



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH
TECHNOLOGIÍ**

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

**MODERNÍ PŘÍSTUP K MĚŘENÍ CITLIVOSTI
MIKROBIÁLNÍCH KULTUR NA ANTIBIOTIKA
S VYUŽITÍM STROJOVÉHO UČENÍ**

A MODERN APPROACH TO MEASURING ANTIBIOTIC SUSCEPTIBILITY OF MICROBIAL
CULTURES USING MACHINE LEARNING

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

Jakub Lepík

VEDOUCÍ PRÁCE

ADVISOR

Ing. Michal Čičatka

BRNO 2024

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Jakub Lepík

ID: 239122

Ročník: 3

Akademický rok: 2023/24

NÁZEV TÉMATU:

Moderní přístup k měření citlivosti mikrobiálních kultur na antibiotika s využitím strojového učení

POKYNY PRO VYPRACOVÁNÍ:

- 1) Nastudujte metody měření citlivosti mikrobiálních kultur na antibiotika (tzv. AST) a algoritmy strojového učení pro detekci objektů v obraze.
- 2) Navrhněte algoritmy pro měření citlivosti mikrobiálních kultur na různé typy antibiotik, s důrazem na využití strojového učení.
- 3) Připravte vhodné datové sady pro vývoj a evaluaci navrženého řešení. Připravte prostředí a skripty pro trénování navržené architektury strojového učení.
- 4) Natrénujte několik modelů v různých konfiguracích. Sledujte parametry jako přesnost a výpočetní náročnost.
- 5) Integrujte řešení do jednoduché webové aplikace.
- 6) Diskutujte dosažené výsledky a jejich význam v kontextu studované problematiky.

Pro semestrální projekt je nutné splnit body 1-3.

Práce probíhá ve spolupráci se společností Bruker s.r.o.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce.

Termín zadání: 5.2.2024

Termín odevzdání: 28.5.2024

Vedoucí práce: Ing. Michal Čičatka

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

UPOZORNĚNÍ:

Autor bakalářské práce nesmí při vytváření bakalářské práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Bakalářská práce se zaměřuje na problematiku měření citlivosti mikrobiálních kultur na antibiotika (AST), konkrétně na vylepšení a automatizaci vyhodnocení diskové difúzní metody pomocí strojového učení a architektur pro detekci objektů v obraze. Díky využití vývojové platformy TensorFlow a rozsáhlé datové sady, na níž byly vytrénovány vlastní detekční modely, jako je EfficientDet, je umožněno zpracování široké škály vstupních dat. To přináší možnost využití mobilních zařízení vedle tradičních laboratorních přístrojů při vyhodnocování této metody. Pomocí dalších technik zpracování obrazu a knihovny OpenCV byl vyvinut vlastní algoritmus na měření velikosti inhibičních zón, který je společně s detekčními modely integrován v rámci modulu do webové aplikace společnosti Bruker Daltonics GmbH & Co. KG. Tento modul, vyvíjený pomocí platformy ASP.NET, je přehledným a užitečným nástrojem pro asistenci pracovníkům v mikrobiologických laboratořích.

KLÍČOVÁ SLOVA

AST, ASP.NET, C++, C#, disková difúzní metoda, Jupyter, konvoluční neuronová síť, OpenCV, Python, strojové učení, TensorFlow, testování citlivosti na antibiotika, umělá inteligence

ABSTRACT

The bachelor's thesis focuses on antibiotic susceptibility testing (AST), specifically enhancing and automating the assessment of the disk diffusion method using machine learning and object detection architectures. Thanks to the TensorFlow development platform and extensive dataset, on which custom detection models like EfficientDet were trained, processing a wide range of input data is enabled. This brings the possibility of using mobile devices alongside traditional laboratory equipment when evaluating this method. By employing additional image processing techniques and the OpenCV library, a custom algorithm for measuring the size of inhibitory zones was developed, which, along with the detection models, is integrated within the application module developed by Bruker Daltonics GmbH & Co. KG. This module, created using the ASP.NET platform, is a precise and valuable tool for assisting personnel in microbiological laboratories.

KEYWORDS

antibiotic susceptibility testing, AST, artificial intelligence, ASP.NET, convolutional neural network, C++, C#, disk diffusion method, EfficientDet, Jupyter, machine learning, OpenCV, Python, TensorFlow

LEPÍK, Jakub. *Moderní přístup k měření citlivosti mikrobiálních kultur na antibiotika s využitím strojového učení*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Vedoucí práce: Ing. Michal Čičatka

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jakub Lepík
VUT ID autora:	239122
Typ práce:	Bakalářská práce
Akademický rok:	2023/24
Téma závěrečné práce:	Moderní přístup k měření citlivosti mikrobiálních kultur na antibiotika s využitím strojového učení

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

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno

.....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce, panu Ing. Michalu Čičatkovi, za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod	11
1 Měření citlivosti na antibiotika	13
1.1 Metody AST	13
1.1.1 Diskový difúzní test	14
1.1.2 Diluční metoda	14
1.1.3 Gradientová difúzní metoda	15
2 Strojové učení	16
2.1 Metody učení	16
2.2 Hluboké učení	16
2.2.1 Umělé neuronové sítě	17
2.3 Konvoluční neuronové sítě	19
2.3.1 Konvoluční vrstva	20
2.3.2 Sdružovací vrstva	22
2.3.3 Modely detekce objektů	23
3 Návrh řešení	25
3.1 Detekce antibiotických disků	26
3.1.1 Příprava trénovacích dat	26
3.1.2 Příprava modelů a trénovacího prostředí	29
3.2 Měření velikosti inhibičních zón	31
4 Implementace	33
4.1 Trénování detekčních modelů	33
4.1.1 Konfigurace	33
4.1.2 Trénování	34
4.1.3 Vyhodnocení	37
4.2 Měření inhibičních zón	39
4.2.1 Vyhodnocení	43
5 Integrace	45
5.1 Detekce antibiotických disků	45
5.2 Měření velikosti inhibičních zón	45
5.3 Webová aplikace	49
Závěr	57
Literatura	59

Seznam symbolů a zkratk	62
A Obsah elektronické přílohy	63

Seznam obrázků

1.1	Diskový difúzní test	14
1.2	Diluční metoda	15
1.3	Gradientová difúzní metoda	15
2.1	Biologický neuron	18
2.2	Model umělého neuronu	18
2.3	Příklady aktivačních funkcí	19
2.4	Příklad první konvoluční vrstvy	21
2.5	Příklad prostorového uspořádání neuronů	22
2.6	Příklad konvoluce	22
2.7	Demonstrace funkce sdružovací vrstvy	23
2.8	Architektura modelů EfficientDet	24
3.1	Princip aplikace	25
3.2	Příklady ideálně zpracovaných snímků	27
3.3	Anotace antibiotických disků	28
3.4	Ukázka sešitu Jupyter	30
3.5	Anotace inhibičních zón	32
4.1	Závislost hodnoty ztrátové funkce	36
4.2	Příklady jednotlivých hodnot IoU.	37
4.3	Postup algoritmu měření inhibičních zón	40
4.4	Vytvoření výsledné binární masky	42
4.5	Proces určení hranice zóny a velikosti	43
5.1	Formulář webové aplikace	50
5.2	Prezentace detekovaných disků a zón uživateli	51
5.3	Prezentace zpracovaného snímku	52
5.4	Klíčování Petriho misky	53
5.5	Táhla jednotlivých rámečků	54
5.6	Celá webová aplikace	56

Seznam výpisů

3.1	Zajištění podpory klávesových zkratk	29
4.1	Konfigurační soubor ve formátu protobuf	35
4.2	Kód z buňky sešitu Jupyter	35
4.3	Export checkpointů do saved_data	36
4.4	Metoda <code>_find_zone</code> v jazyce Python	41
4.5	Metoda <code>_seedfill</code> v jazyce Python	42
4.6	Algoritmus ICE v jazyce Python	43
5.1	Metoda <code>ObtainDiskDetection</code> v jazyce C#	46
5.2	Metoda <code>FindZoneSeedfill</code> v jazyce C++	47
5.3	Zapouzdřovací funkce <code>IFindZoneSeedfill</code> v jazyce C++	48
5.4	Exportace funkcí v jazyce C++	48
5.5	Třída <code>UnsafeNativeMethods</code> v jazyce C#	48
5.6	Část ASP.NET formuláře	49
5.7	Metoda pro odeslání formuláře	51
5.8	Funkce <code>handleMouseDownSel</code> v jazyce JavaScript	55

Úvod

Oblast strojového učení a umělé inteligence je v poslední době velmi rozšířeným fenoménem. Ať už se jedná o cílenou reklamu na webu, personalizovaná doporučení obsahu na sociálních sítích nebo systémy pro rozpoznávání poznávacích značek aut, strojové učení a umělá inteligence mají významný vliv na náš život. Díky jejich schopnostem, kterým je možno nahradit a v některých případech dokonce i předčít ty lidské, se umělá inteligence uplatňuje v celé řadě odvětví.

Umělá inteligence nachází uplatnění i v medicíně a zdravotnictví, které patří mezi nejdůležitější oblasti pro lidskou společnost. Jedním z největších současných problémů v tomto odvětví je neustále rostoucí odolnost mikrobiálních kultur vůči antibiotikům. Proto je nezbytné s antibiotiky zacházet odpovědně a používat je pouze v případech, kdy je jisté, že budou účinná a cílové mikroorganismy na ně budou citlivé. K tomu slouží metody měření citlivosti na antibiotika.

Jednou z klíčových metod tohoto oboru je disková difúzní metoda. Její nevýhodou je, že při jejím vyhodnocení je nutná přítomnost kvalifikovaného lidského operátora. Automatizace a vylepšení postupů při vyhodnocení této metody by znamenala celkové zefektivnění a umožnila by laboratornímu personálu zaměřit se na důležitější aspekty jejich práce.

Tato bakalářská práce se zabývá řešením právě tohoto problému. Aplikuje techniky strojového učení, umělé inteligence a zpracování obrazu k vylepšení postupů při vyhodnocování diskové difúzní metody a její automatizaci. Navrhované řešení klade důraz na co nejširší využití v co největším počtu laboratoří disponujících různým vybavením a snaží se využít i široce dostupná zařízení, jako jsou mobilní telefony.

Samotným výstupem této práce je modul do webové aplikace společnosti Bruker Daltonics GmbH & Co. KG, který umožňuje automatizované zpracování výsledků diskové difúzní metody a jejich efektivní a přehledné prezentování uživateli aplikace.

První dvě kapitoly představují teoretický úvod, ten se nejdříve věnuje procesu měření citlivosti na antibiotika a popisuje ho z mikrobiologické stránky, shrnuje nejčastěji používané metody a jejich lišící se vlastnosti. Druhá kapitola se zabývá problematikou strojového učení, hlubokého učení a konvolučním neuronovým sítím, které jsou klíčové pro zpracování obrazu.

Následující kapitoly se věnují praktické části této bakalářské práce. Třetí kapitola se zabývá návrhem výsledného řešení pro vyhodnocování výsledků diskové difúzní metody a definuje modul webové aplikace. Rozděluje následující vývoj do dvou hlavních částí: detekování antibiotických disků a vyhodnocení velikostí inhibičních zón. V rámci první části pro velkou variabilitu vstupních dat navrhuje řešení v podobě natrénování vlastní architektury detekce objektů v obraze a pro měření velikosti inhibičních zón uvažuje algoritmický postup. Věnuje se mimo uvedené také

přípravě datových sad pro trénování, což zahrnuje zejména anotaci snímků a prezentuje klíčové vývojové platformy pro další práci.

Čtvrtá kapitola se zabývá implementací navržených postupů a jejich vyhodnocení a poslední, pátá kapitola, se věnuje samotné integraci postupů z kapitol předchozích do výsledné webové aplikace.

1 Měření citlivosti na antibiotika

Měření citlivosti na antibiotika, zkráceně AST z anglického Antibiotic Susceptibility Testing hraje klíčovou roli v oblasti klinické mikrobiologie. Pro některé druhy patogenů, čili organismů, které vyvolávají nemoc, existují již zavedené typy léčby. U ostatních patogenů je však nutné nasadit přístup a léčbu individuální. To platí např. u patogenů běžně vykazující odolnost vůči antimikrobiálním léčivům. Příkladem mohou být gram-negativní bakterie v trávicím traktu. Střevní bakterie jsou totiž často imunní vůči antibiotikům a je tedy nutné najít druh léčby, který bude proti danému kmenu patogenu účinný [1].

Antibiotika jsou přírodní látky vytvářeny mikroorganismy k potlačení organismů konkurenčních. Přirozeně je produkují bakterie a houby jako penicilin. Pro lékařské účely jsou často modifikována a uměle syntetizována. Díky svým schopnostem ovlivňovat a potlačovat mikroorganismy včetně patogenů se využívají již desítky let jako léčiva. Existuje mnoho typů antibiotik s různým účinkem na bakterie. Při výběru se v konkrétním případě bere v úvahu, na jaký druh mikroorganismu mají cílit, prostředí, ve kterém mají působit a další faktory. Hodnocení účinku antibiotik zahrnuje například jejich potřebné množství, dobu působení a zda-li požadovaný mikroorganismus zahyne, přestane se šířit nebo se jeho šíření pouze zpomalí [2].

Odolnost mikroorganismů vůči antibiotikům představuje vážný problém v boji proti infekčním onemocněním. Postupem času se jejich odolnost zvyšuje a u některých kombinací i skokově. Mikroorganismy jsou mimořádně proměnlivé a rychle rostou, některé se dokonce reprodukuje každých 30 minut, a tak i často mutují. V přítomnosti přímého ohrožení v prostředí s antibiotiky jsou pak schopny přežít jen odolné mutace [3]. K snižování účinnosti antibiotik přispívá jejich často zbytečné nebo preventivní nasazování, a to jak u lidí, tak zvířat [2].

AST je tedy nepostradatelným diagnostickým nástrojem, jenž umožňuje cílenou a včasnou léčbu bakteriálních onemocnění a zabraňuje nasazování nevhodných a neúčinných antibiotik. To v konečném důsledku také znamená, že zamezuje i zbytečnému zvyšování odolnosti mikrobů [1, 2, 4].

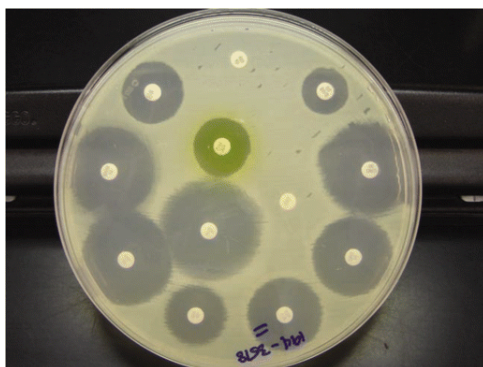
1.1 Metody AST

Aby se mohla citlivost na antibiotika změřit, je prvně nutné vzorky mikrobů odebrat a potřebné mikroby izolovat. Vzorkům se poskytnou vhodné podmínky pro růst a kultivují se [1]. Samotné pěstování mikrobů, nebo-li kultivace, probíhá za přítomnosti takzvaného živného média. Jedná se o směs živin, které dané mikroby pro svůj růst potřebují. Dělí se do dvou kategorií, a to na pevná a tekutá [4]. Tekutými živnými médii mohou být například mléko nebo sladidlo [2], u pevných je základem

agar, který se vyrábí z mořské řasy. Po kultivaci je možné zahájit samotné měření jejich citlivosti [4].

1.1.1 Diskový difúzní test

Disková difúzní metoda je jednoduchý, levný a standardizovaný způsob testování citlivosti bakterií na antibiotika [5, 6]. Před inkubací jsou nanášeny mikroby na Petriho misku obsahující pevné agarové živné médium. Na agarový povrch se pak nanášou za pomoci speciálního dávkovače disky s antibiotiky. Množství antibiotik v jednotlivých discích je pevně stanoveno. Misky se následně umístí do inkubátoru, kde proběhne růst a množení bakterií. Pokud je antibiotikum v daném disku na přítomné mikroby účinné, v oblasti kolem disku se bakterie nemnoží. Tato oblast je nazývána inhibiční. Výsledkem testu je pak jeden z dvou stavů, a tedy jestli jsou dané mikroby na testované antibiotikum citlivé nebo rezistentní [4]. Za přísnějších a standardizovaných podmínek lze zaznamenávat i míru inhibice bakterií v oblasti kolem disku a provádět tak kvantitativní hodnocení účinnosti antibiotika [5]. Výsledek testu se odvíjí od velikosti inhibiční zóny, která se určuje v milimetrech posuvným měřidlem. Pro potřeby určení účinnosti antibiotik jsou velikosti jejich inhibičních zón uvedeny ve standardizovaných tabulkách, které poskytuje například Institut klinických a laboratorních standardů [4, 7].

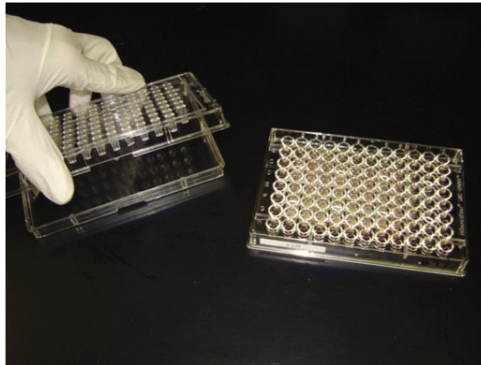


Obr. 1.1: Diskový difúzní test, převzato z [5].

1.1.2 Diluční metoda

U diluční metody se citlivost na antimikrobiální látky hodnotí pomocí minimální inhibiční koncentrace (MIC), což představuje nejmenší množství látky potřebné k úplnému zamezení růstu testovaného organismu v laboratorním prostředí. Oproti diskové difúzní metodě je použito tekuté živné médium. Testování diluční metodou tradičně zahrnuje přípravu série zkumavek s mikroby a různými koncentracemi antimikrobiální látky. MIC je pak stanovena jako nejnižší koncentrace, která zamezuje

růstu organismu [1]. Růst organismu je přitom pozorovatelný okem a platí, že zakalenější vzorky obsahují vyšší koncentraci mikrobů [4]. Výhodou diluční metody je její přesnost, je však časově náročná a finančně nákladná [6].



Obr. 1.2: Diluční metoda, převzato z [5].

1.1.3 Gradientová difúzní metoda

Gradientová difúzní metoda (Etest) využívá tenké plastové proužky obsahující postupně se zvyšující množství antibiotika [5]. Po inkubaci za působení antibiotika vznikne kolem proužku inhibiční zóna a hledá se bod, kde její okraj protíná okraj proužku. Ze stupnice na proužku se pak odečte přibližná příslušná hodnota MIC. Očkování půdy zkoumaným kmenem probíhá před položením testu, rovněž se používá agarové živné médium, což je obdobný postup jako u diskové difúzní metody [4]. Tato metoda může být použita jako levnější a méně časově náročná alternativa k dilučním testům, nevýhodou je však menší přesnost a oproti diskové difúzní metodě stále vyšší cena [6].



Obr. 1.3: Gradientová difúzní metoda, převzato z [5].

2 Strojové učení

Strojové učení je vědní obor zasahující do mnoha dalších jako je informatika, statistika a datová věda. Jeho smyslem je docílit, aby se navržený počítačový systém sám zdokonaloval v závislosti na jeho předešlých zkušenostech a vstupních datech, a tedy se „učil“ [8]. Strojové učení je nedílnou součástí obecnějšího oboru umělé inteligence, která slouží jako náhrada lidského faktoru u typu rozhodování vyžadujících nějakou formu lidské intuice a širší představu o světě např. rozpoznávání objektů nebo řeči.

Naprogramovat takový systém manuálně pomocí logických pravidel je složité a vyžadovalo by velké množství lidských prostředků – programátorů. Ti by zase museli být schopni přesně definovat fungování lidských rozhodovacích procesů a v dostatečné složitosti je převést právě do logických pravidel. Od podobných postupů vytvoření rozsáhlejších systémů umělé inteligence se z důvodu jejich složitosti a nevelkých úspěchů již upustilo [9].

2.1 Metody učení

Učení s učitelem je metoda, kdy se má navržený model za úkol naučit interpretovat určité vzory ve vstupních datech. Pro trénování je pak třeba modelu dodat data ve formě dvojic vstup – výstup, kde jednotlivé výstupy určuje učitel v podobě algoritmu nebo člověka [9]. Příkladem může být binární klasifikace nevyžádaných emailů, kde je model trénován na emailech označených jako vyžádané a nevyžádané.

Učení bez učitele se naopak vyznačuje tím, že se model učí z pozorování a analýzy vzorců namísto z předem definovaných dvojic vstupů a výstupů. Příkladem je takzvané shlukování, kde dochází k rozdělení vstupních dat na menší celky – shluky, objekty v nich pak mají určité společné vlastnosti.

Kombinace učení s učitelem a bez učitele je situace, kdy část trénovacích dat je předem přiřazena k jednotlivým výstupům a část ne.

Zpětnovazebné učení je oproti dříve zmíněným podstatně odlišné. Systém tohoto typu zavádí do předem definovaného prostředí takzvaného agenta, který prostředí pozoruje a koná úkony, při kterých dostává zpětnou vazbu v podobě pozitivní nebo negativní odměny [10].

2.2 Hluboké učení

Hluboké učení se specializuje na řešení velmi náročných úkonů strojového učení, které vyžadují téměř lidské porozumění světu. Hluboké učení je v dnešní době zásadní pro využití umělé inteligence v reálném světě, kde existuje spousta faktorů, které mohou pozorovaná data zásadně ovlivnit. Například při detekci objektů je

nutné docílit toho, aby byl objekt v obraze detekován bez ohledu na to, v jakém prostředí se nachází, jak je osvětlen nebo pod jakým úhlem je pozorován [9]. Jádrem dnešního hlubokého učení tvoří takzvané umělé neuronové sítě [10].

2.2.1 Umělé neuronové sítě

Umělé neuronové sítě jsou komplexní systémy inspirované architekturou biologických neuronů v mozku. Ve své podstatě se však funkcím a procesům v mozku přímo nepodobají a nemají je za cíl simulovat [9].

Umělé neuronové sítě se skládají z velkého množství jednodušších jednotek nazývaných neurony nebo uzly. Tyto jednotky jsou shlukovány do jednotlivých vrstev. Konkrétní uzly mohou napříč vrstvami přijímat výstupy od jiných uzlů nebo z externích zdrojů. Každý uzel dostupné informace zpracuje a vyhodnotí, přičemž může zůstat nečinný nebo se aktivovat a vygenerovat výstup. Toto propojení a kolektivní chování neuronů v síti umožňuje pomocí relativně jednoduchých mechanismů jednoho neuronu modelovat složité vztahy v rámci celé, komplexní sítě [11].

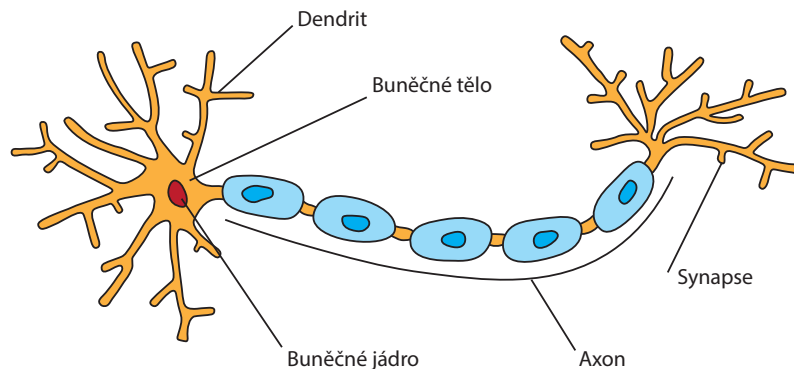
Umělé neuronové sítě se dají rozdělit na *dopředné*, kde se signál a aktivace šíří jedním směrem od vstupu k výstupu a *rekurentní*, které obsahují zpětné vazby a jsou tedy svou architekturou složitější. Umožňují však práci se sekvencemi proměnlivé délky a mají schopnost předpovídat následující události [10]. Umělé neuronové sítě se typicky sestávají ze vstupní vrstvy, jedné nebo více skrytých vrstev a vrstvy výstupní [12].

Biologický neuron

Biologické neurony jsou buňky nervového systému, které se nacházejí především v mozku. Jejich činnost spočívá v generování a zpracovávání krátkých elektrických impulsů – vzruchů. Pomocí systémů těchto buněk nervový systém řídí celý, například lidský, organismus [12].

Kromě buněčného těla, jak je ilustrováno na obrázku 2.1, se neurony sestávají z mnoho větvících se částí zvaných dendrity a části daleko delší než jejich tělo zvanou axon. Na konci se axon dělí na několik větví zakončených drobnými strukturami – synapsemi. Ty slouží k propojení s dendrity nebo buněčnými těly jiných neuronů a právě za jejich pomoci jsou propojeny do neuronových sítí [10].

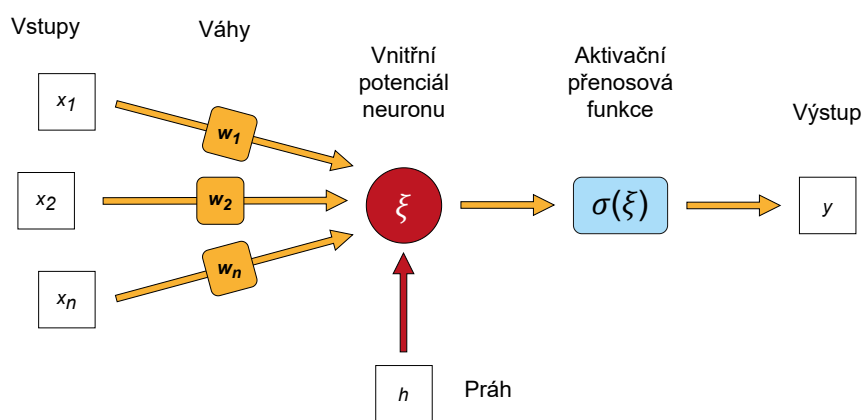
Z pohledu šíření vzruchů představují dendrity vstupy do těla neuronu a axony naopak výstup neuronu. Jednotlivé synapse pak mohou ještě jimi procházené signály utlumovat nebo budit [12].



Obr. 2.1: Biologický neuron

Umělý neuron

Umělý neuron, tedy obdoba toho biologického, tvořící jednotlivé části neuronových sítí, se dá popsat pomocí matematického modelu znázorněného na obrázku 2.2. Vstupy do jeho těla jsou značeny jako x a jsou váženy vahou w . Takto ohodnocené vstupy připomínají synapse biologického neuronu. Do těla ještě vstupuje konstantní hodnota nazývaná práh h . V těle se pak tvoří potenciál neuronu ξ a dále je tady blok aktivační přenosové funkce s a výstup y .



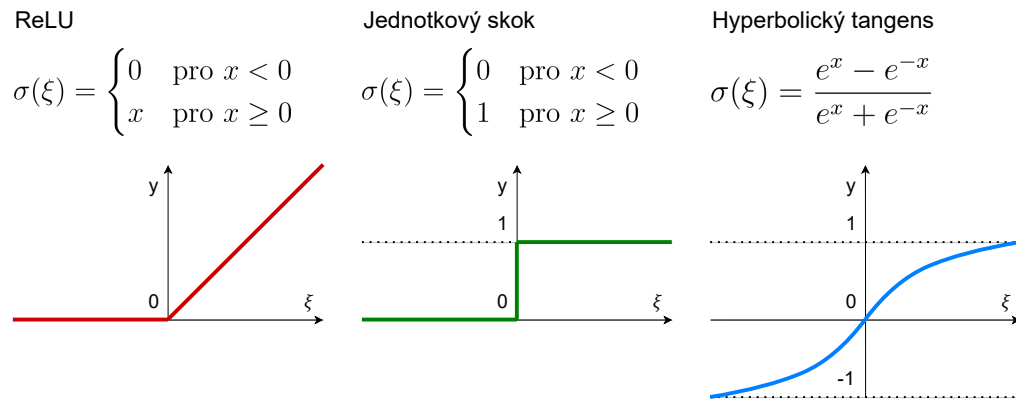
Obr. 2.2: Model umělého neuronu

Hodnoty vstupního vektoru o n prvcích, prahu a vah jsou obecně reálné. Například hodnoty vah mohou nabývat jak kladných, tak záporných hodnot, a tímto způsobem modelovat aktivační a inhibiční synapse.

Jednotlivé prvky vstupního vektoru jsou násobeny odpovídající vahou a následně se tyto součiny sčítají pro všechny prvky vstupního vektoru a od této celkové sumy se odečte hodnota prahu. Tento výsledek se nakonec stává argumentem aktivační funkce a dostáváme výsledek y . Celý vztah lze vyjádřit rovnicí

$$y = \sigma \left(\sum_{i=1}^n w_i x_i - h \right) \quad [12]. \quad (2.1)$$

Aktivační funkce σ je důležitou součástí umělého neuronu, a to hlavně v případě, že požadujeme po neuronové síti řešení komplexních situací. Vnitřní potenciál neuronu ξ je funkcí lineární a pokud bychom v tomhle stavu neurony řetězili a tvořili z nich síť, výstupem by byla opět lineární závislost. Aktivační funkce slouží k nelinearizaci potenciálu a přiblížení výstupu reálnému světu. To umožní výsledné neuronové síti aproximovat teoreticky jakoukoliv spojitou funkci [10, 12]. Na obrázku 2.3 jsou uvedeny příklady častých aktivačních funkcí umělého neuronu.



Obr. 2.3: Příklady častých aktivačních funkcí umělého neuronu

2.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě jsou speciálním druhem neuronových sítí, vznikly na základě vědeckých poznatků o zrakové části biologického mozku. Mnoho neuronů v této části mozku reaguje pouze na malé, specifické oblasti v zorném poli. Některé z těchto neuronů jsou specializovány na detekci horizontálních čar, zatímco jiné reagují na čáry s různými orientacemi. Kromě toho se v této struktuře nacházejí i neurony s větším receptivním polem, které reagují na složitější vzory. Receptivní pole jednotlivých neuronů mají možnost se překrývat, což umožňuje společné pokrytí celého zorného pole a detekci široké škály vizuálních vzorů.

Použití neuronových sítí s plně propojenými vrstvami by pro práci s velkými vstupy bylo příliš výpočetně náročné. Např. v situaci, kdy je vstupem obraz s rozměry 100×100 px (obsahuje 10 000 pixelů) a počáteční vrstva neuronové sítě má 1 000 neuronů, vznikne již pro tuto vrstvu celkem 10 milionů spojení. Konvoluční neuronové sítě tento problém řeší částečně sdílením parametrů a použitím ne zcela propojených vrstev [10].

Konvoluční neuronová síť se sestává z *konvolučních*, *sdužovacích* a *plně propojených* vrstev. Jednotlivé vrstvy mohou být poskládány různě za sebou, a tedy různě kombinovány. Plně propojené vrstvy jsou zpravidla umístěny pouze na konci sítě a jejich vstup je již značně zmenšený [13].

2.3.1 Konvoluční vrstva

Konvoluční vrstva je základní vrstvou konvoluční neuronové sítě. Na rozdíl od plně propojených vrstev jsou neurony v ní propojeny pouze s malou částí neuronů ve vrstvě předchozí, a to v závislosti na předdefinované velikosti receptivního pole jednotlivých neuronů [13]. Úkolem této vrstvy je ze vstupu za pomoci konvolučních filtrů, nazývaných také jako *příznakové detektory* (feature detectors), vygenerovat výstup v podobě takzvaných *příznakových map* (feature maps) [13, 14].

Pokud je vstupem barevný obraz, kde každý pixel obsahuje tři hodnoty (červená, zelená, modrá) nese každý bod v obraze informace nejen o své prostorové poloze (šířce a výšce), ale také o hloubce spojené s uvedenými barevnými kanály. Tento trojrozměrný vstup je následně zpracován prostřednictvím konvolučních filtrů.

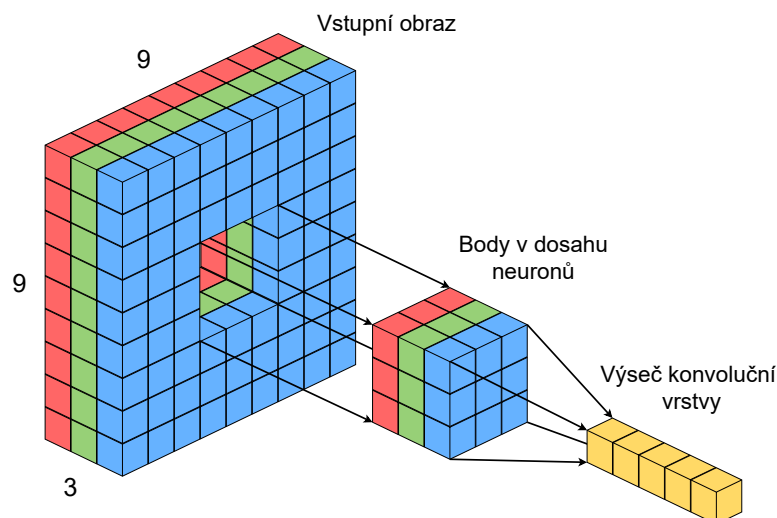
Konvoluční filtry jsou v případě konvoluční vrstvy složeny z jednotlivých spojení neuronů a z příslušných vah a prahu. Každý filtr slouží k detekci určitého vzoru v obraze, nebo-li příznaku. Váhy pak určují, jak moc jsou filtry na dané příznaky citlivé. Tyto filtry jsou aplikovány postupně na celý vstup vrstvy, což představuje proces konvoluce [13].

Aplikací jednotlivých filtrů pomocí konvoluce vznikají mapy příznaků. Na každé z těchto map jsou zvýrazněna místa, kde byly identifikovány příznaky ve vstupních datech pomocí konkrétního filtru [10].

Ilustrativní příklad první konvoluční vrstvy je uveden na obrázku 2.4. Zde je vstupní obraz disponující třemi barevnými kanály s velikostí $9 \times 9 \times 3$, žlutě je pak vyznačena výše konvoluční vrstvy. Jak je z obrázku vidět, každý neuron v konvoluční vrstvě je z hlediska prostorového rozložení propojen pouze se vstupními body v jeho dosahu, pokrývá však celou hloubku vstupu, a tedy všechny tři barevné kanály. Konvoluční vrstva je několik neuronů hluboká, vyznačené neurony mají všechny stejný dosah ve vstupní oblasti a v tomto případě je každý z nich součástí jednoho z pěti konvolučních filtrů [14].

Architektura konvoluční vrstvy

Počet neuronů v konvoluční vrstvě je určen třemi klíčovými parametry, které se také uvádí pod pojmem „hyperparametry“: *hloubkou*, *velikostí kroku* a *velikostí nulové výplně*. Hloubka vrstvy závisí na počtu požadovaných filtrů. Velikost kroku určuje, jak se filtr při konvoluční operaci pohybuje po vstupu. Například při velikosti kroku 1



Obr. 2.4: Příklad první konvoluční vrstvy

se filtr pohybuje pixel po pixelu, při hodnotě 2 dochází k pohybu ob dva pixely. Vyšší hodnota velikosti kroku vede k menšímu počtu neuronů. Hodnota nulové výplně určuje, o kolik nul se má vstup kolem hranice zvětšit. Takto uměle zvětšení vstupu je vhodné zejména v případě, že chceme docílit, aby výstup konvoluční vrstvy měl stejný rozměr jako vstup.

Velikost konvoluční vrstvy X , a tedy počet neuronů v ní, lze vypočítat z velikosti vstupní vrstvy W a za pomoci použitých hyperparametrů z následujícího vztahu:

$$X = \frac{(W - F + 2P)}{S} + 1, \quad (2.2)$$

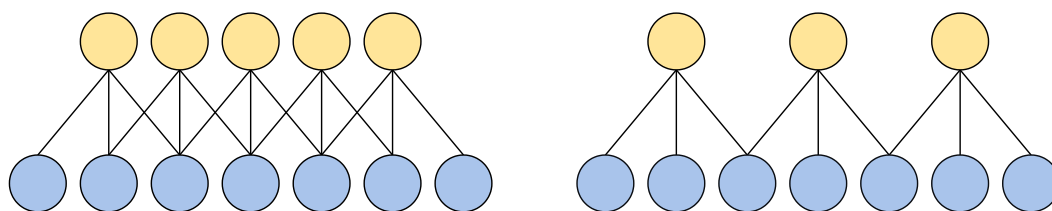
kde F je velikost receptivního pole (filtru), S značí velikost kroku a P je velikost nulové výplně. Pokud máme vstupní vrstvu o velikosti 7×7 , velikost filtru je 3×3 , velikost kroku 1 a nulová výplň je 0, pak bude mít konvoluční vrstva rozměry 5×5 neuronů. Důležité je, aby výstupní hodnota X byla celé číslo.

Příklad prostorového uspořádání neuronů je uveden na obrázku 2.5, pro zjednodušení má vstupní vrstva pouze jeden rozměr. Hodnoty hyperparametrů jsou $F = 3$, $W = 5$, $P = 1$. Na obrázku vlevo je pak velikost kroku S nastavena na hodnotu 1 a vpravo na hodnotu 2. Při $S = 2$ se počet neuronů zmenší z 5 na 3 [14].

Konvoluce

Jak již bylo zmíněno, proces, jakým dostat ze vstupní vrstvy skrz konvoluční vrstvu výstup, zahrnuje matematickou operaci zvanou konvoluce [13].

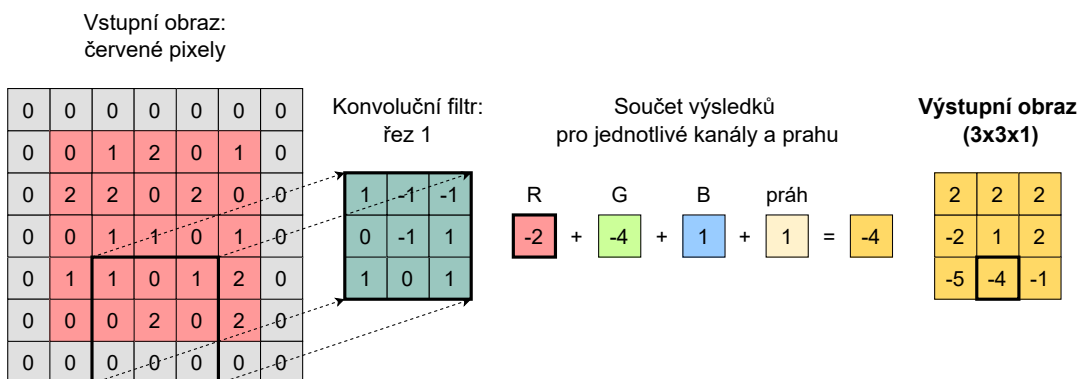
Vzhledem k obtížnosti popisu konvoluce v případě trojrozměrného vstupu lze vstup rozčlenit do dvourozměrných řezů. Například pokud je vstupem obraz s třemi



Obr. 2.5: Příklad prostorového uspořádání neuronů konvoluční vrstvy

barevnými kanály, a tedy je tři hodnoty široký, je tento vstup rozdělen do tří dvou-
 rozměrných řezů. Provedením konvoluce za pomoci filtru na každém z těchto řezů
 vyústí v dvourozměrný výstup. Tyto výstupy lze poté opět zkombinovat naskládá-
 ním na sebe do jednoho, trojrozměrného výstupu.

Proces je vizualizován na obrázku 2.6 pro jednu z výstupních hodnot s jedním ře-
 zem vstupního obrazu symbolizující červený kanál. Vstupní obraz o třech barevných
 kanálech R, G a B má v tomto případě rozměr $5 \times 5 \times 3$ obrazových bodů. Zvolené
 hyperparametry konvoluční vrstvy jsou $F = 3$, $S = 2$, $P = 1$. Počet filtrů K je 1
 a vstup je nadstaven nulovou výplní. Samotný konvoluční filtr má rozměry $3 \times 3 \times 1$.
 Uvedené parametry znamenají, že výstup bude mít rozměry $3 \times 3 \times 1$. Výpočet pro
 každý prvek výstupního obrazu zahrnuje vynásobení příslušných hodnot v jednotli-
 vých řezech vstupu s jednotlivými řezy filtru. Výsledné hodnoty těchto součinů jsou
 sečteny spolu s hodnotou prahu, který je u filtru nastaven na 1.



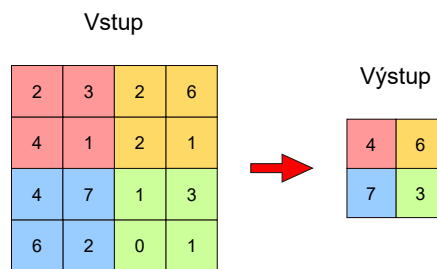
Obr. 2.6: Příklad konvoluce

2.3.2 Sdružovací vrstva

Sdružovací vrstva (pooling layer) má za úkol zmenšit výšku a šířku jejího vstupu,
 hloubku vstupu však neovlivní. Snížení velikosti na jednu stranu vede ke ztrátě méně
 podstatných detailů, na druhou stranu významně redukuje výpočetní náročnosti

následujících vrstev. Nejčastěji probíhá sdružování na základě průměrování hodnot (average pooling) nebo na základě vybírání maximálních hodnot (max pooling) [15].

V praxi se více uplatňuje sdružování na základě maximálních hodnot [14], pomáhá totiž zachování těch nejvýznamnějších rysů ve vstupním obraze. Podobně jako konvoluční vrstva, má i sdružovací vrstva hyperparametry podobné s konvoluční vrstvou. Konkrétně se jedná o velikost kroku S a velikost receptivního pole F . Nejčastějším případem jsou filtry o velikosti 2×2 , které se uplatňují postupně s velikostí kroku $S = 2$ a velikostí receptivního pole $F = 2$. Tato sdružovací vrstva zmenší každý řez trojrozměrného vstupu na polovinu a výsledný trojrozměrný výstup bude rozměrově tvořit pouhých 25 % vstupu. Aplikace tohoto konkrétního filtru je znázorněna na obrázku 2.7, a to na vstupní řez o velikosti 4×4 .



Obr. 2.7: Demonstrace funkce sdružovací vrstvy

2.3.3 Modely detekce objektů

Speciálním případem konvolučních neuronových sítí jsou architektury detekce objektů v obraze. Jejich vstupem je snímek a výstupem mimo jiné pozice jednotlivých detekovaných objektů, jejich typ a jistota jejich určení. Příkladem jsou modely z rodiny EfficientDet.

EfficientDet

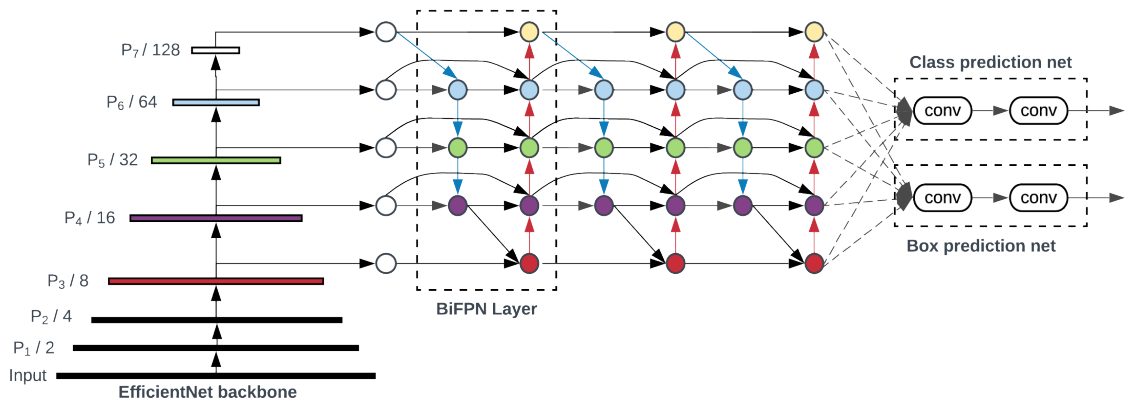
EfficientDet je skupina modelů umělých neuronových sítí vyvinutá společností Google v roce 2020. Byly navrženy pro efektivní a přesnou detekci objektů v obraze a patří mezi jedny z nejmodernějších. Jejich architektura je veřejně dostupná.

Páteří modelu je model EfficientNet, který společnost Google vyvinula již v roce 2019 s úmyslem zefektivnit trénování neuronových sítí. Slouží k extrakci obecných příznaků ze vstupních obrázků.

Obecná architektura modelů ze skupiny EfficientDet je znázorněna na obrázku 2.8. Výstupní příznakové mapy z páteřní sítě jsou v něm označeny od vstupní vrstvy

vzestupně P_1 až P_7 . Architektura modelu implementuje váženou dvousměrnou pyramidovou síť příznaků (BiFPN), která extrahuje příznaky z úrovní P_3 až P_7 základní sítě a následně je opakovaně propojuje jak směrem nahoru, tak směrem dolů. Z těchto finálních map příznaků model pomocí sítě pro klasifikaci a lokalizaci generuje výsledné informace o třídách objektů a jejich umístění v samotném vstupním obraze. Síť BiFPN a sítě pro klasifikaci a lokalizaci mohou být opakovaně použity, a to v závislosti na konkrétním modelu.

Jednotlivé modely z řady EfficientDet jsou označeny od D0 do D7 a každý z nich má různé vstupní rozměry obrazu a počet BiFPN vrstev. EfficientDet D0 pracuje s nejmenšími vstupními rozměry obrazu 512×512 px a obsahuje tři vrstvy BiFPN, což z něj činí nejméně výpočetně náročnou variantu. Naopak D7, s rozměry 1536×1536 px, je nejnáročnější a obsahuje osm vrstev BiFPN [16].

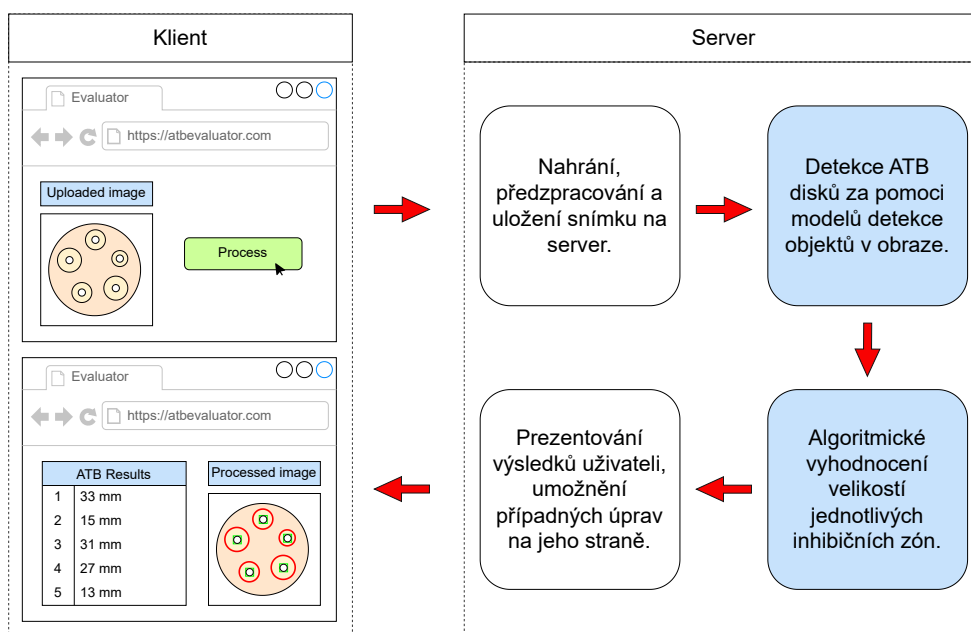


Obr. 2.8: Architektura modelů EfficientDet, převzato z [16].

3 Návrh řešení

Záměrem této práce je vytvořit modul do webové aplikace společnosti Bruker Daltonics GmbH & Co. KG, který co nejvíce zautomatizuje proces vyhodnocení diskových difúzních testů a bude vhodná pro použití v laboratořích. Díky webové implementaci bude umožněno fungování výsledného řešení nezávisle na cílové platformě a koncovém zařízení. Implementací postupů strojového učení a technik zpracování obrazu bude výsledná aplikace schopna analyzovat snímky diskové difúzní metody a minimalizovat v tomto procesu potřebu lidského faktoru a nevýhody, které se s tím pojí.

Základní myšlenka řešení je znázorněna na obrázku 3.1. Modul umožní pracovníkovi laboratoře nahrát snímky pro vyhodnocení na server. Server snímky zpracuje a zobrazí laborantovi výsledky o efektivitě jednotlivých antibiotik.



Obr. 3.1: Princip aplikace

Jelikož koncept diskové difúzní metody spočívá v přesném měření velikosti inhibiční zóny, která se tvoří v okolí jednotlivých disků s antibiotiky, je celý proces rozdělen do dvou hlavních částí. Nejprve je nutné jednotlivé disky ve snímku co nejpresněji lokalizovat a poté, za pomoci dalšího postupu, určit velikost jednotlivých inhibičních zón. Tyto klíčové úlohy serveru jsou na obrázku 3.1 vyznačeny modře.

V průběhu řešení je nutné dbát na takové postupy, aby bylo umožněno zpracování co nejširší množiny snímků, které jsou pořízeny za nejrůznorodějších akvizičních podmínek pomocí různých zařízení. Tento postup, umožňující zpracování snímků

pořízených jak profesionálním laboratorním vybavením, tak mobilními zařízeními, povede k širšímu využití tohoto řešení v laboratořích.

3.1 Detekce antibiotických disků

Jednotlivé antibiotické disky jsou zejména kruhového tvaru, avšak jejich podoba se liší v závislosti na různých akvizičních podmínkách. To lze pozorovat na obrázku 3.2, kde v horní řadě jsou snímky pořízené za standardizovaných podmínek s pomocí řešení MBT Pathfinder® vyvíjeného společností Bruker Daltonics GmbH & Co. KG, v dolní řadě jsou pak snímky, které byly pořízeny mobilními telefony. Z jednotlivých snímků jsou vidět zásadní rozdíly.

Na snímky pořízených mobilním telefonem má totiž dopad větší počet proměnlivých okolností. To je dáno i způsobem, jakým jsou pořizovány. Mnohdy se liší úhel, pod jakým jsou pořízeny, osvětlení, ostrost a další parametry. Jelikož je úkolem s co největší přesností detekovat antibiotické disky i ve výše uvedených snímcích, byla pro detekci disků zvolena aplikace vlastního modelu konvoluční neuronové sítě.

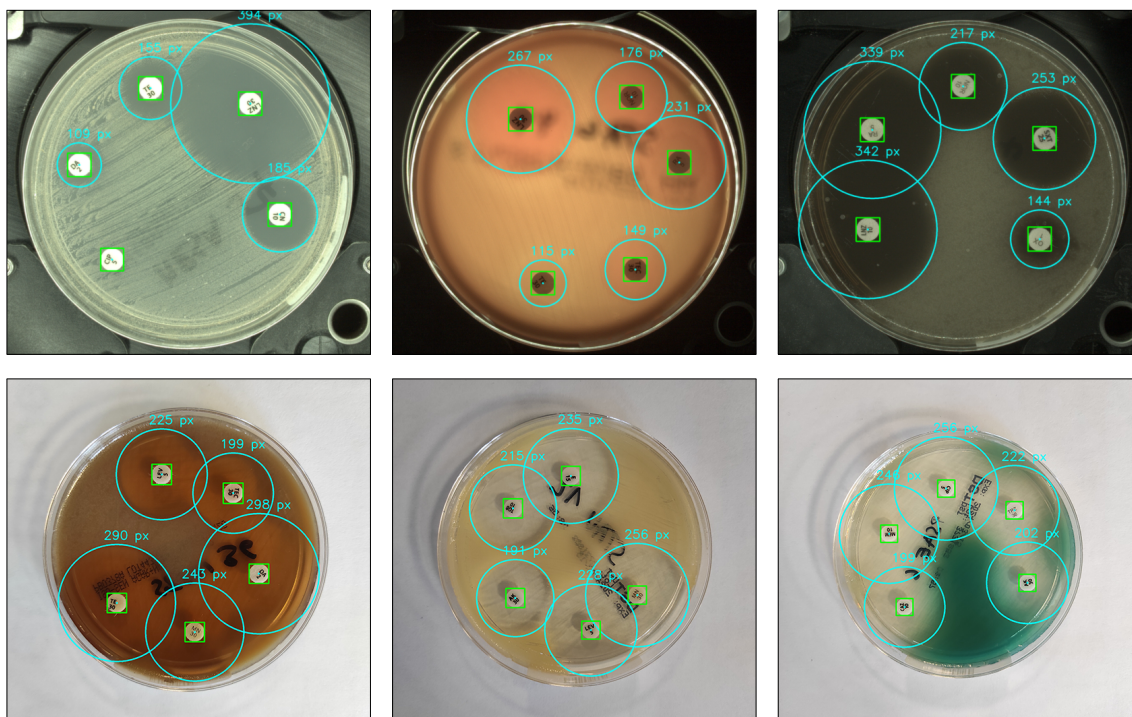
Přenosové učení

Jelikož je trénování velkých konvolučních neuronových sítí s velkým požadavkem na přesnost velice časově a finančně náročné, v praxi se uplatňuje technika tzv. přenosového učení. To znamená, že se použije na velké datové sadě natrénovaný model a následně se dalším trénováním adaptuje na požadovanou aplikaci. Jeho koncept čerpá z teorie přenosu v pedagogické psychologii. Vyplývá z ní, že lidé se efektivněji učí nové dovednosti, pokud je mohou stavět na předchozích zkušenostech. S využitím těchto poznatků je vybrán model, který je zaměřen na rozpoznávání běžných obrazových příznaků vyskytujících se ve všech obrazech. Ten se za pomoci přenosového učení a dalšího trénování dále specializuje na rozpoznávání příznaků potřebných pro požadovanou aplikaci, jako je v tomto případě detekce antibiotických disků [10, 17].

3.1.1 Příprava trénovacích dat

Pro natrénování vlastní architektury na detekci antibiotických disků je zapotřebí připravit odpovídající datovou sadu. To zahrnuje nashromáždění co největšího množství snímků s objekty, které budou předmětem detekce a každý takový snímek této sady, podle techniky *učení s učitelem*, „anotovat“.

Anotace v rámci detekčních modelů je proces, kdy je ve snímku určena poloha a typ hledaného objektu. Nejčastěji se objekty označují obdélníkovými rámečky (bounding box). Pro dobře a přesně fungující model je důležitá přesnost tohoto



Obr. 3.2: Příkladně zpracovaných snímků: snímky nahoře pořízené profesionálním laboratorním vybavením (MBT Pathfinder®), dole mobilními telefony.

vyznačení. Samotná anotace je zdlouhavý a nákladný úkon. Vyžaduje přítomnost lidského operátora, který ručně prochází jeden snímek za druhým a pečlivě vyznačuje všechny objekty s přesností na úrovni jednotlivých pixelů.

V rámci přípravy vstupních dat pro trénování modelu na detekci disků ve snímcích pro diskovou difúzní metodu byl firmou Bruker Daltonics GmbH & Co. KG poskytnuta datová sada čítající přes 3 200 snímků. Výhodou této datové sady pak je, že obsahuje snímky získané širokou škálou zařízení od těch profesionálních po ty mobilní. V jednotlivých snímcích pak byly ručně vyznačeny disky, které jsou na obrázku 3.2 vyznačeny zeleným ohraničením, a to za pomoci bezplatné webové aplikace makesense.ai [18]. Ukázka anotace pomocí makesense.ai je uvedena na obrázku 3.3. Výstupem anotace v této aplikaci mohou být různé formáty souboru. Nejvhodnějším pro náš případ, kdy je dat k anotaci opravdu mnoho, je vytvoření XML souboru se souřadnicemi pro každý jeden snímek datové sady. Díky dvojicím snímek/XML soubor je možné získat lepší pojem o tom, který snímek již anotován byl.

Práce s makesense.ai

Anotační software makesense.ai je volně dostupný, ale zdaleka neintegruje všechny postupy, které umožňují proces anotace urychlit. V tomto případě je dost zásadní absence jakýchkoliv klávesových zkratk. Pomocí vytvořeného skriptu v jazyce Ja-



Obr. 3.3: Anotace antibiotických disků za pomoci webové aplikace makesense.ai dostupné online.

JavaScript, který se dá nahrát přímo do konzole prohlížeče, byl tento problém elegantně vyřešen. Skript mapuje například klávesu W na přepínání mezi anotačním a pohybovým módem kurzoru, zmíněná část skriptu je zobrazena ve výpisu 3.1. Skript využívá metody `document.addEventListener`, a tedy naslouchá na událost stisku klávesy, která je dále v kódu specifikována a ověřována. Obdobně vytvořený skript naslouchá na klávesy jako A a D pro přepínání mezi předchozím a následujícím snímkem pro anotaci. Toto „banální“ ulehčení v podobě mapování příslušných kláves pomocí skriptu ušetřilo velké množství času. A to hlavně v tomto případě, kdy celý proces anotace zabral desítky hodin čistého času.

```
1 document.addEventListener('keydown', function(event) {
2     // Check if the pressed key is 'w'
3     if (event.key === 'W' || event.key === 'w') {
4         // Find the button element using XPath
5         var buttonElement = document.evaluate(
6             "/html/body/div/div/div/div[2]/div[2]/div[1]/div[2]/div[1]/div",
7             document, null, XPathResult.FIRST_ORDERED_NODE_TYPE, null
8         ).singleNodeValue;
9         // Check if the button element is found, click
10        if (buttonElement) {
11            buttonElement.click();
12        } else {
13            console.error('Annotate/move button element not found.');
```

Výpis 3.1: Část ze skriptu v jazyce JavaScript pro zajištění podpory klávesových zkratk a urychlení anotace ve webové aplikaci `makesense.ai`.

Rozdělení datové sady

Anotovanou datovou sadu je pro proces trénování vhodné rozdělit do tří částí. Trénovací část tvoří většinu, na které se model postupně učí a zdokonaluje. Validací skupina snímků je také použita v průběhu trénování, ale pro průběžné ověření efektivity modelu během trénování. Třetí – testovací část datové sady není použita při trénování a od celého procesu je oddělena. Je použita pro ověření efektivity až po finálním natrénování modelu. Díky tomu, že model tyto snímky ještě „nespatřil“, si můžeme být jistí, že účinnost modelu a jeho schopnost generalizace bude po ověření na testovací části datové sady objektivní.

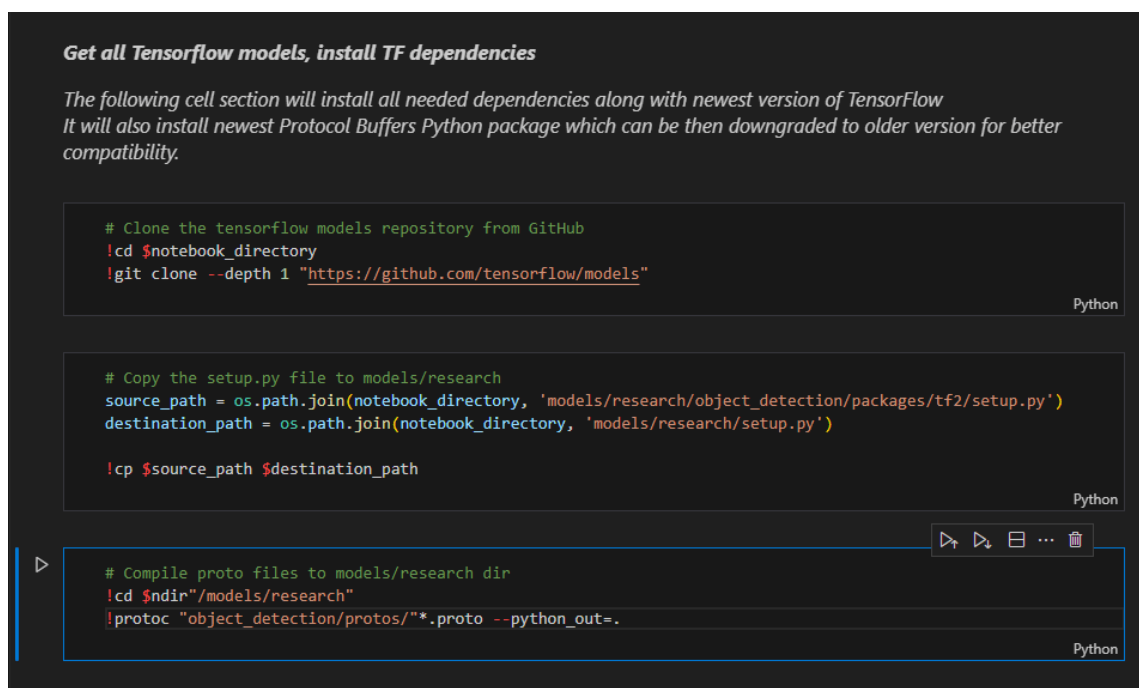
3.1.2 Příprava modelů a trénovacího prostředí

V oblasti umělé inteligence a strojového učení je nejčastěji využíván programovací jazyk Python. Pro práci s umělými neuronovými sítěmi jsou k dispozici speciální vý-

vojové platformy. Mezi nejpopulárnější z nich patří PyTorch, kterou vyvíjí společnost Meta, a TensorFlow [19], která patří pod společnost Google. Obě tyto platformy jsou otevřeným a zdarma dostupným softwarem.

Velmi používaným v kontextu datové vědy a umělé inteligence je projekt Jupyter [20]. Jedná se o otevřenou a zdarma dostupnou platformu, která je vyvíjena za účelem interaktivního programování v různých programovacích jazycích. Implementuje otevřený formát dokumentu Jupyter Notebook a webovou aplikaci umožňující jednotlivé dokumenty, neboli sešity, vytvářet a sdílet. Interaktivita této platformy spočívá v tom, že je v rámci Jupyter sešitu možné spojovat jak kód v různých programovacích jazycích, tak například i text, rovnice, obrázky a další prvky současně.

Jednotlivé Jupyter sešity jsou rozděleny na tzv. „buňky“, každá buňka obsahuje kód jednoho programovacího jazyka, který může být obohacen např. o systémové příkazy. Tato skutečnost je ilustrována na obrázku 3.4, který zobrazuje část Jupyter sešitu otevřeného v editoru Visual Studio Code. Jak je možné vidět, jednotlivé systémové příkazy operačního systému Linux jsou zde uvozovány vykřičníkem. Velice užitečné je také to, že proměnné z jazyka Python je možné přenášet do systémového prostředí pomocí znaku dolaru (a naopak). Jednotlivé buňky lze samostatně nebo jednu po druhé spouštět a použité proměnné jsou uloženy v kontextu celého Jupyter sešitu, což usnadňuje jejich použití v kterékoliv buňce.



```
Get all Tensorflow models, install TF dependencies  
  
The following cell section will install all needed dependencies along with newest version of TensorFlow  
It will also install newest Protocol Buffers Python package which can be then downgraded to older version for better compatibility.  
  
# Clone the tensorflow models repository from GitHub  
!cd $notebook_directory  
!git clone --depth 1 "https://github.com/tensorflow/models"  
  
# Copy the setup.py file to models/research  
source_path = os.path.join(notebook_directory, 'models/research/object_detection/packages/tf2/setup.py')  
destination_path = os.path.join(notebook_directory, 'models/research/setup.py')  
  
!cp $source_path $destination_path  
  
# Compile proto files to models/research dir  
!cd $ndir"/models/research"  
!protoc "object_detection/protos/"*.proto --python_out=.
```

Obr. 3.4: Ukázka sešitu Jupyter obsahující 4 buňky: první buňka je tvořena textem formátovaným pomocí jazyka Markdown, ostatní buňky obsahují kód v jazyce Python doplněný o systémové příkazy.

3.2 Měření velikosti inhibičních zón

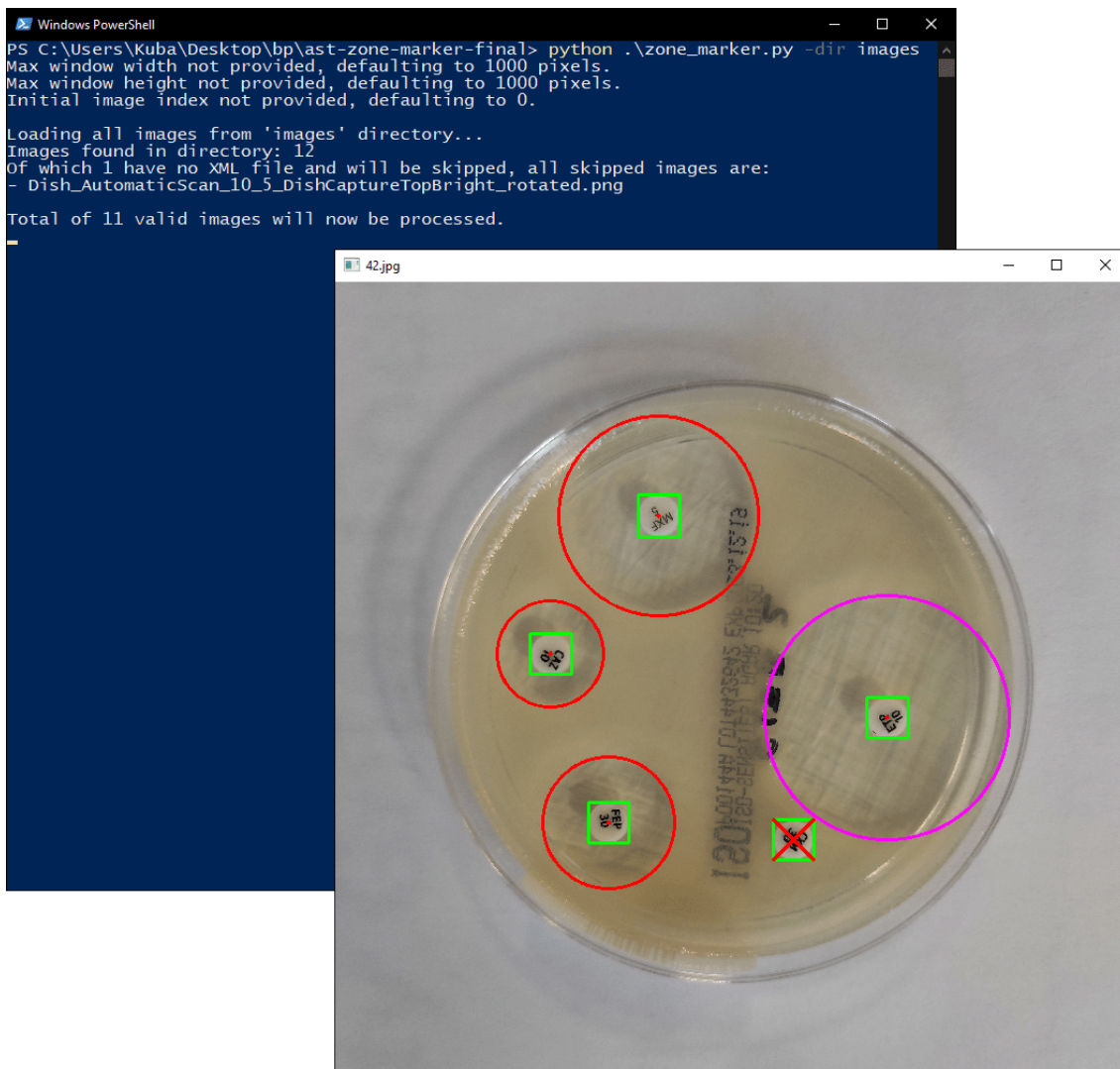
Další část metodiky se zabývá návrhem algoritmu zaměřeném na vyhodnocení velikostí jednotlivých inhibičních zón diskové difúzní metody. Hlavním záměrem je určení poloměrů těchto zón vygenerovaných kolem jednotlivých antibiotických disků. K tomuto poslouží různé algoritmy, včetně těch z knihovny OpenCV [21].

Požadované výsledky tohoto experimentu jsou opět ilustrovány na obrázku 3.2. Tentokrát se jedná o inhibiční zóny, které jsou ohraničené kružnicemi azurové barvy. Nad každou kružnicí je umístěn popisek s velikostí jejího poloměru v pixelech.

Je důležité poznamenat, že pro pozdější hodnocení efektivity vytvořeného algoritmu a pro proces jeho ladění, je nutné po pozici všech antibiotických disků anotovat také velikost poloměrů jednotlivých inhibičních zón.

Pro tento účel byl za pomoci programovacího jazyku Python a v něm podporované knihovny OpenCV vytvořen jednoduchý program. Při jeho spuštění s parametrem `-dir`, tak jako je znázorněno na obrázku 3.5, uživatel specifikuje cestu k adresáři obsahujícímu snímky pro anotaci. Parametry `-mw` a `-mh` umožňují definovat maximální rozměry zobrazovaného okna se snímkem a počáteční index snímku může být nastaven parametrem `-id`. Povinným je však jen parametr `-dir`.

Program umožňuje uživateli procházet jednotlivé snímky ze zvoleného adresáře. V případě potřeby je lze přepínat klávesami A a D. Pro každý snímek program načítá příslušné boxy z XML souboru a uživatel pomocí kurzoru myši určuje poloměr inhibiční zóny kolem středu každého boxu. Aktuální poloměr je vykreslován v reálném čase purpurovou barvou. Po kliknutí levým tlačítkem myši se předchozí purpurový kruh změní na červený a vyznačený poloměr se uloží do souboru ve formátu JSON. Uživatel následně přechází k boxu dalšímu. Pokud daný disk nevygeneroval inhibiční zónu, uživatel pomocí přesunutí kurzoru myši dovnitř označujícího boxu inhibiční zónu „zruší“ a uvnitř daného boxu se vykreslí křížek. Vše zmíněné je k vidění na obrátku 3.5. K odstranění předchozího poloměru slouží pravé tlačítko myši. Po dokončení anotace jednoho, nebo vícero snímků z adresáře, může uživatel anotované snímky procházet a případně libovolný poloměr pravým tlačítkem myši vybrat (aktuálně vybraná inhibiční zóna se vykresluje azurovou barvou). Po stisku levého tlačítka myši takto vybraný poloměr může znovu upravit.



Obr. 3.5: Anotace inhibičních zón za pomoci vytvořeného programu.

4 Implementace

Implementace řešení zahrnuje natrénování modelu pro detekci antibiotických disků a následný vývoj algoritmu pro vyhodnocení velikostí jednotlivých inhibičních zón. Vyhodnocení velikostí zón vychází z předpokladu, že je pozice antibiotického disku již známá, proto je vhodné implementovat tyto části právě v tomto pořadí.

4.1 Trénování detekčních modelů

Pro trénování rozsáhlých modelů konvolučních neuronových sítí se často využívá hardwarové akcelerace pomocí grafických karet. Jejich výhodou je hlavně množství grafické paměti. Paměť grafické karty slouží k načtení celého modelu a ukládání nejen samotných trénovacích a validačních dat, ale také výpočetně náročných mezivýsledků trénovacího procesu. Zvláště při použití složitých modelů s vysokým počtem parametrů je schopnost udržet velké množství informací v paměti klíčová pro úspěšné a efektivní trénování. Pro podporu grafického procesoru je nezbytné upravit prostředí platformy TensorFlow tak, aby bylo kompatibilní s přidaným grafickým procesorem. Tento proces zahrnuje instalaci klíčových knihoven, jako jsou Nvidia CUDA SDK a cuDNN, a to v přesně daných a odpovídajících verzích.

Pro náš postup byly použity předtrénované modely pro detekci objektů z GitHub repozitáře TensorFlow 2 Detection Model Zoo [22]. Pro trénování byla využita serverová infrastruktura společnosti Bruker Daltonics GmbH & Co. KG disponujícími špičkovými grafickými kartami NVIDIA RTX 4090 s 24 GB grafické paměti.

4.1.1 Konfigurace

Každý trénovaný model je potřeba nakonfigurovat za pomoci hyperparametrů, které ovlivňují samotný proces trénování. Hlavními hyperparametry jsou *velikost dávky*, *rychlost učení*, *počet zahřívacích cyklů* a *zahřívací rychlost*. V praxi jsou hodnoty těchto hyperparametrů často experimentálně naladěny na základě výkonu modelu v průběhu trénování. Optimalizace těchto parametrů může mít významný dopad na rychlost trénování a kvalitu naučeného modelu.

Velikost dávky (batch size) určuje, kolik snímků je zpracováno současně během jednoho trénovacího cyklu. Zvětšení velikosti dávky může urychlit trénování na moderních GPU, avšak vyžaduje větší paměťovou kapacitu. Naopak menší velikosti dávky mohou způsobit větší variabilitu při aktualizacích vah, což může přispět k lepšímu výkonu modelu tím, že umožní modelu citlivěji reagovat na různorodost trénovacích vzorků a snižovat riziko přeučení na specifické vzory dat. Přeučení nastává

v situaci, kdy se model až příliš adaptuje na trénovací data a sníží se jeho schopnost generalizace na datech nových.

Rychlost učení (learning rate) definuje, jak rychle by se váhy modelu při trénování měly měnit. Když je tato rychlost nastavena příliš vysoko, může to vést k nekontrolovaným výkyvům nebo přeučení modelu. Pokud je nastavena příliš nízko, může trénování probíhat pomalu a výsledné váhy se mohou zastavit v nějakém lokálním minimu a nedosáhnout optimálních výsledků.

Zahřívací rychlost (warm-up learning rate) určuje rychlost učení během počátečních iterací (kroků) trénování. Touto technikou dochází k postupnému zvyšování rychlosti učení na počátku trénování, což může pomoci stabilizovat a zrychlit konvergenci modelu. *Počet zahřívacích cyklů* (warm-up steps) je pak hodnota cyklů na počátku trénování, ve kterých tento proces probíhá zahřívací rychlostí.

Při trénování s Tensorflow je zapotřebí tyto parametry nastavit v konfiguračním souboru ve formátu protobuf. Jako referenční posloužily hodnoty z výchozích konfiguračních souborů, které byly získány společně s předtrénovanými modely z TensorFlow 2 Detection Model Zoo. Např. původní konfigurační soubor pro model EfficientDet D2 obsahoval velikost dávky 128, rychlost učení 0,08 a zahřívací rychlost 0,001. Samotné trénování se však na grafické kartě s 24 GB grafické paměti podařilo spustit maximálně s velikostí dávky 8, pro větší velikost dávky karta neměla dostatek paměti. Podle poměru původní a nové velikosti dávky jsme následně také upravili parametry rychlosti trénování. Jak je patrné z ukázky konfiguračního souboru ve výpisu 4.1, rychlost učení byla nastavena na hodnotu $4e-4$ a pro zahřívací rychlost byla zvolena hodnota $2.6e-5$.

V konfiguračním souboru je nutné také specifikovat cesty souborů potřebných pro trénování. Prvním je soubor specifikující názvy detekovaných objektů, v tomto případě je pouze jeden – nazvaný „sample“. Dále je potřeba připravit seznam anotovaných snímků, který je převeden ze souborů formátu VOC XML do separátních CSV souborů pro trénovací a validační data. Pro tento účel využijeme skript v jazyce Python. Tyto CSV soubory jsou následně použity k vytvoření souborů ve formátu `.tfrecord`, což je formát platformy TensorFlow, který je pro tento účel nezbytný.

4.1.2 Trénování

Pro trénování jsme zvolili dva modely z rodiny EfficientDet s různými velikostmi vstupních obrazů: D0 s 512×512 px a D2 s 768×768 px. Tyto modely se jeví jako nejvhodnější z dostupných v TensorFlow 2 Detection Model Zoo [22]. Dosahují dobrého poměru mezi výpočetní náročností a přesností detekce. V repozitáři modelů jsou uvedeny i přesnější architektury, které jsou však podle uvedených metrik mnohem výpočetně náročnější. Pro lepší představu o tomto kompromisu mezi přesností

```

1 # SSD with EfficientNet-b2 + BiFPN feature extractor,
2 # shared box predictor and focal loss (a.k.a EfficientDet-d2).
...
140 train_config {
...
144   batch_size: 8
...
161   optimizer {
162     momentum_optimizer {
163       learning_rate {
164         cosine_decay_learning_rate {
165           learning_rate_base: 4e-4
166           total_steps: 300000
167           warmup_learning_rate: 2.6e-5
168           warmup_steps: 2500
169         }
170       }
171       momentum_optimizer_value: 0.9
172     }
173     use_moving_average: false
174   }
...
199 }

```

Výpis 4.1: Ukázka části konfiguračního souboru ve formátu protobuf s nastavením hyperparametrů použitým pro trénování modelu EfficientDet D2.

a náročností jsme do trénovaných modelů zařadili i Mask R-CNN Inception ResNet V2 (dále jen Mask R-CNN), který má z výše uvedených i největší velikost vstupu, a to 1024×1024 px. Pro každý model detekce objektů byl vytvořen samostatný Jupyter sešit, trénování na serveru probíhalo s využitím operačním systémem Ubuntu 22.04 LTS.

Trénování je spuštěno pomocí skriptu napsaného v jazyce Python z repozitáře TensorFlow 2 Detection Model Zoo [22]. Jak je vidět z výpisu 4.2, který obsahuje kód z buňky z jednoho z trénovacích sešitů, s využitím parametrů je nutné specifikovat výše popsaný konfigurační soubor (parametr `--pipeline_config_path`), následně také uvést cestu k samotnému předtrénovanému modelu (parametr `--model_dir`).

```

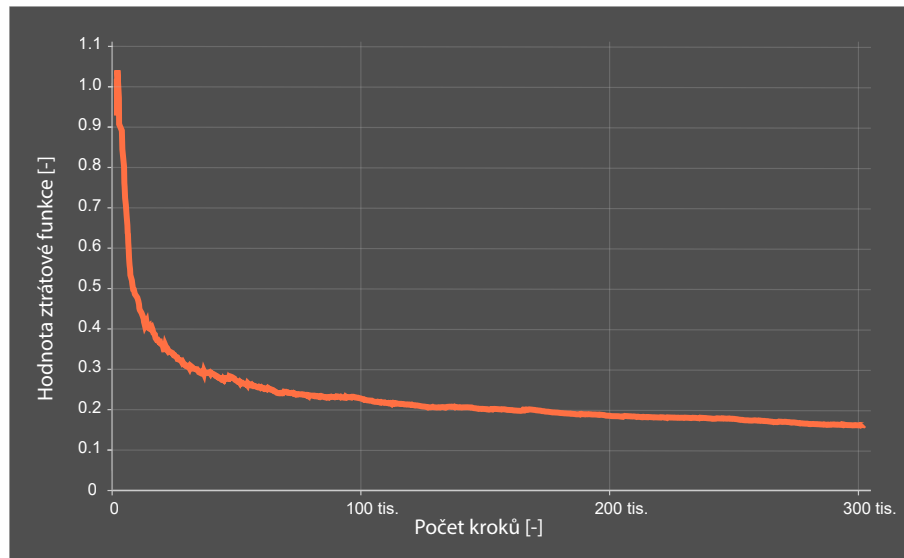
1 # Run training!
2 !python3 $notebook_directory"models/research/object_detection/model_main_tf2.
   ↪ py" \
3   --pipeline_config_path={pipeline_file} \
4   --model_dir={model_dir} \
5   --alsologtostderr

```

Výpis 4.2: Kód z buňky sešitu Jupyter pro spuštění trénovacího skriptu.

Se spuštěním trénování a využitím nástroje TensorBoard je možné v reálném čase pozorovat řadu metrik. Jedna z nejdůležitějších je „total_loss“, neboli hodnota celkové ztrátové funkce. Hodnota ztrátové funkce vyjadřuje průběžný rozdíl mezi

predikcemi modelu a skutečnými hodnotami na validačních snímcích. Celková ztráta pak zahrnuje hodnoty ztrátových funkcí na všech validačních snímcích datasetu, a tedy určuje, zdali se model správně učí a zdokonaluje ve svých predikcích. V grafu 4.1 je vynesena závislost hodnoty této funkce na počtu kroků při trénování modelu EfficientDet D2.



Obr. 4.1: Závislost hodnoty ztrátové funkce na počtu kroků (iterací) z nástroje TensorBoard, vyhlazeno.

Z počátku trénování jsou její hodnoty vyšší a prudce klesají, rychlost poklesu klesá a s ní taky i efektivita učení modelu. Trénování modelu je vhodné rovněž včas ukončit, aby nedošlo k přetrénování (přeučení).

Výstupem trénování vlastního modelu pomocí výše uvedeného postupu vzniknou tzv. „checkpointy“, čili soubory obsahující průběžně ukládané váhy modelu. Pro další práci s modelem a jeho nasazení je nutné checkpointy exportovat do formátu s názvem `saved_data`. To je možné učinit opět pomocí skriptu spouštěného z Jupyter buňky, jak je uvedeno ve výpisu 4.3.

```
1 # Export the model checkpoints to saved_data format
2 !python $notebook_directory"models/research/object_detection/exporter_main_v2
   ↪ .py" \
3 --pipeline_config_path={pipeline_file} \
4 --trained_checkpoint_dir={last_model_path} \
5 --output_directory={output_directory}
```

Výpis 4.3: Kód z buňky sešitu Jupyter pro spuštění skriptu pro export z checkpointů do formátu `saved_data`.

4.1.3 Vyhodnocení

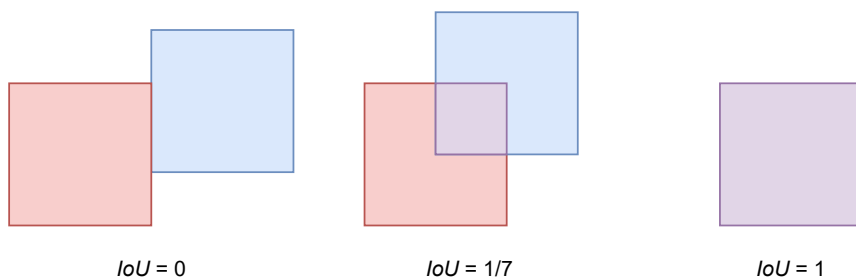
Výsledky natrénovaných modelů detekce objektů lze posoudit jak subjektivními, tak i objektivními metodami. Pro posouzení přesnosti byla vybrána objektivní metrika mAP (Mean Average Precision), která je v oblasti detekčních modelů rozšířená. Tato metrika byla poprvé definována v [23] a pro samotné vyhodnocení výsledků pomocí této metriky bude využit skript v jazyce Python, který byl speciálně vytvořen pro použití v [24]. Jak již bylo dříve nastíněno, pro modely s vyšší přesností je typická i vyšší výpočetní náročnost a během vyhodnocení účinnosti bude brána v úvahu i průměrná doba, za kterou dojde k zpracování jednoho snímku modelem.

Metrika pro vyhodnocení přesnosti

Metrika mAP se vypočítá jako průměrná hodnota z dílčích průměrných přesností detekce AP (Average Precision) jednotlivých objektů v obraze pro určité hodnoty IoU (Intersection over Union). Hodnota IoU vyjadřuje míru překryvu mezi predikovanou oblastí objektu a skutečnou oblastí objektu ve snímku. Výpočet IoU je založen na poměru plochy překryvu a sjednocení těchto dvou oblastí:

$$IoU = \frac{S_P}{S_S}, \quad (4.1)$$

kde plocha sjednocení S_S představuje celkovou plochu, kterou zahrnují oba objekty (predikovaný a skutečný) včetně oblasti překryvu S_P . Plocha překryvu S_P představuje oblast, kde se predikovaný objekt a skutečný objekt v obraze vzájemně překrývají. Na obrázku 4.2 je znázorněn princip IoU , skutečný objekt je vyznačen modře, predikovaný objekt červeně, fialově je pak vyznačena oblast překryvu S_P [25].



Obr. 4.2: Příklady jednotlivých hodnot IoU .

Celková hodnota mAP je v tomto případě určena z deseti dílčích hodnot průměrné přesnosti AP při deseti různých IoU , a to od $IoU = 0,5$ až do $IoU = 0,95$ při kroku $0,05$. Pro výpočet AP je nutné u každého objektu podrobeného detekci určit *přesnost* a *úplnost*.

Pro určení těchto hodnot je třeba znát následující termíny: *Falešně pozitivní* detekce nastává, když model nesprávně identifikuje objekt, který ve skutečnosti v obraze není. Naopak *pravdivě pozitivní* detekce odpovídá správné identifikaci skutečného objektu. *Falešně negativní* detekce se vyskytuje, když model neidentifikuje objekt, který ve skutečnosti v obraze existuje.

Určením přesnosti (precision) P měříme přesnost predikce, a tedy kolik z nich je skutečně správných:

$$P = \frac{PP}{PP + FP}, \quad (4.2)$$

kde PP je počet pravdivě pozitivních výsledků a FP je počet falešně pozitivních výsledků.

Úplnost (recall) R určuje s jakou efektivitou jsou identifikovány všechny skutečné pozitivní případy vzhledem k celkovému počtu pozitivních případů:

$$R = \frac{PP}{PP + FN}, \quad (4.3)$$

kde PP je počet pravdivě pozitivních výsledků a FN počet falešně negativních výsledků.

Pro danou třídu objektu a IoU se takto vypočítá přesnost P a úplnost R a vynese se do závislosti, z té se pak určí průměrná přesnost AP . Průměrnou přesnost AP lze získat výpočtem plochy pod křivkou závislosti přesnosti na úplnosti:

$$AP = \int_0^1 P dR. \quad (4.4)$$

Celková hodnota mAP se tak za pomoci dílčích hodnot AP vypočítá jako:

$$mAP = \frac{1}{10} \sum_{j=0,5}^{0,95} AP_j = \frac{1}{10} (AP_{0,5} + AP_{0,55} + \dots + AP_{0,95}), \quad (4.5)$$

kde $AP_{0,5}$ je určena při hodnotě $IoU = 0,5$, hodnota $AP_{0,55}$ při $IoU = 0,55$ a tak dále [25].

Výsledky hodnocení

Hodnocení natrénovaných modelů pomocí metriky mAP a průměrné doby zpracování jednoho snímku jsou uvedeny v tabulce 4.1. Detekční modely byly testovány na testovací části použitého datasetu, která zahrnuje 10% z celkového počtu snímků, a to na notebooku s procesorem AMD Ryzen 5800H bez grafické akcelerace.

Z výsledků vyplývá, že Mask R-CNN dosáhl nejvyššího mAP , a to 94,37%. Na druhou stranu však tento model potřebuje v průměru 3896 ms na zpracování jednoho snímku při použití výše uvedeného hardwaru. I přes to, že modely z rodiny

Tab. 4.1: Porovnání přesnosti detekce a výpočetní náročnosti natrénovaných modelů.

Architektura modelu	Průměrná doba zpracování snímku [ms]	mAP [%]
EfficientDet D0	237	87,43
EfficientDet D2	454	88,62
Mask R-CNN	3 896	94,37

EfficientDet vykazují mírně nižší mAP skóre ve srovnání s Mask R-CNN, jsou výrazně méně výpočetně náročné. EfficientDet D0 a D2 dosáhly mAP skóre 87,43 % a 88,62 %, s průměrnou dobou na zpracování jednoho snímku 236 ms a 453 ms.

Při porovnání modelů D2 a D0 rodiny EfficientDet je patrný dvojnásobný nárůst výpočetní doby. Dalo by se předpokládat, že s rostoucí vstupní velikostí obrazu a i časem potřebným pro detekci, se zvýší i její přesnost. V tomto případě však EfficientDet D2, se vstupními rozměry snímku 768×768 px, přináší pouze minimální nárůst přesnosti oproti modelu D0, který pracuje s rozměry 512×512 px.

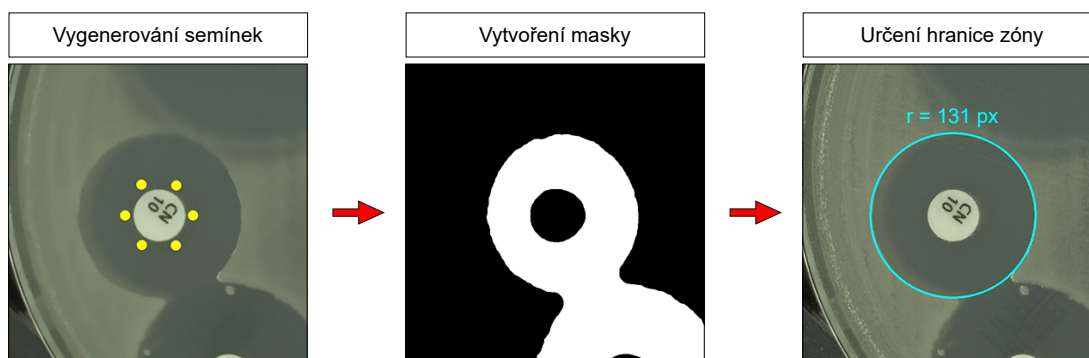
Díky výsledkům z tohoto vyhodnocení bylo ověřeno, že natrénované modely pro detekci antibiotických vzorků ve snímcích jsou dostatečně účinné a vhodné pro použití v dalším postupu této práce.

4.2 Měření inhibičních zón

Pro náš postup a jeho postupné ladění a optimalizaci využijeme programovací jazyk Python a v něm dostupnou knihovnu OpenCV.

V rámci předzpracování dat se nejprve snímek, na kterém bude provedeno měření inhibičních zón, „vyhladí“ za pomoci algoritmu „non-local means“ na redukci šumu. Následně se aplikuje Gaussovský filtr a snímek se mírně rozmaže. Další postup je ilustrován na obrázku 4.3. Dojde k využití přesně získaných pozic antibiotických disků, které jsou výstupem natrénovaných detekčních modelů ze sekce 4.1. V okolí každého disku dojde k vygenerování množiny bodů – tzv. „semínek“, které jsou v první části obrázku 4.3 vyznačeny žlutými body. Tyto body využijeme jako vstupní data pro tzv. „flood-fill“ algoritmus. Který se snaží v rámci obrazu vyplnit, neboli „zaplavit“ jeho vizuálně „uzavřenou oblast“. Výstupem tohoto procesu je binární maska s vyplněnou oblastí. Následně pak pomocí navrženého algoritmu pojmenovaného ICE (Iterative Circle Enlargement) dochází k postupnému rozšiřování zóny okolo disku a zkoumá se, zdali se současná oblast již nenachází mimo zaplavenou část.

Navržený algoritmus pro určení poloměru inhibiční zóny implementuje následující metodu `_find_zone`, která je zobrazena ve výpisu 4.4. Nejprve se vytvoří kopie



Obr. 4.3: Postup algoritmu pro měření inhibičních zón antibiotických disků.

původního snímku, metoda `_generate_seeds_around_center` pak vygeneruje seznam semínek. Funkce tyto body, reprezentované dvojicemi souřadnic, vybírá v okolí antibiotického disku. Z nich pak vypočte průměrný rozdíl jasu pro každý barevný kanál metodou `_calculate_seeds_average_brightness`, do které vstupuje právě seznam semínek a výstup tvoří trojice hodnot rozdílů jednotlivých barevných kanálů. Po získání této referenční hodnoty se pro každé jedno semínko v rámci jeho pozice od této hodnoty vypočte rozdíl. Jak je vidět z výpisu 4.4, takto získané rozdíly hodnot vstupují do vytvořené metody `_seedfill`. Její obsah je zobrazen ve výpisu 4.5. Vyvinutá metoda implementuje tzv. „flood-fill“ funkci z knihovny OpenCV. Tato funkce se snaží v rámci obrazu vyplnit jeho „uzavřenou oblast“, a to na základě zadaného počátečního bodu (semínka) a maximální dolní a horní hranice rozdílů jasu (`loDiff`, `upDiff`). V okolí semínka pak algoritmus postupně rozhoduje, zda podle těchto kritérií sousední pixely zahrne do zaplavené oblasti.

Z výpisu 4.5 je zřejmé, že se podle kompenzačních hodnot prvně určí parametry `up_diff` a `lo_diff` pro všechny barevné kanály. Chování flood-fill algoritmu z knihovny OpenCV lze ještě ovlivnit využitím „vlajek“. Vlajka s názvem `cv2.FLOODFILL_MASK_ONLY`, určuje, že výstupem bude pouze maska s „rozlitým“ prostorem. Vlajka (`255 << 8`) rozšíří oblast rozhodování algoritmu v rámci jednotlivých semínek tak, že oproti výchozí vlajce (`255 << 4`) budou zahrnuty i pixely sdílející roh se semínkem. Poslední vlajkou je `cv2.FLOODFILL_FIXED_RANGE`, která zajišťuje, že rozhodující rozdíl pro rozšíření na další pixel v hodnotách barev mezi aktuálním pixelem a pixelem semínka bude v pevném rozsahu (`lo_diff` a `up_diff`).

Poté, co jsou pomocí výše uvedené metody získány jednotlivé masky, sloučí se do jedné, což je patrné z části začínající na řádce 21 výpisu 4.4. Následně se aplikuje morfologická operace pro uzavření případných děr, které se mohly v masce vyskytnout. Tento postup je naznačen na obrázku 4.4. Takto vytvořená maska vstupuje do algoritmu ICE (`_iterative_circle_enlarging`), která má za úkol přetrasformovat masku na poloměr inhibiční zóny v pixelech.


```

1 def _find_zone(self, image, bbox_center, bbox_width, bbox_height, seed_num,
  ↪ seed_offset_comp, seed_offset_rand, seed_lo_comp, seed_up_comp,
  ↪ ice_rad_end, ice_thickness, ice_thresh, ice_step, exclude_extremes,
  ↪ debug):
2     image_processed = image.copy()
3     max_height, max_width = image_processed.shape[:2]
4
5     # Generate the seeds
6     seeds = self._generate_seeds_around_center(bbox_center, bbox_width,
  ↪ bbox_height, max_height, max_width, seed_num, seed_offset_comp,
  ↪ seed_offset_rand, debug)
7     seed_brightness_avg = self._calculate_seeds_average_brightness(
  ↪ image_processed, seeds, exclude_extremes, debug)
8
9     masks_filled = []
10
11    # Iterate over the seeds and feed them into the seedfill
12    for seed in seeds:
13        current_seed_diff = self._calculate_seed_diff(image_processed, seed,
  ↪ seed_brightness_avg, debug)
14        mask = image_processed.copy()
15        if len(current_seed_diff) > 1:
16            mask = self._seedfill(mask, seed, current_seed_diff[0],
  ↪ current_seed_diff[1], current_seed_diff[2], seed_lo_comp,
  ↪ seed_up_comp, debug)
17        else:
18            mask = self._seedfill_grayscale(mask, seed, current_seed_diff[0],
  ↪ seed_lo_comp, seed_up_comp, debug)
19        masks_filled.append(mask[1:-1, 1:-1])
20
21    mask_conc = np.zeros(image.shape[:2], dtype=np.uint8)
22
23    # Concatenate the flood-filled masks
24    for image_filled in masks_filled:
25        mask_conc = cv2.max(mask_conc, image_filled)
26
27    cv2.morphologyEx(mask_conc, cv2.MORPH_CLOSE, cv2.getStructuringElement(
  ↪ cv2.MORPH_ELLIPSE, (10, 10)))
28    start_radius = round(math.sqrt((bbox_width / 2) ** 2 + (bbox_height / 2)
  ↪ ** 2))
29    found_radius = self._iterative_circle_enlarging(mask_conc, bbox_center,
  ↪ start_radius, ice_rad_end, ice_thickness, ice_step, ice_thresh,
  ↪ debug)
30    return found_radius

```

Výpis 4.4: Metoda `_find_zone` v jazyce Python pro měření velikosti inhibičních zón.

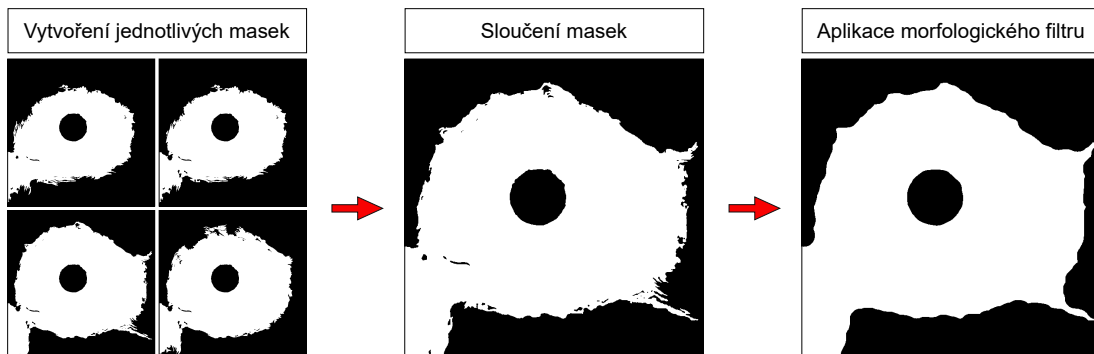
Obsah algoritmu ICE je uveden ve výpisu 4.6. Jak je z něj patrné, dochází k postupné iteraci přes rozsah hodnot poloměrů s krokem určeným hodnotou `step`. Počáteční hodnota je tvořena poloměrem antibiotického disku a koncová je pevně daná. V každé iteraci dojde k vytvoření „oblasti zájmu“ (z anglického ROI – Region of Interest), tedy duté kruhové masky, jejíž tloušťka je určena parametrem `thickness`. V této oblasti se pak vyextrahují všechny pixely z binární masky, které nabývají hodnoty jasu 1.

```

1 def _seedfill(self, img, seed, r_diff, g_diff, b_diff, lo_comp, up_comp,
  ↪ debug):
2     up_diff = [int(r_diff + up_comp), int(g_diff + up_comp), int(b_diff +
  ↪ up_comp)]
3     lo_diff = [int(r_diff + lo_comp), int(g_diff + lo_comp), int(b_diff +
  ↪ lo_comp)]
4
5     # Crop the values
6     for i in range(3):
7         up_diff[i] = min(max(up_diff[i], 0), 255)
8         lo_diff[i] = min(max(lo_diff[i], 0), 255)
9
10    floodflags = 4
11    floodflags |= cv2.FLOODFILL_MASK_ONLY
12    floodflags |= (255 << 8)
13    floodflags |= cv2.FLOODFILL_FIXED_RANGE
14
15    mask = np.zeros((img.shape[0] + 2, img.shape[1] + 2), dtype=np.uint8)
16
17    cv2.floodFill(img, mask, seed, (255, 255, 255), loDiff=(lo_diff[0],
  ↪ lo_diff[1], lo_diff[2]), upDiff=(up_diff[0], up_diff[1], up_diff
  ↪ [2]), flags=floodflags)
18    return mask

```

Výpis 4.5: Metoda `_seedfill` v jazyce Python implementující tzv. „flood-fill“ algoritmus knihovny OpenCV.



Obr. 4.4: Proces vytvoření výsledné binární masky.

Na řádku 13 (výpisu 4.6) se pak vypočítá počet všech pixelů v rámci vyextrahované oblasti, které nabývají hodnoty jasu větší než 0. Totéž se provede pro samotnou oblast zájmů. Nakonec se za pomoci těchto dvou hodnot vypočte poměr a pakliže je menší nebo roven stanovené prahové hodnotě `thresh`, funkce vrátí aktuální iterovaný poloměr. Pokud se během iterace nesplní prahová podmínka, metoda vrátí hodnotu `1e7`, což indikuje, že inhibiční zóna nebyla nalezena (pravděpodobně disk žádnou nevygeneroval).

Celý výše uvedený postup algoritmu ICE je naznačen na obrázku 4.5. Pro větší zřetelnost je parametr `thickness` nastaven na 30 px, stejně tak jako počáteční hod-

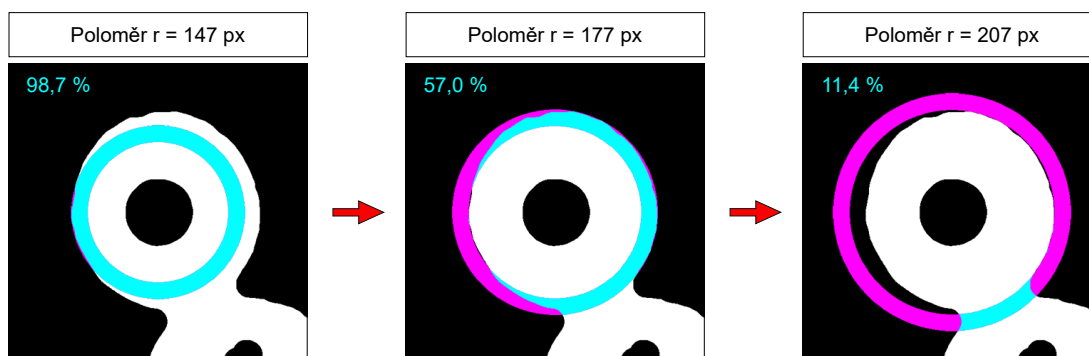
```

1 def _iterative_circle_enlarging(self, img, center, rad_start, rad_end,
  ↪ thickness, step, thresh, debug):
2     for radius in range(rad_start, rad_end, step):
3
4         # Create a hollow circular mask of a given thickness
5         mask = np.zeros(img.shape[:2], dtype=np.uint8)
6         cv2.circle(mask, center, radius, 255, thickness)
7
8         # Extract the region of interest using the circular mask
9         masked = cv2.bitwise_and(img, img, mask=mask)
10        total_non_zero = cv2.countNonZero(mask)
11        roi_non_zero = cv2.countNonZero(masked)
12
13        if roi_non_zero <= total_non_zero * thresh:
14            return radius
15    return int(1e7)

```

Výpis 4.6: Algoritmus ICE, metoda v jazyce Python pro určení poloměru inhibiční zóny v pixelech ze vstupní masky.

nota poloměru je 147 px. Iterace probíhá s krokem 30 px. Maska určena zvětšujícím se poloměrem je zde znázorněna růžovou barvou, vyextrahované pixely pak barvou tyrkysovou. V prvním kroku je poměr vyextrahovaných pixelů k počtu pixelů oblasti zájmů 98,7 %, při druhé iteraci je tento poměr 57,0 %. Pokud bychom nastavili prahovou hodnotu `thresh` na 0,5, čili 50 %, k její překročení by došlo právě při třetí iteraci, kde je poloměr již 207 px a výsledkem ICE by tedy byl právě poloměr o velikosti 207 px.



Obr. 4.5: Proces určení hranice zóny a její výsledné velikosti pomocí algoritmu ICE.

4.2.1 Vyhodnocení

Navržený algoritmus pro měření velikosti inhibičních zón byl otestován na množině tvořené 60 profesionálně pořízenými snímky (snímky z MBT Pathfinder®) a 60

snímky pořízenými mobilními telefony. Pozice disků byly určeny z anotovaných hodnot. Detailní výsledky jsou uvedeny v tabulce 4.2. Relativní chyba zde uvádí průměrnou odchylku měřeného poloměru od jeho skutečné velikosti a nabývá 12,61 %. Hodnota „Správné výsledky“ značí součet všech pravdivě pozitivních a pravdivě negativních výsledků. Z výsledků vyplývá, že z celkového počtu 545 poloměrů dokázal náš algoritmus určit pro 519 z nich to, zda daný antibiotický disk inhibiční zónu vygeneroval, či nikoliv. Algoritmus v této metrice tedy dosahuje úspěšnosti 95,23 %. Díky tomuto postupu byla funkčnost navrženého algoritmu úspěšně ověřena.

Tab. 4.2: Výsledky přesnosti algoritmu pro měření velikosti inhibičních zón.

Relativní chyba [%]	Správné výsledky [-]	Špatné výsledky [-]	Úspěšnost [%]
12,61	519	26	95,23

5 Integrace

Všechny postupy uvedené v předchozí kapitole jsou integrovány ve formě modulu webové aplikace společnosti Bruker Daltonics GmbH & Co. KG, která využívá platformu ASP.NET vyvíjenou společností Microsoft. Jednotlivé postupy je nutné převést do takové podoby, abychom zajistili jejich podporu v rámci platformy ASP.NET a mohli je tak integrovat do výsledné webové aplikace.

5.1 Detekce antibiotických disků

Trénování modelů detekce antibiotických disků proběhlo za využití jazyka Python, prostředí Jupyter a platformy TensorFlow. Modely byly vyexportovány z checkpointů do TensorFlow formátu `saved_data`. Integraci natrénovaných modelů v rámci prostředí .NET zajistí ONNX. Jedná se o ekosystém umožňující flexibilní přecházení mezi různými platformami zabývajícími se strojovým učením a umělou inteligencí. Pro práci s ONNX Runtime, což je část ekosystému ONNX, která umožňuje spouštět modely formátu ONNX, je nutné nainstalovat příslušný .NET balíček a vyexportované modely převést do ONNX formátu. K tomu je vhodné použít `tf2onnx` balíček jazyka Python, který lze nainstalovat pomocí balíčkového manažeru `pip`.

O detekci antibiotických disků se v rámci navrhované aplikace stará třída s názvem `ATBDiskDetector`, implementovaná v jazyce C#. Ve výpisu 5.1 je zobrazena metoda `ObtainDiskDetection` této třídy. Při jejím zavolání se nejprve získá informace o šířce a výšce vstupního snímku a obrazová data se převedou do formátu kompatibilního s modelem. Z dat se vytvoří tenzor, který se vloží do seznamu vstupů pro predikci. Následně se spustí predikce, díky které model vygeneruje seznam výstupů zahrnujících rámečky ohraničující detekované disky. Výstup pro každý takto detekovaný disk dále obsahuje desetinnou hodnotu určující míru jistoty predikce.

Po získání hodnot z výstupního tensoru metoda projde výsledné detekce a pokud je jistota jejich predikce menší než prahová hodnota, detekovaný disk zahodí. Pro zbylé disky se z výstupních dat extrahují souřadnice jejich ohraničujících rámečků a pomocí `Convert.ToInt32` se převedou na celočíselné hodnoty odpovídající pixelům v obrázku. Návrátovou hodnotou této metody je výsledek detekce v podobě seznamu s objekty typu `BoundingBox`.

5.2 Měření velikosti inhibičních zón

Během práci na vývoji algoritmu pro měření velikostí inhibičních zón byl využíván programovací jazyk Python a knihovna OpenCV, kterou podporuje. Spolu s prací se sešity Jupyter tento postup umožňoval snadné změny a ladění. Jelikož není knihovna

```

1 public List<BoundingBox> ObtainObjectDetection(BVIBitmap dishImage)
2 {
3     int width = dishImage.Width;
4     int height = dishImage.Height;
5     var data = MachineLearningConverter.ConvertImageByte(dishImage, swapRB:
6         ↪ true, MachineLearningConverter.Format.NHWC);
7
8     DenseTensor<byte> tensor = new(data, new[] { 1, height, width, 3 });
9
10    List<NamedOnnxValue> inputs = new() { NamedOnnxValue.CreateFromTensor(
11        ↪ _inputColumnName, tensor) };
12
13    var prediction = _session.Run(inputs);
14    var detectionBoxes = prediction.ElementAt(1).AsTensor<float>();
15    var detectionScores = prediction.ElementAt(4).AsTensor<float>();
16
17    List<BoundingBox> resultBoundingBoxes = new();
18    for (int i = 0; i < detectionScores.Length; i++)
19    {
20        var confidence = detectionScores.ElementAt(i);
21        if (confidence < _confidence) { continue; }
22        var ymin = detectionBoxes.ElementAt(i * 4);
23        var xmin = detectionBoxes.ElementAt(i * 4 + 1);
24        var ymax = detectionBoxes.ElementAt(i * 4 + 2);
25        var xmax = detectionBoxes.ElementAt(i * 4 + 3);
26
27        var left = Convert.ToInt32(xmin * width);
28        var top = Convert.ToInt32(ymin * height);
29        var right = Convert.ToInt32(xmax * width);
30        var bottom = Convert.ToInt32(ymax * height);
31        resultBoundingBoxes.Add(new(left, top, right, bottom, confidence));
32    }
33    return resultBoundingBoxes;
34 }

```

Výpis 5.1: Metoda ObtainDiskDetection třídy ATBDiskDetector v jazyce C# obstarávající detekci antibiotických disků.

OpenCV přímo podporována v rámci .NET a jazyka C# a spouštění kódu v jazyce Python např. pomocí implementace IronPython v C# není pro výpočetně náročné aplikace optimální, je nutné kód napsaný v jazyce Python přepsat do jazyka C++. Díky tomuto postupu bude možné přepsaný kód zkompileovat jako dynamickou knihovnu DLL a její obsah volat pomocí PInvoke z .NET prostředí, a tudíž i z výsledné webové aplikace.

Ve výpisu 5.2 je metoda FindZoneSeedfill ze třídy AstDiffusionZoneFinder v jazyce C++. Obstarává tutéž funkci jako metoda `_find_zone` ve výpisu 4.4 ze sekce 4.2 v jazyce Python. Aby bylo možné metodu zkompileovat jako dynamickou knihovnu DLL a její obsah mohl interagovat s kódem v jazyce C# a prostředí .NET, je nutné před kompilací uvažované metody patřičně označit. Z tohoto důvodu byly pro každou metodu vytvořeny zapouzdřovací funkce, ta pro metodu FindZoneSeedfill je zobrazena ve výpisu 5.3.

```

1  int32_t AstDiffusionZoneFinder::FindZoneSeedfill(const cv::Mat image, const cv
    ↳ ::Point2i bboxCenter, int32_t bboxWidth, int32_t bboxHeight, int32_t
    ↳ seedNum, double seedOffsetComp, double seedOffsetRand, double
    ↳ seedLoComp, double seedUpComp, int32_t IceRadEnd, int32_t IceThickness,
    ↳ double IceThresh, int32_t IceStep, int32_t debug) {
2      cv::Mat imageProcessed = image.clone();
3
4      // Get image dimensions
5      int32_t maxHeight = imageProcessed.rows;
6      int32_t maxWidth = imageProcessed.cols;
7      // Generate the seeds
8      std::vector<cv::Point2i> seeds = AstDiffusionZoneFinder::
    ↳ GenerateSeedsAroundCenter(bboxCenter, bboxWidth, bboxHeight,
    ↳ maxHeight, maxWidth, seedNum, seedOffsetComp, seedOffsetRand);
9      std::vector<double> seedBrightnessAvg = AstDiffusionZoneFinder::
    ↳ CalculateSeedsAverageBrightness(imageProcessed, seeds, debug);
10
11     std::vector<cv::Mat> masksFilled;
12     for (const auto& seed : seeds) {
13         // Calculate the difference in brightness for the seed
14         std::vector<double> currentSeedDiff = AstDiffusionZoneFinder::
    ↳ CalculateSeedDiff(imageProcessed, seed, seedBrightnessAvg,
    ↳ debug);
15         cv::Mat mask = imageProcessed.clone();
16         if (currentSeedDiff.size() > 1) { mask = Seedfill(mask, seed,
    ↳ currentSeedDiff[0], currentSeedDiff[1], currentSeedDiff[2],
    ↳ seedLoComp, seedUpComp, debug); }
17         else { mask = SeedfillGrayscale(mask, seed, currentSeedDiff[0],
    ↳ seedLoComp, seedUpComp, debug); }
18         masksFilled.push_back(mask(cv::Rect(1, 1, mask.cols - 2, mask.rows -
    ↳ 2)));
19     }
20     // Concatenate the flood-filled masks
21     cv::Mat maskConc = cv::Mat::zeros(image.size(), CV_8UC1);
22     for (auto& imageFilled : masksFilled) { maskConc = cv::max(maskConc,
    ↳ imageFilled); }
23     cv::morphologyEx(maskConc, maskConc, cv::MORPH_CLOSE, cv::
    ↳ getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(10, 10)));
24     int32_t found_radius = AstDiffusionZoneFinder::IterativeCircleEnlarging(
    ↳ maskConc, bboxCenter, cvRound(sqrt(pow(bboxWidth / 2, 2) + pow(
    ↳ bboxHeight / 2, 2))), IceRadEnd, IceThickness, IceStep, IceThresh);
25     return found_radius;
26 }

```

Výpis 5.2: Metoda FindZoneSeedfill třídy AstDiffusionZoneFinder v jazyce C++ obstarávající tutéž funkci jako metoda `_find_zone` v jazyce Python ve výpisu 4.4 ze sekce 4.2.

Poté je možné vytvořené zapouzdřovací funkce v hlavičkovém souboru `.h` ohraničit klíčovou kombinací `extern "C"` tak, jak je naznačeno ve výpisu 5.4. V `.NET` projektu bude kompilátor informován o tom, že definice této funkce existuje jinde a měla by být při generování finálního spustitelného souboru propojena. Před definicí každé funkce ještě použijeme makro `EXPORT`, funkce pak bude viditelná i mimo současný modul nebo knihovnu.

```

1  int32_t IFindZoneSeedfill(IABase::BVIBitmap* image, int32_t bboxCenterX,
   ↪ int32_t bboxCenterY, int32_t bboxWidth, int32_t bboxHeight, int32_t
   ↪ seedNum, double seedOffsetComp, double seedOffsetRand, double
   ↪ seedLoComp, double seedUpComp, int32_t IceRadEnd, int32_t IceThickness,
   ↪ double IceCircleThresh, int32_t IceStep, int32_t debug) {
2  AstDiffusionZoneFinder zoneFinder;
3  return zoneFinder.FindZoneSeedfill(image->getImage(), cv::Point2i(
   ↪ bboxCenterX, bboxCenterY), bboxWidth, bboxHeight, seedNum,
   ↪ seedOffsetComp, seedOffsetRand, seedLoComp, seedUpComp, IceRadEnd,
   ↪ IceThickness, IceCircleThresh, IceStep, debug);
4  }

```

Výpis 5.3: Zapouzdřovací funkce IFindZoneSeedfill v jazyce C++ sloužící jako rozhraní pro metodu FindZoneSeedfill třídy AstDiffusionZoneFinder.

```

1  extern "C" {
2  EXPORT int32_t IFindZoneSeedfill(IABase::BVIBitmap* image, int32_t
   ↪ bboxCenterX, int32_t bboxCenterY, int32_t bboxWidth, int32_t
   ↪ bboxHeight, int32_t seedNum, double seedOffsetComp, double
   ↪ seedOffsetRand, double seedLoComp, double seedUpComp, int32_t
   ↪ IceRadEnd, int32_t IceThickness, double IceCircleThresh, int32_t
   ↪ IceStep, int32_t debug);
...
6  }

```

Výpis 5.4: Exportace funkcí v jazyce C++ před kompilací dynamické knihovny za pomoci klíčové kombinace `extern "C"` a makra `EXPORT`.

Po zkompileování dynamické knihovny za pomoci patřičně nastaveného souboru `CMakeLists.txt`, propojení projektů a přidání správných cest, již můžeme volat funkce v rámci projektového prostředí webové aplikace. O interakci s DLL modulem pomocí `PInvoke` se v rámci `.NET` aplikace stará zapouzdřovací statická třída `UnsafeNativeMethods`, která je zobrazena ve výpise 5.5. Samotný algoritmus hledání kružnic a další funkce přejaté z dynamické knihovny DLL jsou pak volány pomocí jednotlivých metod statické třídy `ZoneFinder`.

```

1  internal static partial class UnsafeNativeMethods
2  {
...
6  [DllImport("Bruker.ImageAnalysisPlus.IAAS", CallingConvention =
   ↪ CallingConvention.Cdecl)]
7  unsafe public static extern int IFindZoneSeedfill(BitmapSafeHandle image,
   ↪ int bboxCenterX, int bboxCenterY, int bboxWidth, int bboxHeight,
   ↪ int seedNum, double seedOffsetComp, double seedOffsetRand, double
   ↪ seedLoComp, double seedUpComp, int IceRadEnd, int IceThickness,
   ↪ double IceCircleThresh, int IceStep, int debug);
...
15 }

```

Výpis 5.5: Statická třída `UnsafeNativeMethods` v jazyce C# pro volání funkcí z dynamické knihovny DLL.

5.3 Webová aplikace

V předchozí kapitole byly vytvořeny třídy `ATBDiskDetector` a `ZoneFinder` v programovacím jazyce `C#`, které se starají o zpracování snímků. Na vývoj serverové části aplikace využijeme prostředí `ASP.NET` a jazyka `C#`. Výsledky o účinnosti antibiotických vzorků budou prezentovány na straně klienta, kde je uživatel taktéž bude moci upravit. Právě o tuto funkcionalitu se bude starat část aplikace napsaná v jazyce `JavaScript`. Přehlednou grafickou prezentaci uživateli v prohlížeči zajistí použití vhodných kaskádových stylů `CSS`.

Nahrávání snímku, konfigurace

Pro nahrání samotného snímku na serverovou část poslouží formulář, který bude uživatelem vyplněn a odeslán na server. Formulář obsahuje velké množství položek. Mezi ty základní patří tlačítko, pomocí kterého se formulář odešle, vstupní snímek, tlačítko pro zobrazení podrobné konfigurace a k němu přidružená skrytá vstupní hodnota. Ukázka z této části `ASP.NET` projektu je zobrazena ve výpisu 5.6. Jednotlivé elementy jsou propojeny s `JavaScript` kódem: např. tlačítko `SubmitButton` je při načtení stránky deaktivováno (atribut `disabled`) a až po nahrání snímku dochází k jeho aktivaci. Tlačítko `EnableConfButton` je zase propojeno s `JavaScript` funkcí `toggleAdvSettings`, ta slouží k zapnutí, či vypnutí konfigurační nabídky v rámci formuláře. Podstatným je také skrytý vstup s časovou značkou, ta se generuje v prohlížeči za pomoci skriptu jazyka `JavaScript` a na straně serveru se ověřuje. Jedná se o jednoduchý mechanismus zabránění opětovného odeslání stejného formuláře.

```
1 <form id="submitForm" asp-page-handler="Async" method="post" enctype="
  ↳ multipart/form-data" class="content">
2   <div class="form-container-main">
3     <button class="button" type="submit" id="SubmitButton" disabled>
4       ↳ Process image </button>
5     <input type="file" name="FormFile" id="FormFile" />
6     <button class="button" type="button" id="EnableConfButton"
7       ↳ aria-checked="false" onclick="toggleAdvSettings()">Enable
8       ↳ configuration </button>
9     <input type="hidden" asp-for="FormData.ADVConf">
10    @* Send the timestamp *@
11    <input type="hidden" asp-for="FormData.Timestamp" />
12  </div>
13  ...
14  ...
15  </form>
```

Výpis 5.6: Část `ASP.NET` formuláře pro odeslání snímku a konfiguračních dat na server.

Na obrázku 5.1 je zobrazen náhled části webové stránky se zmíněným formulářem, na kterou byly aplikovány kaskádové styly se snadno škálovatelným prostředím

flexbox. Uživatel v horní části vybere snímek, v dolní části (nepovinné a zobrazitelné pomocí tlačítka) může ladit parametry dříve uvedených algoritmů, které po stisknutí tlačítka s nápisem „Process image“ snímek zpracují.

Obr. 5.1: Formulář webové aplikace pro konfiguraci a nahrání snímku ke zpracování.

Zpracování snímku

Při odesílání formuláře je využita metoda `OnPostAsync` zobrazena ve výpisu 5.7. Nastavené možnosti jsou serializovány a uloženy do dat relace `TempData`, kterou je možno v ASP.NET prostředí využít. Po opětovném načtení stránky uživatel nemusí nastavovat hodnoty znovu. Data jsou deserializována a načtena právě z dat relace. Obsah formuláře je následně zkontrolován metodou `IsValid` statické třídy `FormDataVerifier` a pakliže jsou data správná, dojde k pokusu o jejich načtení za pomoci třídy `MemoryStream`. Po načtení je snímek oříznut a zmenšen (pokud je povoleno) a uložen pod náhodným jménem na server, aby jej bylo možné uživateli na webové stránce zobrazit. Následně se snímek pomocí funkce volané z dynamické knihovny DLL vyhledá a rozmaže Gaussovským filtrem a je možné spustit detekci antibiotických disků a na ni navazující proces měření velikosti inhibičních zón.

Po zpracování snímku se detekované disky a jejich přidružené zóny zobrazí v podobě výpisu v horní části obrazovce uživateli tak, jako to je znázorněno na ukázce 5.2. Jsou zde vypsány výsledky pro jednotlivé detekované disky: jednoznačný identifikátor ID, hodnota jistoty detekce `Confidence` a průměr kruhové inhibiční zóny v pixelech. Pro přehledné a škálovatelné zobrazení těchto dat bylo opět využito prostředí **flexbox**, každý disk pak tvoří jeho další položku.

V dolní části je již zpracovaný (a podle potřeby i ořezaný) snímek, který byl nahrán uživatelem na server. Ukázka této části webové aplikace je znázorněna na

```

1 public async Task<IActionResult> OnPostAsync(IFormFile formFile)
2 {
3     InferenceDone = false;
4     // Store useful FormData to TempData
5     TempData.Clear();
6     TempData["FormData"] = SerializeFormData(FormData.GetFormDataToStore());
7     TempData.Keep("FormData");
8
9     FormData.FileData = formFile;
10    if (!FormDataVerifier.IsValid(FormData, TempData, _logger, LOG))
11    {
12        return Page();
13    }
14    // Attempt to upload the file
15    var memoryStream = new MemoryStream();
16    try
17    {
18        var fileStream = FormData.FileData.OpenReadStream();
19        // Copy the file stream to the memory stream
20        await fileStream.CopyToAsync(memoryStream);
21        // Reset the memory stream position to the beginning
22        memoryStream.Seek(0, SeekOrigin.Begin);
23    }
24    ...
25    ...
61 }

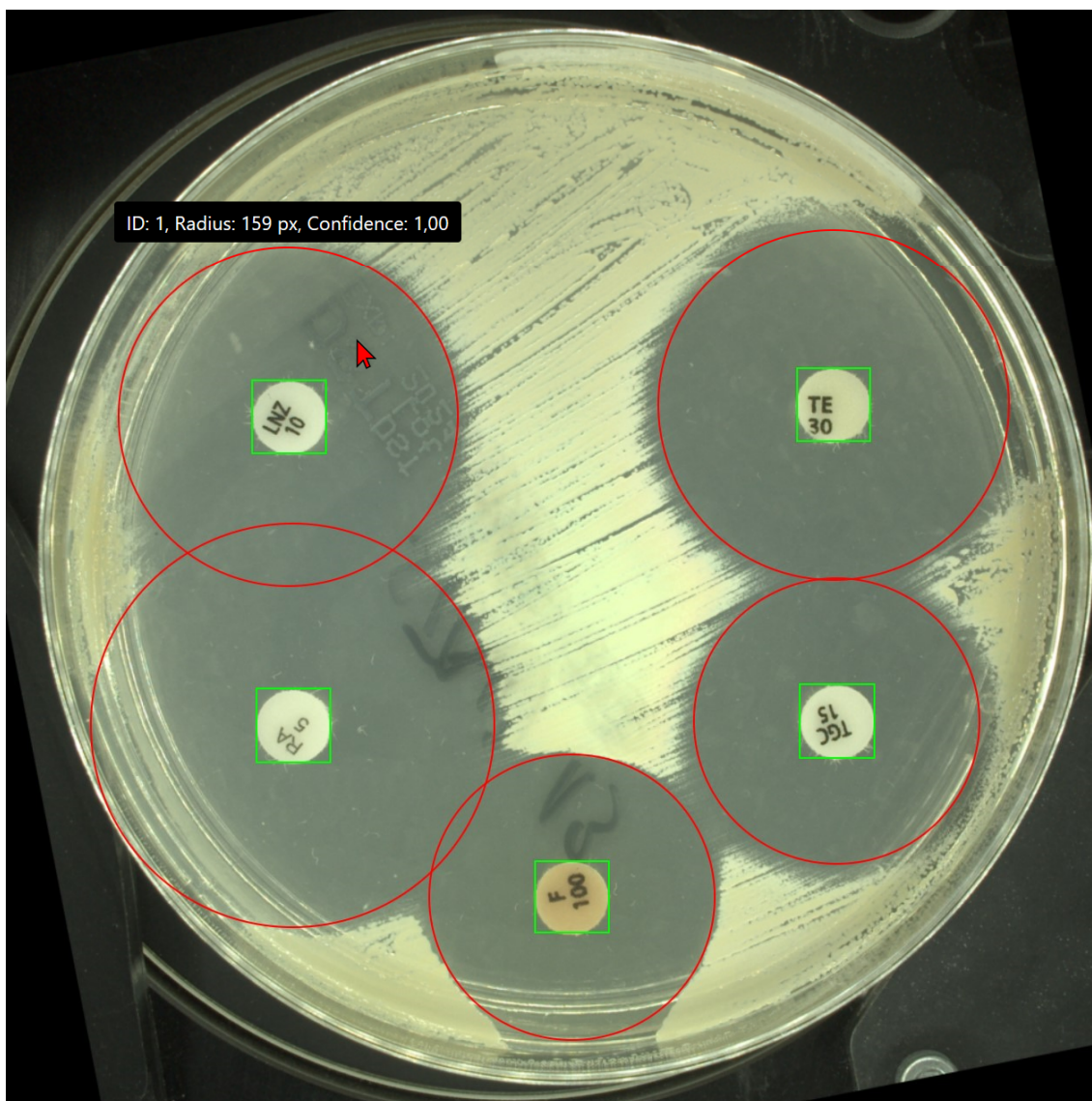
```

Výpis 5.7: Metoda využívána při odeslání formuláře v rámci projektu webové aplikace v ASP.NET.

ID:	0	1	2	3	4
Confidence:	1,00	1,00	1,00	1,00	1,00
Measured diameter (px):	328	318	268	268	378
Measured diameter (mm):

Obr. 5.2: Prezentace detekovaných antibiotických disků a jim přidružených inhibičních zón uživateli v rámci webové aplikace.

obrázku 5.3. Zelené rámečky ohraničující detekované disky a červené kružnice ohraničující inhibiční zóny jsou umístěny v rámci vrstvy překrývající element se snímkem. Jsou tvořeny elementy tříd `circle` a `bounding-box`. Jejich pozice je určena absolutně přepočtem z bodů na procenta pomocí funkce `calc`, kterou prohlížeče podporující CSS nabízí. Důležité je také nastavení správné hodnoty `z-index`. Ta udává, v jakém pořadí jsou vrstvy na sebe skládány. Po najetí kurzorem na patřičnou zónu či disk je za pomoci kódu v jazyce JavaScript zobrazen popisek s identifikátorem ID a dalšími hodnotami.



Obr. 5.3: Presentace zpracovaného snímku uživateli v rámci webové aplikace.

Uživatelské úpravy, výsledky měření zón

Jak již bylo naznačeno, samotné vyhodnocení výsledků probíhá na straně klienta, a proto je tato funkcionality zpracována zejména v rámci jazyka JavaScript. Aby bylo možné výsledky v pixelech přepočítat do hodnot v milimetrech, je nutné umožnit Petriho misku uživateli vyklíčovat a uvést její rozměr. Na obrázku 5.4 je uvedena část webové aplikace, která se zabývá právě tímto postupem.

Ukázka popisuje stav po stisknutí tlačítka s názvem „Segment out the dish“. Uprostřed snímku je vykreslen kruh tyrkysové barvy, který znázorňuje právě klíčovanou Petriho misku. Dříve viditelné elementy jsou skryty. Kruh je možné pomocí kurzoru myši a podržením levého tlačítka uvnitř čtvercového táhla libovolně přemísťovat. Kruh je také možné zmenšit přejetím a tažením kurzoru směrem do nebo

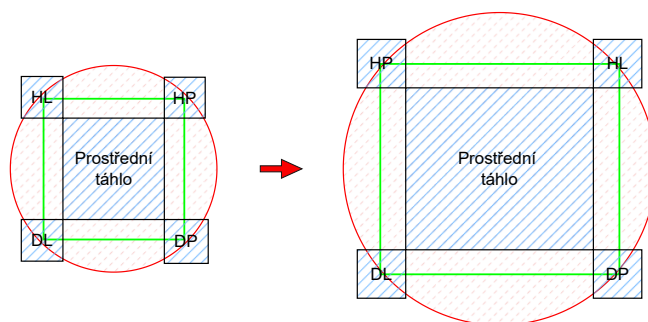


Obr. 5.4: Část webová aplikace, kde je umožněno uživateli vyklíčovat Petriho misku, a tak umožnit přepočet velikostí inhibičních zón v pixelech na milimetry.

od středu táhla. Pakliže je uživatel s vyznačením misky spokojen, pomocí stejného tlačítka postup uloží a původní elementy jsou opět zobrazeny. Po zadání poloměru vyklíčované misky v milimetrech do pole uvedeného jako „Dish diameter“, jsou v reálném čase průměry inhibičních zón přepočteny do velikostí v milimetrech, což je vidět na obrázku 5.6.

Během používání aplikace je možné, že bude zapotřebí upravit automaticky vygenerovanou pozici antibiotického disku, nebo velikost vygenerované inhibiční zóny v jeho blízkosti. Proto lze každý rámeček označující antibiotický disk libovolně přemístit, zvětšit, zmenšit a nebo upravit jeho poměr stran. Jelikož je jeden výsledek

reprezentován několika překrývajícími se elementy a u každého by měla být možnost jeho interaktivní úpravy uživatelem, byl implementován systém táhel zobrazený na obrázku 5.5.



Obr. 5.5: Táhla jednotlivých rámečků sloužící pro změnu pozice a rozměrů výsledku.

Každý výsledek zobrazený na snímku je tedy ve skutečnosti reprezentován nejen zeleným ohraničujícím rámečkem a červeným kruhem, ale i pěti táhly, které jsou viditelné pro kurzor myši, avšak nikoli pro uživatele. Prostřední táhlo slouží k přemístování libovolného rámečku v rámci snímku. Táhla označená na obrázku 5.5 jako „HL“, „HP“, „DL“ a „DP“ jsou určena pro změnu rozměrů rámečku, a to v jim odpovídajících směrech. Např. táhlo „HL“ značí horní levý roh umožňuje změnu rozměrů rámečku ve dvou směrech současně, a to od jeho středu doleva a nahoru. Zelený rámeček je pro kurzor myši úplně neviditelný, ale červeně ohraničený kruh slouží ke změně velikosti inhibiční zóny. Jelikož nejmenší velikost inhibiční zóny je daná velikostí vepsané kružnice zeleného rámečku, došlo by právě při této situaci ke kompletnímu překrytí těchto dvou objektů. Řešením je fakt, že prostřední táhlo tvoří vždy 90 % rozměrů zeleného rámečku, a tak je možné v jakémkoliv případě s kruhem reprezentující inhibiční zónu spolehlivě interagovat. Při změně pozice boxu ohraničující disk nebo jeho rozměrů se v přehledu jednotlivých detekcí změní i jistota detekce z desetinné hodnoty na slovo „user“ a v plovoucím popisku pak bude tato hodnota nastavena na „User Defined“, a to tak, jak je ukázáno na obrázku 5.6.

Rámečky označující antibiotické disky a jimi vygenerované zóny je také možné odstraňovat a přidávat. Pro odstraňování nechtěných výsledků detekčního algoritmu je možné využít tlačítka „Remove selected box“, ale nejdříve je zapotřebí daný box označit, a to kliknutím levým tlačítkem myši. To je možné pomocí funkce `handleMouseDownSel`, která je zobrazena ve výpise 5.8.

Funkce vybraný box vybarví žlutou barvou a následně je možné vybraný box spolu se zónou odstranit. Tento postup je zobrazen na obrázku 5.6. K přidávání nových výsledků slouží tlačítko s názvem „Add new ATB box“, po jeho stisknutí je vyvolána JavaScriptová funkce a uprostřed snímku se objeví nový box a uživatel

```

1 function handleMouseDownSel(event) {
2
3     // Get parent - bounding box
4     const boundingBox = event.target.parentNode;
5     const value = parseInt(boundingBox.getAttribute("value"));
6     resetMouseDownSel();
7
8     // Reset on second click
9     if (selectedBoxId === value) {
10         selectedBoxId = -1;
11         return;
12     }
13     selectedBoxId = value;
14     boundingBox.style.border = "2px dashed yellow";
15     boundingBox.style.backgroundColor = "rgba(255, 255, 0, 0.2)";
16
17     // Get result container with result-id = value
18     const resultContainer = document.querySelector('#atb-result-value[value='
19         ↳ `${value}`']');
20     resultContainer.style.border = "1px dashed black";
21     resultContainer.querySelector("#result-id").style.backgroundColor = "rgba
22         ↳ (255, 255, 0, 0.5)";
23     resultContainer.querySelector("#result-px").style.backgroundColor = "rgba
24         ↳ (255, 255, 0, 0.5)";
25
26     // Update removeBoxButton onclick to removeBoundingBox(value)
27     const removeButton = document.querySelector("#removeBoxButton");
28     removeButton.setAttribute("onclick", `removeBoundingBox(${value})`);
29 }

```

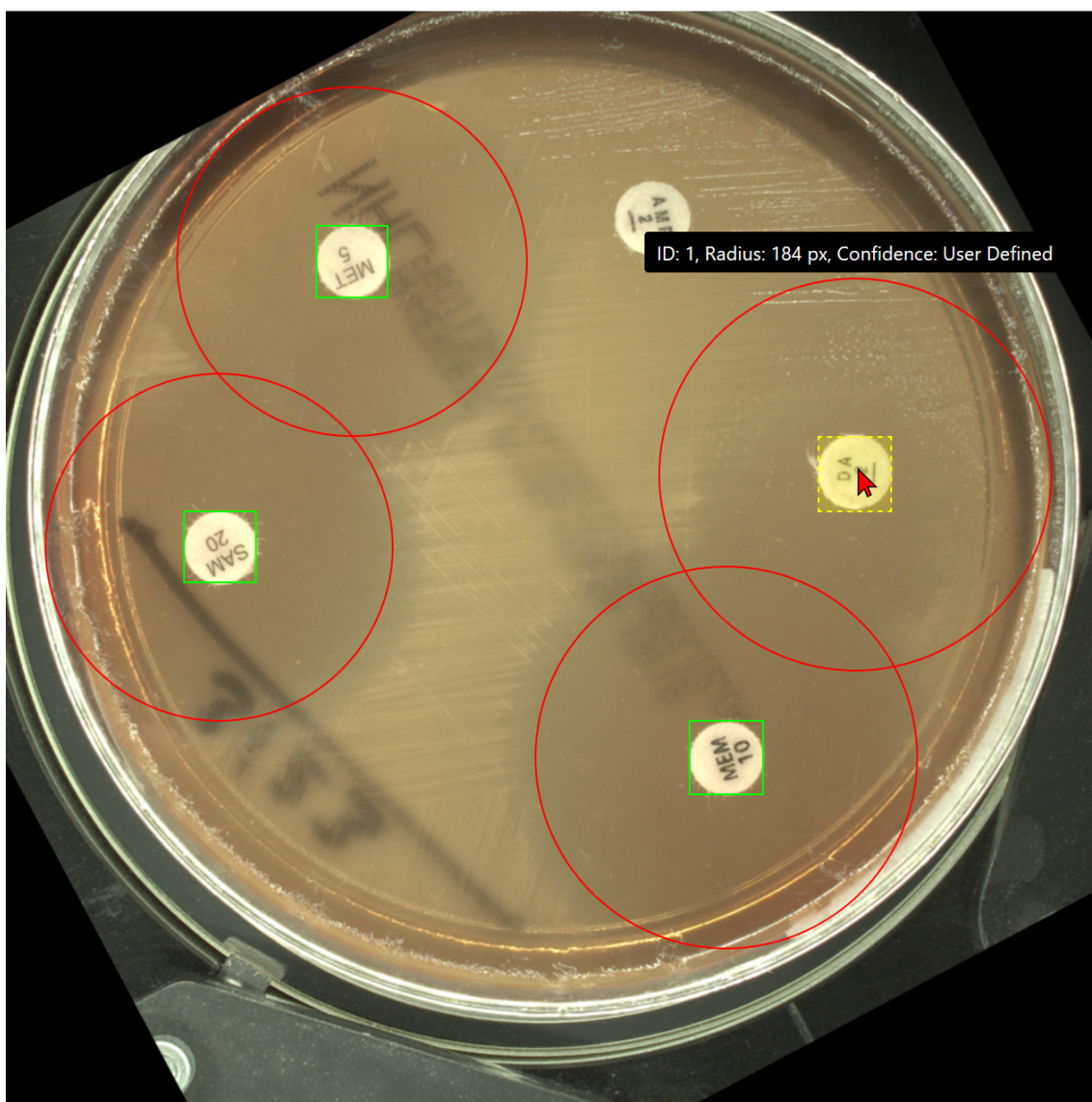
Výpis 5.8: Funkce `handleMouseDownSel` v jazyce JavaScript obstarávající výběrání jednotlivých detekovaných disků uživatelem.

jej může podle potřeb přemístit a jeho inhibiční zónu interaktivně zvětšit. Pomocí tohoto postupu a využití jazyka JavaScript je možné vše výše uvedené provádět dynamicky a usnadnit tak práci uživatele.

Process image [Procházet...](#) Soubor nevybrán. [Enable configuration](#)

ID:	0	1	2	4	Add new ATB box
Confidence:	user	user	1,00	1,00	Remove selected box
Measured diameter (px):	328	368	358	326	
Measured diameter (mm):	31.68	35.54	34.57	31.48	

[Segment out the dish](#) Dish diameter: 90 mm



Obr. 5.6: Ukázka celé webové aplikace se zaměřením na úpravu výsledků uživatelem.

Závěr

Záměrem této bakalářské práce bylo aplikovat metody umělé inteligence, strojového učení a algoritmů zpracování obrazu v oblasti měření citlivosti na antibiotika. Pomocí těchto postupů pak konkrétně zautomatizovat a vylepšit proces vyhodnocení diskové difúzní metody, která v současném světě hraje klíčovou roli při stanovování účinnosti antibiotik na mikrobiální kultury.

První kapitola této práce nazvaná Měření citlivosti na antibiotika se věnovala úvodu do problematiky oboru měření citlivosti na antibiotika a popisu současně používaných metod v tomto odvětví. Druhá kapitola s názvem Strojové učení, pojednávala o oblasti strojového učení, umělé inteligence a s nimi souvisejícími technologiemi zpracování obrazu, které využívají modely detekce objektů v obraze. Byla podrobně popsána architektura sloužící pro detekci objektů – EfficientDet.

V třetí kapitole, Návrh řešení, byl nastíněn návrh celkového postupu při implementaci webové aplikace určené pro použití v laboratořích a byly podrobněji rozebrány její klíčové prvky. Další postup byl rozdělen na dvě hlavní části: detekci antibiotických disků a měření velikosti inhibičních zón. V rámci přípravy datových sad pro první část byla provedena anotace více než 3 200 snímků diskové difúzní metody, které byly poskytnuty společností Bruker Daltonics GmbH & Co. KG. Jednotlivé snímky této sady byly pořízeny jak pomocí profesionálního laboratorního vybavení (MBT Pathfinder®), tak mobilními telefony. Anotované snímky byly rozděleny na tři části: část pro trénování, validační a testovací část. Při přípravě podkladů pro vývoj algoritmu na měření velikostí inhibičních zón byl vytvořen jednoduchý program v jazyce Python umožňující jejich anotaci. Díky dat získaných pomocí tohoto programu bylo možné v následujících krocích objektivně vyhodnotit účinnost navrženého algoritmického řešení pro měření velikostí inhibičních zón.

Čtvrtá kapitola, která byla nazvána Implementace, pojednává o vývoji řešení jednotlivých komponent pro navrhovanou aplikaci. Byly natrénovány modely detekce objektů v obraze pro detekování antibiotických disků za pomoci platformy TensorFlow, sešitů Jupyter a programovacího jazyka Python.

V rámci vyhodnocení modelů byla definována metrika mAP a byla porovnána přesnost detekce natrénovaných architektur. Model Mask R-CNN Inception ResNet V2 dosáhl mAP 94,37 % a je nejpresnější, ale také nejvíce výpočetně náročným, jelikož průměrná doba na zpracování jednoho snímku tímto modelem činí 3 896 ms. Modely z rodiny EfficientDet dosahují lepšího poměru mezi nutným výpočetním výkonem a přesností detekce. EfficientDet D0 dosáhl hodnoty mAP 87,43 % s dobou zpracování 237 ms a model D2 hodnoty 88,62 % při 454 ms. Z vyhodnocení vyplývá, že modely jsou spolehlivé a jsou vhodné pro další postup.

Při implementaci algoritmu pro měření velikostí inhibičních zón bylo využito

knihovny OpenCV v jazyce Python a výsledky algoritmu byly statisticky shrnuty. Výskyt inhibiční zóny je algoritmus schopný určit správně v 95,23 % případů a relativní chyba velikosti poloměru inhibiční zóny je 12,61 %, což je vzhledem k vysoké variabilitě vstupních dat výborný výsledek. Algoritmus je tedy vhodné integrovat do výsledné aplikace.

Poslední, pátá kapitola, nesoucí název Integrace, shrnula znalosti a postupy z předchozích kapitol do modulu webové aplikace společnosti Bruker Daltonics GmbH & Co. KG. Samotná aplikace byla vyvinuta pomocí platformy ASP.NET a programovacích jazyků C#, JavaScript a CSS. Obě klíčové části byly pro implementaci v rámci webové aplikace upraveny. Pro integraci natrénovaných modelů byl použit ekosystém ONNX. Algoritmický postup vyhodnocení velikosti inhibičních zón byl přepsán do jazyka C++, což umožnilo jeho integraci do dynamické knihovny DLL a jeho využití ve výsledné webové aplikaci.

Výsledkem celého vývoje, a tedy i celé této bakalářské práce, je funkční a přehledná webová aplikace, která je vhodná pro použití v široké škále laboratoří. To je umožněno využitím detekčních algoritmů a neuronových sítí, pomocí kterých lze ve výsledné aplikaci zpracovat různě variabilní vstupní data, včetně snímků pořízených mobilním telefonem. Tento přístup vede k dalšímu rozšíření metody a k celkovému usnadnění postupu při vyhodnocování diskových difúzních testů.

Jednotlivé postupy uvedené a později integrované do výsledného řešení byly prezentovány jako vědecký článek na studentské konferenci EEICT 2024, což mimo jiné umožnilo získat cennou zpětnou vazbu, která do této bakalářské práce mohla být zakomponována.

Literatura

- [1] MADIGAN, Michael T.; BENDER, Kelly S.; BUCKLEY, Daniel H.; SATTLE, W. Matthew a STAHL, David A. *Brock Biology of Microorganisms*. Fifteenth Edition. Pearson, 2018. ISBN 9780134261928.
- [2] KOPECKÁ, Jana a ROTKOVÁ, Gabriela. *Skripta ke cvičení z obecné mikrobiologie, cytologie a morfologie bakterií*. Online. Brno: Masarykova univerzita, 2017. ISBN 9788021087873. Dostupné z: <https://is.muni.cz/elportal/?id=1383503>. [cit. 2023-11-18].
- [3] SPÍŽEK, Jaroslav. *Boj s rezistencí na antibiotika: výzkumný program Potraviný pro budoucnost. Strategie AV21*. Praha: Středisko společných činností AV ČR, v.v.i., pro Kancelář Akademie věd ČR, 2016. ISBN 9788027031146.
- [4] BEDNÁŘ, Marek; SMÍŠEK, Jan; SCHINDLER, Jiří; NĚMEČKOVÁ, Veronika; ADÁMKOVÁ, Václava et al. *Příručka mikrobiologie pro bakaláře 3. LF UK*. Online. 2009. Dostupné z: <http://mikrobiologie.lf3.cuni.cz/mikrobiologie-nova/>. [cit. 2023-11-20].
- [5] RELLER, L. Barth; WEINSTEIN, Melvin; JORGENSEN, James H. a FERRARO, Mary Jane. Antimicrobial Susceptibility Testing: A Review of General Principles and Contemporary Practices. Online. *Clinical Infectious Diseases*. 2009, vol. 49, no. 11, s. 1749-1755. Dostupné z: <https://doi.org/https://doi.org/10.1086/647952>. [cit. 2023-11-20].
- [6] DESJARDINS, Michaël; LEFEBVRE, Brigitte; LAVALLÉE, Christian; LABBÉ, Annie-claude et al. Gradient diffusion susceptibility testing for *Neisseria gonorrhoeae*: an accurate alternative to agar dilution in high-MIC strains? Online. *Access microbiology*. 2020, vol. 2, no. 5. ISSN 2516-8290. Dostupné z: <https://doi.org/10.1099/acmi.0.000116>. [cit. 2023-11-23].
- [7] CLINICAL AND LABORATORY STANDARDS INSTITUTE. *The History of CLSI*. Online. 2023. Dostupné z: <https://clsi.org/about/clsi-s-history/>. [cit. 2023-11-21].
- [8] JORDAN, M. I. a MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. Online. *Science (American Association for the Advancement of Science)*. 2015, roč. 349, č. 6245, s. 255-260. ISSN 0036-8075. Dostupné z: <https://doi.org/10.1126/science.aaa8415>. [cit. 2023-10-15].

- [9] GOODFELLOW, Ian; BENGIO, Yoshua a COURVILLE, Aaron. *Deep learning*. Online. Online version. MIT Press, 2016. Dostupné z: <https://www.deeplearningbook.org>. [cit. 2023-10-15].
- [10] GÉRON, Aurélien. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Third edition. Sebastopol, CA: O'Reilly, 2022. ISBN 978-1-098-12597-4.
- [11] Explained: Neural networks. Online. HARDESTY, Larry. *MIT News*. 2017, vol. 14. Dostupné z: <https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414>. [cit. 2023-11-03].
- [12] BLAHA, Milan; HOLČÍK, Jiří a KOMEDA, Martin. Umělá inteligence. Online. In: *Matematická biologie: e-learningová učebnice*. Brno: Masarykova univerzita, 2015. ISBN 978-80-210-8095-9. Dostupné z: <https://portal.matematickabiologie.cz/index.php?pg=analiza-a-hodnoceni-biologickych-dat--umela-inteligence>. [cit. 2023-11-05].
- [13] BUDUMA, Nithin; PAPA, Joe a LOCASCIO, Nicholas. *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*. Second edition. Sebastopol: O'Reilly, 2022. ISBN 978-1-492-08218-7.
- [14] KARPATY, Andrej. *CS231n Convolutional Neural Networks for Visual Recognition*. Online. 2015. Dostupné z: <https://cs231n.github.io>. [cit. 2023-11-17].
- [15] VOULODIMOS, Athanasios; DOULAMIS, Nikolaos; DOULAMIS, Anastasios a PROTOPAPADAKIS, Eftychios. Deep Learning for Computer Vision: A Brief Review. Online. *Computational intelligence and neuroscience*. 2018, roč. 2018, article 7068349. ISSN 1687-5265. Dostupné z: <https://doi.org/10.1155/2018/7068349>. [cit. 2023-11-13].
- [16] TAN, Mingxing; PANG, Ruoming a LE, Quoc V. EfficientDet: Scalable and efficient object detection. Online. In: TAN, Mingxing; PANG, Ruoming a V. LE, Quoc. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Seattle, WA, USA: IEEE, 2020, s. 10778-10787. ISBN 978-1-7281-7168-5. ISSN 1063-6919. Dostupné z: <https://doi.org/10.1109/CVPR42600.2020.01079>. [cit. 2023-11-25].
- [17] ZHUANG, Fuzhen; QI, Zhiyuan; DUAN, Keyu; XI, Dongbo; ZHU, Yongchun et al. A Comprehensive Survey on Transfer Learning. Online. *Proceedings of*

- the IEEE*. 2021, vol. 109, no. 1, s. 43-76. ISSN 0018-9219. Dostupné z: <https://doi.org/10.1109/JPROC.2020.3004555>. [cit. 2023-11-24].
- [18] SKALSKI, Piotr. *Makesense.ai*. Online. 2019. Dostupné z: <https://github.com/SkalskiP/make-sense>. [cit. 2023-11-22].
- [19] ABADI, Martín; AGARWAL, Ashish; BARHAM, Paul; BREVDO, Eugene; CHEN, Zhifeng et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Online. *ArXiv.org*. 2016. Dostupné z: <https://doi.org/10.48550/arxiv.1603.04467>. [cit. 2024-03-09].
- [20] GRANGER, Brian E. a PEREZ, Fernando. Jupyter: Thinking and Storytelling With Code and Data. Online. *Computing in science & engineering*. 2021, vol. 23, no. 2, s. 7-14. ISSN 1521-9615. Dostupné z: <https://doi.org/10.1109/MCSE.2021.3059263>. [cit. 2023-11-24].
- [21] BRADSKI, Gary. The OpenCV Library. Online. *Dr. Dobb's Journal of Software Tools*. 2000, no. 25. Dostupné z: <https://www.drdobbs.com/open-source/the-opencv-library/184404319>. [cit. 2024-05-20].
- [22] *TensorFlow 2 Detection Model Zoo*. Online. 2021. Dostupné z: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md. [cit. 2023-11-25].
- [23] EVERINGHAM, M.; VAN GOOL, L.; WILLIAMS, C. K. I.; WINN, J. a ZISSERMAN, A. *PASCAL VOC2012 Challenge Results*. Online. 2012. Dostupné z: <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>. [cit. 2024-05-20].
- [24] CARTUCHO, J.; VENTURA, R. a VELOSO, M. Robust Object Recognition Through Symbiotic Deep Learning In Mobile Robots. Online. *International Conference on Intelligent Robots and Systems*. 2018, roč. 2018, s. 2336-2341. Dostupné z: <https://doi.org/10.1109/IRoS.2018.8594067>. [cit. 2024-05-20].
- [25] HUI, Jonathan. *MAP (mean Average Precision) for Object Detection*. Online. 2018. Dostupné z: <https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173>. [cit. 2023-11-27].

Seznam symbolů a zkratek

<i>AP</i>	Průměrná přesnost detekce – Average Precision
AST	Měření citlivosti na antibiotika – Antibiotic Susceptibility Testing
ATB	Antibiotika, antibiotický
BiFPN	Vážená dvousměrná pyramidová síť příznaků – Weighted Bi-Directional Feature Pyramid Network
DLL	Dynamická linkovaná knihovna – Dynamic Link Library
GPU	Grafický procesor – Graphics Processing Unit
<i>mAP</i>	Metrika pro hodnocení účinnosti detekčních modelů – Mean Average Precision
ICE	Algoritmus iterativního rozšiřování kruhu – Iterative Circle Enlarging
<i>IoU</i>	Míra překryvu mezi predikovanou oblastí a skutečnou oblastí – Intersection over Union
MIC	Minimální inhibiční koncentrace – Minimum inhibitory concentration
ReLU	Typ aktivační funkce jedné reálné proměnné – Rectified Linear Unit
ROI	Oblast zájmu – Region of Interest

A Obsah elektronické přílohy

V archivu elektronické přílohy se nalézají řada zdrojových souborů. Skript vytvořený v jazyce JavaScript, který zajišťuje podporu klávesových zkratk ve webové aplikaci `makesense.ai` ze sekce 3.1, je uveden v souboru `KeyBindings.js`. Soubory vytvořeného programu pro anotaci inhibičních zón ze sekce 3.2 jsou uvedeny v adresáři `ZoneMarker`. Zde uvedený soubor `zone_marker.py` slouží ke spuštění programu a podadresář `zm_data` obsahuje data potřebná pro jeho chod. V sešitu `ATBDiskModelTraining.ipynb` je uveden postup pro trénování modelů ze sekce 4.1. Algoritmus pro měření velikostí inhibičních zón vyvíjený v jazyce Python, který byl uvedený v sekci 4.2, se nachází v Jupyter sešitu `InhibitionZoneFinder.ipynb`. Zdrojový kód v jazyce C++, který byl použit pro vytvoření dynamické knihovny v sekci 5.2, je umístěn v adresáři `InhibitionZoneFinderDLL`. Další soubory jsou součástí vyvíjeného webového modulu, který je popsán v sekci 5.3 a nachází se v adresáři `WebAppModule`. Jednotlivé třídy pro zpracování snímků jsou v podadresáři `ASTImageProcessing`. Soubory z webové stránky vytvořené pomocí ASP.NET jsou uvedeny v podadresáři `Pages`. Kaskádové styly a JavaScriptový kód je uveden v podadresáři `wwwroot`.

```
elektronicka-priloha ..... kořenový adresář přiloženého archivu
├── ATBDiskModelTraining.ipynb
├── InhibitionZoneFinder.ipynb
├── InhibitionZoneFinderDLL
│   ├── AstDiffusionZoneFinder.cpp
│   └── AstDiffusionZoneFinder.h
├── KeyBindings.js
├── WebAppModule
│   ├── ASTImageProcessing
│   │   ├── ATBDiskDetector.cs
│   │   └── ZoneFinderWrapper.cs
│   ├── DataStructures
│   │   └── BoundingBox.cs
│   ├── Pages
│   │   ├── AST.cshtml
│   │   ├── AST.cshtml.cs
│   │   ├── AST.cshtml.FormData.cs
│   │   ├── AST.cshtml.FormDataVerifier.cs
│   │   └── AST.cshtml.FormInputHandler.cs
│   └── wwwroot
│       ├── AST.css
│       └── AST.js
└── ZoneMarker
    ├── zm_data
    └── zone_marker.py
```