pro YOLO architekturu ve formátu COCO a konvertujeme ji do nativního formátu YOLO s normalizací hodnot.

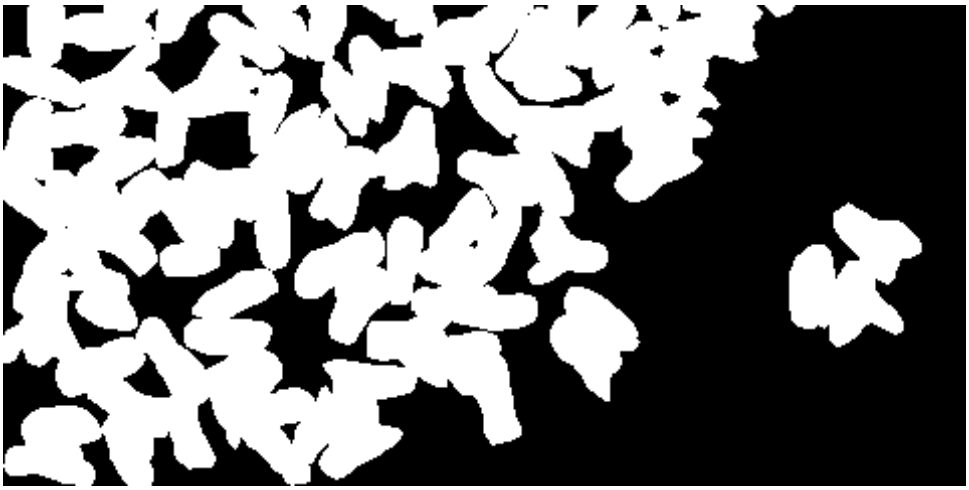
4.4.2 UNet datový formát

UNet anotace je obrázek ve formě masky, kde každý pixel v obraze je přiřazen k jednomu z několika segmentů.

UNet anotace můžeme vytvořit s použitím nástrojů na anotování obrázku nebo můžeme konvertovat jiný anotační formát na jeden z formátů masek UNet, viz 4.2.

Anotaci vytvořenou pro YOLO síť COCO konvertujeme do formátu binární masky, protože segmentujeme objekty pouze jedné třídy.

Obrázek 14 vizualizuje konvertovanou UNet anotaci v binárním formátu.



Obrázek 14 – UNet anotace – vizualizace, [vlastní zdroj]

UNet očekává ve své kořenové složce dvě další složky, **images** a **masks**, s příslušnými obrazy a anotacemi.

```
data
|
|__images
|   |
|   |__1.png
|   |__2.png
|   |   ...
|
|__masks
    |
    |__1.png
    |__2.png
       ...
```

Zdrojový kód 8 – UNet struktura datové sady

5 Implementace

5.1 Vytvoření datové sady

Vytvoříme anotace ve formátu COCO pomocí softwarového nástroje CVAT [22].

Anotujeme 50 obrazů pro učení a 15 pro validaci.

Konvertujeme COCO do YOLO formátu s normalizací pomocí skriptu B v příloze. Skript B přijímá na vstup cesty do obrázků, jejich anotace a výstupní složku. Pro každý obrázek z COCO anotace extrahujeme segmenty (masky). Dostaneme pro každý obrázek textový soubor YOLO, popsaný v kapitole 4.4.1.

```
python coco2yolo.py --input_images images --input_annotation annotation --
output_path output
```

Zdrojový kód 9 – COCO2YOLO skript spouštění

Konvertujeme COCO do formátu binární masky pro UNet pomocí skriptu C v příloze. Skript C přijímá na vstup cestu do anotačního souboru a výstupní složku. Pro každou anotaci z obrázku převádíme masku na polygon a vytváříme binární masku z polygonů.

```
python coco2unet.py --input_annotation annotation --output_path output
```

Zdrojový kód 10 – COCO2UNet skript spouštění

5.2 Augmentace

Pro **YOLO** nastavíme příslušné hyperparametry pro automatickou generaci augmentací.

```
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 180.0 # image rotation (+/- deg)
flipud: 0.8  # image flip up-down (probability)
fliplr: 0.7  # image flip left-right (probability)
```

Zdrojový kód 11 – YOLO augmentace pomocí hyperparametrů

Pro **UNet** vygenerujeme anotace pomocí skriptu D v příloze s použitím knihovny Albumentations [25]. Skript přijímá na vstup sadu augmentací s pravděpodobností aplikace augmentace a její parametry definované v metodě `define_transforms`, cesty pro ukládání augmetovaných obrázků a anotací a počet generovaných augmentací na jeden obrázek.

```
python unet_aug.py --images_dir images --masks_dir masks --
image_transforms define_transforms --aug_images_output images_output --
aug_masks_output masks_output --aug_per_image 2
```

Zdrojový kód 12 – UNetAug skript spouštění

```
import albumentations as A

def define_transforms():
    ...

    image_transform = A.Compose([
        horizontal_flip,
        vertical_flip,
        brightness_contrast
    ])

    return image_transform
```

Zdrojový kód 13 – UNet augmentace pomocí knihovny „Albumentations“

5.3 YOLOv5 instalace a spuštění

Stáhneme repozitář YOLOv5: `git clone https://github.com/ultralytics/yolov5`

Rozšíříme implementaci o další optimalizátory Adamax, NAdam, RAdam a RMSProp. Modifikujeme metodu `smart_optimizer` v `utils/torch_utils.py` a zařádíme nové optimalizátory do výběrového algoritmu.

```
if name == "Adam":
        optimizer = torch.optim.Adam(g[2], lr=lr, betas=(momentum,
0.999))
    elif name == "AdamW":
        optimizer = torch.optim.AdamW(g[2], lr=lr, betas=(momentum,
0.999), weight_decay=0.0)
    elif name == "Adamax":
        optimizer = torch.optim.Adamax(g[2], lr=lr,
betas=(momentum, 0.999), weight_decay=0.0)
    elif name == "NAdam":
        optimizer = torch.optim.NAdam(g[2], lr=lr, betas=(momentum,
0.999), weight_decay=0.0)
    elif name == "RAdam":
        optimizer = torch.optim.RAdam(g[2], lr=lr, betas=(momentum,
0.999), weight_decay=0.0)
    elif name == "RMSProp":
        optimizer = torch.optim.RMSprop(g[2], lr=lr,
momentum=momentum)
    elif name == "SGD":
        optimizer = torch.optim.SGD(g[2], lr=lr, momentum=momentum,
nesterov=True)
    else:
        raise NotImplementedError(f"Optimizer {name} not
```

```
implemented.")
```

Zdrojový kód 14 – YOLOv5 rozšířený výběr optimalizátoru

Spustíme učení s parametry, kde **data** je cesta ke konfiguračnímu souboru z kapitoly 4.3.1.

Parametr `weights` určuje typ a velikost modelu, který budeme učit. Písmeno za **yolov*** určuje velikost: **n**ano, **s**mall, **m**edium, **l**arge a e**x**tra-large. Sufix určuje typ sítě: segmentační, detekční atd.

Pokud chceme využít přednaučený model a přeučit ho na své třídy (doporučeno autorem YOLO), použijeme váhy s příponou **.pt**, pokud chceme naučit model z nuly, použijeme váhy s příponou **.yaml**.

```
python segment/train.py --img 1024 --batch-size 8 --epochs 50 --
data data.yaml --weights yolov5l-seg.pt
```

Zdrojový kód 15 – YOLOv5 spouštění učení

5.4 YOLOv8 instalace a spuštění

Provedeme instalaci: `pip install ultralytics`

Inicializujeme model a spustíme učení s parametry.

```
model = YOLO('yolov8l-seg.pt')
model.train(
    data=data,
    epochs=50,
    batch=8,
    imgsz=1024
)
```

Zdrojový kód 16 – YOLOv8 spouštění učení

5.5 UNet instalace a spuštění

Stáhneme repozitář UNet: `git clone https://github.com/milesial/Pytorch-UNet`

Rozšíříme metodu `train_model` o další optimalizátory: Adam, Adamax, AdamW, NAdam, RAdam a SGD podobným způsobem jako ve zdrojovém kódu 14.

Přidáme možnost přeškálování velikostí obrázků na zadané rozlišení.

```
    @staticmethod
    def preprocess(mask_values, pil_img, scale, is_mask,
use_default_image_size, image_size):
        w, h = pil_img.size
        newW, newH = image_size, image_size
```

```
        assert newW > 0 and newH > 0, 'Scale is too small, resized
images would have no pixel'
        pil_img = pil_img.resize((newW, newH),
resample=Image.NEAREST if is_mask else Image.BICUBIC)
        img = np.asarray(pil_img)
```

Zdrojový kód 17 – UNet přeškálování obrázku na zadané rozlišení

Spustíme učení spolu s novými parametry: `optimizer` a `image-size`. Parameter `amp` zapíná automatickou smíšenou přesnost, která umožňuje modelu používat méně paměti a být rychlejší na GPU.

```
python train.py --optimizer adam --learning-rate 0.01 --epochs 50 -classes
2 --image-size 1024 --amp
```

Zdrojový kód 18 – UNet spouštění učení

Pro porovnání metrik YOLO a UNet vypočteme společné metriky pomocí skriptu E v příloze. Skript E přijímá na vstup cestu do anotačních masek, naučený model a zařízení, na kterém probíhá predikce (CPU nebo GPU). Naučený model predikuje masky a pro každý příslušný pár predikované a anotační masky se počítají metriky. Metriky pro každý pár se průměrují a ukládají do výsledné tabulky.

6 Experimenty

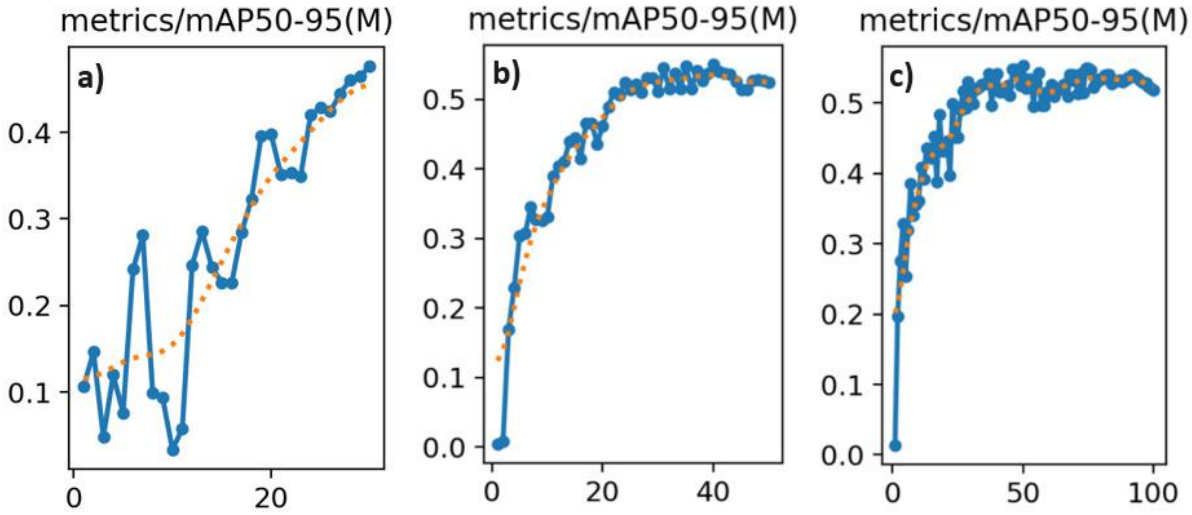
Experimenty rozdělíme na několik skupin:

1. Originální obrázky (YOLO, UNet).
2. Originální obrázky s interními mechanismy augmentace (YOLO).
3. Originální a námi vytvořené augmentované obrázky (UNet).

Provedeme experimenty na datové sadě vytvořené v kapitole 5.1 pro tři rychlosti učení (LR, learning rate): 0.01 (defaultní), 0.1, 0.001 a sedm optimalizátorů: Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, SGD.

Všechny experimenty provedeme na 50 epoch. Tento počet epoch byl uznán za optimální na základě sérií experimentů pro volbu délky učení.

Obrázek 15 ukazuje výsledky těchto experimentů – průběhy učení, 15.a) je průběh pro 30 epoch, 15.b) je průběh pro 50 epoch a 15.c) je průběh pro 100 epoch.



Obrázek 15 – Volba délky učení – experiment, [vlastní zdroj]

Z experimentů plyne, že 30 epoch není dostačující počet epoch pro síť, aby prokázala své nejlepší hodnoty. Doba ve 100 epochách je příliš dlouhá, a nevede k navýšení metrik. Optimální je délka učení v 50 epochách, protože metriky nabývají svých maximálních hodnot.

Prozkoumáme metriky neuronových sítí a vyznačíme nejlepší model, příp. modely, zelenou barvou, na kterých provedeme učení s augmentačními obrazy. Provedeme porovnání nejlepších modelů obou architektur a vizuální segmentaci.

6.1 YOLO

Experimenty s YOLO provedeme na všech velikostech modelu. Tabulka 1 popisuje velikosti YOLO modelu a počet jejich parametrů (v milionech) pro každou verzi architektury.

Velikost	Parametry(M)v5	Parametry(M)v8
n	1.9	3.2
s	7.2	11.2
m	21.2	25.9
l	46.5	43.7
x	86.7	68.2

Tabulka 1 – Porovnání počtů parametrů pro různé velikosti YOLO v5 a v8

6.1.1 YOLOv5

Originální obrázky. Learning rate 0.01.

Tabulky z přílohy A, A.1–A.7 zobrazují výsledné metriky pro učení modelů všech velikostí, se všemi optimalizátory, s LR 0.01.

Pouze 2 optimalizátory prokázaly metriky výrazně odlišné od nuly – RAdam a SGD. Tabulka A.5 ukazuje výsledné metriky pro RAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.4353	0.4316	0.3761	0.0892
s	0.6066	0.4885	0.4940	0.1333
m	0.5951	0.5461	0.5353	0.1629
l	0.6380	0.5184	0.5420	0.1600
x	0.5305	0.4370	0.4609	0.1300

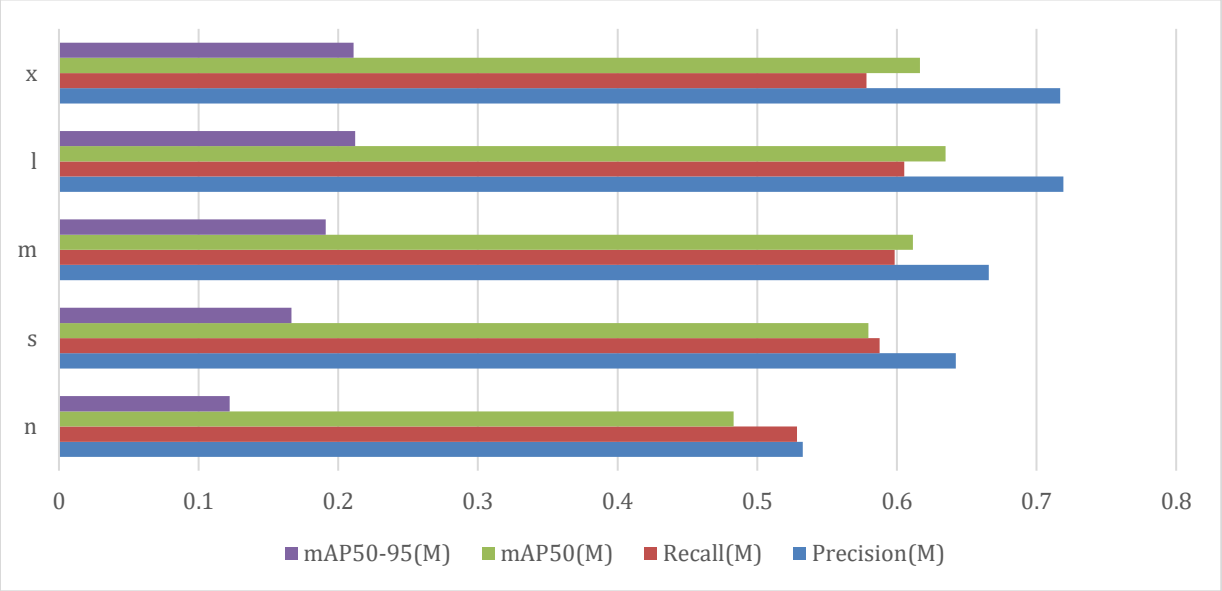
Tabulka A.5 – YOLOv5, RAdam optimalizátor, 50 epoch, LR 0.01

RAdam dosáhl největších hodnot mAP50 a mAP50-95 – 54 % a 16 % při velikosti **large**. Tabulka A.7 ukazuje výsledné metriky pro SGD optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.5325	0.5284	0.4830	0.1222
s	0.6423	0.5876	0.5796	0.1665
m	0.6659	0.5983	0.6115	0.1911
l	0.7192	0.6052	0.6348	0.2120
x	0.7169	0.5783	0.6166	0.2110

Tabulka A.7 – YOLOv5, SGD optimalizátor, 50 epoch, LR 0.01

Graf 1 vizualizuje hodnoty metrik z tabulku A.**7** pro všechny velikosti.



Graf 1 –Vizualizace tabulky A.7

SGD se jevil nejlepším optimalizátorem pro tento LR a dosáhl největších hodnot mAP50 a mAP50-95 – 63 % a 21 % při velikosti **large**.

Modely s ostatními optimalizátory, tabulky A.1–A.4, A.6, mají metriky blízké k nule, což je očekávané vzhledem k velikosti datové sady.

Na základě tohoto experimentu můžeme předpokládat, že správný optimalizátor může mít pozitivní vliv na kvalitu učení a umožní síti se naučit i s malým objemem dat. Tento předpoklad ověříme během dalších experimentů.

Originální obrázky. Learning rate 0.1.

Tabulky A.8–A.14 zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.1.

Výsledky všech optimalizátorů jsou téměř nulové. Toto platí i pro RAdam a SGD optimalizátory, které se prokázaly nejlépe v předchozích experimentech.

Tabulka A.12 ukazuje výsledné metriky pro RAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.1199	0.1713	0.0671	0.0130
s	0.2646	0.2450	0.1788	0.0314
m	0.0413	0.1429	0.0248	0.0045
l	0.0209	0.0722	0.0113	0.0019
x	0.0300	0.1037	0.0170	0.0026

Tabulka A.12 –YOLOv5, RAdam optimalizátor, 50 epoch, LR 0.1

RAdam ve velikosti small je jediným případem ze všech tabulek, kde např. hodnota mAP50 pro masky je zhruba 18 %, ale mAP50-95 již pouze 3 %. LR 0.1 se jeví příliš rychlým.

Originální obrázky. LR 0.001.

Tabulky A.15–A.21 zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.001.

Čtyři optimalizátory: Adam (A.15), Adamax (A.16), AdamW (A.17), NAdam (A.18) a RAdam (A.19) prokázaly hodnoty mAP50-95 v rozmezí 12–21 % ve velikostech nad **nano**: **small**, **medium**, **large** a **extra-large**.

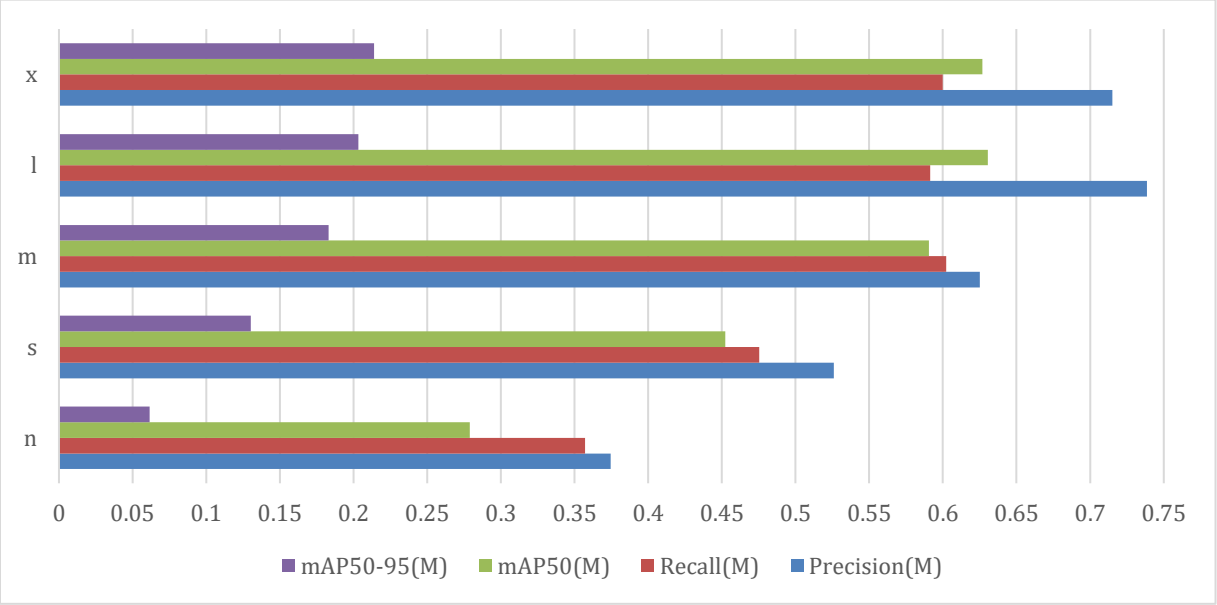
Nejlepší hodnoty dosáhl **extra-large** model s Adamax – 21 %.

Tabulka A.16 ukazuje výsledné metriky pro Adamax optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.3745	0.3571	0.2789	0.0616
s	0.5259	0.4754	0.4524	0.1303
m	0.6252	0.6022	0.5905	0.1831
l	0.7386	0.5914	0.6306	0.2033
x	0.7152	0.5999	0.6269	0.2140

Tabulka A.16 –YOLOv5, Adamax optimalizátor, 50 epoch, LR 0.001

Graf 2 vizualizuje hodnoty metrik z tabulku A.16 pro všechny velikosti.



Graf 2 –Vizualizace tabulky A.16

Adam, AdamW a NAdam, RAdam nepřesáhly 20 %. Tabulka A.18 ukazuje výsledné metriky pro NAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.4419	0.4316	0.3825	0.0897
s	0.5312	0.4608	0.4500	0.1220
m	0.5504	0.4578	0.4764	0.1241
l	0.4018	0.3264	0.2970	0.0654
x	0.2639	0.1912	0.1655	0.0340

Tabulka A.18–YOLOv5, NAdam optimalizátor, 50 epoch, LR 0.001

NAdam (A.18), dosáhl mAP50-95 12 % pouze na **small** a **medium** velikosti, což je v porovnání s předchozími optimalizátory relativně málo. Nicméně, pomalejší LR se jevil vhodnější pro NAdam optimalizátor.

Tabulka A.19 ukazuje výsledné metriky pro RAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.0669	0.1498	0.0407	0.0067
s	0.2889	0.3395	0.2228	0.0488
m	0.4466	0.4793	0.4016	0.1098
l	0.4885	0.4877	0.4548	0.1210
x	0.5536	0.5737	0.5026	0.1530

Tabulka A.19–YOLOv5, RAdam optimalizátor, 50 epoch, LR 0.001

RAdam, který se jevil jako jeden z nejlepších modelů pro LR 0.01, prokázal mAP50-95 maximálně 15 % a nedosáhl podobné kvality s pomalejším LR.

RMSProp (A.20), stejně jako 2 předchozí experimenty, nabyl nulové hodnoty.

SGD (A.21) nepřesáhl 8 % pro mAP50-95.

Nižší LR umožňuje ukázat se jiným optimalizátorům, které prokázaly nulové metriky s defaultním LR.

Modely s RAdam a SGD optimalizátory s LR 0.001 dosáhly horších hodnot než s LR 0.01.

Provedené experimenty naznačují, že existují optimalizátory vhodnější k použití s rychlejším (0.01) a pomalejším (0.001) LR.

Z experimentů s RMSProp se můžeme domnívat, že nevhodný optimalizátor může pokazit celé učení bez ohledu na jakékoliv parametry.

Originální a augmentované obrázky.

Pro nejlepší optimalizátory z příslušných LR: RAdam a SGD s LR 0.01 a Adam, Adamax, AdamW a RAdam s LR 0.001 provedeme učení s augmentacemi a prozkoumáme jejich vliv na kvalitu učení.

Tabulky A.64–A.69 zobrazují výsledné metriky pro vybrané optimalizátory a LR.

RAdam s LR 0.01 dosáhl mírného zvětšení metriky mAP50-95 a zároveň mírného zmenšení mAP50. Augmentace neměly kvalitativní vliv na segmentační schopnost modelu.

Tabulka A.65 ukazuje výsledné metriky pro SGD optimalizátor s LR 0.01.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.45278	0.46928	0.38014	0.086002
s	0.43257	0.51843	0.4209	0.1011
m	0.54753	0.55223	0.54058	0.14914
l	0.53323	0.50922	0.49359	0.13028
x	0.55414	0.51382	0.49091	0.12137

Tabulka A.65 –YOLOv5, SGD optimalizátor, 50 epoch, LR 0.01, interní augmentace

SGD s LR 0.01 dosáhl vylepšení mAP50 o 7–13 procentních bodů a mAP50-95 o 1–3 procentní body.

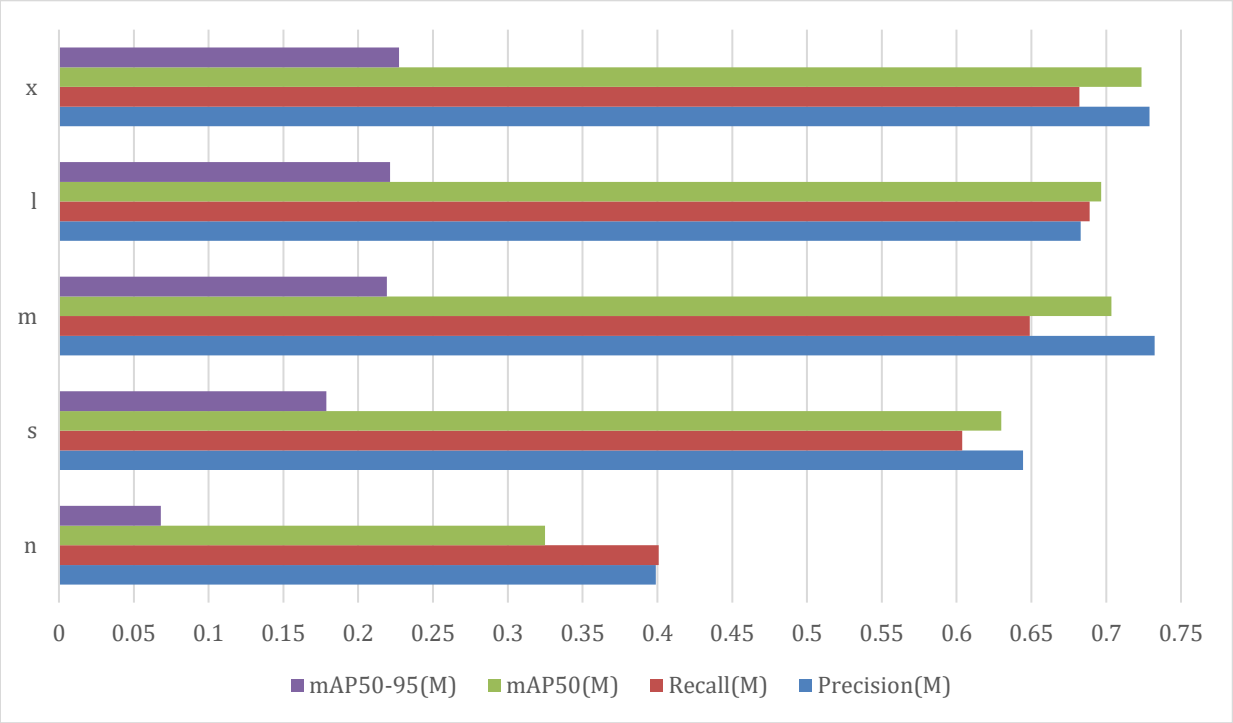
Adamax, AdamW a RAdam s LR 0.001, všechny dosáhly nárůstu mAP50 o 6–15 procentních bodů. Metrika mAP50-95 zůstala téměř stejná, u některých velikostí je vylepšení o 1–3 procentní body.

Tabulka A.67 ukazuje výsledné metriky pro Adamax optimalizátor s LR 0.01, který dosáhl vylepšení mAP50 o 15 procentních bodů ve velikosti **small** a o 10 ve velikosti **extra-large**.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.39894	0.40092	0.32494	0.068003
s	0.64451	0.60369	0.62993	0.17876
m	0.73228	0.649	0.70344	0.21927
l	0.68299	0.68894	0.69654	0.22132
x	0.72899	0.68203	0.72366	0.22721

Tabulka A.67 –YOLOv5, Adamax optimalizátor, 50 epoch, LR 0.001, interní augmentace

Graf 3 vizualizuje hodnoty metrik z tabulku A.67 pro všechny velikosti.



Graf 3 –Vizualizace tabulky A.67

Adam s LR 0.001 měl mírný pokles metrik přes všechny velikosti.

Interní augmentace, generované za běhu učení YOLOv5, měly pozitivní vliv na počet detekovaných včel, což lze pozorovat z navýšení mAP50 u některých kombinací, jako Adamax a RAdam s LR 0.001. Ostatní kombinace buď měly nevýznamný přírůstek, nebo pokles.

Následně provedeme experimenty pro YOLOv8 a vybereme model s nejlepšími metrikami.

6.1.2 YOLOv8

Originální obrázky. LR 0.01.

Tabulky z přílohy A, A.22–A.28, zobrazují výsledné metriky pro učení modelů všech velikostí, se všemi optimalizátory s LR 0.01.

Stejně jako u předchozí verze architektury prospěly pouze 2 optimalizátory – RAdam a SGD, kde oba dosahují hodnot mAP50–95 kolem 40 %. Zbývající optimalizátory mají hodnoty kolem 0.

Oproti předchozí verzi architektury, pro RAdam a SGD, je to kvalitativní vylepšení mAP50–95 o 20–30 procentních bodů, viz tabulky A5, A7.

Tabulka A.26 ukazuje výsledné metriky pro RAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.6729	0.6989	0.7009	0.3049
s	0.7487	0.7051	0.7506	0.3487
m	0.7833	0.7717	0.7991	0.3990
l	0.7867	0.7834	0.8295	0.4230
x	0.7685	0.7857	0.8257	0.4150

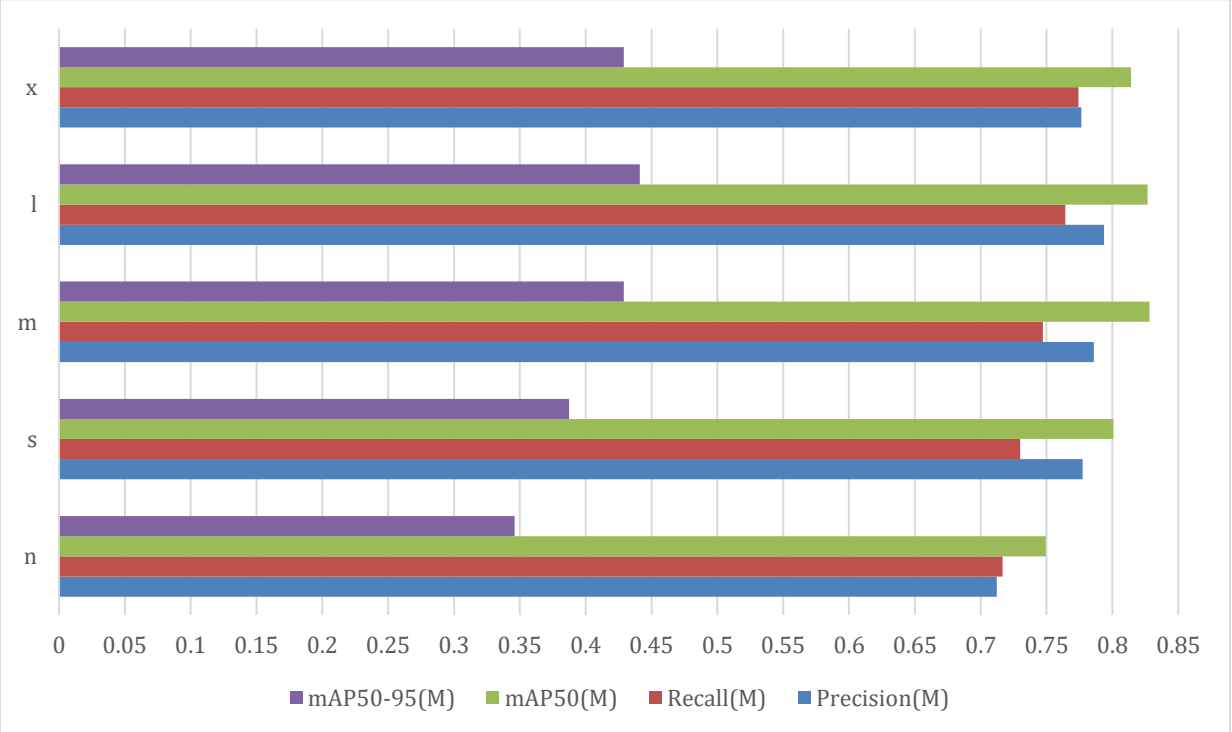
Tabulka A.26 –YOLOv8, RAdam optimalizátor, 50 epoch, LR 0.01

Tabulka A.28 ukazuje výsledné metriky pro SGD optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.7123	0.7166	0.7495	0.3460
s	0.7775	0.7301	0.8008	0.3874
m	0.7860	0.7473	0.8281	0.4290
l	0.7938	0.7642	0.8268	0.4410
x	0.7764	0.7742	0.8140	0.4290

Tabulka A.28 –YOLOv8, SGD optimalizátor, 50 epoch, LR 0.01

Graf 4 vizualizuje hodnoty metrik z tabulku A.28 pro všechny velikosti.



Graf 4 –Vizualizace tabulky A.28

Originální obrázky. LR 0.1.

Tabulky A.29–A.35 zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.1.

Výsledky jsou velmi podobné předchozí verzi. LR 0.1 se jeví moc rychlým pro většinu optimalizátorů.

Tabulka A.33 ukazuje výsledné metriky pro RAdam optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.6033	0.6705	0.6312	0.2487
s	0.4483	0.4639	0.4229	0.1112
m	0.3275	0.3848	0.2886	0.0675
l	0.1640	0.3095	0.1260	0.0283
x	0.2830	0.2819	0.2067	0.0460

Tabulka A.33–YOLOv8, RAdam optimalizátor, 50 epoch, LR 0.1

RAdam je jediný optimalizátor, kde jsou metriky odlišné od 0. RAdam ve velikosti nano dosáhl mAP50 63 % a mAP50-95 25 %. Zvětšení modelu vede ke zhoršení metrik, např. model velikostí small dosáhl mAP50 42 %, ale mAP50-95 již pouze 11 %. Nano model je jediným použitelným modelem s tímto LR a optimalizátorem.

Originální obrázky. LR 0.001.

Tabulky A.36–A.42 zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.001.

Výsledek je podobný YOLOv5. Pomalejší LR umožnil prokázat svou činnost ostatním optimalizátorům (Adam, Adamax, AdamW a NAdam).

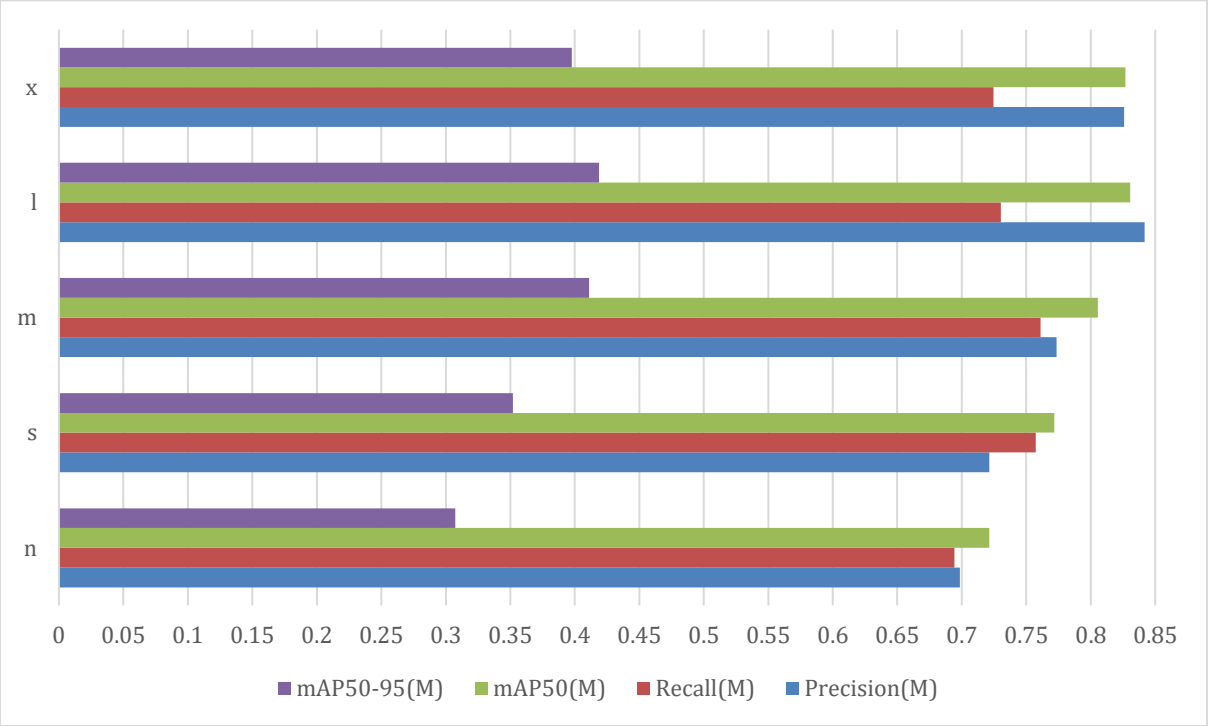
Nejlépe se prokázal opět Adamax ve velikosti **large** s mAP50-95 – 42%

Tabulka A.37 ukazuje výsledné metriky pro Adamax optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.6985	0.6943	0.7214	0.3072
s	0.7213	0.7573	0.7719	0.3521
m	0.7735	0.7611	0.8055	0.4112
l	0.8417	0.7304	0.8306	0.4187
x	0.8258	0.7246	0.8270	0.3978

Tabulka A.37 –YOLOv8, Adamax optimalizátor, 50 epoch, LR 0.001

Graf 5 vizualizuje hodnoty metrik z tabulku A.37 pro všechny velikosti.



Graf 5 –Vizualizace tabulky A.37

Výsledky pro Adam, AdamW a NAdam kolísají v rozmezí 33 až 38 % pro mAP50-95.

Pomalejší LR zhoršil výkon RAdam a SGD, které se lépe prokázaly s LR 0.01.

Například SGD optimalizátor ztratil 5–6 procentních bodů mAP50-95 oproti defaultnímu LR.

Tabulka A.42 ukazuje výsledné metriky pro SGD optimalizátor.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.4659	0.4823	0.4375	0.1839
s	0.7200	0.6575	0.7078	0.3159
m	0.6993	0.6790	0.7293	0.3537
l	0.6841	0.7135	0.7415	0.3580
x	0.7574	0.7135	0.7829	0.3880

Tabulka A.42–YOLOv8, SGD optimalizátor, 50 epoch, LR 0.001

RMSProp (A.41), podobně jako v minulých experimentech, má nulové hodnoty. Tato skutečnost může potvrdit naši domněnku, že „nesprávný“ optimalizátor může zkazit výsledek celého učení bez ohledu na jiné parametry.

Přesnost modelů s LR 0.01 je ve všech případech větší než přesnost modelů s LR 0.001.

Originální a augmentované obrázky. LR 0.01.

Tabulky A.70–A.77 zobrazují výsledné metriky učení s augmentacemi, provedené pro RAdam a SGD optimalizátory s LR 0.01 a Adam, Adamax, AdamW, NAdam, RAdam a SGD s LR 0.001.

Augmentace zvýšily hodnoty metrik pro všechny kombinace velikostí a optimalizátorů kromě SGD s LR 0.001, kde došlo k mírnému snížení.

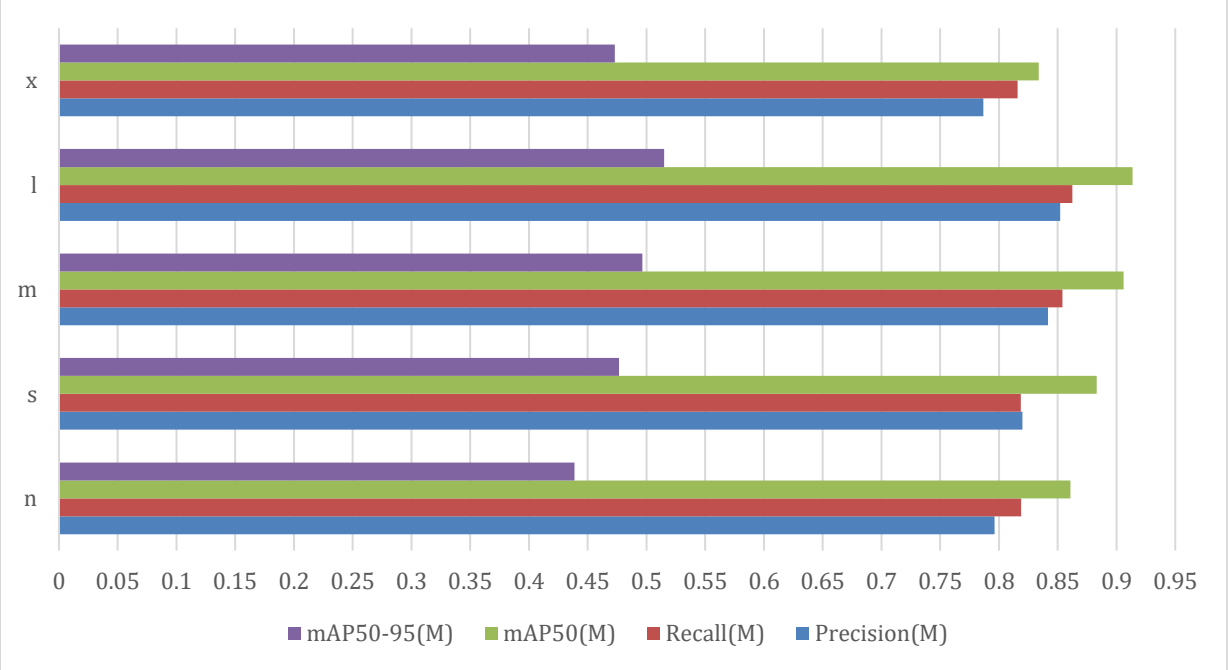
RAdam s defaultním LR a augmentacemi předčil model se stejným nastavením bez augmentace o 6–13 procentních bodů. RAdam velikosti **large** vylepšil se z 0.42 do 0.51 mAP50-95, viz A.26.

Tabulka A.70 ukazuje výsledné metriky pro RAdam optimalizátor s LR 0.01.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.7961	0.8187	0.8608	0.4387
s	0.8200	0.8186	0.8831	0.4765
m	0.8417	0.8541	0.9059	0.4964
l	0.8521	0.8625	0.9136	0.5150
x	0.7866	0.8157	0.8339	0.4730

Tabulka A.70 –YOLOv8, RAdam optimalizátor, 50 epoch, LR 0.01, interní augmentace

Graf 6 vizualizuje hodnoty metrik z tabulku A.70 pro všechny velikosti.



Graf 6 –Vizualizace tabulky A.70

SGD s defaultním LR a augmentacemi dosáhl skoro stejných výsledků jako bez, pouze o 2–3 procentní body horší pro menší (**nano**, **small**, **medium**) velikosti a o 3–6 procentních bodů lepší pro vetší modely (**large**, **extra-large**).

Tabulka A.71 ukazuje výsledné metriky pro SGD optimalizátor s LR 0.01.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.7697	0.6621	0.7574	0.3380
s	0.8102	0.6523	0.8060	0.3994
m	0.6572	0.7744	0.6283	0.3179
l	0.8065	0.8195	0.8764	0.4729
x	0.8791	0.7803	0.8795	0.4821

Tabulka A.71–YOLOv8, SGD optimalizátor, 50 epoch, LR 0.01, interní augmentace

Pro všechny optimalizátory z LR 0.001 výrazné vylepšení prokázal pouze NAdam s vylepšením mAP50-95 o 10 procentních bodů pro velikost **large** a 8 procentních bodů pro velikost **extra-large**.

Tabulka A.75 ukazuje výsledné metriky pro NAdam optimalizátor s LR 0.001.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.7654	0.7519	0.8118	0.3878
s	0.8259	0.7658	0.8587	0.4355
m	0.7628	0.7535	0.8147	0.4021
l	0.7759	0.7197	0.7986	0.3489
x	0.6872	0.5876	0.6459	0.2205

Tabulka A.75–YOLOv8, NAdam optimalizátor, 50 epoch, LR 0.001, interní augmentace

Výsledky pro ostatní optimalizátory: Adam, Adamax, AdamW, RAdam a SGD jsou téměř stejné. Některé velikosti modelů jsou o 2–3 procentní body horší nebo lepší oproti modelům učených pouze na originálních obrázcích. Tabulka A.77 ukazuje výsledné metriky pro SGD optimalizátor s LR 0.001.

Velikost	Precision(M)	Recall(M)	mAP50(M)	mAP50-95(M)
n	0.5055	0.5561	0.4699	0.1858
s	0.6502	0.6137	0.6371	0.2698
m	0.6881	0.6413	0.6647	0.2909
l	0.7013	0.7016	0.7173	0.3418
x	0.7414	0.6644	0.7382	0.3549

Tabulka A.77–YOLOv8, SGD optimalizátor, 50 epoch, LR 0.001, interní augmentace

YOLOv8 výrazně přesahuje YOLOv5 při stejných kombinacích parametrů.

RAdam s LR 0.01 u YOLOv8 přesahuje YOLOv5 o 40 procentních bodů pro mAP50 a o 30 pro mAP50-95.

SGD s LR 0.01 u YOLOv8 přesahuje YOLOv5 o 10–30 procentních bodů pro mAP50 a o 18–23 pro mAP50-95 pro různé velikosti.

Optimalizátory s pomalejším LR přesahují analogicky z YOLO5 o 10–30 procentních bodů pro mAP50 a o 15–25 procentních bodů pro mAP50-95 pro různé velikosti modelů.

Ze všech otestovaných verzí architektur a kombinací, optimalizátorů, LR a velikostí sítě, model je YOLOv8 s augmentacemi, RAdam optimalizátor, LR 0.01 o velikosti **large,** má největší hodnoty mAP50 a mAP50-95. Porovnáme tento model s nejlepším UNet modelem.

6.2 UNet

Originální obrázky. LR 0.01

Tabulka 2 je spojením tabulek A.43–A.49, které zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.01.

Optimalizátor/Metrika	Precision	Recall	F1	IoU
Adam	0.5901	0.2215	0.3221	0.2081
Adamax	0.5231	0.2852	0.3686	0.1867
AdamW	0.5757	0.3371	0.4252	0.2903
NAdam	0.5510	0.4803	0.5132	0.4041
RAdam	0.6343	0.2827	0.3911	0.2432
RMSProp	0.5458	0.4772	0.5092	0.3838
SGD	0.5618	0.3490	0.4306	0.2787

Tabulka 2 – UNet, všechny optimalizátory, 50 epoch, LR 0.01

Nejlépe se prokázaly NAdam a RMSProp optimalizátory, které dosáhly 40 % a 38 % IoU.

AdamW, RAdam a SGD dosáhly IoU v rozmezí 24–35 %.

Adam a Adamax sotva dosahují 20 % IoU.

Originální obrázky. LR 0.1

Tabulka 3 je spojením tabulek A.50–A.56, které zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.1.

Optimalizátor/Metrika	Precision	Recall	F1	IoU
Adam	0.5375	0.4119	0.4664	0.3507
Adamax	0.5167	0.4006	0.4501	0.3154
AdamW	0	0	0	0
NAdam	0.5800	0.3977	0.4719	0.3629
RAdam	0.5845	0.3909	0.4685	0.3339
RMSProp	0.5362	0.3723	0.4395	0.3014
SGD	0.5677	0.4813	0.5210	0.3738

Tabulka 3–UNet, všechny optimalizátory, 50 epoch, LR 0.1

Výsledné IoU pro všechny optimalizátory kolísá v rozmezí 30–40 %.

Adam, NAdam, SGD dosáhly 35, 36 a 37 % IoU

Adamax, RMSProp, RAdam dosáhly 31, 30 a 33 % IoU.

AdamW je jediný optimalizátor, který dosáhl nulové metriky.

Vzhledem k jednoduchosti UNet architektury LR 0.1 není moc rychlý, jak to platí pro YOLO.

Originální obrázky. LR 0.001

Tabulka 4 je spojením tabulek A.57–A.63, které zobrazují výsledné metriky pro učení všech optimalizátorů s LR 0.001.

Optimalizátor/Metrika	Precision	Recall	F1	IoU
Adam	0.5530	0.3790	0.4498	0.3202
Adamax	0.5150	0.2450	0.3318	0.2650
AdamW	0.5897	0.1482	0.2368	0.1417
NAdam	0.6002	0.6062	0.6032	0.4556
RAdam	0.4193	0.1289	0.1972	0.0808
RMSProp	0.5737	0.3107	0.4031	0.2859
SGD	0.4743	0.5208	0.4965	0.3144

Tabulka 4 –UNet, všechny optimalizátory, 50 epoch, LR 0.001

NAdam se jevil nejlepším optimalizátorem a dosáhl hodnoty IoU 45 %.

Adam, Adamax, RMSProp a SGD dosáhly 32, 26, 28 a 31 % IoU.

AdamW a RAdam prokázaly nejhorší výsledky, 14 % a 8 % IoU.

Ze všech vyzkoušených kombinací NAdam prokázal dobré konzistentní metriky se všemi LR.

Originální a augmentované obrázky.

Provedeme učení modelů pro vybrané optimalizátory s augmentovanými obrazy a prozkoumáme jejich vliv na kvalitu segmentace.

Tabulka 5 je spojením tabulek A.78–A.80, které zobrazují výsledné metriky pro učení NAdam optimalizátoru se všemi LR.

LR/Metrika	Precision	Recall	F1	IoU
0.01	0.6748	0.5128	0.5827	0.4551
0.1	0.5854	0.6771	0.6279	0.5069
0.001	0.2258	0.0468	0.0776	0.0422

Tabulka 5 –UNet, vybrané nejlepší optimalizátory, 50 epoch, LR 0.001

NAdam s LR 0.1 a 0.01 měl přírůst IoU o 5 a 10 procentních bodů.

Takový pokrok ve kvalitě je výrazný a neočekávaný. Tato skutečnost může sloužit náznakem přeučení sítě, obzvlášť pro relativně jednoduchou architekturu jako UNet. Ověříme to v další části experimentů.

NAdam s LR 0.001 prokázal hodnotu IoU 4 %. Takový model není použitelný.

Tabulka A.81 ukazuje metriky pro SGD optimalizátor s LR 0.1.

Metrika	Hodnota
Precision	0.6244
Recall	0.5498
F1	0.5847
IoU	0.4148

Tabulka A. 81 –UNet, SGD optimalizátor, 50 epoch, LR 0.1, generované augmentační obrazy

SGD s LR 0.1 vylepšil svou hodnotu IoU o 4 procentních body.

6.3 Porovnání YOLO a UNet

Pro srovnání modelů YOLO a UNet vybereme nejlepší model a vypočteme společné metriky pro nejlepší modely YOLO a UNet a porovnáme jejich kvalitu. Porovnání provedeme pro metriky: přesnost (**precision**), úplnost (**recall**), **F1** skóre a průnik ku sjednocení (**IoU**).

Za nejlepší YOLO model byl uznán YOLOv8 o velikosti **large**, RAdam optimalizátor, LR 0.01 s použitím interních augmentací.

Tabulka A.82 ukazuje porovnání společně vypočitatelných metrik YOLO a UNet.

	Precision	**Recall**	**F1**	**IoU**
YOLOv8	0.8284	0.5955	0.6929	0.5482
UNet	0.5854	0.6771	0.6279	0.5069

Tabulka A.82 – Porovnání nejlepších modelů YOLOv8a UNet

YOLO má výrazně vyšší **precision**, což naznačuje, že YOLO má větší přesnost predikce a produkuje méně falešně pozitivních výsledků. YOLO má nižší **recall** než UNet, což naznačuje schopnost UNet identifikovat více správných segmentů v porovnání s YOLO. Podle metriky IoU je YOLOv8 trochu lepší, ale ne zásadně. Takový malý rozdíl v přesnosti správně predikovaných segmentů je alarmující, vzhledem k rozdílu složitostí architektur YOLO a UNet. Je málo pravděpodobné, aby relativně jednoduchá architektura UNet měla takový velký přírůst hodnot metrik.

Pro výběr vhodného modelu je třeba provést vizuální srovnání a zajistit, že metriky odrážejí správnou segmentaci, a ne přeučení na konkrétních datech.

6.4 Vizuální hodnocení segmentace

Pro vizuální zhodnocení segmentace provedeme inference na nejlepších modelech YOLO a UNet na dva typy obrázků: validačních z datové sady, kde je známý počet včel z anotace, a předem neučených obrázků včelích plástů a zhodnotíme generalizační schopnost sítě.

Provedeme vizuální hodnocení počtu detekovaných včel a kvality přidělení segmentační masky.

Obrázek 16 vizualizuje výsledky inference YOLOv8 a UNet na validačním obrázku, 16.a) je výsledek YOLO, 16.b) je výsledek UNet.

Obrázek 16 – Inference validačního obrázku, YOLO a) a UNet b), [vlastní zdroj]

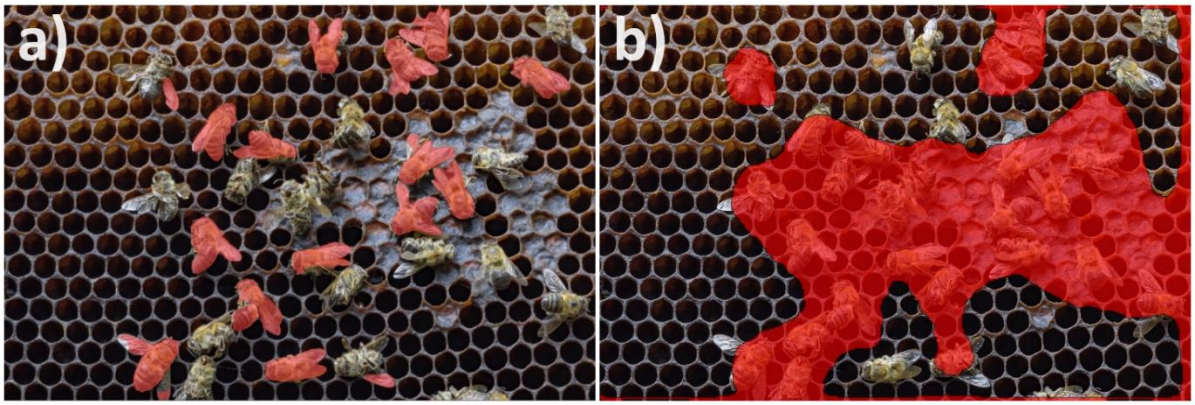
YOLO segmentuje individuálně instance včel a zachovává jejich obrysy. Díky tomuto má YOLO vyšší **precision** a nižší **recall** kvůli nedetekovaným včelám (s konfidencí nepřesahující práh), které se započítávají do metriky.

UNet segmentuje včely plochým vybarvením celé oblasti shluku včel na plástu. Tato skutečnost vede k navýšení metriky **recall**.

Vizuální experiment potvrzuje výsledky metrik: YOLO má vyšší metriku **precision**, protože detekuje každou včelu zvlášť, a UNet má vyšší recall kvůli vybarvení celého obrázku segmentační maskou.

Otestujeme schopnost práce YOLO a UNet s obrázky, které neměly ani v trénovací, ani ve validační sadě, a provedeme inferenci obrázků, které sítě předtím neviděly.

Obrázek 17 vizualizuje výsledky inference YOLO a UNet na neučeném obrázku, 17.a) je výsledek YOLO, 17.b) je výsledek UNet.



Obrázek 17 – Inference neučeného obrázku, YOLO a) a UNet b), [vlastní zdroj]

Oba modely prokázaly schopnost generalizace a extrahovaly segmenty z předtím neučených obrázků.

Platí to, co platilo pro inferenci validačních obrázků. YOLO zachovává obrysy včel, UNet vybarvuje celou oblast, kde jsou včely přítomné.

UNet docela nepřesně určuje obrys jednotlivé včely. Taková segmentace je jen těžce uplatnitelná v praxi.

Nepřesnost obrysu a vybarvení celé plochy může být způsobeno charakterem trénovacích dat, kde převážná množina obrázků má velké husté plochy křížcích se včel.

Takový charakter dat a augmentace s překlopením a otočením mohly působit přeučení na blízkých lokacích včel a zhoršení schopností segmentace obrysu.

Zkusíme provést porovnání nejlepšího YOLO modelu s nejlepším modelem UNet bez augmentace (NAdam, LR 0.001) a zhodnotíme schopnost UNet určovaní obrysů jednotlivých včel.

Obrázek 18 vizualizuje výsledky inference nejlepšího YOLO a nejlepšího UNet bez augmentace na validačním obrázku, 18.a) je výsledek YOLO, 18.b) je výsledek UNet.

Obrázek 18 – Inference validačního obrázku, nejlepší YOLO a) a nejlepší UNet b) bez augmentace, [vlastní zdroj]

UNet bez augmentace prokázal lepší segmentační schopnost než augmentovaný.

Model bez augmentace vybarvuje jednotlivé včely a nechává nevybarvený prostor na plástech mezi jednotlivými včelami oproti vybarvení celého včelího shluku jednou maskou.

Stále má potíže s přesností stanovení obrysů a rozlišením jednotlivých včel mezi sebou a ukazuje mnohem více falešně pozitivních predikcí.

Také testujeme schopnost UNet ke generalizaci a porovnáme s YOLO na obrázcích, které sítě předtím neviděly.

Obrázek 19 vizualizuje výsledky inference nejlepšího YOLO a nejlepšího UNet bez augmentace na neučeném obrázku, 19.a) je výsledek YOLO, 19.b) je výsledek UNet.

Obrázek 19 – Inference neučeného obrázku, nejlepší YOLO a) a nejlepší UNet b) bez augmentace, [vlastní zdroj]

UNet prokázal schopnost generalizace s poměrně slabou přesností určení obrysu včely a přidělení segmentační masky.

YOLO je pro náš účel celkově lepší architekturou, a to díky své schopnost rozlišovat jednotlivé včely mezi sebou a přesně stanovit jejich obrys. Během vizuálních experimentů YOLO prokázal méně falešně pozitivních odpovědí a detekoval více včelích instancí.

Provedeme experiment pro hodnocení eliminace vlivu kvality obrázku v různých jasových podmínkách na výslednou segmentaci.

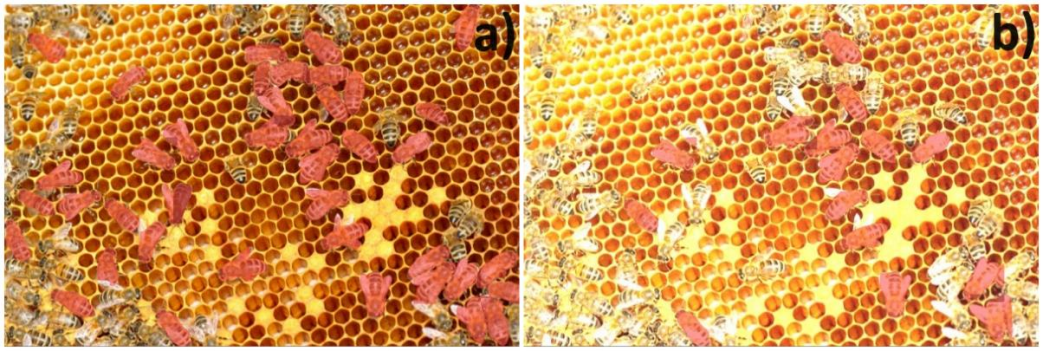
Vygenerujeme augmentační obrázky včel na plástech, simulující různé jasové podmínky během snímání fotografie.

Provedeme inference YOLO modelů se stejnými parametry na originální a augmentované inferenční obrázky. Jeden model bude naučen pouze na originálních obrázcích a druhý s použitím augmentací. Zvolíme proto nejlepší model s augmentacemi a jeho verzi bez augmentací – RAdam, LR 0.01 o velikosti **large**.

Eliminace vlivu jasových podmínek na kvalitu segmentace

Porovnáme počet detekovaných včel a přesnost přidělení segmentační masky a určíme, jestli augmentace přispěly k eliminaci vlivu kvality jasových podmínek na snímek.

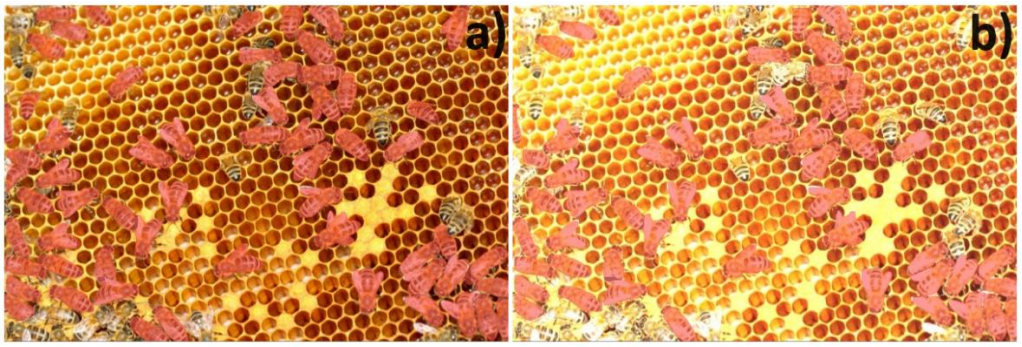
Obrázek 20 vizualizuje výsledky inference YOLO modelu učeného bez augmentace na originální a augmentovaný obrázek, 20.a) je výsledek z originálního obrázku, 20.b) je výsledek z augmentovaného obrázku.



Obrázek 20 – YOLO inference modelu učeného bez augmentace na originální a) a augmentovaný b) obrázek, Experiment 1, [vlastní zdroj]

Na obrázku 20.a) bylo segmentováno 41 včel, na 20.b) bylo segmentováno pouze 10 včel. Segmentováno o 76 % méně včel bylo na augmentovaném obrázku. Z obrázku 20 je patrné, že při změně jasových podmínek se výrazně zhoršuje počet detekovaných včel.

Obrázek 21 vizualizuje výsledky inference YOLO modelu učeného s augmentací na originální a augmentovaný obrázek, 21.a) je výsledek z originálního obrázku, 21.b) je výsledek z augmentovaného obrázku.



Obrázek 21 – YOLO inference modelu učeného s augmentací na originální a) a augmentovaný b) obrázek, Experiment 1, [vlastní zdroj]

Na obrázku 21.a) bylo segmentováno 60 včel, na 21.b) bylo segmentováno 41 včel. Segmentováno o 32 % méně včel bylo na augmentovaném obrázku.

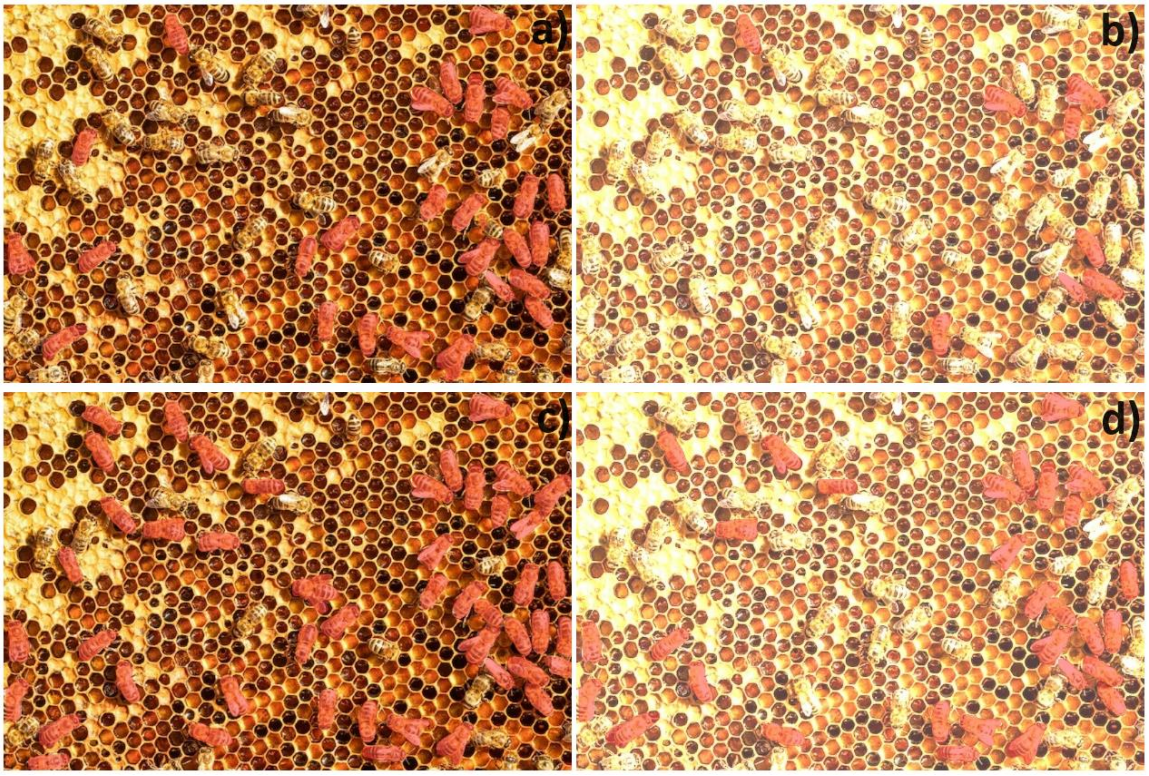
Model učený s augmentacemi segmentoval o 46 % více včel na originálním obrázku a o 310 % více včel na augmentovaném obrázku oproti modelu učenému bez augmentací.

Model učený s augmentacemi je odolnější vůči změně jasových podmínek a ztrácí o 2krát méně segmentů při jejich změně.

Augmentace měly pozitivní vliv na celkový počet segmentovaných včel a odolnost modelu vůči různým jasovým podmínkám.

Obrázky 22–24 ukazují výsledky stejného experimentu, kde obrázky a), b) jsou výsledky inference modelu učeného bez augmentace, a c), d) modelu učeného s augmentací na originální a augmentovaný obrázek.

Obrázek 22 ukazuje výsledky experimentu 2.



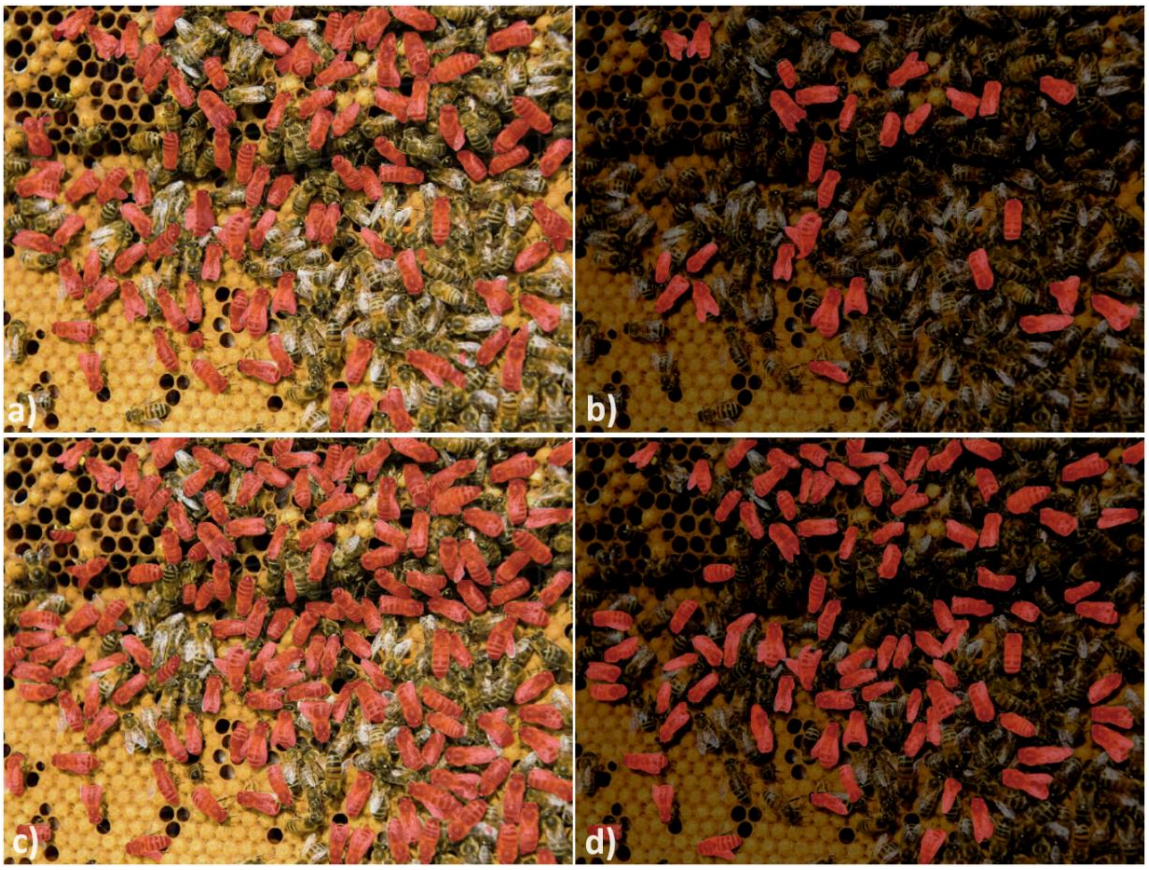
Obrázek 22 – Eliminace vlivu kvality obrázku na segmentaci, Experiment 2, [vlastní zdroj]

Na obrázku 22.a) bylo segmentováno 27 včel, na 22.b) bylo segmentováno pouze 13 včel. Segmentováno o 52 % méně včel bylo na augmentovaném obrázku.

Na obrázku 22.c) bylo segmentováno 47 včel, na 22.d) bylo segmentováno 33 včel. Segmentováno o 30 % méně včel bylo na augmentovaném obrázku.

Model učený s augmentacemi segmentoval o 74 % více včel na originálním obrázku a o 154 % více včel na augmentovaném obrázku oproti modelu učenému bez augmentací.

Obrázek 23 ukazuje výsledky experimentu 3.



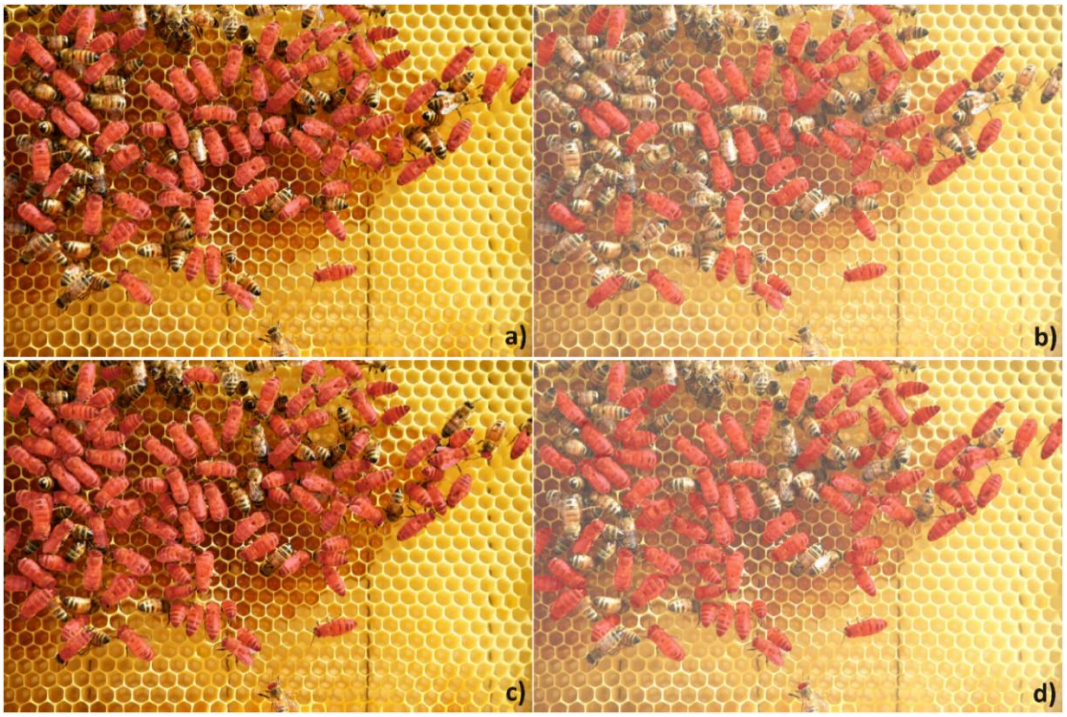
Obrázek 23 – Eliminace vlivu kvality obrázku na segmentaci, Experiment 3, [vlastní zdroj]

Na obrázku 23.a) bylo segmentováno 90 včel, na 23.b) bylo segmentováno pouze 29 včel. Segmentováno o 68 % méně včel bylo na augmentovaném obrázku.

Na obrázku 23.c) bylo segmentováno 140 včel, na 23.d) bylo segmentována 90 včel. Segmentováno o 36 % méně včel bylo na augmentovaném obrázku.

Model učený s augmentacemi segmentoval o 56 % více včel na originálním obrázku a o 210 % více včel na augmentovaném obrázku oproti modelu učenému bez augmentací.

Obrázek 24 ukazuje výsledky experimentu 4.



Obrázek 24 – Eliminace vlivu kvality obrázku na segmentaci, Experiment 4, [vlastní zdroj]

Na obrázku 24.a) bylo segmentováno 72 včel, na 24.b) bylo segmentováno 50 včel. Segmentováno o 30 % méně včel bylo na augmentovaném obrázku.

Na obrázku 24.c) bylo segmentováno 97 včel, na 24.d) bylo segmentováno 82 včel. Segmentováno o 15 % méně včel bylo na augmentovaném obrázku.

Model učený s augmentacemi segmentoval o 35 % více včel na originálním obrázku a o 64 % více včel na augmentovaném obrázku oproti modelu učenému bez augmentací.

Z provedených experimentů je patrné, že, v závislosti na kvalitě fotografie, model učený s augmentacemi segmentuje o 35–75 % více včel než model bez augmentace. Tímto je prokázán pozitivní vliv augmentace na počet detekovaných včel a následnou kvalitu segmentaci.

Model učený bez augmentace je velmi citlivý na změny jasových podmínek. V závislosti na kvalitě obrázku se počet detekovaných včel se může snížit o 30 až 80 %.

Oproti tomu je model učený s augmentacemi odolnější vůči změně jasových podmínek a produkuje robustní výsledky. Při stejných změnách jasových podmínek byla průměrná ztráta detekovaných včel v rozmezí od 15 do 40 %.

Toto ukazuje na pozitivní vliv augmentace na eliminaci vlivu kvality obrázku v různých jasových podmínkách.

7 Výsledky a diskuze

V rámci práce byla provedena série experimentů určených k hledání nejlepší kombinace hyperparametrů, jako je optimalizátor a learing rate, pro dosažení optimálních metrik s disponovanou datovou sadou. Bylo provedeno 21 experimentů pro každou implementaci architektur YOLO a UNet: YOLOv5, YOLOv8 a Pytorch-UNet.

Bylo vybráno šest kombinací optimalizátorů a rychlostí učení pro YOLOv5, osm pro YOLOv8 a čtyři pro UNet:

- YOLOv5:
    - RAdam a SGD s LR 0.01,
    - Adam, Adamax, AdamW a RAdam s LR 0.001.
- YOLOv8:
    - RAdam a SGD s LR 0.01,
    - Adam, Adamax, AdamW, NAdam, RAdam a SGD s LR 0.001.
- UNet:
    - NAdam se všechny LR: 0.1, 0.01 a 0.001,
    - SGD s LR 0.01.

Pro vybrané kombinace bylo provedeno dodatečné učení s augmentacemi a porovnání jejich výsledných metrik s předchozím učením bez augmentace a vzájemně pro vybrání nejvhodnějšího modelu architektury. U YOLO architektury zvítězila verze 8, která přesáhla přesnost verze 5 ve všech kombinacích a byla zvolena pro další porovnání s UNet.

Pro vizuální porovnání byly vybrané dva modely obou architektur s největšími společnými metrikami: YOLOv8 o velikosti **large**, RAdam optimalizátor, s LR 0.01 s metrikami IoU – 0.54, mAP50 – 0.91, mAP50-95 – 0.51 a UNet, NAdam optimalizátor s LR 0.1 s metrikou IoU – 50.

Vybraný YOLO přinesl vizuálně dobré výsledky. Přesněji stanoví a vyděluje obrysy včel bez ohledu na jejich polohu a orientaci.

Vybraný UNet ukázal neuspokojivé segmentační výsledky při vizuálním hodnocení. Toto je nejspíše způsobeno typem fotografii v datové sadě – malou rozmanitostí scén na obrázcích

plástů a hustým umístěním včel na nich, které síti zabránilo dobře se naučit rozlišovat obrys včel jak jednotlivých, tak i včel v okolí jiných včel. UNet ukázal nízkou přesnost přidělení segmentační masky. Shluky včel byly označeny jednou velkou maskou bez možnosti vizuálního rozdělení mezi instancemi včel. Maska pro jednu včelu buď významně přesahovala skutečný obrys, nebo pokrývala včelu jenom částečně necelou maskou nebo více menšími maskami.

Pro porovnání s nejlepším YOLO byl vybrán UNet s nejvyššími metrikami bez augmentace – NAdam s LR 0.01 s IoU – 0.45. Tento model již méně trpěl zaučením lokací a generoval segmentační masku, která pokrývala jednotlivé včely, a zároveň nechával neoznačený prostor na plástech mezi nimi. Přesto se segmentační masky včel ve shlucích často skládaly na sebe a vypadaly jako jeden segment. Kromě překrývání masek existuje problém přesnosti jejich přidělení, obzvlášť při různých orientacích včel na plástech.

Soubor těchto problémů brání praktické aplikaci UNet pro náš specifický úkol segmentace včel na obrázcích plástů a zamezuje dalšímu rozšíření tohoto úkolu, jako například segmentaci více druhů včel na plástech.

Následně byla provedena řada experimentů vizuálního a statistického porovnání výsledků inference stejných modelů učených s augmentacemi a bez nich.

Model učený s augmentacemi segmentoval o 35–75 % více včel než model bez augmentace. Model učený s augmentacemi je odolnější vůči změně jasových podmínek a průměrná ztráta detekovaných včel činila 15 až 40 %. Oproti tomu model učený bez augmentace byl výrazně ovlivněn změnou jasu a ztrácel 30 až 80 % včel.

Úskalím práce zůstává velikost datové množiny – 50 obrázků a 5269 instancí včel pro trénování a 15 obrázků a 1302 instancí včel pro validaci. Vyzkoušené neuronové sítě byly schopné dosáhnout vysokých výstupních hodnot: mAP50 – 0.91 a mAP50-95 – 0.51 při vybrání správné kombinace optimalizátorů a learning rate. Kvůli malému objemu trénovacích dat není možné soudit o vhodnosti ostatních kombinací, zda jsou aplikovatelné pro náš úkol.

YOLOv8 bylo uznáno za vítěznou architekturu pro náš úkol, vzhledem k jejímu vysokému počtu detekovaných včel, přesnosti segmentace, schopnosti interní augmentace a odolnosti vůči změnám jasových podmínek na obrázcích.

8 Závěr

Tato bakalářská práce se zabývala segmentací včel na obrázcích plástů.

Byla provedena rešerše přístupů pro segmentaci obrázků a klasifikaci textur klasickými přístupy a neuronovými sítěmi se zaměřením na metody, kdy včela bude moci být považována za texturu s určitou orientací.

V rámci rešerše přístupů byly rozebrány klasické metody sémantické segmentace obrázků založené na prostorových příznacích objektů, různé architektury segmentačních neuronových sítí a metody segmentace textur klasickými metodami nebo neuronovými sítěmi.

Během rešerše nebylo nalezeno žádné vhodné segmentační řešení, které by provádělo sémantickou segmentaci textur. Po dohodě s vedoucím práce bylo od požadavku na zaměření na textury upuštěno.

Na základě rešerše byla práce směřována na sémantickou segmentaci neuronovými sítěmi, vzhledem ke složitosti úkolu a omezené účinnosti klasických metod segmentace. Provedli jsme rešerši metrik pro hodnocení kvality segmentace neuronovými sítěmi a porovnali 5 architektur. Z 5 srovnaných architektur byly vybrány 2 – YOLO a UNet. U architektury YOLO jsme zvolili 2 její nejpopulárnější verze – YOLOv5 a YOLOv8. Byl proveden detailní přehled architektur a srovnání obou verzí YOLO a UNet, jejich principů segmentace, kvalitativních metrik, formátů datové sady, anotací a augmentací. Byl vytvořen skript pro výpočet společných kvalitativních metrik odlišných architektur.

Byla sestavena datová sada z 65 obrázků včel na plástech s 6571 instancemi včel na nich. Datová sada byla anotována pro obě architektury. Pro YOLO byla vytvořena anotace ve formátu COCO a pomocí skriptu konvertována do nativního formátu YOLO se stejným jménem. Anotace UNet byla odvozena z COCO anotace také pomocí konverzního skriptu a byla vytvořena anotace ve formátu binární segmentační masky.

Na sestavenou datovou sadu byly aplikované augmentace imitující obrázky zabrané za různých jasových podmínek. Byly využity tři druhy augmentace: náhodná překlopení, otočení a nastavení kontrastu, jasu a saturace. Pro YOLO byl použit jeho interní mechanismus augmentace (hyperparametry v konfiguračním souboru), který generuje augmentace k obrázku a anotaci během učení. Pro UNet bylo pomocí vlastního skriptu generováno 65 dodatečných obrázků a anotace do datové sady. Celkem byly během práce

vytvořeny 4 skripty: dva pro konverzi formátů anotací, jeden pro výpočet společných metrik a jeden pro augmentaci UNet.

Vybrané implementace neuronových sítí byly rozšířeny tak, aby podporovaly stejné sady optimalizátorů. Byla provedena řada experimentů pro každou síť a vybrány kombinace optimalizátorů a learning rate pro dosažení maximálních kvalitativních hodnot na naší datové sadě.

Pro vybrané kombinace bylo provedeno učení s augmentacemi a zhodnocen jejich vliv na výsledné metriky sítě. Na UNet augmentace působily negativně a vedly k přeučení na blízkých lokacích včel a zhoršené schopnosti extrakce segmentů. YOLOv5 mírně vylepšilo své metriky díky svým interním augmentacím. YOLOv8 prokázalo velký vliv augmentací k vylepšení segmentační schopnosti sítě.

Na základě YOLOv8 byly provedeny čtyři experimenty, které dokázaly účinnost augmentací pro zvýšení kvality segmentací a eliminací vlivu různých jasových podmínek na výsledek segmentace.

9 Seznam použité literatury, obrázků, zdrojových kódů, rovnic, tabulek a grafů

9.1 Seznam použité literatury

[1] KUNC, Martin, Pavel DOBEŠ, Jana HURYCHOVÁ, et al. The Year of the Honey Bee (Apis mellifera L.) with Respect to Its Physiology and Immunity: A Search for Biochemical Markers of Longevity [online]. [cit. 2024-02-02]. Dostupné z: https://doi.org/10.3390/insects10080244

[2] AZAD, Reza, Moein HEIDARI, Kadir YILMAZ, Michael HÜTTEMANN, Sanaz KARIMIJAFARBIGLOO, Yuli WU, Anke SCHMEINK a Dorit MERHOF. Loss Functions In The Era Of Semantic Segmentation: A Survey And Outlook [online]. [cit. 2024-02-02]. Dostupné z: https://browse.arxiv.org/pdf/2312.05391.pdf

[3] EMEK SOYLU, Busra, Mehmet Serdar GUZEL, Gazi Erkan BOSTANCI, Fatih EKINCI, Tunc ASUROGLU a Koray ACICI. Deep-Learning-Based Approaches for Semantic Segmentation of Natural Scene Images: A Review [online]. [cit. 2024-02-02]. Dostupné z: https://www.semanticscholar.org/reader/88d28e14b21a0c4478ac0f904009f3f96636a2f1

[4] HLAVÁČ, Václav a ŠONKA, Milan. Počítačové vidění. Praha: Grada, 1992. ISBN isbn80-85424-67-3

[5] LOWE, David G. Distinctive Image Features from Scale-Invariant Keypoints [online]. [cit. 2024-02-01]. Dostupné z: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[6] DALAL, Navneet a Bill TRIGGS. Histograms of Oriented Gradients for Human Detection [online]. [cit. 2024-02-01]. Dostupné z: https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

[7] CORTES, Corinna a Vladimir VAPNIK. Support-vector networks [online]. [cit. 2024-02-01]. Dostupné z: https://link.springer.com/article/10.1007/BF00994018

[8] TERVEN, Juan R. a Diana M. Cordova-Esparza. A Comprehensive Review Of YOLO Architectures In Computer Vision: From YOLOV1 to YOLOV8 and YOLO-NAS [online]. [cit. 2024-02-22]. Dostupné z: https://arxiv.org/pdf/2304.00501.pdf

[9] RONNEBERGER, Olaf, Philipp FISCHER a Thomas BROX. U-Net: Convolutional Networks for Biomedical Image Segmentation [online]. [cit. 2024-02-09]. Dostupné z: https://arxiv.org/pdf/1505.04597.pdf

[10] HE, Kaiming, Georgia GKIOXARI, Piotr DOLLAR a Ross GIRSHICK. Mask R-CNN [online]. [cit. 2024-02-22]. Dostupné z: https://arxiv.org/pdf/1703.06870v3.pdf

[11] CHEN, Liang-Chieh, George PAPANDREOU, Iasonas KOKKINOS, Kevin MURPHY a Alan L. YUILLE. Semantic Image Segmentation With Deep Convolutional Nets and Fully Connected CRFS [online]. [cit. 2024-02-22]. Dostupné z: https://arxiv.org/pdf/1412.7062v4.pdf

[12] ZHAO, Hengshuang, Jianping SHI, Xiaojuan QI, Xiaogang WANG a Jiaya JIA. Pyramid Scene Parsing Network [online]. [cit. 2024-02-22]. Dostupné z: https://arxiv.org/pdf/1612.01105v2.pdf

[13] MADASU, Vamsi Krishna a Prasad YARLAGADDA. An in depth comparison of four texture segmentation methods [online]. [cit. 2024-02-08]. Dostupné z: https://eprints.qut.edu.au/12743/1/12743.pdf

[14] KARABAG, Cefa, Jo VERHOEVEN, Naomi Rachel MILLER a Constantino Carlos REYES-ALDASORO. Texture Segmentation: An Objective Comparison between Five Traditional Algorithms and a Deep-Learning U-Net Architecture [online]. [cit. 2024-02-08]. Dostupné z: https://www.mdpi.com/2076-3417/9/18/3900

[15] ANDREARCZYK, Vincent a Paul F. WHELAN. Texture segmentation with Fully Convolutional Networks [online]. [cit. 2024-02-08]. Dostupné z: https://arxiv.org/pdf/1703.05230v1.pdf

[16] HASAN, Md. Mahedi, Aritejh Kr GOIL, Henryk CHAN, et al. A Novel Application for Real-time Arrhythmia Detection using YOLOv8 [online]. [cit. 2024-02-08]. Dostupné z: https://doi.org/10.48550/arXiv.2305.16727

[17] BOCHKOVSKIY, Alexey, Chien-Yao WANG a Hong-Yuan Mark LIAO. YOLOv4: Optimal Speed and Accuracy of Object Detection [online]. [cit. 2024-01-26]. Dostupné z: https://arxiv.org/pdf/2004.10934.pdf

[18] YOLOv5 repozitář [online]. [cit. 2024-01-26]. Dostupné z: https://github.com/ultralytics/yolov5

[19] PyTorch oficiální stránka [online]. [cit. 2024-02-09]. Dostupné z: https://pytorch.org/

[20] YOLOv8 repozitář [online]. [cit. 2024-01-26]. Dostupné z: https://github.com/ultralytics

[21] Pytorch-UNet github [online]. [cit. 2024-02-09]. Dostupné z: https://github.com/milesial/Pytorch-UNet

[22] CVAT oficiální stránka [online]. [cit. 2024-01-26]. Dostupné z: https://www.cvat.ai/

[23] VoTT github [online]. [cit. 2024-01-26]. Dostupné z: https://github.com/microsoft/VoTT

[24] Imaug oficiální stránka [online]. [cit. 2024-02-09]. Dostupné z: https://imgaug.readthedocs.io/en/latest/

[25] Albumentations oficiální stránka [online]. [cit. 2024-02-09]. Dostupné z: https://albumentations.ai/

9.2 Seznam obrázků, zdrojových kódů, rovnic, tabulek a grafů

9.2.1 Seznam obrázků

9.2.2 Seznam zdrojových kódů

9.2.3 Seznam rovnic

9.2.4 Seznam tabulek

9.2.5 Seznam grafů

Seznam Příloh

Přílohy

A. Tabulky experimentů

Jednotlivé tabulky jsou uložené do samostatného souboru v příloze k této práci.

B. COCO2YOLO skript

```
import json
import os

import numpy as np
from PIL import Image


def get_img_ann(image_id, data):
    img_ann = []
    isFound = False
    for ann in data['annotations']:
        if ann['image_id'] == image_id:
            img_ann.append(ann)
            isFound = True
    if isFound:
        return img_ann
    else:
        return None


def get_img(input_images, filename, data):
    for img in data['images']:
        if img['file_name'] == filename:
            img['img_data'] = Image.open(os.path.join(input_images,
filename))
            return img


def create_labels(
        input_images,
        input_json,
        output_labels,
        heigh_thresh=0
):
    file_names = os.listdir(input_images)

    f = open(input_json)
    data = json.load(f)
    f.close()

    count = 0
    for filename in file_names:
        img = get_img(input_images, filename, data)
        img_id = img['id']
        img_w = img['width']
        img_h = img['height']
        img_ann = get_img_ann(img_id, data)

        if img_ann:
            file_object =
open(f"{output_labels}/{filename.split('.jpg')[0]}.txt", "a")

            for ann in img_ann:
```

```
                # Subtracting 1 from category id because class
labels in yolo format starts from 0
                current_category = ann['category_id'] - 1

                if heigh_thresh is not None and ann['bbox'][3] <
heigh_thresh:
                    continue

                s = [j for i in ann['segmentation'] for j in i]  #
all segments concatenated
                s = (np.array(s).reshape(-1, 2) / np.array([img_w,
img_h])).reshape(-1).tolist()
                file_object.write(f"{current_category} {'
'.join(str(x) for x in s)}\n")

            file_object.close()

        count += 1
```

C. COCO2UNetBinaryMask skript

```
import os

import cv2
import numpy as np
from pycocotools.coco import COCO


def coco_to_unet_binary_mask(coco_ann_file, output_folder):
    coco = COCO(coco_ann_file)

    os.makedirs(output_folder, exist_ok=True)

    for img_id in coco.getImgIds():
        img_info = coco.loadImgs(img_id)[0]
        img_filename = img_info['file_name']
        img_width = img_info['width']
        img_height = img_info['height']

        ann_ids = coco.getAnnIds(imgIds=img_id)
        anns = coco.loadAnns(ann_ids)

        mask = np.zeros((img_height, img_width), dtype=np.uint8)

        for ann in anns:
            polygon = ann['segmentation'][0]
            polygon = np.array(polygon).reshape((-1,
2)).astype(np.int32)
            cv2.fillPoly(mask, [polygon], 1)

        mask_filename = os.path.join(output_folder,
f"{img_filename.split('.')[0]}.jpg")
        cv2.imwrite(mask_filename, mask * 255)

        cv2.imwrite(mask_filename, mask * 255)
```

D. UNet augmentace skript

```
import os
import cv2
import albumentations as A


def define_transforms():
    horizontal_flip = A.HorizontalFlip(p=1)
    vertical_flip = A.VerticalFlip(p=1)
    brightness_contrast = A.RandomBrightnessContrast(p=0.95)

    image_transform = A.Compose([
        horizontal_flip,
        vertical_flip,
        brightness_contrast
    ])

    return image_transform


def augment_folder(
        images_dir,
        masks_dir,
        image_transform,
        aug_images,
        aug_masks,
        aug_per_image
):
    image_files = os.listdir(images_dir)

    for image_file in image_files:
        image = cv2.imread(os.path.join(images_dir, image_file))
        mask = cv2.imread(os.path.join(masks_dir, image_file),
cv2.IMREAD_GRAYSCALE)
        if image is None or mask is None:
            print(f"Failed to load image or mask for {image_file}")
            continue

        for i in range(aug_per_image):
            augmented = image_transform(image=image, mask=mask)
            augmented_image = augmented['image']
            augmented_mask = augmented['mask']

            image_name, _ = os.path.splitext(image_file)
            output_image_path = os.path.join(aug_images,
f"{image_name}_{i}.jpg")
            output_mask_path = os.path.join(aug_masks,
f"{image_name}_{i}.jpg")
            cv2.imwrite(output_image_path, augmented_image)
            cv2.imwrite(output_mask_path, augmented_mask)

            print(f"Augmentation completed for
{output_image_path}")
```

E. YOLO a UNet výpočet společných metrik skript

```
def predict_img(
        net,
        full_img,
        device,
        scale_factor=1,
        out_threshold=0.5
):
    net.eval()
    img = torch.from_numpy(BasicDataset.preprocess(None, full_img,
scale_factor, is_mask=False))
    img = img.unsqueeze(0)
    img = img.to(device=device, dtype=torch.float32)

    with torch.no_grad():
        output = net(img).cpu()
        output = F.interpolate(output, (full_img.size[1],
full_img.size[0]), mode='bilinear')
        if net.n_classes > 1:
            mask = output.argmax(dim=1)
        else:
            mask = torch.sigmoid(output) > out_threshold

    return mask[0].long().squeeze().numpy()


def binaryMaskIOU(mask1, mask2):
    mask1_area = np.count_nonzero(mask1 == 1)
    mask2_area = np.count_nonzero(mask2 == 1)
    intersection = np.count_nonzero(np.logical_and(mask1 == 1,
mask2 == 1))
    iou = intersection / (mask1_area + mask2_area - intersection)
    return iou


def calc_metrics(unet_val_masks, net, device):
    ious = []
    precisions = []
    recalls = []
    for image in os.listdir(unet_val_images):

        orig_image = Image.open(os.path.join(unet_val_images,
image))

        predicted_mask = predict_img(net, orig_image, device)
        gt_mask = Image.open(os.path.join(unet_val_masks,
image.split('.')[0] + '.png')).convert('L').resize(
            (predicted_mask.shape[1], predicted_mask.shape[0]))

        np_img = predicted_mask
        np_img[np_img > 0] = 1
        predicted_mask = np_img

        np_img = np.array(gt_mask)
```

```
            np_img[np_img > 0] = 1
            gt_mask = np_img

            iou = binaryMaskIOU(gt_mask, predicted_mask)
            ious.append(iou)

            tn, fp, fn, tp = confusion_matrix(gt_mask.flatten(),
predicted_mask.flatten()).ravel()
            if tp + fp != 0:
                precisions.append(tp / (tp + fp))
            else:
                precisions.append(0)

            if tp + fp != 0:
                recalls.append(tp / (tp + fn))
            else:
                recalls.append(0)

        iou = np.mean(ious)
        precision = np.mean(precisions)
        recall = np.mean(recalls)
        if precision + recall != 0:
            f1_score = 2 * precision * recall / (precision + recall)
        else:
            f1_score = 0
        print("IoU: {}, Precision: {}, Recall: {}, F1-score:
{}".format(iou, precision, recall, f1_score))
        return iou, precision, recall, f1_score
```