

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-16607-104537

**DETEKCIA A SEGMENTÁCIA OBJEKTOV V
HISTORICKÝCH ŠIFROVANÝCH RUKOPISOCH**

DIPLOMOVÁ PRÁCA

2024

Bc. Filip Mikuš

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-16607-104537

**DETEKCIA A SEGMENTÁCIA OBJEKTOV V
HISTORICKÝCH ŠIFROVANÝCH RUKOPISOCH**
DIPLOMOVÁ PRÁCA

Študijný program: Aplikovaná informatika

Názov študijného odboru: Informatika

Školiace pracovisko: Ústav informatiky a matematiky

Vedúci záverečnej práce: Ing. Pavol Marák, PhD.

Bratislava 2024

Bc. Filip Mikuš



ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Filip Mikuš**

ID študenta: 104537

Študijný program: aplikovaná informatika

Študijný odbor: informatika

Vedúci práce: Ing. Pavol Marák, PhD.

Vedúci pracoviska: doc. Ing. Milan Vojvoda, PhD.

Názov práce: **Detekcia a segmentácia objektov v historických šifrovaných rukopisoch**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Cieľom práce je preskúmať oblasť detekcie a segmentácie objektov s malými rozmermi, špeciálne, rukou písaných znakov v historických šifrovaných dokumentoch. Na tento účel sa využijú súčasné AI modely, konkrétnie detektory z rodiny YOLOv8, zero-shot detektor SAM (Segment Anything Model) a zvolený ViT (Vision Transformer) model. Historické rukopisy predstavujú obrázky s vysokým počtom malých objektov (znaky). Z toho dôvodu je nevyhnutné použiť SAHI (Slicing Aided Hyper Inference) techniku na zvýšenie presnosti odhalenia znakov. Študent bude pracovať s vysokokvalitným a rozsiahlym datasetom znakov a symbolov, ktoré boli manuálne anotované (polygonálne anotácie vo formáte COCO a YOLO) pracovníkmi a študentmi na pracovisku ÚIM FEI STU. Výstupom práce budú natrénované vyššie spomenuté detektory, ktoré na vstupnom obraze vo vysokom rozlíšení odhalia a klasifikujú znaky a symboly. Funkcionalita detektorov bude sprístupnená aj bežným používateľom v rámci počítačovej siete pomocou klient-server API rozhrania. Táto práca sa bude podieľať na výskumnom projekte VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov.

Úlohy:

1. Preskúmajte problematiku detekcie a segmentácie objektov v obraze s dôrazom na malé objekty.
2. Zdokumentujte detektory YOLOv8, SAM a RT-DETR.
3. Zdokumentujte SAHI techniku.
4. Získajte a analyzujte dataset znakov a symbolov z historických šifrovaných rukopisov.
5. Natrénujte jednotlivé detektory na získanom datasete.
6. Pomocou FastAPI frameworku implementujte web rozhranie na detekciu objektov na vzdialenom ML (machine learning) serveri a tzv. data explorer pre vyhľadávanie v datasete pomocou SQL dopytov.
7. Vyhodnoťte úspešnosť detekcie a segmentácie jednotlivých detektorov, pričom za účelom zlepšenia inferencie využite SAHI techniku.
8. Exportujte natrénované detektory vo formáte ONNX (Open Neural Network Exchange).
9. Vytvorte písomnú dokumentáciu.

Zoznam odbornej literatúry:

1. AKYON, F. C. et al. Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection. 2022 IEEE International Conference on Image Processing (ICIP), <http://dx.doi.org/10.1109/ICIP46576.2022.9897990>
2. ANTAL, E. – MARÁK, P. 2023. Automated transcription of historical encrypted manuscripts. In Tatra Mountains Mathematical Publications, vol. 82, no.2, pp. 65-86. 1210-3195. DOI: <https://doi.org/10.2478/tmmp-2022-0019>

Termín odovzdania diplomovej práce: 10. 05. 2024
Dátum schválenia zadania diplomovej práce: 09. 02. 2024
Zadanie diplomovej práce schválil: prof. Dr. Ing. Miloš Oravec – garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:

Aplikovaná informatika

Autor:

Bc. Filip Mikuš

Diplomová práca:

Detekcia a segmentácia objektov v historických
šifrovaných rukopisoch

Vedúci záverečnej práce:

Ing. Pavol Marák, PhD.

Miesto a rok predloženia práce:

Bratislava 2024

S narastajúcim počtom dostupných digitalizovaných historických dokumentov v archívoch, vývojom výpočtového hardvéru a modelov strojového učenia (ML) sa zintenzívňuje snaha o automatizované spracovanie historických šifrovaných dokumentov. Tento proces prácu s historickými prameňmi urýchľuje aplikovaním automatizovanej transkripcie a detektie/segmentácie symbolov ML modelmi. Cieľom tejto práce bolo oboznámiť sa s podobou historických šifrovaných rukopisov, zdokumentovať, dotrénovať a otestovať najmodernejšie detekčné/segmentačné modely, respektíve techniky a zahrnúť ich do webového aplikačného rozhrania (API). Na základe získaných poznatkov sme na poskytnutých datasetoch číslic a symbolov (glyfov) natrénovali a otestovali modely YOLOv8, YOLOv9, YOLO-NAS, YOLO-Worldv2, RT-DETR a FastSAM. Na zlepšenie detektie malých objektov sme využili techniku SAHI (Slicing Aided Hyper Inference). Natrénované modely boli exportované do univerzálneho ONNX formátu, ktorého inferenčné časy boli porovnané s natívnym exportným formátom PyTorch. Na automatizovanú transkripciu detegovaných symbolov sme navrhli a implementovali vlastný transkripčný mechanizmus s využitím strojového učenia bez učiteľa (UL). Natrénované vyexportované modely a mechanizmus na automatizovanú transkripciu s využitím UL sme spolu s nástrojmi na automatizované generovanie anotácií s využitím ML a na pokročilé prehľadávanie datasetov s využitím ML zahrnuli do nami navrhnutého a implementovaného webového API typu REST. Výsledné API sme otestovali pomocou HTTP testov a záťažových (performance) testov.

Kľúčové slová: historické šifrované rukopisy, nomenklátory, detekcia, segmentácia, transkripcia, číslice, glyfy, umelá inteligencia, strojové učenie, YOLO, DETR, SAM, SAHI, ONNX, webové REST API

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Bc. Filip Mikuš
Master's thesis:	Object detection and segmentation in historical encrypted manuscripts
Supervisor:	Ing. Pavol Marák, PhD.
Place and year of submission:	Bratislava 2024

With the increasing number of digitized historical encrypted documents available in archives, the development of computing hardware and machine learning (ML), efforts to automate the processing of historical encrypted documents are intensifying. This process speeds up the work with historical sources by applying automated transcription using detection and segmentation of symbols by ML models. The aim of our work was to study the form of historical encrypted manuscripts, to document, train and test state-of-the-art detection/segmentation models or techniques, and integrate them into a web application programming interface (API). Based on the collected theoretical knowledge, we trained and tested the detection/segmentation models YOLOv8, YOLOv9, YOLO-NAS, YOLO-Worldv2, RT-DETR and FastSAM on the provided datasets of digits and symbols (glyphs). We have used the SAHI technique (Slicing Aided Hyper Inference) to improve the detection of small objects. The trained models were exported to the universal ONNX format, whose inference times were compared with the native PyTorch export format. For automated transcription of the detected symbols, we designed and implemented a custom transcription mechanism using unsupervised learning (UL). We integrated the trained exported models and the mechanism for automated transcription using UL, along with tools for automated annotation generation using ML and for advanced dataset browsing using ML, into a REST web API that we designed and implemented. We tested the API using HTTP tests and performance tests.

Keywords: historical encrypted manuscripts, nomenclators, detection, segmentation, transcription, digits, glyphs, artificial intelligence, machine learning, YOLO, DETR, SAM, SAHI, ONNX, web REST API

Vyhľásenie

Ja, dolu podpísaný Bc. Filip Mikuš čestne vyhlasujem, že som diplomovú prácu Detektia a segmentácia objektov v historických šifrovaných rukopisoch vypracoval samostatne na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Uvedenú prácu som vypracoval pod vedením Ing. Pavla Maráka, PhD.

Bratislava, dňa 10.5.2024

.....

podpis autora

Podakovanie

Chcem sa podakovať vedúcemu záverečnej práce, ktorým bol Ing. Pavol Marák, PhD., za odborné vedenie, rady a pripomienky, ktoré mi pomohli pri vypracovaní tejto diplomovej práce.

Ďalej by som sa chcel podakovať vedúcemu výskumného projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov, Ing. Eugenovi Antalovi, PhD., vďaka ktorému nám boli poskytnuté materiály a dátá pre náš výskum.

Obsah

Úvod	1
1 Historické rukopisy	2
1.1 Historické šifrované rukopisy	2
1.2 Nomenklátor	2
1.3 Zbierka historických rukopisov	5
2 Spracovanie historických šifrovaných rukopisov	7
2.1 Digitalizácia historických šifrovaných rukopisov	8
2.2 Detekcia a segmentácia symbolov v historických šifrovaných rukopisoch	8
2.3 Transkripcia detegovaných symbolov v historických šifrovaných rukopisoch	9
3 Detekcia a segmentácia objektov	10
3.1 Detekcia a segmentácia	10
3.1.1 YOLOv8	11
3.1.2 YOLO-NAS	13
3.1.3 YOLOv9	14
3.1.4 RT-DETR	15
3.2 Zero-Shot (Promptable) segmentácia	17
3.2.1 SAM	19
3.2.2 FastSAM	20
3.3 Open-Vocabulary detekcia	21
3.3.1 YOLO-World	21
3.4 Detekcia a segmentácia malých objektov	22
3.4.1 SAHI	24
4 Analýza a návrh riešenia	25
4.1 Požiadavky	25
4.1.1 Funkcionálne požiadavky	25
4.1.2 Nefunkcionálne požiadavky	25
4.2 Datasetsy	26
4.2.1 Číslice	28
4.2.2 Glyfy	29

4.3	Modely	31
4.3.1	Metriky	31
4.3.2	ONNX formát	33
4.4	Mechanizmus na automatizovanú transkripciu	35
4.5	Webové API	36
4.6	Volba nástrojov	41
5	Implementácia a dosiahnuté výsledky	42
5.1	Modely	42
5.1.1	YOLOv8	42
5.1.2	YOLO-NAS	47
5.1.3	YOLOv9	51
5.1.4	RT-DETR	55
5.1.5	YOLO-World	59
5.1.6	FastSAM	64
5.1.7	SAHI	65
5.1.8	Porovnanie rýchlosťi inferencie modelov vo formátoch PyTorch a ONNX	67
5.2	Nástroj na automatizovanú transkripciu symbolov	69
5.3	Nástroj na automatizovanú anotáciu obrázkov	71
5.4	Nástroj na prehľadávanie datasetov s využitím AI	72
5.4.1	Prehľadávanie na základe podobnosti	72
5.4.2	Prehľadávanie na základe SQL dopytu	73
5.4.3	Prehľadávanie na základe textového dopytu	74
5.5	Webové API	74
5.5.1	Architektúra API	75
5.5.2	Prístupové body API	77
5.5.3	Testovanie API	78
Záver		80
Zoznam použitej literatúry		82
Prílohy		I
A Štruktúra elektronickej prílohy s praktickým výstupom		II

B Používateľská príručka	IV
C Ukážky výstupov detekcie na celých historických rukopisoch pri využití YOLOv8 + SAHI	V
D Prehľad úspešnosti detekčných modelov	XI

Zoznam obrázkov a tabuliek

Obrázok 1	Historický rukopis obsahujúci nomenklátorový kľúč	3
Obrázok 2	Historický rukopis obsahujúci šifrovaný text	4
Obrázok 3	Ukážka časti historického rukopisu s otvoreným textom napísaným pod riadkami šifrovaného textu	5
Obrázok 4	Ukážka časti historického rukopisu so šifrovaným textom	6
Obrázok 5	Ukážka časti historického rukopisu so šifrovaným textom	6
Obrázok 6	Ukážka časti historického rukopisu (denníka) so šifrovaným . .	6
Obrázok 7	Schéma automatizovaného spracovania historických šifrovaných rukopisov	9
Obrázok 8	Ukážka detekcie a segmentácie číslíc (vľavo) a glyfov (vpravo) dotrénovanými modelmi YOLOv8 v historických šifrovaných rukopisoch	10
Obrázok 9	Schéma architektúry YOLOv8	12
Obrázok 10	Schéma architektúry YOLO-NAS	13
Obrázok 11	Schéma architektúry YOLOv9	14
Obrázok 12	Schéma architektúry siete GELAN modelu YOLOv9	15
Obrázok 13	Schéma architektúry RT-DETR	16
Obrázok 14	Ukážka zero-shot segmentácie číslíc (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Segment everything mód	17
Obrázok 15	Ukážka zero-shot promptable segmentácie číslíc (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Box prompt mód	18
Obrázok 16	Ukážka zero-shot promptable segmentácie číslíc (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Point prompt mód	18
Obrázok 17	Schéma architektúry SAM	19
Obrázok 18	Schéma architektúry FastSAM	20
Obrázok 19	Schéma architektúry YOLO-World	21
Obrázok 20	Ukážka detekcie malých číslíc v historických šifrovaných rukopisoch dotrénovaným modelom YOLOv8 (vľavo) a dotrénovaným modelom YOLOv8 s využitím techniky SAHI (vpravo)	23

Obrázok 21	Schéma fungovania konceptu SAHI	24
Obrázok 22	Ukážka ručného kreslenia anotácií	26
Obrázok 23	Ukážka vzoriek VEGA datasetov číslic (vľavo) a glyfov (vpravo)	27
Obrázok 24	Ukážka vzoriek VEGA Cropped datasetov číslic (hore) a glyfov (dole)	27
Obrázok 25	Graf distribúcie tried (10) anotácií číslic v trénovacej množine .	28
Obrázok 26	Graf distribúcie tried (10) anotácií číslic vo validačnej množine .	28
Obrázok 27	Graf distribúcie tried (10) anotácií číslic v testovacej množine .	29
Obrázok 28	Graf distribúcie tried (162) anotácií glyfov v trénovacej množine	29
Obrázok 29	Graf distribúcie tried (162) anotácií glyfov vo validačnej množine	30
Obrázok 30	Graf distribúcie tried (162) anotácií glyfov v testovacej množine	30
Obrázok 31	Schéma výpočtového grafu formátu ONNX	34
Obrázok 32	Schéma exportu a importu modelu vo formáte ONNX	34
Obrázok 33	Vývojový diagram algoritmu na zoskupenie detegovaných symbolov do riadkov s využitím DBSCAN zhľukovania	36
Obrázok 34	Diagram komponentov webového ML REST API	36
Obrázok 35	Diagram aktivít funkcionality detekcie a segmentácie objektov webového ML REST API	37
Obrázok 36	Diagram aktivít funkcionality detekcie a transkripcie objektov webového ML REST API	38
Obrázok 37	Diagram aktivít funkcionality automatizovaného generovania anotácií datasetov webového ML REST API	39
Obrázok 38	Diagram aktivít funkcionality analýzy datasetov webového ML REST API	40
Obrázok 39	Ukážka detekcie a segmentácie číslic modelom YOLOv8 na testovacích vzorkách VEGA Cropped	43
Obrázok 40	Ukážka detekcie a segmentácie glyfov modelom YOLOv8 na testovacích vzorkách VEGA Cropped	43
Obrázok 41	Konfúzna matica modelu YOLOv8 na testovacej množine číslic datasetu VEGA Cropped	44
Obrázok 42	Konfúzna matica modelu YOLOv8 na testovacej množine glyfov datasetu VEGA Cropped	45
Obrázok 43	Ukážka detekcie číslic modelom YOLOv8 na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI)	46

Obrázok 44	Ukážka detekcie číslíc modelom YOLO-NAS na testovacích vzorkách VEGA Cropped	47
Obrázok 45	Ukážka detekcie glyfov modelom YOLO-NAS na testovacích vzorkách VEGA Cropped	48
Obrázok 46	Konfúzna matica modelu YOLO-NAS na testovacej množine číslic datasetu VEGA Cropped	49
Obrázok 47	Konfúzna matica modelu YOLO-NAS na testovacej množine glyfov datasetu VEGA Cropped	49
Obrázok 48	Ukážka detekcie číslíc modelom YOLO-NAS na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI) . .	50
Obrázok 49	Ukážka detekcie a segmentácie číslic modelom YOLOv9 na testovacích vzorkách VEGA Cropped	51
Obrázok 50	Ukážka detekcie a segmentácie glyfov modelom YOLOv9 na testovacích vzorkách VEGA Cropped	52
Obrázok 51	Konfúzna matica modelu YOLOv9 na testovacej množine číslic datasetu VEGA Cropped	53
Obrázok 52	Konfúzna matica modelu YOLOv9 na testovacej množine glyfov datasetu VEGA Cropped	53
Obrázok 53	Ukážka detekcie číslic modelom YOLOv9 na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI)	54
Obrázok 54	Ukážka detekcie číslic modelom RT-DETR na testovacích vzorkách VEGA Cropped	55
Obrázok 55	Ukážka detekcie glyfov modelom RT-DETR na testovacích vzorkách VEGA Cropped	56
Obrázok 56	Konfúzna matica modelu RT-DETR na testovacej množine číslic datasetu VEGA Cropped	57
Obrázok 57	Konfúzna matica modelu RT-DETR na testovacej množine glyfov datasetu VEGA Cropped	57
Obrázok 58	Ukážka detekcie číslic modelom RT-DETR na VEGA vzorke celého historického dokumentu - model nedetegoval žiadnu číslicu (bez použitia techniky SAHI)	58
Obrázok 59	Ukážka detekcie číslic modelom YOLO-Worldv2 na testovacích vzorkách VEGA Cropped	60

Obrázok 60	Ukážka detekcie glyfov modelom YOLO-Worldv2 na testovacích vzorkách VEGA Cropped	60
Obrázok 61	Konfúzna matica modelu YOLO-Worldv2 na testovacej množine číslic datasetu VEGA Cropped	61
Obrázok 62	Konfúzna matica modelu YOLO-Worldv2 na testovacej množine glyfov datasetu VEGA Cropped	62
Obrázok 63	Ukážka detekcie číslic modelom YOLO-Worldv2 na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI) . .	63
Obrázok 64	Ukážka zero-shot segmentácie číslic modelom FastSAM na testovacích vzorkách VEGA Cropped	64
Obrázok 65	Ukážka zero-shot segmentácie glyfov modelom FastSAM na testovacích vzorkách VEGA Cropped	64
Obrázok 66	Ukážka detekcie číslic modelom YOLOv8 (vľavo) a riešenia SAHI + YOLOv8 (vpravo) na VEGA vzorke celého historického dokumentu	66
Obrázok 67	Graf porovnania inferenčných časov modelu vo formátoch PyTorch a ONNX	68
Obrázok 68	Ukážka transkripcie číslic (vpravo) detegovaných modelom YOLOv8 na testovacej vzorke VEGA Cropped s mierne rotovanými riadkami	69
Obrázok 69	Ukážka transkripcie číslic (vpravo) detegovaných riešením SAHI + YOLOv8 na testovacej VEGA vzorke celého historického dokumentu	70
Obrázok 70	Ukážka konfigurácie funkcionality na automatizované generovanie anotácií	71
Obrázok 71	Ukážka automatizovane vygenerovaných anotácií funkctionalitou Annotator	71
Obrázok 72	Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe podobnosti	73
Obrázok 73	Ukážka vyhľadaných vzoriek (v strede, vpravo) podobných k pôvodnej vzorke (vľavo) funkctionalitou Explorer	73
Obrázok 74	Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe SQL dopytu	73

Obrázok 75	Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe textového dopytu	74
Obrázok 76	Diagram tried webového ML REST API	76
Obrázok 77	Graf porovnania reakčných časov koncových bodov webového REST API	79
Obrázok C.1	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 1 - YOLOv8 + SAHI	V
Obrázok C.2	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 2 - YOLOv8 + SAHI	VI
Obrázok C.3	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 3 - YOLOv8 + SAHI	VII
Obrázok C.4	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 4 - YOLOv8 + SAHI	VIII
Obrázok C.5	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 5 - YOLOv8 + SAHI	IX
Obrázok C.6	Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 6 - YOLOv8 + SAHI	X
Tabuľka 2	Úspešnosť modelu YOLOv8l-seg na testovacej množine referenčného datasetu COCO	43
Tabuľka 3	Úspešnosť modelu YOLOv8 na testovacej množine číslic datasetu VEGA Cropped	44
Tabuľka 4	Úspešnosť modelu YOLOv8 na testovacej množine glyfov datasetu VEGA Cropped	44
Tabuľka 5	Úspešnosť modelu YOLO-NAS-l na testovacej množine referenčného datasetu COCO	47
Tabuľka 6	Úspešnosť modelu YOLO-NAS na testovacej množine číslic datasetu VEGA Cropped	48
Tabuľka 7	Úspešnosť modelu YOLO-NAS na testovacej množine glyfov datasetu VEGA Cropped	48
Tabuľka 8	Úspešnosť modelu YOLOv9c-seg na testovacej množine referenčného datasetu COCO	51
Tabuľka 9	Úspešnosť modelu YOLOv9 na testovacej množine číslic datasetu VEGA Cropped	52

Tabuľka 10	Úspešnosť modelu YOLOv9 na testovacej množine glyfov datasetu VEGA Cropped	52
Tabuľka 11	Úspešnosť modelu RT-DETRl na testovacej množine referenčného datasetu COCO	55
Tabuľka 12	Úspešnosť modelu RT-DETR na testovacej množine číslic datasetu VEGA Cropped	56
Tabuľka 13	Úspešnosť modelu RT-DETR na testovacej množine glyfov datasetu VEGA Cropped	56
Tabuľka 14	Úspešnosť modelu YOLO-Worldv2l na testovacej množine referenčného datasetu COCO	59
Tabuľka 15	Úspešnosť modelu YOLO-Worldv2 pri dynamickej špecifikácii tried na testovacej množine číslic datasetu VEGA Cropped . . .	59
Tabuľka 16	Úspešnosť modelu YOLO-Worldv2 pri dynamickej špecifikácii tried na testovacej množine glyfov datasetu VEGA Cropped . .	59
Tabuľka 17	Úspešnosť modelu YOLO-Worldv2 na testovacej množine číslic datasetu VEGA Cropped	61
Tabuľka 18	Úspešnosť modelu YOLO-Worldv2 na testovacej množine glyfov datasetu VEGA Cropped	61
Tabuľka 19	Úspešnosť modelu VFNet a riešenia SAHI + VFNet na testovacej množine referenčného datasetu xView	65
Tabuľka 20	Porovnanie inferenčných časov modelu YOLOv8 a riešenia SAHI + YOLOv8 na vzorke celého historického dokumentu datasetu VEGA	67
Tabuľka 21	Porovnanie inferenčných časov modelu vo formátoch PyTorch a ONNX	68
Tabuľka 22	Porovnanie reakčných časov koncových bodov webového REST API	79

Zoznam skratiek

AI	Artificial Intelligence
AIFI	Attention based Intra Scale Feature Interaction
AIS	All-instance Segmentation
AP	Average Precision
API	Application Programming Interface
AutoNAC	Automated Neural Architecture Construction
B	Byte
CCFM	CNN based Cross-scale Feature-fusion Module
CLIP	Contrastive Language-Image Pre-training
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DETR	Detection Transformer
Fast R-CNN	Fast Region based Convolutional Neural Network
Faster R-CNN	Faster Region based Convolutional Neural Network
FastSAM	Fast Segment Anything Model
FN	False Negative
FP	False Positive
FPGA	Field Programmable Gate Array
FPN	Feature Pyramid Network
GELAN	Generalized Efficient Layer Aggregation Network
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IoU	Intersection over Union
M	Mega
mAP	Mean Average Precision

Mask R-CNN	Mask Region based Convolutional Neural Network
ML	Machine Learning
MVC	Model-View-Controller
NAS	Neural Architecture Search
NLP	Natural Language Processing
NMS	Non-Maximum Suppression
NPU	Neural Processing Unit
ONNX	Open Neural Network Exchange
OVD	Open Vocabulary Object Detection
P	Precision
PAN	Path Aggregation Network
PGI	Programmable Gradient Information
PGS	Prompt-Guided Selection
px	pixel
R	Recall
R-CNN	Region based Convolutional Neural Network
REST	Representational State Transfer
RT-DETR	Real-Time Detection Transformer
SA-1B	Segment Anything 1 Billion Mask
SAHI	Slicing Aided Hyper Inference
SAM	Segment Anything Model
SOTA	State of the Art
SSD	Single Shot Detector
TP	True Positive
TPU	Tensor Processing Unit
TSD	Two Shot Detector
UL	Unsupervised Learning
VFNet	Varifocal Net
ViT	Vision Transformer
YOLO	You Only Look Once
YOLO-NAS	You Only Look Once-Neural Architecture Search
YOLO-World	You Only Look Once-World

ÚIM FEI STU Ústav informatiky a matematiky, Fakulta elektrotechniky a informatiky Slovenskej technickej univerzity v Bratislave

Úvod

Šifrovanie ako prostriedok pre dosiahnutie ochrany dôvernosti a utajenia informácií, má hlboké historické korene. Už v starovekom Egypte, Grécku, Mezopotámii alebo Ríme boli využité rôzne techniky utajenia citlivých informácií pomocou rozličných šifrovacích systémov. Šifrovaním boli často chránené strategické informácie, osobné údaje alebo citlivá komunikácia.

V súčasnej dobe digitalizácie a informatizácie, kedy disponujeme digitalizovanými dokumentami historického dobového charakteru, sa objavujú nové široké možnosti na ich analýzu a interpretáciu. Medzi tieto zámery patrí aj transkripcia a následné lúštenie šifrovaných historických dokumentov. S rozmachom informačných technológií je však možné tieto procesy rádovo urýchliť a zefektívniť.

Výskumný projekt VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov, do ktorého je zapojená aj naša práca, je zameraný práve na analýzu, návrh a implementáciu systému na spracovanie digitalizovaných historických šifrovaných archívnych dokumentov (rukopisov).

Vzhľadom na skutočnosť, že pre spracovanie a analýzu týchto typov dokumentov je nevyhnutná detekcia alebo segmentácia objektov, ako číslice alebo špecifické symboly (glyfy), bolo primárnym cieľom našej práce natrénovanie a porovnanie najaktuálnejších a najpokročilejších (SOTA) detekčných a segmentačných modelov strojového učenia (ML), respektíve počítačového videnia (CV) a umelej inteligencie (AI).

Po zmapovaní problematiky a analýze súčasného stavu sme natrénovali a porovnali výsledky rôznych modelov, ako napríklad YOLOv8, YOLO-NAS, YOLOv9, YOLO-World, RT-DETR alebo FastSAM. Pre zlepšenie detekcie a segmentácie malých objektov sme použili techniku SAHI.

Modely sme exportovali do univerzálneho formátu pre hlboké neurónové siete (ONNX), ktorých inferenčné časy sme porovnali so štandardným exportným formátom PyTorch.

Na automatizovanú transkripciu detegovaných symbolov sme navrhli a implementovali mechanizmus s využitím techniky strojového učenia bez učiteľa (UL).

Najrelevantnejšie natrénované modely a implementovaný mechanizmus na automatizovanú transkripciu s využitím UL sme následne zahrnuli do webového aplikačného rozhrania (API), ktoré sme rozšírili nástrojmi na automatizované generovanie anotácií s využitím ML a pokročilé prehľadávanie datasetov s využitím ML. Webové API sme napokon podrobili testovaniu pomocou testov typu HTTP a záťažových testov.

1 Historické rukopisy

V súčasnosti sa v archívoch a knižniciach po celej Európe vďaka rozmachu informačných technológií a s ním spojenou digitalizáciou dokumentov na prelome milénii nachádza veľké množstvo historických rukopisov rôzneho charakteru. Tieto rukopisy môžu byť rôzneho typu, okrem základného uchovania informácií môžu plniť korešpondenčnú funkciu rôzneho druhu od súkromnej komunikácie až po diplomatickú alebo vojenskú [1].

1.1 Historické šifrované rukopisy

Vzhľadom na skutočnosť, že informácie a dátá obsiahnuté v historických dokumentoch a rukopisoch môžu byť súkromné alebo citlivé, sú na ich utajenie často použité rôzne kryptosystémy a šifry. Forma šifrovaných rukopisov bola teda väčšinou odlišná od tých bežných, no zároveň bola často vo vyššej miere štruktúrovaná, čo bolo priamym dôsledkom typu konkrétneho šifrovacieho systému [2, 1].

Samotné šifrovanie reprezentuje proces transformácie dát alebo informácií z čitateľnej formy na nečitatelnú formu. Proces na spätnú transformáciu, teda z nečitatelnej formy na čitateľnú formu, nazývame dešifrovanie. Skúmaním týchto metodík sa zaoberá vedný odbor kryptológia, ktorá sa následne delí na kryptografiu a kryptoanalýzu. Kryptografia sa zameriava na návrh systémov utajenia, pričom kryptoanalýza sa sústredí na narušenie takýchto systémov [2, 3, 4].

1.2 Nomenklátory

Pojem nomenklátor v kryptológií reprezentuje substitučný kryptosystém využívajúci rôzne jednoduchšie prístupy šifrovania, používaný najmä v období medzi 14. a 19. storočím. Nomenklátory vznikli rozšírením jednoduchých alebo homofónnych, bigramových a trigramových substitúcií o kódové knihy alebo klamače, pomocou ktorých boli šifrované citlivé osobné údaje panovníkov a historických predstaviteľov [2, 3, 4, 5].

Klasické šifry sa vyznačujú jednoduchým šifrovaním a dešifrovaním v rôznych podmienkach a situáciách, pričom pod pojmom substitučné šifry rozumieme kryptosystém fungujúci na báze nahradenia (substitúcie) znakov otvoreného textu pri dodržaní stanovených pravidiel [2, 3, 4, 5].

Pravidlá substitučného šifrovania a dešifrovania sú v prípade nomenklátorov uvedené v nomenklátorových kľúčoch [2, 3, 4, 5].

Napriek širokej variabilite foriem a štruktúr sú pre nomenklátorové klúče charakteristické znaky [1, 5, 6]:

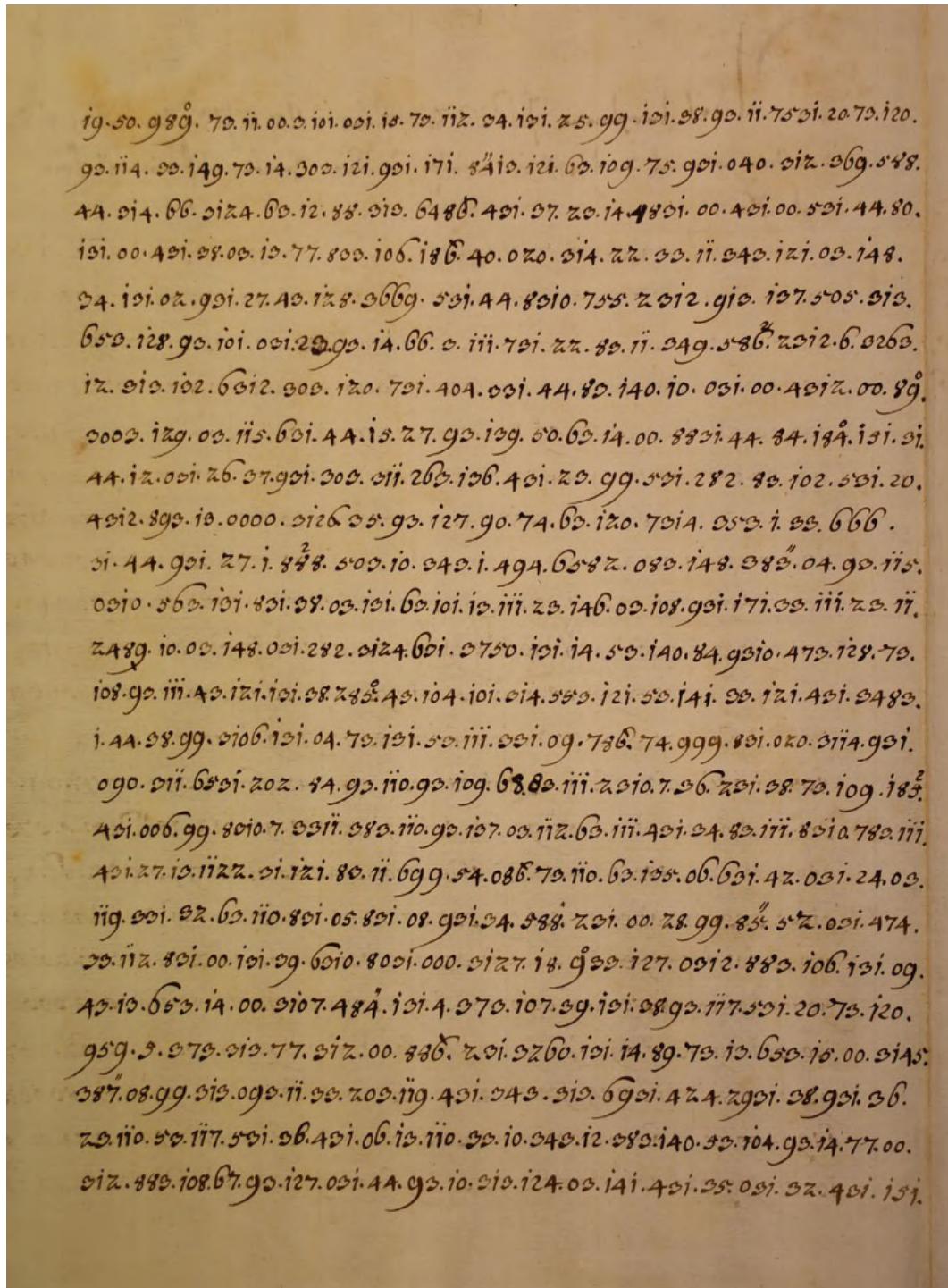
- šifrovací kľúč je zakreslený na celom hárku,
- jednotlivé šifrovacie subsystémy sú graficky oddelené,
- znaky šifrovaného textu sú reprezentované pomocou číslic, písmen, špecifických symbolov (glyfov) a ich kombináciami,
- šifrovací kľúč zväčša obsahuje kódy, kódové skupiny alebo celé kódové knihy,
- šifrovací kľúč môže obsahovať klamače,
- šifrovací kľúč v niektorých prípadoch obsahuje okrem časti na šifrovanie aj časť na dešifrovanie.

A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
21	29	36	23	41	26	35	42	20	31	40	44	27	37	22	43	32	24	38	34	28	39	33	25	
45	52	67	59	51	66	46	58	53	61	47	57	30	50	56	64	54	48	62	55	65	63	74	49	
60	84	91	77	70	90	85	89	75	78	68	86	71	69	79	72	80	87	73	92	88	81	93	82	
76	98	87		83	99	11		94	16	13	10	95	12	19	14	96	15	97					18	
dd	bb	oo	aa	ff	cc	gg	ee	kk	ll	ii	qq	rr	nn	tt	pp	ss	xx	yy	zz	uu	ww			
ll	ei	eu	ie	iu	oi	re	si	sp	ox	ll	nn	rr	zz	tt	pp	ss	hh	uu	cl	cc	tz			
K	W	Ø	♀	X	Ø	Ø	♂	Ø	†	♂	♀	Ø	Ø	Ø	Ø	*	+	+	+	+	+	+	+	
Ab	--	a		Sas	--	hh		Ray	--	o		Naam.	uuu		zi	--	3							
abov	--	b		fan	--	ii		Naan.	--	o		enil.	www		zi	--	4							
abo	--	c		fin	--	KKK		niel	--	z		enib.	xxx		zi	--	5							
om	--	d		gar	--	lll		nof	--	z		enib.	yyy		ziik	--	6							
an	--	e		gant	--	mm		num	--	s		enir	eee		zuar	--	7							
ang	--	f		gong	--	nnn		nur	--	t		enir	zzz		zuar	--	8							
ant	--	g		gan	--	ooo		ob	--	u		enir	...		zuar	--								
ant	--	h		gong	--	ppp		ofe	--	v		wot	...											

Obr. 1: Historický rukopis obsahujúci nomenklátorový kľúč [1, 7]

Aj napriek vysokej popularite so sebou kryptosystém nomenklátor prinášal aj negatíva. Prvotný zlý návrh celého šifrovacieho systému často komplikoval samotný proces šifrovania a dešifrovania, pričom ich špecifická forma a štruktúra zapríčinila, že zmena alebo rozšírenie kľúča bolo veľmi nákladné a zdĺhavé [1, 5, 8].

Vyššie spomenuté skutočnosti a charakteristiky však prispeli k širokému využitiu nomenklátorov, teda aj k súčasnemu logickému úsiliu tento typ systému analyzovať a interpretovať.



Obr. 2: Historický rukopis obsahujúci šifrovaný text [1, 7]

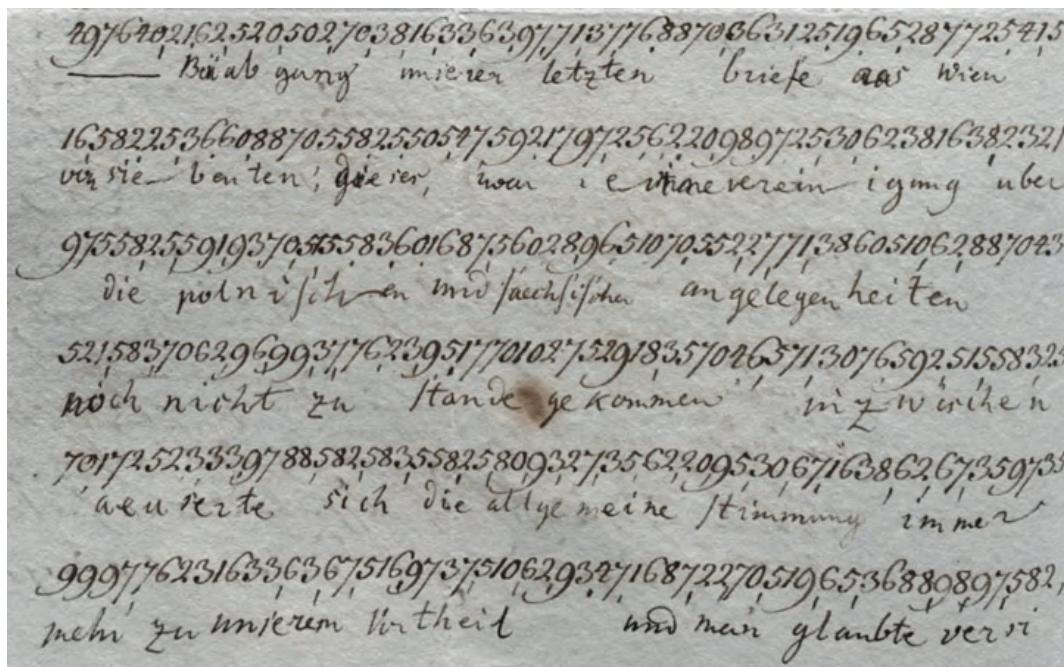
1.3 Zbierka historických rukopisov

V našej práci pracujeme so zbierkou niekolko sto strán historických šifrovaných rukopisov a šifrovacích kľúčov dostupnou v rámci výskumného projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov [7].

Zbierka historických rukopisov pochádza zo Slovenského národného archívu z 3 rôznych fondov šľachtických rodov, konkrétnie Esterházi, Pálffy-Daun a Amade-Üchtritz [7].

Zozbierané šifrované rukopisy je podľa ich štruktúry tiež možné rozdeliť do typov [7]:

- úplne alebo čiastočne zašifrované správy,
- zašifrované správy, kde je otvorený text napísaný nad/pod riadkami šifrového textu,
- zašifrované časti v denníkoch,
- koncepty správ obsahujúce zašifrované časti,
- koncepty správ obsahujúce zašifrované časti, kde je otvorený text nad/pod riadkami šifrového textu.



Obr. 3: Ukážka časti historického rukopisu s otvoreným textom napísaným pod riadkami šifrovaného textu [7]

019. 104. 93. 11. 202. 111. 22. 103. 48. 201. 28. 28. 66. 87. 201. 22. 02. 11. 653. 109.
67. 90. 118. 601. 04. 101. 149. 118. 101. 3729. 116. 80. 111. 42. 121. 11. 93. 121. 18. 77.
801. 44. 121. 3120. 126. 021. 22. 82. 136. 20. 10. 99. 20. 11. 360. 1. 44. 120. 148. 288.
66. 202. 71. 202. 2012. 2101. 00. 301. 293. 0. 111. 787. 888. 99. 82. 11. 693. 105.

Obr. 4: Ukážka časti historického rukopisu so šifrovaným textom [7]

87561466670024088003686481608890165585909630899
726783466696666260162660283044639854166666567777
68501650919311850009470511684540271188980158603062
67196623279107020981992370114151244120208176994
636891020819019059105999990908289391237825441

Obr. 5: Ukážka časti historického rukopisu so šifrovaným textom [7]

Thn 28 ^{thu} May.
Zelln ist nun zuerst nach dem alten Fuß; ich bin jedoch zu niedrig
77503557143443193535327118219443441341931312747 und fahre daher
332715256813313119352725431935 fußlos mit. Lynden haben wir nun sehr
viele neue und frische Pflanzen gesehen.

Thn 1 ^{thu} June.
Benzel ist bei allen früheren Besuchten und Reisen kein Tropf
mit Kompass und Karte; die Welt wird nur in den Kreiseln und

Obr. 6: Ukážka časti historického rukopisu (denníka) so šifrovaným textom [7]

2 Spracovanie historických šifrovaných rukopisov

Na prelome milénia, paralelne s pokrokom v informatizácii a informačných technológiach, sa začala vyvíjať snaha a úsilie o digitalizáciu historických archívnych rukopisov. S narastajúcim počtom dostupných digitalizovaných šifrovaných dokumentov historického charakteru v archívoch alebo iných výskumných inštitúciach, vývojom hardvéru a prelomovými poznatkami v oblastiach, ako je kryptoanalýza, počítačové videnie alebo strojové učenie sa tiež objavila iniciatíva na automatizované spracovanie historických šifrovaných rukopisov [2, 9, 7].

Napriek mňovým pokrokom v spomenutých oblastiach sa nejedná o triviálny problém a aj napriek výskumu danej problematiky, v súčasnosti stále neexistuje komplexné univerzálne riešenie, ktoré by naplnilo špecifické požiadavky konkrétneho účelu využitia [9, 7, 10, 11].

Systémy na spracovanie historických šifrovaných rukopisov môžu plniť odlišné úlohy a mať rôznu štruktúru a architektúru. Od systémov na digitalizáciu a manuálne spracovanie, ako napríklad DECRYPT alebo HCPortal až po prvé pokusy o automatizované spracovanie, ktoré proces pri rozsiahlejšej kolekcii rukopisov môže zrýchliť a zefektívniť [9, 7, 10, 11, 12, 13].

Automatizované spracovanie historických šifrovaných rukopisov vo svojej elementárnej podobe proces urýchluje aplikovaním automatizovanej transkripcie, s využitím detekcie a segmentácie symbolov modelmi strojového učenia (ML). Architektúra systému automatizovaného spracovania historických šifrovaných rukopisov vo väčšine prípadov pozostáva z nasledovných krokov [7, 10, 11]:

- digitalizácia historických šifrovaných rukopisov,
- detekcia a segmentácia symbolov v historických šifrovaných rukopisoch,
- transkripcia detegovaných symbolov v historických šifrovaných rukopisoch.

Funkcionalitu a prípady využitia je však možné pomocou strojového učenia ďalej rozšíriť aj na iné úlohy, ako napríklad [1, 9, 7]:

- klasifikácia typov historických šifrovaných rukopisov,
- detekcia a segmentácia častí rozloženia historických šifrovaných rukopisov.

2.1 Digitalizácia historických šifrovaných rukopisov

Digitalizácia historických šifrovaných rukopisov reprezentuje proces prevodu fyzických historických dokumentov a listín do digitálnej obrazovej podoby, napríklad fotografie [2, 14].

Digitalizácia historických dokumentov so sebou prináša nezanebatelné výhody a superlatíva oproti fyzickým médiám, ako napríklad zachovanie kultúrneho dedičstva, zastavenie dodatočnej degradácie dokumentov, zníženie a minimalizovanie nákladov na konzerváciu dokumentov alebo sprístupnenie dokumentov pre pospolitú verejnosť [9, 2].

Historické dobové dokumenty však pri digitalizácii predstavujú aj niekoľko problémov, počnúc stavom samotných dobových dokumentov, cez kvalitu snímača, ktorou je dokument digitalizovaný a podmienkami v prostredí, kde je digitalizovaný, končiac vysokými pamäťovými nákladmi na perzistentné uchovanie dokumentov [9, 2, 7].

Samotná digitalizácia je teda pre následné fungovanie zvyšku systému na spracovanie historických dokumentov klúčová a vo väčšine prípadov vyžaduje profesionálne vybavenie a vhodné svetelné podmienky [7].

2.2 Detekcia a segmentácia symbolov v historických šifrovaných rukopisoch

Pojem detekcia a segmentácia symbolov v historických dokumentoch pod sebou zastrešuje proces lokalizácie (detektie), separácie (segmentácie) a identifikácie (klasifikácie) symbolov na obrazovom vstupe, v našom prípade na digitalizovaných historických rukopisoch. Na výstupe sa následne nachádzajú súradnice ohraničujúcich boxov, v ktorých sa lokalizované symboly nachádzajú, súradnice binárnych masiek alebo polygónových ohraničujúcich boxov, pomocou ktorých sú lokalizované symboly separované od zvyšku obrazového vstupu a identifikátor príslušnosti k triede, do ktorej lokalizovaný a separovaný symbol spadá. V súčasnosti sa na druhú väčšinu úloh tohto typu využívajú AI/ML/CV modely a algoritmy, takzvané detektory [6, 15].

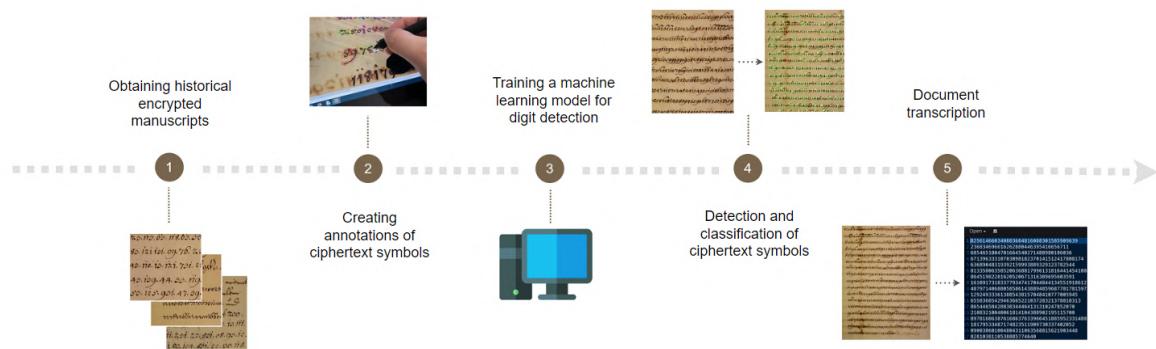
Detekcia a segmentácia v digitalizovaných historických rukopisoch je však špecifická, nakolko vstupy prinášajú komplikácie, napríklad poškodený papier, vyblednutý atrament, atramentové šmuhy alebo rozdielny sklon, prekrytie a rotácie symbolov [9, 2, 15].

Čo najpresnejšia detekcia a segmentácia je pre fungovanie následných častí celého riešenia na spracovanie dobových dokumentov nevyhnutná. Potrebný je teda vhodne natrénovaný model, ako aj obrazové dátá, na ktorých trénovanie prebieha [9, 2, 15].

2.3 Transkripcia detegovaných symbolov v historických šifrovaných rukopisoch

Ďalší významný prvok pri spracovaní historických šifrovaných rukopisov predstavuje transkripciu. Transkripcia vo všeobecnosti predstavuje prepis alebo prevod textu z jedného formátu do druhého, pri automatizovanej transkripcii dobových rukopisov ide o prevod do textovej digitálnej reprezentácie. Na výstupe sa teda nachádza textová dátová štruktúra alebo celý súbor s prepisom obsahu digitalizovaného vstupného dokumentu [9, 7, 16].

Na automatizovanú transkripciu dokumentov sa v súčasnosti využívajú rôzne prístupy a algoritmy, ukotvenie symbolov podľa súradníc na vstupe do výstupnej mriežky, detekcia/segmentácia riadkov a častí rozloženia dokumentu alebo využitie lokálnych maxím histogramov stredov ohraničujúcich boxov na zoskupenie do riadkov implementované počas riešenia výskumného projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov na Ústave informatiky a matematiky, Fakulty elektrotechniky a informatiky Slovenskej technickej univerzity v Bratislave (ÚIM FEI STU) [9, 7].



Obr. 7: Schéma automatizovaného spracovania historických šifrovaných rukopisov [7]

V súčasnosti existuje niekoľko riešení na automatizované spracovanie historických šifrovaných rukopisov. Od komerčnej aplikácie Transkribus, detekčného modelu na báze YOLOv3, DIGITNET až po detekčný a segmentačný model na báze Mask R-CNN vyvinutý pracovníkmi a študentmi na ÚIM FEI STU v rámci grantu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov [7, 17, 18].

Avšak vzhľadom na neaktuálnosť spomenutých riešení a pokrok v AI/ML oblasti sme sa v našej práci zamerali na proces detekcie/segmentácie číslí a symbolov (glyfov) v historických šifrovaných rukopisoch, konkrétnie na najmodernejšie a najpokročilejšie SOTA detekčné a segmentačné modely, ktoré majú potenciál zlepšiť, zefektívniť a skvalitniť spracovanie historických šifrovaných rukopisov.

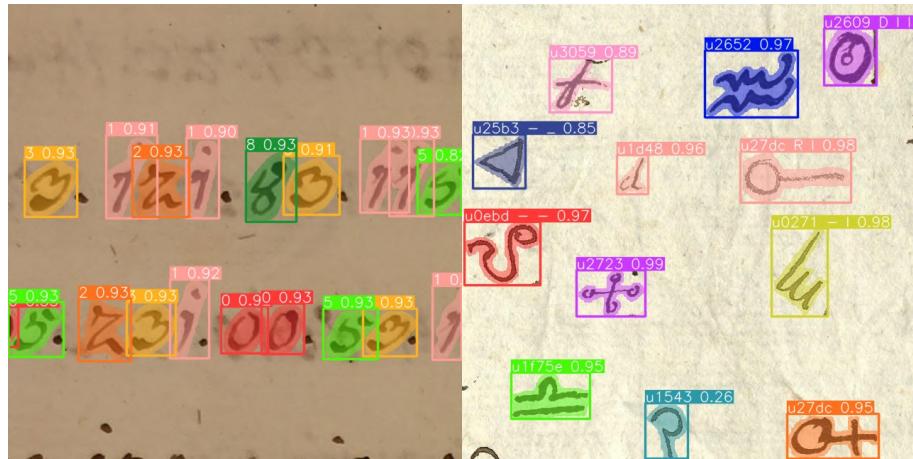
3 Detekcia a segmentácia objektov

Detekcia a segmentácia objektov predstavuje na poli strojového učenia, respektíve počítačového videnia, významnú problematiku. Detekčné a segmentačné modely majú široké spektrum využitia v rôznych odvetviach, napríklad v doprave, stavebníctve, logistike alebo medicíne.

Nakolko v našom prípade pri detekcii a segmentácii symbolov v historických šifrovaných rukopisoch ide o špecifický problém, rozhodli sme sa preskúmať najmodernejšie a najpokročilejšie prístupy rôznych druhov a architektúr, ktoré majú potenciál priniesť pokrok do riešenia tejto problematiky.

3.1 Detekcia a segmentácia

Prvým typom prístupu je detekcia a segmentácia objektov vo svojej elementárnej forme. Pod detekciou objektov rozumieme proces lokalizácie objektov na obrazovom vstupe pomocou ohraničujúcich boxov (bounding box) a ich následnú klasifikáciu do tried. Segmentácia, ktorá rozširuje a priamo nadväzuje na detekciu, reprezentuje proces separácie lokalizovaných objektov od okolia pomocou binárnych masiek (mask), ktoré sú rovnako klasifikované do tried [2, 19].



Obr. 8: Ukážka detekcie a segmentácie číslíc (vľavo) a glyfov (vpravo) dotrénovanými modelmi YOLOv8 v historických šifrovaných rukopisoch

Detekčné modely teda produkujú výstup, ktorý obsahuje x-ové a y-ové súradnice ľavého horného rohu boxu, dolného pravého rohu boxu, alebo výšku a šírku boxu, v závislosti od formátu konkrétneho detekčného modelu. Výstup rovnako zahŕňa klasifikačný identifikátor (classification id) triedy pre každý detegovaný objekt a mieru istoty (confidence), s ktorou sú objekty detegované a klasifikované [2, 19].

Segmentáčné modely k výstupu detekčných modelov pridávajú binárne masky segmentovaných objektov vo formáte matice s rozmerom celého obrazového vstupu a hodnotami 1 alebo True pre pixely, ktorými je maska tvorená, respektíve hodnotami 0 alebo False pre zvyšok pixelov. Druhou alternatívou sú x-ové a y-ové súradnice polygónu, ktorý reprezentuje samotnú masku [2, 19].

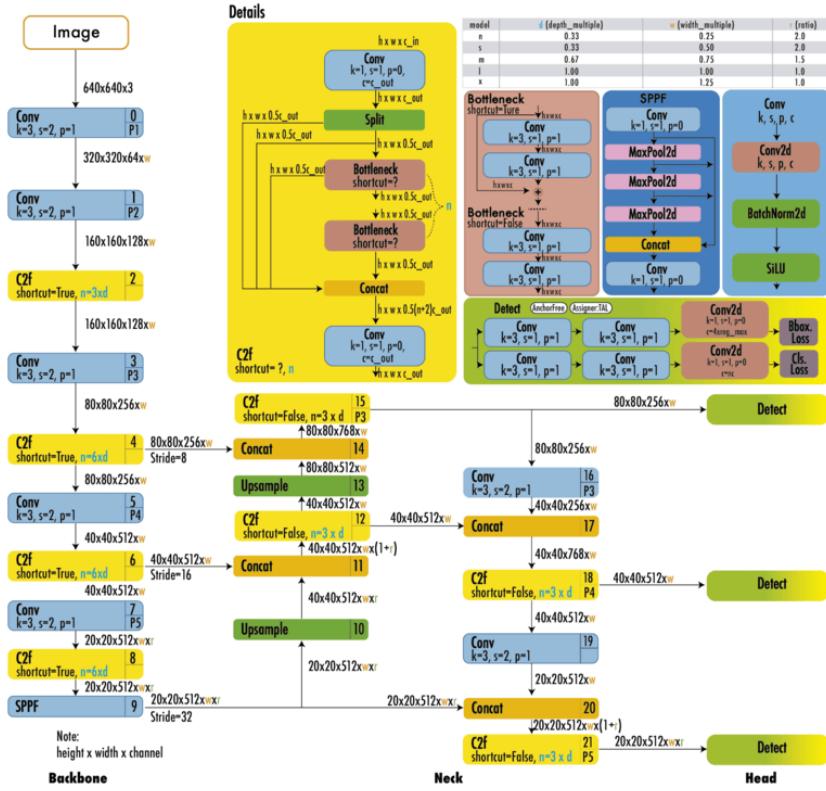
V súčasnosti existuje niekoľko modelov na detekciu alebo segmentáciu, od historicky starších dvojúrovňových modelov (TSD) na báze konvolučných neurónových sietí (CNN) ako R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN cez jednoúrovňové modely (SSD) typu You Only Look Once (YOLO) vo verziach 1 až 7, respektíve najnovšie iterácie YOLOv8, YOLO-NAS, YOLOv9, až po najmodernejšie a najkomplexnejšie modely na báze obrazových transformerov (ViT), napríklad RT-DETR [2, 7, 19, 20, 21].

3.1.1 YOLOv8

YOLOv8 predstavuje jednu z posledných iterácií SOTA YOLO detektorov. Bol vyvinutý spoločnosťou Ultralytics v roku 2023, pričom vo vývoji priamo nadväzuje na ich predošlý model, YOLOv5, ktorý vo svojej dobe so sebou priniesol značné zlepšenie v oblasti detekcie objektov. Model YOLOv8 prináša značné vylepšenia oproti jeho predchodcovi vo verzii 5 [22].

Prvým významným vylepšením je architektúra, ktorá napomáha zvýšeniu úspešnosti a rýchlosťi samotného modelu. Táto skutočnosť je dosiahnutá využitím konceptov Feature Pyramid Network (FPN) and Path Aggregation Network (PAN), ktoré napomáhajú lepsiemu zachytávaniu príznakov pri rôznych veľkostach a mierkach objektov. FPN postupne znižuje priestorové rozlíšenie, zároveň zvyšuje počet kanálov s príznakmi, čo spoločne napomáha vytvoreniu komplexnejších a obsiahlejších príznakových máp. PAN na druhej strane agreguje a spája príznaky z rôznych úrovní siete pomocou preskakovania jej spojení. Tieto koncepty priamo napomáhajú modelu detegovať a segmentovať objekty rôznych mierok, rozlíšení a tvarov [22].

Ďalším rozdielom 8. iterácie je implementácia sofistikovanejšieho postprocesingu, konkrétnie Soft-NMS, ktorý rozširuje algoritmus Non-Maximum Suppression (NMS) pôvodného YOLOv5 určeného na elimináciu, redundantných ohraničujúcich boxov (bounding box). Prístup Soft-NMS využíva jemnejší prah (threshold), ktorý naroziel od pôvodného prístupu NMS neodstraňuje väčšinu prekrývajúcich sa boxov, čím pomáha k zachovaniu cenných obrazových informácií a príznakov pre presnejšiu detekciu a segmentáciu [22].



Obr. 9: Schéma architektúry YOLOv8 [23]

Jedno z najvýznamnejších zlepšení oproti ostatným YOLO modelom predstavuje koncept tzv. Anchor-Free detekcie. Anchor-Free Detection oproti striktným malým, stredným a veľkým kotviacim boxom v predošlých verziách implementuje mechanizmus detekcie bez kotvy pomocou priamej predikcie stredu objektu, čo proces detekcie zrýchluje a zefektívnuje [22].

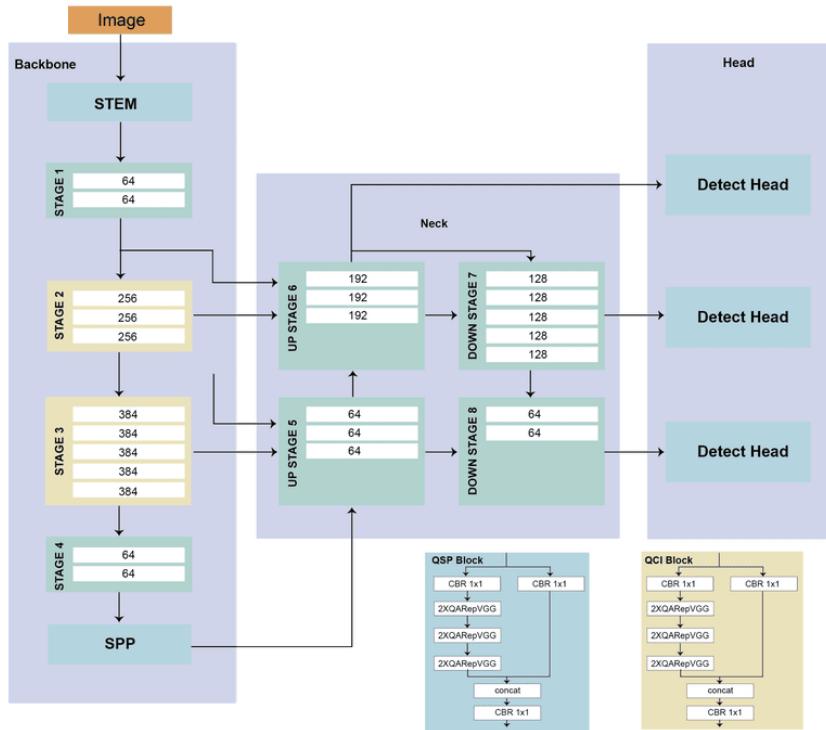
YOLOv8 rovnako ako jeho predchodca YOLOv5 využíva mozaikovú augmentáciu dát, ktorá kombinuje 4 náhodné obrazové vzorky z trénovacej dátovej sady do jednej dátovej vzorky, čo zvyšuje modelu schopnosť generalizácie, čím tiež zvyšuje presnosť detekcie a segmentácie [22].

YOLOv8 v súčasnosti predstavuje jeden z najpokročilejších a najpoužívanejších detekčných modelov. Ponúka detekčný model (det), segmentačný model (seg) a model s odhadom polohy špecifických bodov na obraze (pose) natrénované na datasete COCO, klasifikačný model (cls) natrénovaný na ImageNet datasete a detekčný model podporujúci rotované ohraničujúce boxy (obb) natrénovaný na datasete DOTAv1. Všetky varianty modelu YOLOv8 sú dostupné vo veľkostiach n (nano), s (small), m (medium), l (large) a x (extra large) [24].

3.1.2 YOLO-NAS

YOLO-NAS reprezentuje model z roku 2023 od spoločnosti Deci AI. Model YOLO-NAS bol synteticky vygenerovaný pomocou riešenia AutoNAC s využitím techniky NAS s dôrazom na zlepšenie detekcie malých objektov, presnosti lokalizácie a využitie v reálnom čase [20].

Technológia Automated Neural Architecture Construction (AutoNAC) je algoritmickej optimalizačný mechanizmus využívajúci nástroj Neural Architecture Search (NAS) na vytvorenie, čo najoptimálnejšej architektúry modelu pre konkrétny hardvér, na ktorom bude model prevádzkovaný, povahu dát s ktorými bude model pracovať a scenár použitia, ktorý by mal model plniť [20, 25].



Obr. 10: Schéma architektúry YOLO-NAS [23]

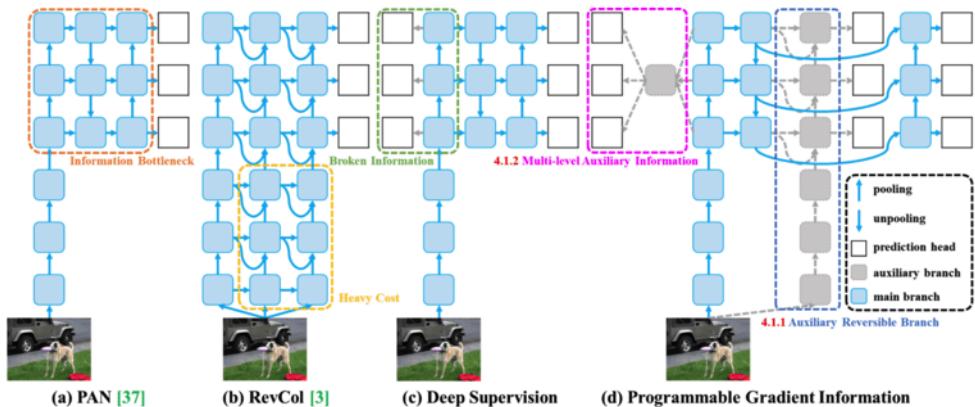
Významné vylepšenie pri detektore YOLO-NAS predstavuje možnosť kvantovaného modelu, čo v praxi predstavuje využitie parametrov a funkcií s nižšou presnosťou. Pri využití reprezentácie s nižšou presnosťou, napríklad 8-bitovej reprezentácie s pevnou desatinou čiarkou alebo binárnej reprezentácie, model zabera menej pamäťového miesta a je tiež výpočtovo efektívnejší. Kvantované modely je vďaka ich nižším nárokom možné nasadiť na zariadenia s obmedzenými výpočtovými zdrojmi, napríklad na mobilné zariadenia alebo zariadenia internetu vecí (IoT) [20, 25].

YOLO-NAS rovnako ako YOLOv8 využíva princíp Anchor-Free detekcie, teda princíp detekcie bez kotvy s priamou predikciou stredu objektu [20, 25].

YOLO-NAS je ukážkou SOTA techniky generovania syntetických modelov na báze požadovaných vlastností. Je k dispozícii v detekčnej verzii (det) a vo verzii pre odhad polohy špecifických bodov na obraze (pose) natrénovaných na COCO datasete. Obe varianty modelu sú k dispozícii vo veľkostiach s (small), m (medium) a l (large) s a bez kvantovania [25].

3.1.3 YOLOv9

YOLOv9 predstavuje poslednú iteráciu modelu z rodiny YOLO vyvinutú autormi YOLOv7 v roku 2024, pričom vychádza z implementácie YOLOv5 od spoločnosti Ultralytics. YOLOv9 predstavuje niekolko citelných zmien a vylepšení, napríklad techniky Programmable Gradient Information (PGI) alebo Generalized Efficient Layer Aggregation Network (GELAN). Tieto inovatívne metódy napomáhajú modelu k zachovaniu kľúčových informácií počas trénovalia aj samotnej detekcie, a zvyšujú jeho schopnosť učiť sa, vďaka čomu dosahuje výnimočnú presnosť a výkonnosť [26].

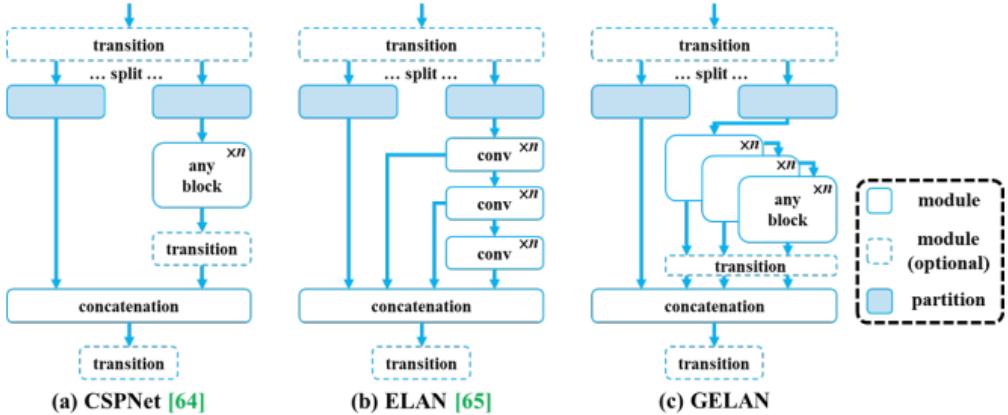


Obr. 11: Schéma architektúry YOLOv9 [26]

Prvým inovatívnym konceptom sú reverzibilné funkcie. Reverzibilné funkcie je možné invertovať bez straty informácie, čo modelom hlbokého učenia umožňuje zachovať plynulý tok informácií. Architektúra YOLOv9 využíva tento koncept na zmiernenie degradácie informácií v jej hlbších vrstvách, čo priamo napomáha k zachovaniu kritických údajov pri detekcii objektov [21, 26].

Ďalšie významné vylepšenie predstavuje technika Programmable Gradient Information (PGI), ktorá pomáha zachovať informácie a údaje v celej hlbke modelu. Tento fakt zlepšuje generovanie gradientov a následnú aktualizáciu parametrov modelu pri procese trénovalia, čo prispieva k zlepšeniu celkovej výkonnosti modelu [21, 26].

GELAN, Generalized Efficient Layer Aggregation Network reprezentuje ďalší architektonický pokrok 9. verzie detektora YOLO. GELAN umožňuje integrovať rôzne výpočtové bloky, ktoré umožňujú modelu fungovať v rôznych scenároch bez straty rýchlosťi a prenosnosti [21, 26].



Obr. 12: Schéma architektúry siete GELAN modelu YOLOv9 [26]

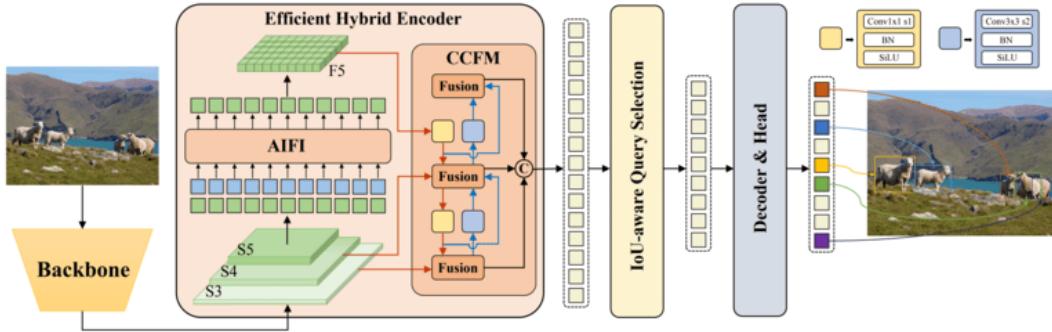
Model YOLOv9 zhodne ako predošlé modely pracuje s Anchor-Free detekciou bez kotvy s priamou predikciou stredu objektu [24, 26].

Model YOLO v poslednej verzii 9 demonštruje pokrok v efektivite, prispôsobivosti, presnosti a nastavuje nové referenčné hodnoty v detekčných a klasifikačných metrikách na COCO datasete, na ktorom je model trénovaný. Projekt YOLOv9 navyše demonštruje schopnosť kooperácie výskumnej komunity v oblasti umelej inteligencie (AI). Model je k dispozícii v detekčnej verzii (det) vo veľkostach t, s, m, c, e [24, 26].

3.1.4 RT-DETR

Ďalšou modernou SOTA alternatívou na poli modelov počítačového videnia (CV) je architektúra transformer využívaná v ML pôvodne pri spracovaní prirodzeného jazyka (NLP). Konkrétnie ide o model obrazového transformera (ViT) typu DETR, RT-DETR. Detektor RT-DETR bol vyvinutý výskumníkmi v spoločnosti Baidu s cieľom dosiahnuť rýchlejšiu a presnejšiu detekciu, ako ponúkali dostupné YOLO varianty v roku 2023 [21, 27].

Špecifikom modelu RT-DETR je jeho architektúra v kompletnej podobe, nakoľko aplikuje princípy z oblasti spracovania prirodzeného jazyka (NLP) do oblasti počítačového videnia (CV) a od YOLO modelov sa značne odlišuje. Hlavné stavebné komponenty modelu RT-DETR tvoria Backbone, Hybrid Encoder, IoU Aware Query Selector a Transformer Decoder & Head [21, 27].



Obr. 13: Schéma architektúry RT-DETR [21]

Backbone modelu RT-DETR vychádza zo starších ML modelov ResNet a HGNetv2. Backbone extrahuje príznaky, pričom na vstup do enkódera sú privedené príznakové mapy z posledných 3 vrstiev [21, 27].

Hybrid Encoder zodpovedá za transformáciu príznakových máp do sekvenie príznakov, pričom využíva moduly Attention based Intra Scale Feature Interaction (AIFI) a CNN based Cross-scale Feature-fusion Module (CCFM), ktoré za pomoci využitia konvolučných jadier, zachytávajú príznaky a ich sémantický kontext v rámci pôvodnej príznakovej mapy [21, 27].

IoU Aware Query Selector slúži ako filter, ktorý počas trénovalia slúži na zosúladenie vysokého detekčného a klasifikačného skóre, čo napomáha výberu relevantnejších príznakov pre dekóder. Transformer Decoder následne za pomocí pomocných predikčných hláv (Head) generuje ohraničujúce boxy (bounding box) a ich klasifikačné skóre istoty (confidence) [21, 27].

RT-DETR po vzore ostatných SOTA YOLO prístupov pracuje s technikou Anchor-Free Detection, teda s detekciou bez kotvy s priamou predikciou stredu objektu [21, 27].

RT-DETR je ďalšou ukážkou pokroku v AI/ML/CV oblasti, ktorý kombinuje NLP prístup využitý na obrazové dátu, vysokú rýchlosť a úspešnosť detekcie. Je k dispozícii v detekčnej verzii (det) natrénovanej na COCO datasete vo veľkostach l (large) a x (extra large) [21, 24].

3.2 Zero-Shot (Promptable) segmentácia

Ďalší SOTA prístup z oblasti počítačového videnia predstavuje zero-shot segmentácia. Pod pojmom segmentácia rozumieme proces separácie objektov od zvyšku obrazového vstupu pomocou binárnych masiek (mask) [19].

Zero-shot segmentácia predstavuje rozšírenie, ktorého modely a implementácie sú vďaka robustným architektúram a komplexným trénovacím dátam schopné segmentovať objekty patriace do tried, ktoré neboli súčasťou samotného trénovania modelu. Táto skutočnosť pomáha jednoduchému použitiu v rôznych oblastiach a scenároch bez potreby dodatočného dotrénovania modelu [28].

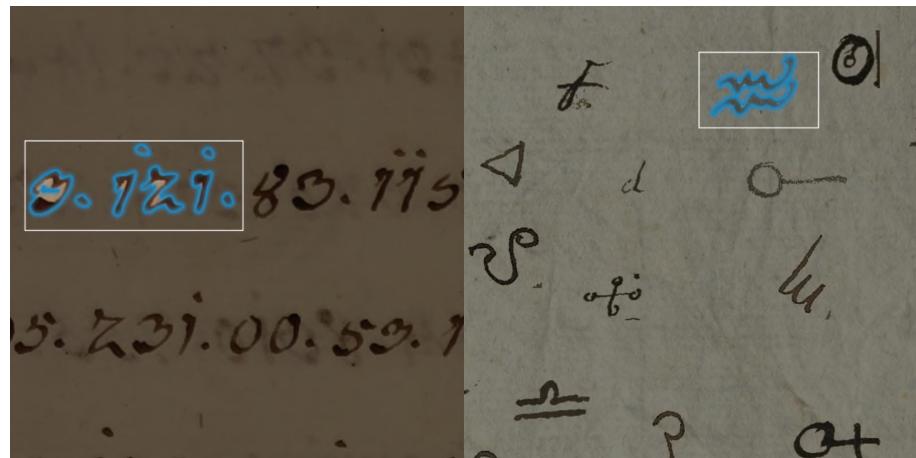


Obr. 14: Ukážka zero-shot segmentácie číslíc (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Segment everything mód

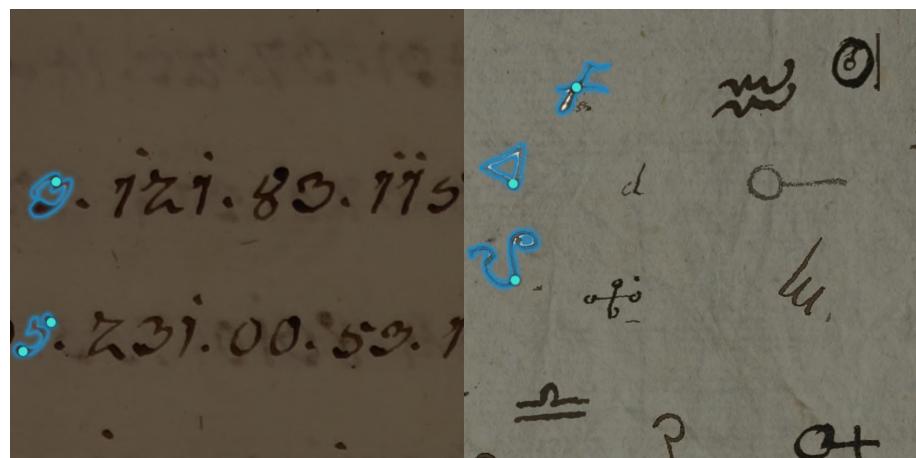
Ďalšou priamo nadväzujúcou SOTA technikou je takzvaná Zero-Shot Promptable segmentácia. Tento typ segmentácie pridáva k zero-shot segmentácii nadstavbu v podobe schopnosti personalizácie segmentácie pomocou takzvaných príkazov (prompt). Vďaka nim je možné pomocou zadefinovania ohraničujúcich boxov (Box prompt) alebo konkrétnych bodov (Point prompt) špecifikovať oblasť, na ktorej má byť segmentácia vykonaná. To prispieva k zvýšeniu flexibility pri využívaní takýchto modelov [29].

Box prompt mód konkrétnie umožňuje pomocou ohraničujúceho boxu (bounding box) špecifikovať oblasť, vo vnútri ktorej prebehne zero-shot segmentácia, kým Point prompt mód umožňuje pomocou určenej obrazovej súradnice (point) špecifikovať objekt, ktorý má byť oddelený od okolia prostredníctvom zero-shot segmentácie.

Výstupom rovnako ako pri štandardnej segmentácii sú binárne masky vo formáte matice s rozmerom celého obrazového vstupu a kladnými, respektíve zápornými hodnotami, alebo vo formáte pola súradníc ohraničujúcich polygónov [19].



Obr. 15: Ukážka zero-shot promptable segmentácie číslic (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Box prompt mód



Obr. 16: Ukážka zero-shot promptable segmentácie číslic (vľavo) a glyfov (vpravo) modelom SAM v historických šifrovaných rukopisoch - Point prompt mód

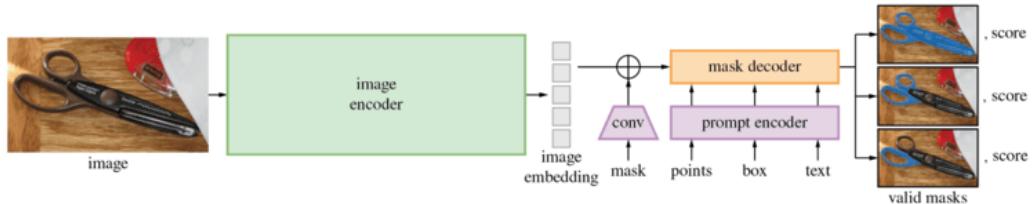
Prelomovým modelom v oblasti zero-shot promptable segmentácie je Segment Anything Model (SAM) vyvinutý výskumným tímom spoločnosti Meta v prvej polovici roku 2023 [29].

3.2.1 SAM

Model od výskumníkov zo spoločnosti Meta, Segment Anything Model (SAM), znamená pre odvetvie ML masívny krok vpred. SAM reprezentuje model typu Foundation, ktoré sú predtrénované na obrovskom množstve dát, čo spoločne s vhodnými príkazmi (prompt engineering) dopomáha ich schopnosti generalizácie pri nových dátach a úlohách. SAM bol vyvinutý s dôrazom na segmentáciu objektov pomocou príkazov, ktorými je možné modelu úlohu upresniť na základe dodatočných priestorových indícií, pričom je schopný fungovať v 3 hlavných režimoch [29, 30]:

- režim úplnej segmentácie (segment everything),
- režim segmentácie na základe ohraničujúcich boxov (box prompt segmentation),
- režim segmentácie na základe klúčových bodov (point prompt segmentation).

Pri segmentácii na základe príkazov (prompt) je ďalším prelomovým konceptom trénovacia dátová množina SA-1B. Dataset Segment Anything 1 Billion Mask (SA-1B) je najrobustnejším segmentačným datasetom, ktorý bol vytvorený práve pre potreby vývoja modelu SAM. S lepšou úspešnosťou modelu paralelne rástol aj počet vysoko kvalitných anotácií a vzoriek v datasete, až do finálnej podoby s 1.1 miliardami masiek na 11 miliónoch obrazových vzoriek [29, 30].



Obr. 17: Schéma architektúry SAM [29]

Architektúra modelu SAM rovnako prináša viaceré zaujímavosti a špecifiká, pričom ju tvoria 3 klúčové komponenty, Image Encoder, Prompt Encoder a Mask Decoder.

Komponent Image Encoder predstavuje predtrénovaný obrazový transformer (ViT), ktorý v prvej fáze modelu generuje sekvenciu príznakov s dôrazom na ich pôvodný sémantický kontext na vstupe, ktorá je privedená na vstup dekódera masiek [29, 30].

Príkazový enkóder, respektíve Prompt Encoder, zodpovedá za transformáciu príkazov v podobe klúčových bodov alebo ohraničujúcich boxov do vektorovej číselnej reprezentácie (embedding vector), ktorá je rovnako privedená do dekódera masiek [29, 30].

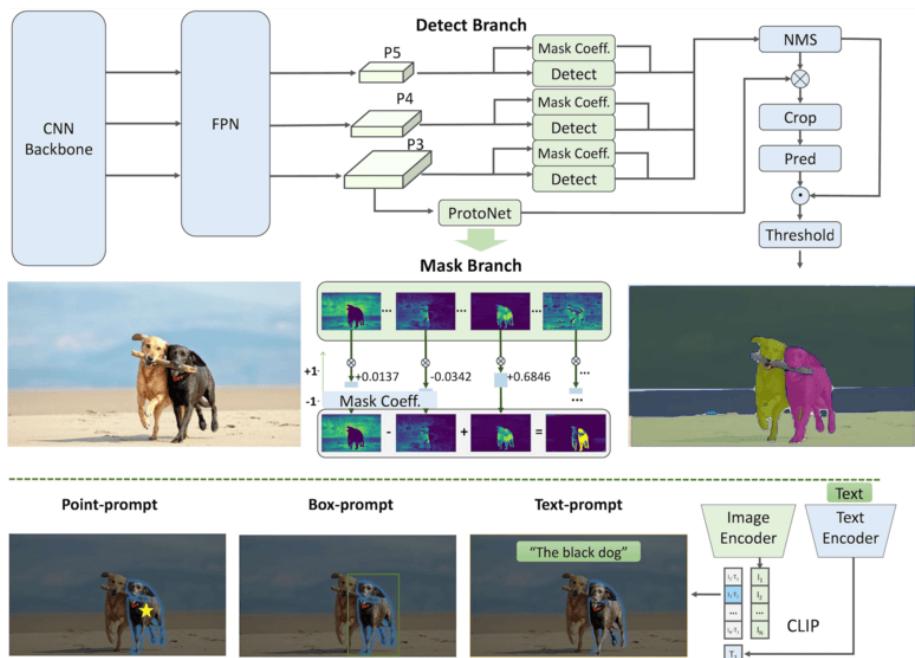
Za výslednú predikciu samotných masiek na základe vektorových reprezentácií príznakov a príkazov je zodpovedný Mask Decoder. Architektúra modelu SAM využíva na generovanie výsledných masiek okrem modifikovaného dekódera aj pomocné predikčné hlavy (Head). Štruktúra tohto komponentu je teda priamo inšpirovaná modelmi typu transformer, čo dopomáha jej schopnosti segmentácie v reálnom čase [29, 30].

SAM je dostupný v 3 veľkostiacach architektúry, mobile, b (base) a l (large) [24, 29].

3.2.2 FastSAM

Napriek významnému pokroku a novým možnostiam využitia, ktoré model SAM priniesol, bolo jeho nasadenie do bežných scenárov použitia limitované jeho vysokými výpočtovými a hardvérovými nárokmi. Z tohto dôvodu bola o pár mesiacov neskôr vyvinutá jeho alternatívna verzia s dôrazom na zníženie výpočtových nárokov a s minimálnym dopadom na presnosť, Fast Segment Anything Model (FastSAM) [31].

Fast Segment Anything Model je tvorený 2 hlavnými komponentami, All-instance Segmentation (AIS) and Prompt-guided Selection (PGS) [31].



Obr. 18: Schéma architektúry FastSAM [31]

AIS modul využíva segmentačný model YOLOv8 na získanie masiek z obrazového vstupu, čím masívne zrýchluje fungovanie celého systému [31].

Po segmentácii všetkých objektov na vstupe modulom AIS je úlohou PGS modulu selekcia relevantných segmentovaných objektov zodpovedajúcich vloženým príkazom obsahujúcich ohraňičujúce boxy alebo kľúčové body [31].

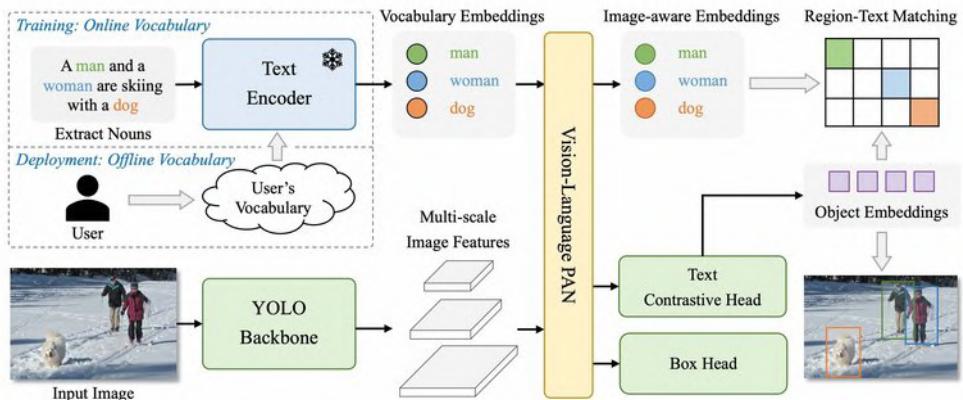
FastSAM trénovaný na 2% pôvodného SA-1B datasetu je k dispozícii vo veľkostiach s (small) a x (extra large) [24, 31].

3.3 Open-Vocabulary detekcia

Koncept detektie pomocou otvorenej slovnej zásoby, respektíve Open-Vocabulary Detection, predstavuje ďalší SOTA prístup priamo spájajúci pokrok a najnovšie poznatky z oblastí počítačového videnia (CV) a spracovania prirodzeného jazyka (NLP). Tento prístup sa zameriava na open-set detekciu, ktorej modely sú schopné detegovať objekty, ktoré neboli súčasťou trénovacej množiny. Táto skutočnosť je možná vďaka hybridnému prístupu, kedy model okrem obrazových vlastností (rozloženie, kompozícia obrazového vstupu) pracuje aj s textovými kontextovými opismi objektov (text-to-image grounding). Detekciu je následne bez dodatočného trénovania možné personalizovať pomocou kontextových textových príkazov (prompt) alebo špecifikáciou vlastnej množiny tried, ktoré si želáme detegovať [32, 33].

3.3.1 YOLO-World

Foundation model z rodiny YOLO, YOLO-World je model strojového učenia podporujúci Zero-Shot Real-Time Open-Vocabulary detekciu objektov. YOLO-World prináša nový unikátny prístup detektie využitím textových opisov objektov (text-to-image grounding) a obrazových vlastností objektov, ktorý mu umožňuje detekciu objektov bez ich prítomnosti v trénovacej dátovej sade, čo v kombinácii s jeho rýchlosťou a efektivitou podporuje jeho široké využitie v rôznych scenároch reálneho sveta. K tejto skutočnosti prispieva predovšetkým jeho dizajn a architektúra, ktorá pozostáva z Text Encoder, YOLO Backbone a Vision-Language PAN modulov [32, 33].



Obr. 19: Schéma architektúry YOLO-World [32]

Modul YOLO Backbone slúži na extrakciu obrazových príznakov zo vstupu pomocou modelu YOLOv8. Extrahované obrazové príznakové mapy sú následné privedené na vstup Vision-Language Path Aggregation siete [32, 33].

Ďalšie dôležité vstupy pre modul Vision-Language PAN generuje textový enkóder (Text Encoder), ktorý je postavený na princípoch modelu CLIP (Contrastive Language-Image Pre-training), natrénovaný na rozsiahlych dátových množinách pre účel porozumenia vzťahov medzi obrazovými vstupmi a ich textovými opismi (text-to-image grounding). Tento koncept modelu zabezpečuje prístup ku kontextovým informáciám, čo zvyšuje jeho schopnosť interpretácie vizuálneho obsahu [32, 33].

Vision-Language Path Aggregation Network (PAN) slúži na agregáciu obrazových a jazykových (textových) príznakov z predoších 2 modulov. Táto fúzia modelu zlepšuje reprezentácie objektov. Pomocné hlavy (Text Contrastive Head a Box Head) následne generujú podobnosti vizuálnej a textovej reprezentácie objektov pre ich finálnu detekciu [32, 33].

Model je distribuovaný v pôvodnej verzii a vo verzii 2 (v2) vo veľkostiach s (small), m (medium), l (large) a x (extra large). Predtrénovaný bol na datasetoch Objects365v1, GQA a Flickr30k [24, 32].

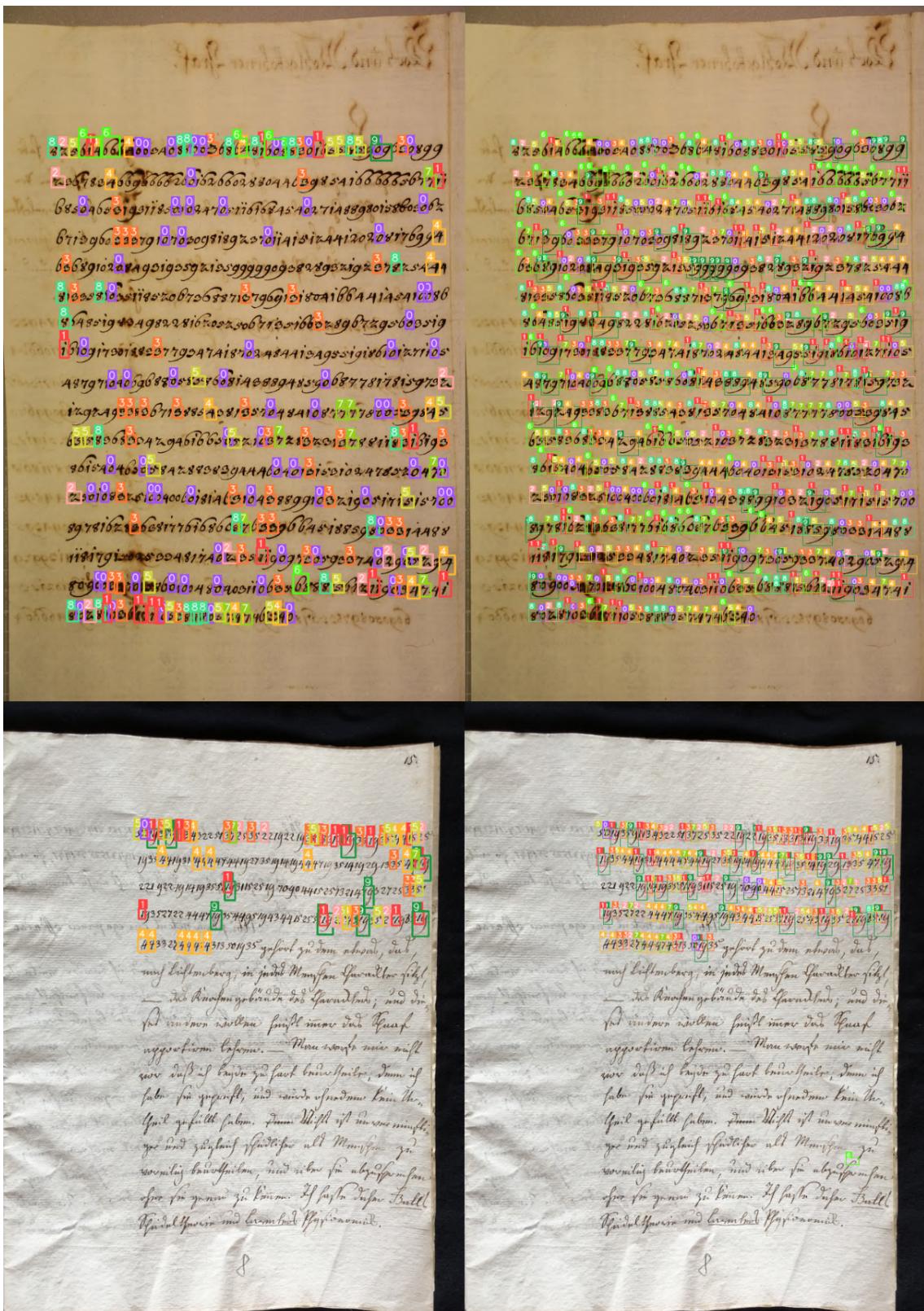
3.4 Detekcia a segmentácia malých objektov

Napriek pokročilým technikám detekcie a komplexným architektúram modelov predstavuje detekcia a segmentácia husto rozmiestnených objektov malých rozmerov významný problém.

Malé a vzdialené objekty sú vo väčšine prípadov reprezentované minimálnym počtom pixelov na obrazovom vstupe, čo znižuje počet detailov a zabraňuje presnej detekcii a segmentácii pomocou štandardných detekčných modelov [34, 35].

Napriek modelom alebo prístupom, ktoré boli vyvíjané práve s dôrazom na riešenie problému detekcie a segmentácie malých objektov, ako YOLO-NAS, YOLOv9, FPN alebo klízavé okno (Sliding Window), nie sú tieto modely a prístupy vo svojej štandardnej podobe schopné tento problém úplne eliminovať. Táto skutočnosť je často zapríčinená povahou samotných datasetov, na ktorých je väčšina modelov trénovaná, napríklad COCO alebo ImageNet, ktoré primárne obsahujú obrázky v nižšom rozlíšení (640 x 480 pixelov), na ktorých sa primárne nachádzajú väčšie objekty, čo vedie k limitáciám pri detekcii a segmentácii malých objektov [34, 35].

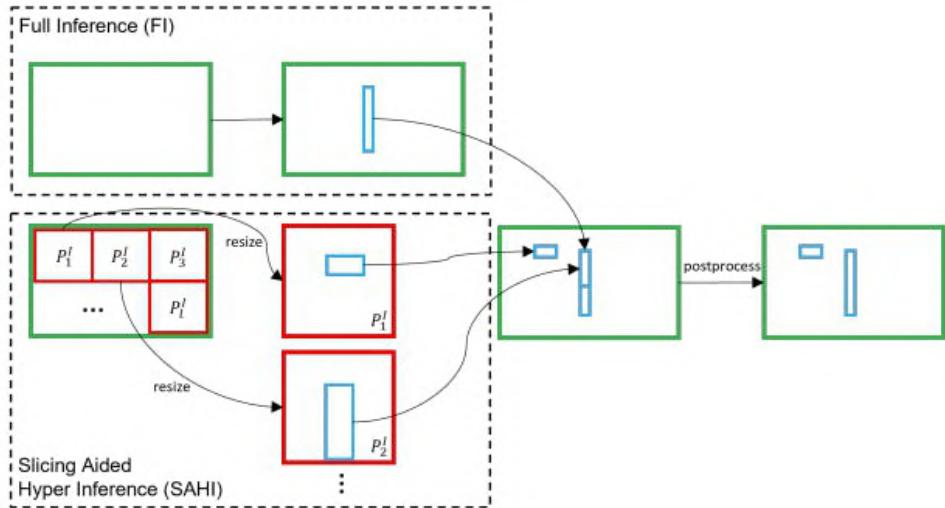
Nový prístup s potenciálom razantného zlepšenia detekcie malých objektov predstavuje Slicing Aided Hyper Inference (SAHI) [34, 35].



Obr. 20: Ukážka detekcie malých číslíc v historických šifrovaných rukopisoch dotrénovaným modelom YOLOv8 (vľavo) a dotrénovaným modelom YOLOv8 s využitím techniky SAHI (vpravo)

3.4.1 SAHI

SAHI, respektívne Slicing Aided Hyper Inference, predstavuje inovatívny postup, ktorý vďaka rozdeleniu obrazového vstupu na bloky, detekcii objektov na blokoch a následnej agregácii detekcií zlepšuje úspešnosť detekcie malých objektov [34, 35].



Obr. 21: Schéma fungovania konceptu SAHI [34]

Prvým krokom prístupu SAHI je rozdelenie pôvodného obrazového vstupu na $N \times N$ blokov s miernym prekrytím. Na každý blok je následne aplikovaný ľubovolne zvolený detekčný model, ktorý je na výsekokach schopný zachytiť aj jemné detaily. Pre zvýšenie presnosti je možné detekčný model aplikovať aj na celý obrazový vstup (Full Inference) [34, 35].

Na všetky detegované objekty z blokov, prípadne celého obrazového vstupu je následne aplikovaný algoritmus Non-Maximum Suppression (NMS), ktorý pri neprekročení prahu hodnoty prekrycia objektov IoU vyradí duplicitné detegované objekty a zachová len najrelevantnejšie z nich. Tento proces filtrovania a agregácie konsoliduje detekcie z celého obrazového vstupu, čo dopomáha k presne detegovaným objektom rôznych veľkostí [34, 35].

4 Analýza a návrh riešenia

Po oboznámení sa s problematikou a osvojení si teoretických poznatkov je ďalším krokom našej práce analýza a návrh riešenia. V tejto časti sa zameriavame na analýzu datasetov, ktoré sú použité na proces trénovania modelov, analýzu dostupných ML modelov, návrh mechanizmu na automatizovanú transkripciu detegovaných symbolov, návrh použitia webového aplikačného rozhrania (API) a v neposlednom rade na analýzu a voľbu dostupných softvérových nástrojov, ktoré sú použité pri implementácii konečného riešenia.

4.1 Požiadavky

Pre vypracovanie analýzy a správnu implementáciu je nevyhnutné sformulovanie požiadaviek na nami vyvíjané riešenie. Požiadavky na funkciaľitu finálneho riešenia rozčleníme na funkcionálne a nefunkcionálne požiadavky [36].

Funkcionálne požiadavky reflektujú fungovanie softvérového riešenia a jeho komponentov. Nefunkcionálne požiadavky zase odzrkadľujú vlastnosti fungovania softvérového riešenia a jeho komponentov [36].

4.1.1 Funkcionálne požiadavky

- trénovanie vybraných modelov na detekciu a segmentáciu objektov v historických šifrovaných rukopisoch, ich export do ONNX formátu a ich následná evaluácia v kombinácii s a bez techniky SAHI,
- sprístupnenie detekčnej a segmentačnej funkcionality vybraných riešení a natrénovaných modelov pomocou webového API,
- sprístupnenie funkcionality automatizovaného generovania anotácií vo formáte YOLO pomocou webového API,
- sprístupnenie funkcionality pokročilého prehľadávania datasetov vo formáte YOLO pomocou webového API.

4.1.2 Nefunkcionálne požiadavky

- využitie najpokročilejších a najmodernejších detekčných/segmentačných modelov a techniky SAHI,
- jednotné a intuitívne rozhranie webového API,
- využitie paralelizácie s cieľom zvýšenia efektivity a zníženia výpočtového času pri obsluhe požiadaviek webovým API.

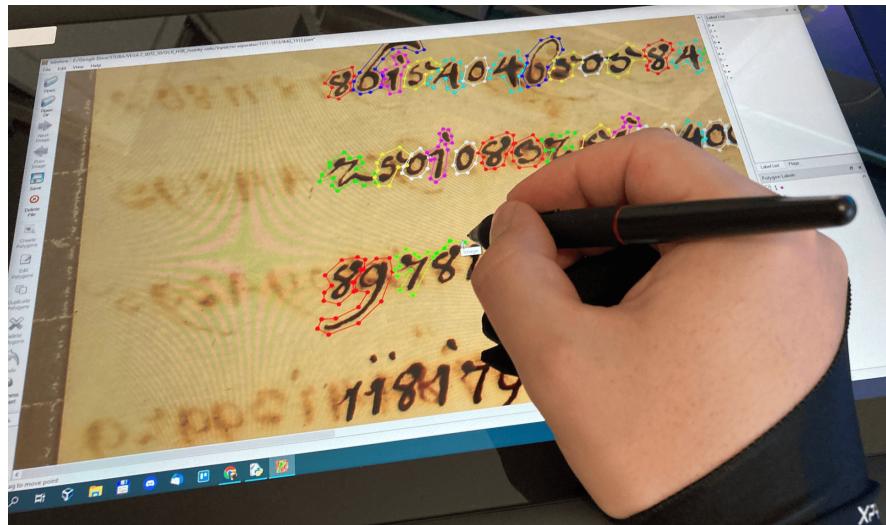
4.2 Datasetsy

Klúčový faktor pre korektné a optimálne fungovanie modelov strojového učenia (ML) predstavujú dátá, na ktorých sú natrénované. Prvým bodom pri analýze je teda preskúmanie nami zvolených dátových množín. Napriek pokroku v ML oblasti a stále zvyšujúcemu sa počtu dostupných datasetov stále neexistuje detekčný/segmentačný dataset, ktorý by reflektoval problematiku historických rukopisov.

Ku klasifikačným dátovým množinám typu ARDIS alebo DIDA v súčasnosti neexistuje relevantná varianta pre potreby detekcie a segmentácie. Z tohto dôvodu v našej práci využívame vysokokvalitný a rozsiahly dataset historických rukou písaných číslíc a symbolov vytvorený v rámci grantového projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov na pracovisku ÚIM FEI STU [2, 7].

Ako už bolo spomenuté, historické rukopisy pochádzajú zo zbierky dokumentov Slovenského národného archívu z 3 rôznych fondov šľachtických rodov, pričom ich digitálne kópie boli zhotovené vo vysokom rozlíšení (4160×6240 pixelov) pomocou fotoaparátu pripojeného na stojane s dodatočnými zdrojmi svetla, vďaka čomu digitalizované rukopisy vo väčšine prípadov obsahujú čisté písmo, symboly sú jasne oddelené a dobre čitateľné [7].

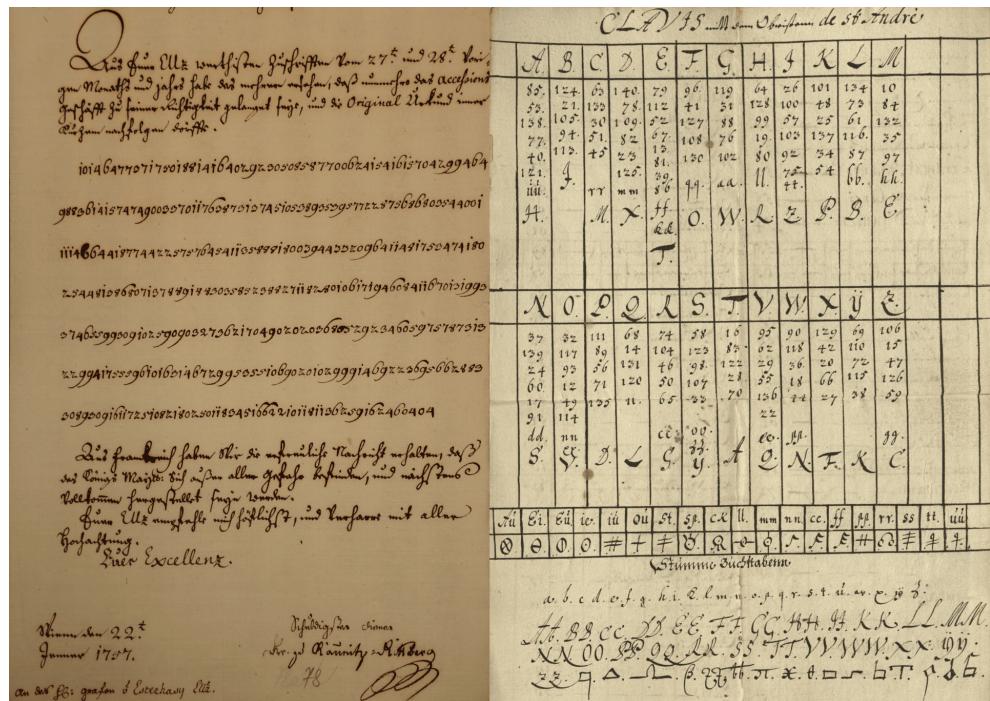
Anotácie, respektíve polygonálne masky číslíc a symbolov, boli zhotovené ručne pomocou tabletu, dotykového pera (stylus) a softvérového nástroja *LabelMe*, ktorý umožňuje kreslenie geometrických tvarov na ohraničenie objektu na obrazovom vstupe [7].



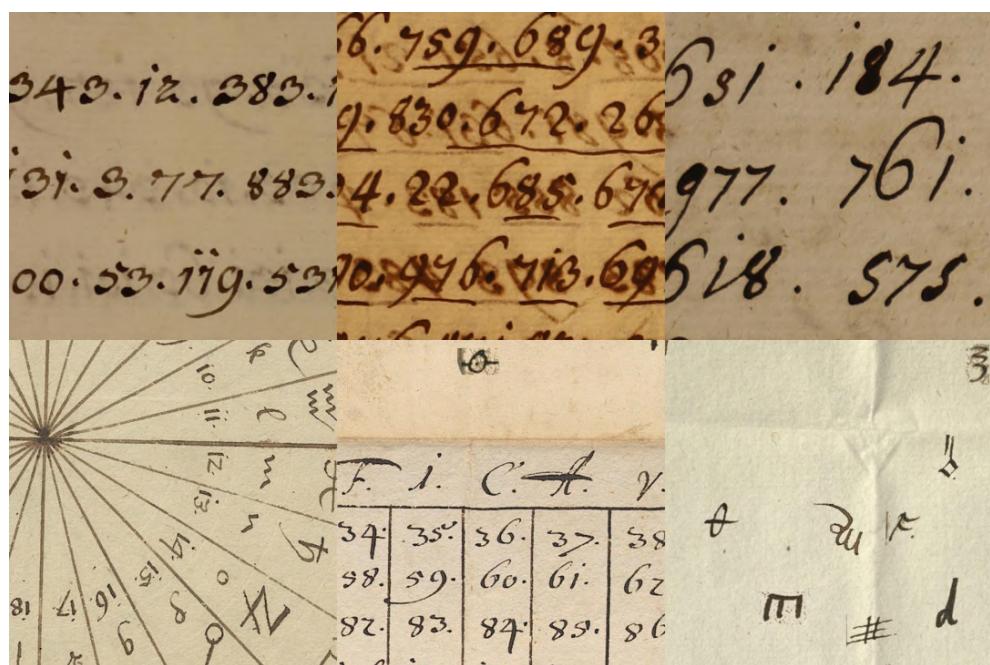
Obr. 22: Ukážka ručného kreslenia anotácií [7]

Dátové vzorky a príslušné anotácie boli následne pre efektívnejšie trénovanie a zvýšenie úspešnosti detekcie a segmentácie výsledného modelu rozdelené do blokových výsekov o veľkosti 640×640 pixelov datasetu VEGA Cropped [2, 7].

VEGA a VEGA Cropped dataset je tvorený 2 primárnymi dátovými podmnožinami, datasetom číslí a datasetom špecifických symbolov, respektíve glyfov. Je dostupný vo formáte COCO a vo formáte YOLO v detekčnej aj segmentačnej variante [6, 15].



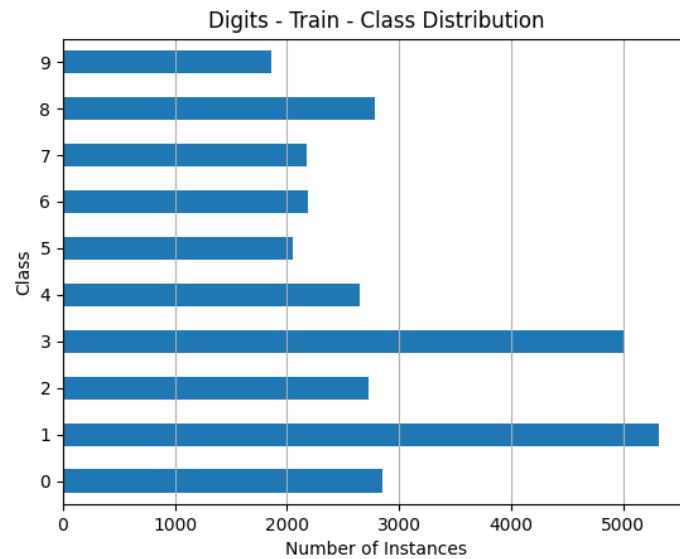
Obr. 23: Ukážka vzoriek VEGA datasetov číslí (vľavo) a glyfov (vpravo) [7]



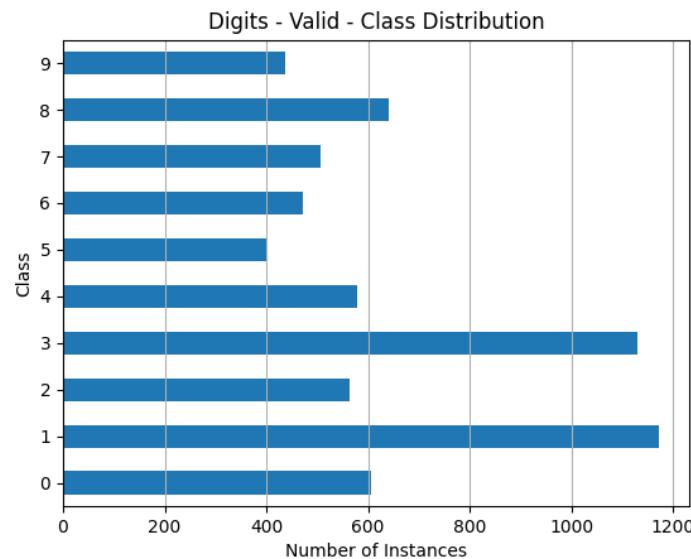
Obr. 24: Ukážka vzoriek VEGA Cropped datasetov číslí (hore) a glyfov (dole) [7]

4.2.1 Číslice

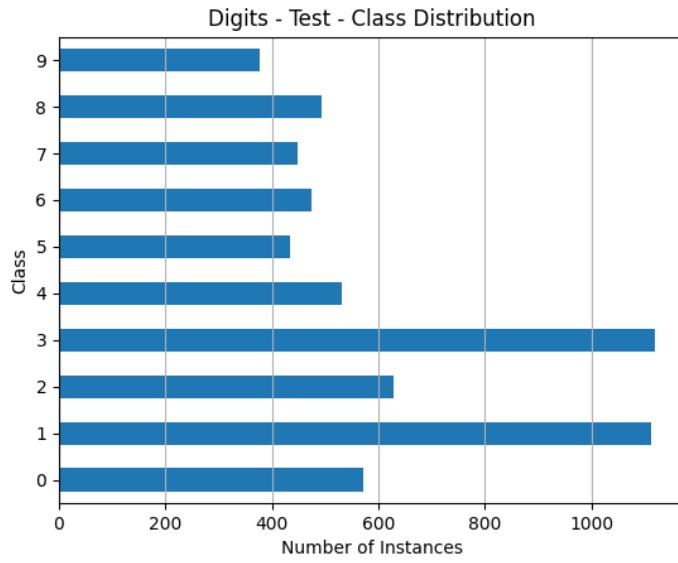
VEGA Cropped dataset číslíc je tvorený 10 triedami obsahujúcich 29625 inštancií na 2107 obrazových vzorkách v trénovacej, 6512 inštancií na 452 obrazových vzorkách vo validačnej a 6188 inštancií na 451 obrazových vzorkách v testovacej množine. Celkový počet digitalizovaných dokumentov, z ktorých sú výseky zhotovené je 43 [15].



Obr. 25: Graf distribúcie tried (10) anotácií číslíc v trénovacej množine



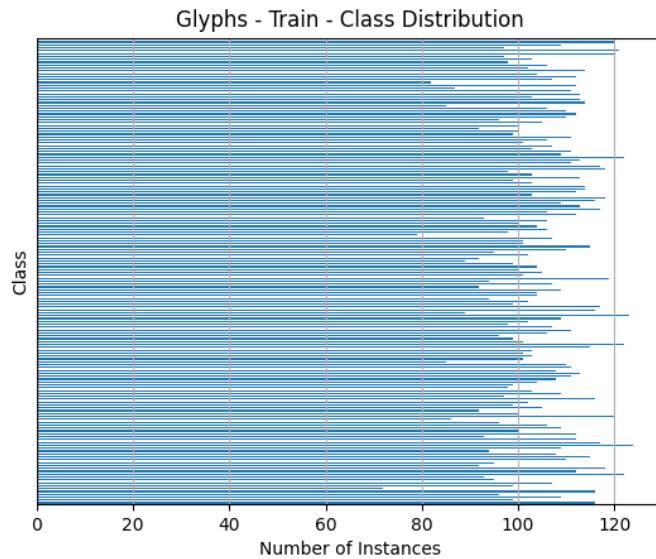
Obr. 26: Graf distribúcie tried (10) anotácií číslíc vo validačnej množine



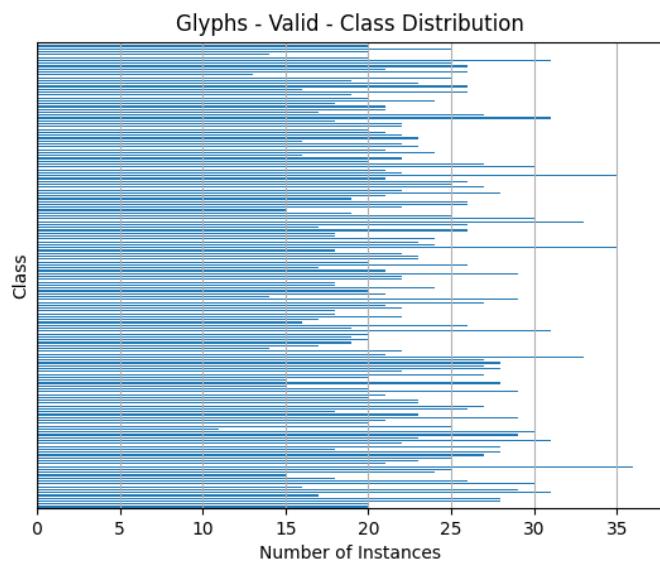
Obr. 27: Graf distribúcie tried (10) anotácií číslic v testovacej množine

4.2.2 Glyfy

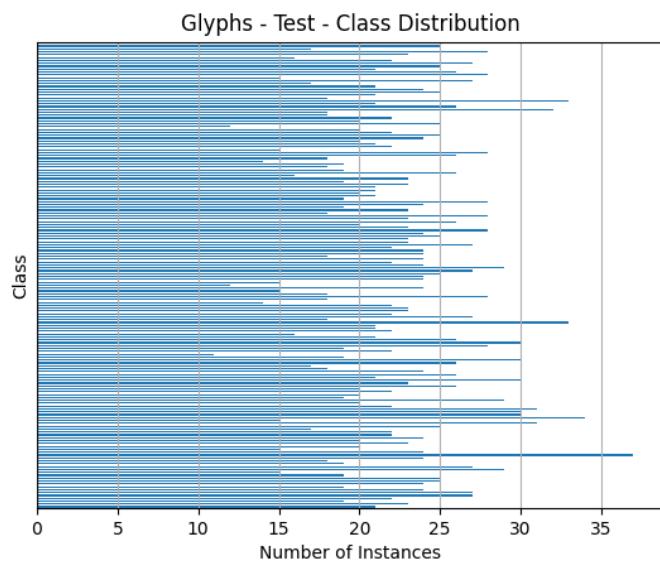
VEGA Cropped dataset glyfov je tvorený 162 triedami obsahujúcich 17022 inštancií na 7591 obrazových vzorkách v trénovacej, 3698 inštancií na 1627 obrazových vzorkách vo validačnej a 3656 inštancií na 1627 obrazových vzorkách v testovacej množine. Celkový počet digitalizovaných dokumentov, z ktorých sú výseky zhotovené je 512 [6].



Obr. 28: Graf distribúcie tried (162) anotácií glyfov v trénovacej množine



Obr. 29: Graf distribúcie tried (162) anotácií glyfov vo validačnej množine



Obr. 30: Graf distribúcie tried (162) anotácií glyfov v testovacej množine

4.3 Modely

Ďalším krokom analýzy a návrhu je výber modelov strojového učenia (ML), ktoré sú v rámci implementačnej časti našej práce trénované a testované na VEGA Cropped datasetoch čísel a glyfov. Na základe nadobudnutých teoretických poznatkov, charakteristík modelov a profilu nášho problému sme vyselektovali modely:

- YOLOv8,
- YOLO-NAS,
- YOLOv9,
- RT-DETR,
- YOLO-World,
- FastSAM.

Na zlepšenie úspešnosti modelov pri detekcii a segmentácii malých objektov taktiež aplikujeme techniku Slicing Aided Hyper Inference (**SAHI**).

Výsledky testovania modelov porovnáme podľa štandardných AI/ML metrík. Najúspešnejšie modely a konfigurácie následne okrem integrácie do webového API, z dôvodu ľahšej prenositeľnosti exportujeme do formátu **ONNX**.

4.3.1 Metriky

Dôležitú súčasť testovania a následného porovnania ML modelov predstavujú rôzne metriky. V našej práci pracujeme so štandardnými metrikami na vyhodnocovanie úspešnosti detekcie a segmentácie, konkrétnie pôjde o Intersection over Union (**IoU**), Precision (**P**), Recall (**R**), Average Precision (**AP**) a Mean Average Precision (**mAP**) [37].

Prvú dôležitú metriku pri vyhodnocovaní úspešnosti detekcie a segmentácie predstavuje ukazovateľ **Intersection over Union (IoU)**. Metrika IoU znázorňuje mieru prekrycia medzi detegovaným ohraničujúcim boxom alebo segmentovaným polygónom a pravdivým ohraničujúcim boxom, respektívne pravdivým polygónom. IoU teda predstavuje mieru prekrycia medzi predikciou (prediction) a skutočnosťou (ground truth), pričom sa pohybuje v intervale medzi 0, teda nulovým prekrytím, a 1, teda úplným prekrytím. Vzorec IoU vyzerá nasledovne [6, 37]:

$$IoU = \frac{\text{IntersectionArea}}{\text{UnionArea}} \quad (1)$$

- IoU – miera prekrytia medzi detekciou/segmentáciou a skutočnosťou,
- $IntersectionArea$ – obsah príeniku plôch detekcie/segmentácie a skutočnosti,
- $UnionArea$ – obsah zjednotenia plôch detekcie/segmentácie a skutočnosti.

Ďalším významným ukazovateľom pri evaluácii ML systémov je **Precision (P)**, teda presnosť, v našom prípade detekcie a segmentácie. Precision konkrétnie znázorňuje pomer počtu správne detegovaných, respektíve segmentovaných, objektov (TP) k počtu všetkých detegovaných a segmentovaných objektov (TP + FP). Vzorec má nasledovnú podobu [6, 37]:

$$P = \frac{TP}{TP + FP} \quad (2)$$

- P – miera presnosti detekcie/segmentácie,
- TP – počet správne detegovaných/segmentovaných objektov,
- FP – počet nesprávne detegovaných/segmentovaných objektov.

Metriku hodnotiacu citlivosť modelu predstavuje **Recall (R)**. Recall teda vyčísluje pomer počtu správne detegovaných, respektíve segmentovaných, objektov (TP) k súčtu správne detegovaných, respektíve segmentovaných, objektov a nesprávne nedetegovaných, respektíve nesegmentovaných, objektov (TP + FN). Vypočítava sa pomocou vzorca [6, 37]:

$$R = \frac{TP}{TP + FN} \quad (3)$$

- R – miera citlivosti detekcie/segmentácie,
- TP – počet správne detegovaných/segmentovaných objektov,
- FN – počet nesprávne nedetegovaných/nesegmentovaných objektov.

Ďalšou dôležitou metrikou problematiky detekcie a segmentácie je **Average Precision (AP)**. AP vyjadruje priemernú presnosť modelu pri rôznych nastaveniach prahovej hodnoty miery istoty (confidence). AP je vyčíslené pomocou obsahu plochy pod **PR krivkou**, ktorá graficky znázorňuje Precision a Recall pri rôznych prahových hodnotách miery istoty (confidence). Metrika Average Precision teda sumarizuje PR krivku do jednej skalárnej hodnoty pomocou interpolácie presnosti na pevných úrovniach návratnosti alebo Riemannovho súčtu. Výpočet vyzierá nasledovne [6, 37]:

$$AP = \sum_n (R_n - R_{n-1}) P_n \quad (4)$$

- AP – miera priemernej presnosti detekcie/segmentácie,
- P – miera presnosti detekcie/segmentácie,
- R – miera citlivosti detekcie/segmentácie,
- n – n-tá prahová hodnota miery istoty.

Poslednú kľúčovú metriku pre hodnotenie úspešnosti detekcie a segmentácie reprezentuje **Mean Average Precision (mAP)**, ktorá vyjadruje priemernú hodnotu AP naprieč všetkými triedami. mAP je široko využívaný ukazovateľ detekčných a segmentačných schopností modelov, nakoľko zohľadňuje presnosť a citlosť modelu pri rôznych hodnotách prahu miery istoty (confidence) naprieč všetkými triedami. Výpočet prebieha na základe vzorca [6, 37]:

$$mAP = \frac{1}{k} \sum_k^i AP_i \quad (5)$$

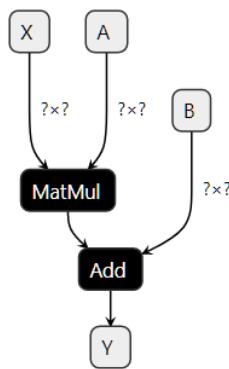
- mAP – miera priemernej presnosti detekcie/segmentácie naprieč triedami,
- AP – miera priemernej presnosti detekcie/segmentácie,
- k – počet detekčných/segmentačných tried,
- i – i-tá detekčná/segmentačná trieda.

4.3.2 ONNX formát

V súčasnosti existuje hneď niekoľko široko využívaných knižníc strojového učenia, ktoré využívajú rôzne proprietárne formáty exportu modelov. Ide napríklad o formáty TensorFlow, PyTorch, Caffe2 alebo Theano. Z dôvodu štandardizácie exportu a následného nasadzovania ML modelov bol spoločnosťami Meta a Microsoft vyvinutý štandardizovaný formát Open Neural Network Exchange (ONNX) [38, 39].

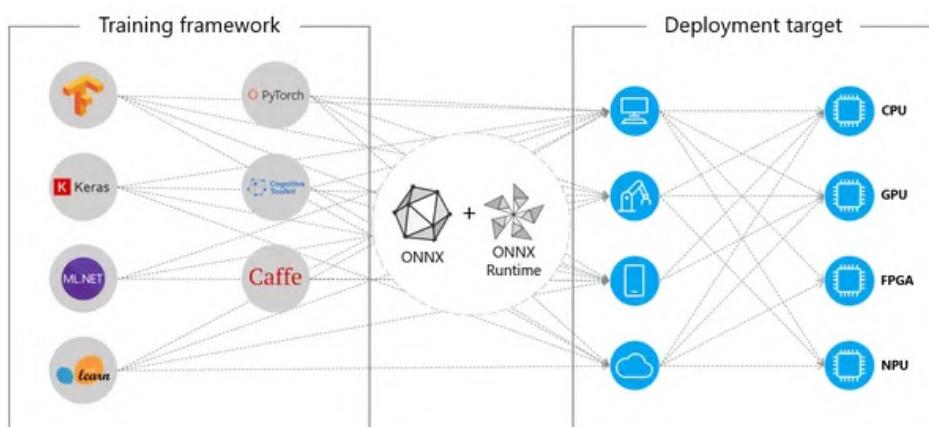
Medzi kľúčové výhody formátu ONNX patria jeho flexibilita a konzistentnosť pri importe a exporte modelov medzi rôznymi knižnicami, napríklad TensorFlow, PyTorch, Caffe2 alebo Theano, rôznymi programovacími jazykmi, napríklad Python, C++, C# alebo Java a rôznymi hardvérovými platformami, ako sú CPU, GPU, TPU, NPU alebo FPGA. Vďaka spolupráci najväčších technologických hráčov pri vývoji tohto formátu je ONNX navyše podporovaný aj v rámci clouдовých riešení Amazon Web Services (AWS), Microsoft Azure a Google Cloud [38, 39].

Základný koncept formátu ONNX pozostáva z univerzálnej reprezentácie výpočtových grafov. Výpočtové grafy sú definované uzlami reprezentujúcimi operácie a hranami reprezentujúcimi tok dát. Každý uzol grafu teda reprezentuje špecifickú operáciu, napríklad konvolúciu, pooling alebo aktiváciu, a obsahuje atribúty, ktoré definujú jeho správanie. Na definovanie výpočtových grafov v rámci ONNX sa využíva dátový formát ProtoBuff, ktorý je nezávislý od jazyka a platformy. ONNX tiež obsahuje štandardizovaný súbor funkcií, typov a atribútov, ktoré špecifikujú výpočty a akcie v rámci uzlov v grafe. Formát ONNX rovnako štandardizuje a definuje podobu vstupných a výstupných tenzorových štruktúr [38, 39].



Obr. 31: Schéma výpočtového grafu formátu ONNX [40]

Jednoduchá prenositeľnosť a nasadzovanie modelov v ONNX formáte na rôznych hardvérových platformách je dosiahnuté vďaka univerzálному inferenčnému virtuálnemu prostrediu ONNX Runtime, v ktorom sú modely vo formáte ONNX akcelerované. [38, 39].



Obr. 32: Schéma exportu a importu modelu vo formáte ONNX [39]

4.4 Mechanizmus na automatizovanú transkripciu

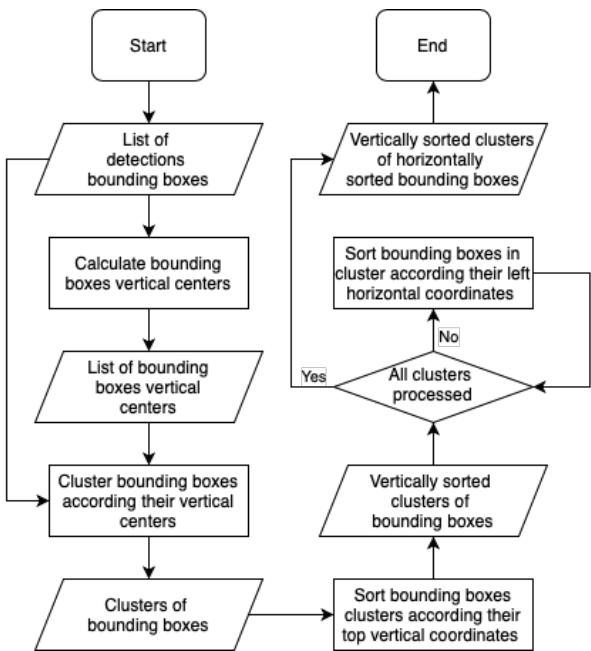
Pre optimálne fungovanie systémov na automatizované spracovanie historických rukopisov je nevyhnutná transkripcia detegovaných symbolov, teda prevod detekčného výstupu do textovej digitálnej reprezentácie. Nakoľko v našej práci vo väčšine prípadov pracujeme s rukopismi vo forme listov alebo denníkov, je pre transkripciu tohto typu dokumentov nevyhnutná separácia detegovaných symbolov do riadkov.

Súčasný mechanizmus na transkripciu detegovaných symbolov, fungujúci na báze hľadania lokálnych maxím histogramov stredov ohraničujúcich boxov, ktorý bol vyvinutý v rámci projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov, však prináša obmedzenie pri zoskupovaní mierne zakrivených a rotovaných riadkov. Z tohto dôvodu sme sa rozhodli pre návrh nového transkripčného algoritmu odolného voči miernemu zakriveniu riadkov, ktorý na ich separáciu využíva techniku strojového učenia bez učiteľa (UL), konkrétnie priestorové zhľukovanie (clustering) na základe hustoty (DBSCAN) [7].

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) predstavuje techniku zhľukovania na základe hustoty. DBSCAN funguje na princípe zoskupovania (zhľukovania) bodov na základe merania ich vzájomných vzdialenosí a minimálneho počtu bodov. Na rozdiel od iných zhľukovacích techník nevyžaduje explicitnú špecifikáciu počtu výsledných zhľukov [41, 42].

Nami vyvinutý mechanizmus na automatizovanú transkripciu detegovaných symbolov pri dostupnosti výsledných detekčných ohraničujúcich boxov (bounding box) pozostáva z krokov:

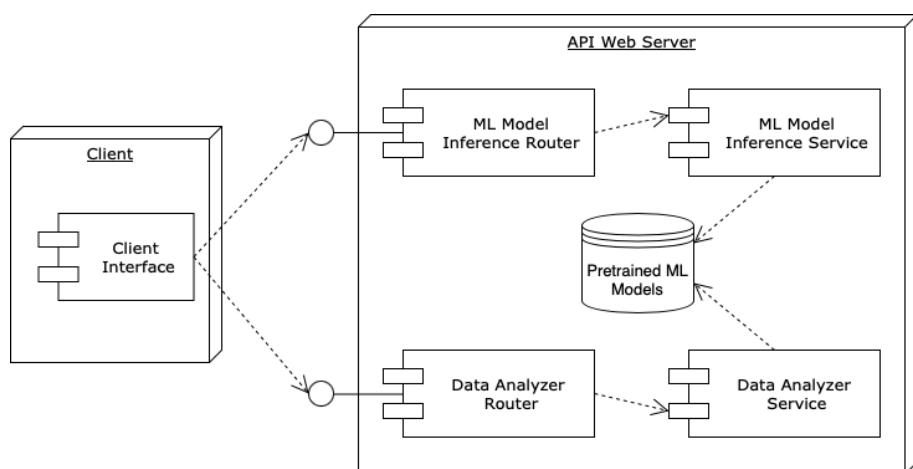
1. extrakcia vertikálnych stredových súradníc výsledných detekčných boxov,
2. zhľukovanie výsledných detekčných boxov technikou DBSCAN na základe vertikálnych stredových súradníc do riadkov,
3. vertikálne zoradenie výsledných zhľukov (riadkov) detekčných boxov,
4. horizontálne zoradenie detekčných boxov vo výsledných zhľukoch (riadkoch),
5. prepis identifikátorov/názvov tried do textovej reprezentácie na základe ich rozdelenia do zhľukov (riadkov) a vertikálneho, respektíve horizontálneho zoradenia.



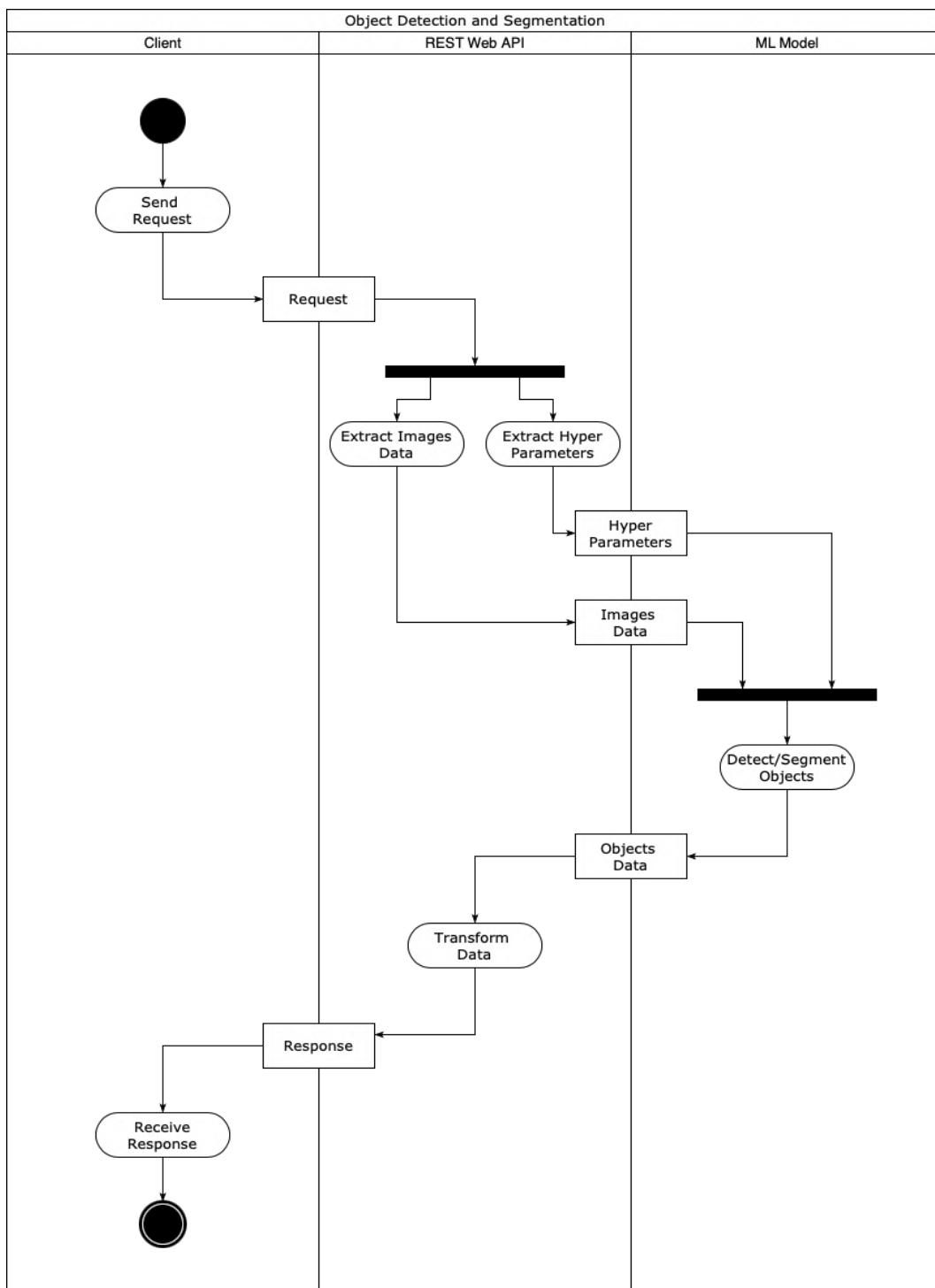
Obr. 33: Vývojový diagram algoritmu na zoskupenie detegovaných symbolov do riadkov s využitím DBSCAN zhľukovania

4.5 Webové API

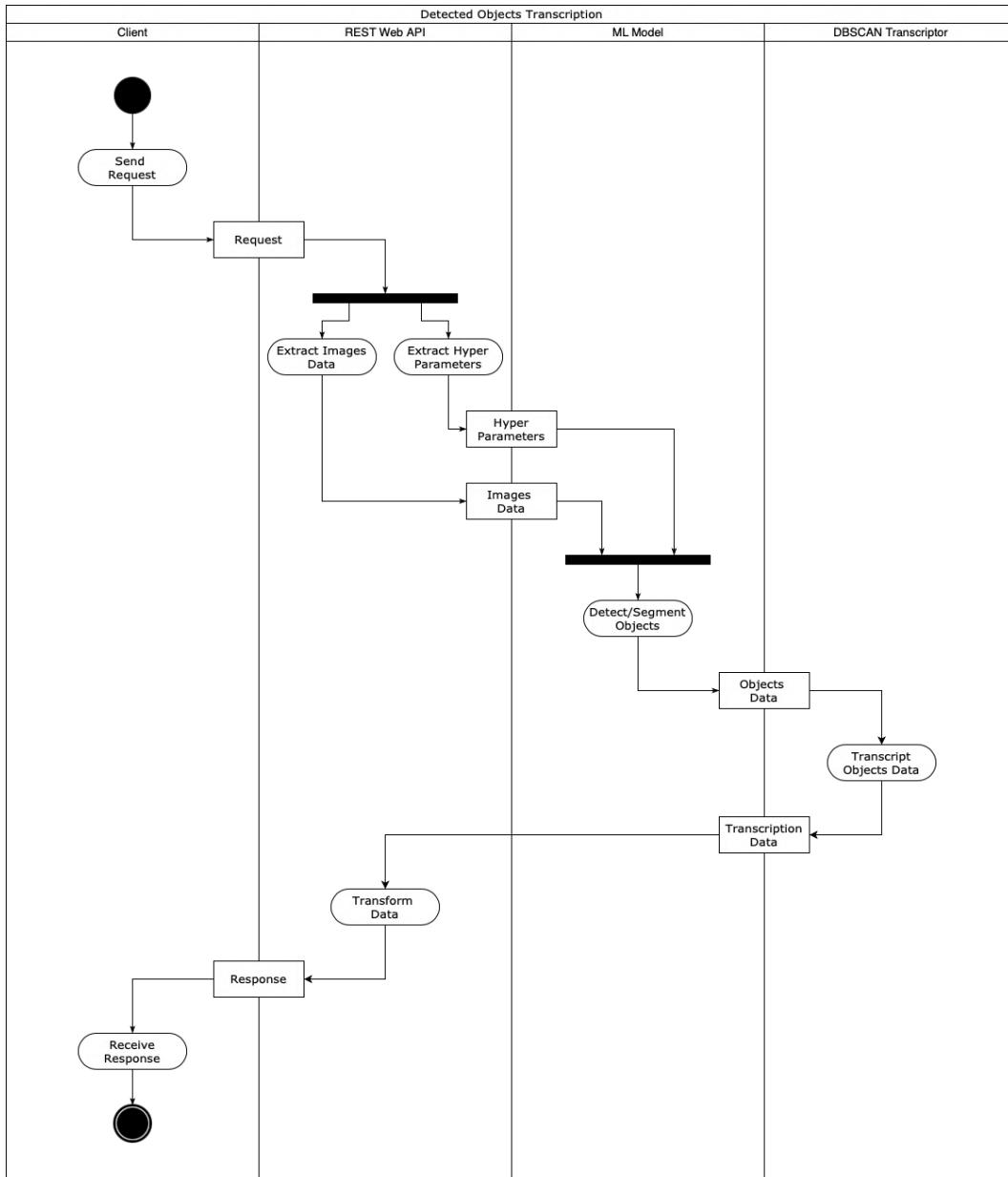
Po dotrénovaní modelov, výbere najúspešnejších z nich a implementácií mechanizmu na automatizovanú transkripciu je ďalším krokom integrácia týchto modelov a tohto mechanizmu do nami implementovaného webového aplikačného rozhrania (API), ktoré okrem detekčnej, segmentačnej a transkripčnej funkcionality obsahuje aj funkcionality na automatizované generovanie anotácií datasetov a pokročilé prehľadávanie datasetov, a to bez potreby hlbšej znalosti aplikačných rozhraní použitých knižníc a modulov.



Obr. 34: Diagram komponentov webového ML REST API

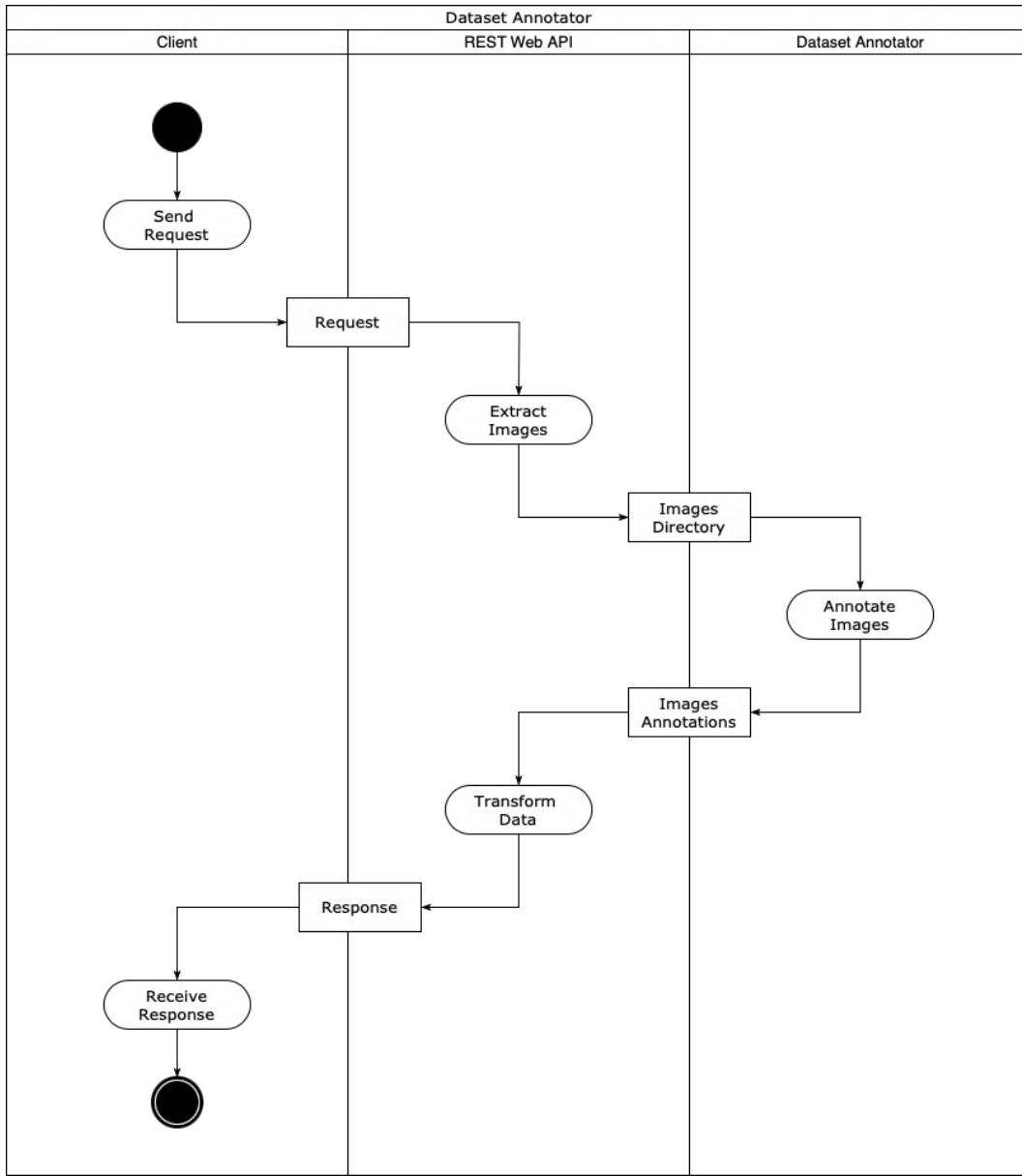


Obr. 35: Diagram aktivít funkcionality detekcie a segmentácie objektov webového ML REST API



Obr. 36: Diagram aktivít funkcionality detekcie a transkripcie objektov webového ML REST API

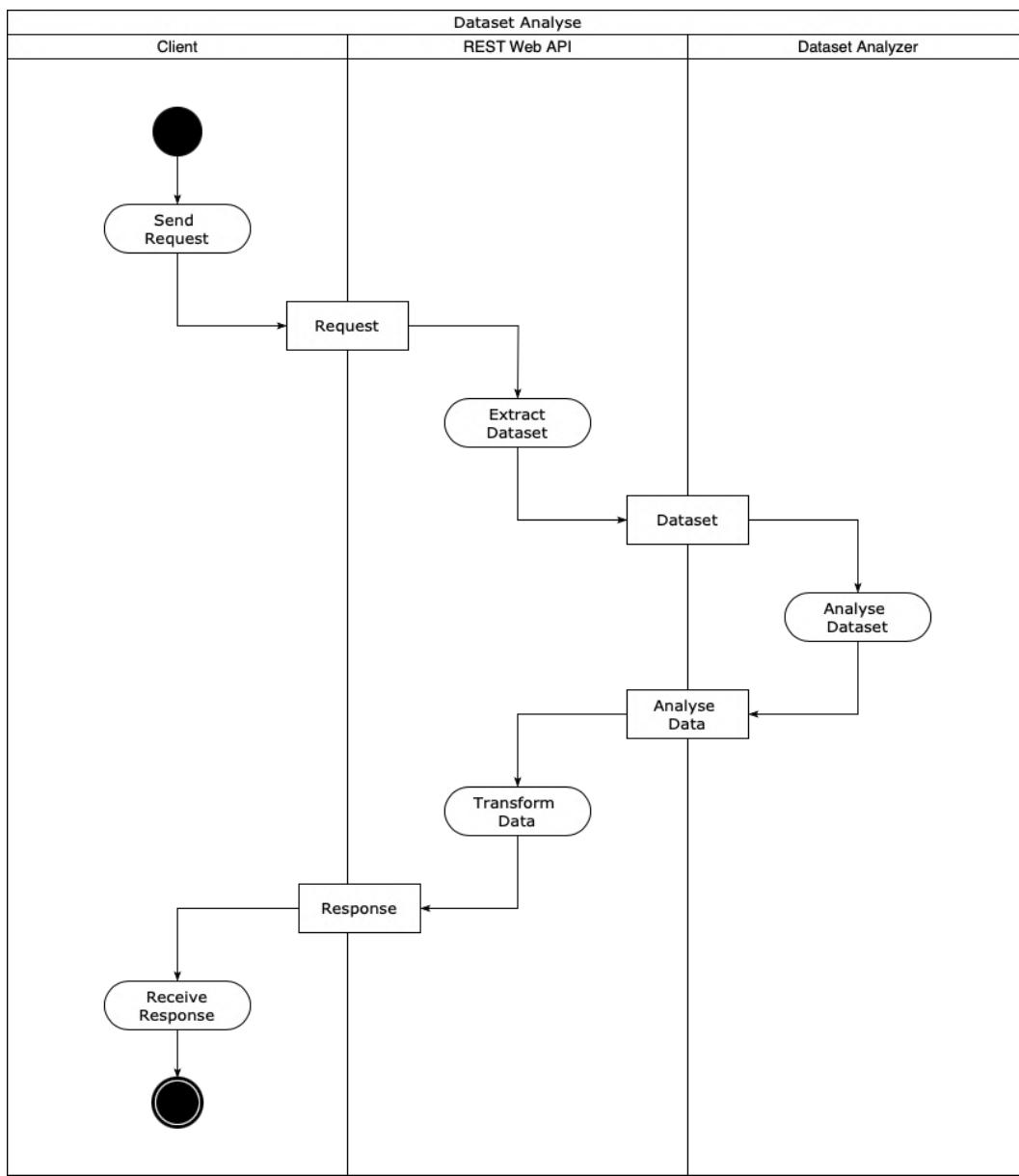
Funkcionality webového API sú sprostredkované pomocou požiadaviek (request) a odpovedí (response) cez HTTP/HTTPS protokol. Rozhranie webového API je typu REST a riešenie je implementované podľa architektúry Model-View-Controller (MVC), pričom v našej práci pokrývame komponenty Model a Controller [43, 44].



Obr. 37: Diagram aktivít funkcionality automatizovaného generovania anotácií datasetov webového ML REST API

Implementácia taktiež splňa Three-Tier klient-server architektúru, jej časti tým pádom sú rozdelené do [43, 44]:

- Presentation tier (Routers, Models),
- Logic tier (Services),
- Data tier (Model weights).



Obr. 38: Diagram aktivít funkcionality analýzy datasetov webového ML REST API

4.6 Volba nástrojov

Vzhľadom na povahu problematiky našej diplomovej práce implementácia prebehla v programovacom jazyku *Python* vo verzii *3.10*. Python ponúka širokú paletu knižníc na prácu s dátami a aplikačných rámcov s implementáciami najmodernejších ML modelov, pričom na ich správu využívame správcu balíčkov *Pip* [45, 46].

Pri práci s vektormi, maticami, tenzormi a viacrozmernými polami využívame knižnicu *NumPy*, na prácu s obrazovými súbormi využívame knižnicu *Pillow*. Na automatizovanú transkripciu detegovaných symbolov využívame implementáciu zhlukovacej techniky DB-SCAN z knižnice *Scikit-Learn* [47, 48, 49].

Na prácu s obrazovými dátami využívame knižnicu počítačového videnia *Supervision*, ktorá obsahuje aj implementáciu techniky SAHI na detekciu malých objektov [50].

Ďalej využívame knižnicu strojového učenia a počítačového videnia *Ultralytics*, ktorá okrem implementácie modelov YOLOv8, YOLOv9, RT-DETR, FastSAM a YOLO-World obsahuje aj modul Explorer s pokročilými AI funkcionálitami na analýzu a prehľadávanie obrazových datasetov vo formáte YOLO a modul Annotator na AI automatizované vytváranie anotácií vo formáte YOLO [24].

Na dotrénovanie modelu YOLO-NAS využívame knižnicu *SuperGradients* obsahujúcu jeho pôvodnú implementáciu [25].

Na implementáciu webového REST API rozhrania využívame rýchlu a flexibilnú knižnicu na tvorbu REST rozhranií v jazyku Python, *FastAPI*, ktorá podporuje automatickú validáciu formátu tiel požiadaviek (request) a odpovedí (response) pomocou knižnice *Pydantic*, a na zaručenie asynchronného (konkurentného) vykonávania služieb (service) pracuje s knižnicou *Starlette*. Riešenie následne beží na serveri *Uvicorn*. Na vykonanie zátažových testov webového REST API využívame aplikačný rámec *Locust* [51, 52].

Uvedené knižnice a moduly na pozadí pracujú s ďalšími knižnicami, ktoré však v rámci implementácie nevyužívame priamo, v našej práci teda nie sú ďalej spomenuté.

5 Implementácia a dosiahnuté výsledky

Nami osvojené teoretické poznatky, vypracovanú analýzu a návrh riešenia využívame v implementačnej časti našej práce. Implementačná časť v sebe zahŕňa trénovanie, testovanie a export vybraných detekčných/segmentačných modelov. Ďalej sa v tejto časti zameriavame na implementáciu nástrojov na automatizovanú transkripciu symbolov, na pokročilé prehľadávanie datasetov a generovanie YOLO anotácií s využitím AI. Tieto nástroje a vyexportované modely následne integrujeme do implementácie webového aplikačného rozhrania (API) typu REST. Štruktúru elektronickej prílohy s kompletným výstupom z implementačnej časti uvádzame v prílohe A, používateľská príručka je uvedená v prílohe B.

5.1 Modely

Prvý krok implementačnej časti našej práce predstavuje trénovanie, testovanie, evaluácia a export detekčných/segmentačných ML modelov. Pre potreby nami riešenej problematiky vyberáme modely YOLOv8, YOLO-NAS, YOLOv9, RT-DETR, YOLO-World a FastSAM, pričom na zlepšenie detekcie malých objektov využívame techniku SAHI.

Predtrénovaný referenčný model dotrénujeme na VEGA Cropped datasetoch číslic a glyfov počas 300 epoch, pričom pri trénovaní využívame predvolené hyperparametre. Počet epoch bol zvolený na základe konvergencie hodnôt chýb modelov pri trénovaní k nižším hodnotám.

Schopnosti modelov testujeme a vyhodnocujeme pomocou štandardných ML metrík, konkrétnie Precision (P), Recall (R) a Mean Average Precision (mAP50, mAP50-95).

Modely po natrénovaní a evaluácii následne exportujeme do formátov PyTorch a ONNX, ktorých inferenčné časy porovnáme.

5.1.1 YOLOv8

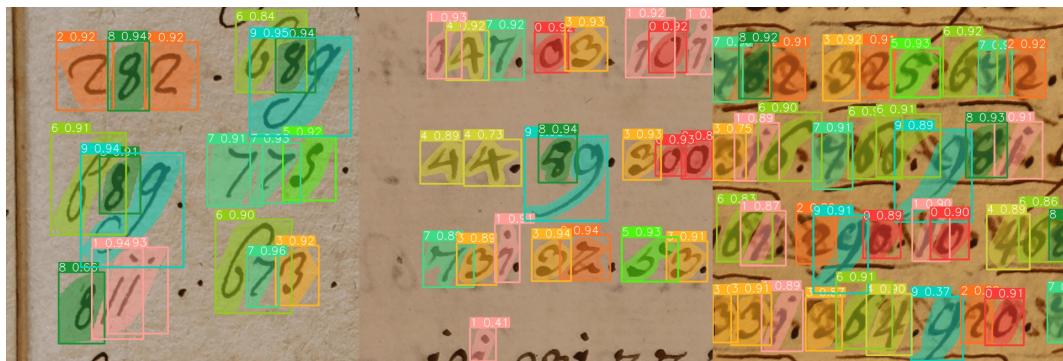
Prvý využitý detektor v našej práci je model z rodiny YOLO vo verzii 8. V našej práci pracujeme s implementáciou modelu YOLOv8 obsiahnutou v knižnici *Ultralytics*, pričom využívame jeho segmentačnú alternatívu vo veľkosti l (YOLOv8l-seg) s 46.0 miliónmi parametrov, ktorá na referenčnom COCO datasete dosahuje nasledujúce výsledky [24]:

Referenčný model dotrénujeme na VEGA Cropped datasetoch číslic a glyfov, pričom sú zachované predvolené hyperparametre. Trénovanie prebieha v 300 epochách (*epochs*) s veľkosťou vstupu (*imgsz*) 640 pixelov, veľkosťou dávky (*batch*) 4, aktívnou validáciou (*val*) a predčasným zastavením trénovania (early stopping) s trpezlivosťou (*patience*) 50 epoch [24].

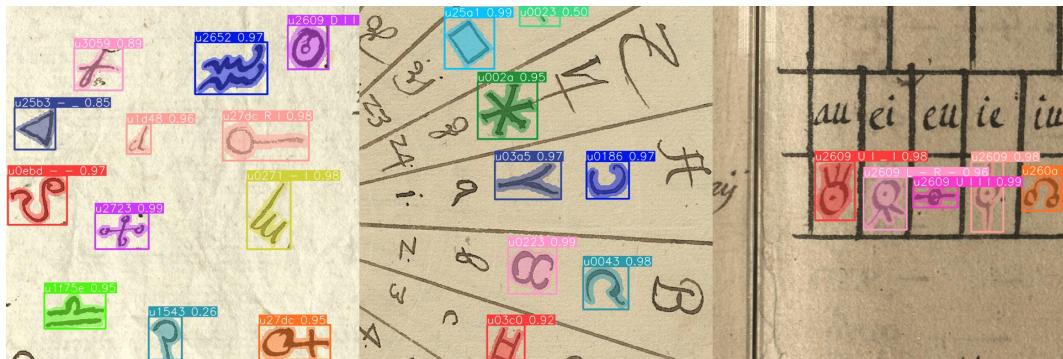
| | Size
(px) | Box
(mAP50-95) | Mask
(mAP50-95) | Params
(M) | FLOPs
(B) |
|-------------|--------------|-------------------|--------------------|---------------|--------------|
| YOLOv8l-seg | 640 | 0.523 | 0.426 | 46.0 | 220.5 |

Tabuľka 2: Úspešnosť modelu YOLOv8l-seg na testovacej množine referenčného datasetu COCO [24]

Po dokončení trénovalia modelov pomocou inferenčného módu spúšťame detekciu a segmentáciu na náhodných vzorkách z testovacej množiny pri veľkosti vstupu (*imgsz*) 640 x 640 pixelov a prahovej hodnote (*conf*) 0.25 [24].



Obr. 39: Ukážka detekcie a segmentácie číslí modelom YOLOv8 na testovacích vzorkách VEGA Cropped



Obr. 40: Ukážka detekcie a segmentácie glyfov modelom YOLOv8 na testovacích vzorkách VEGA Cropped

Pre evaluáciu úspešnosti trénovalého modelu je nutné model spustiť vo validačnom režime na testovacích dátach. Schopnosť modelu detekcie a segmentácie objektov vyčíslime pomocou ML metrič pre ohraničujúce detekčné boxy (Box) a segmentačné polygonálne masky (Mask) [24].

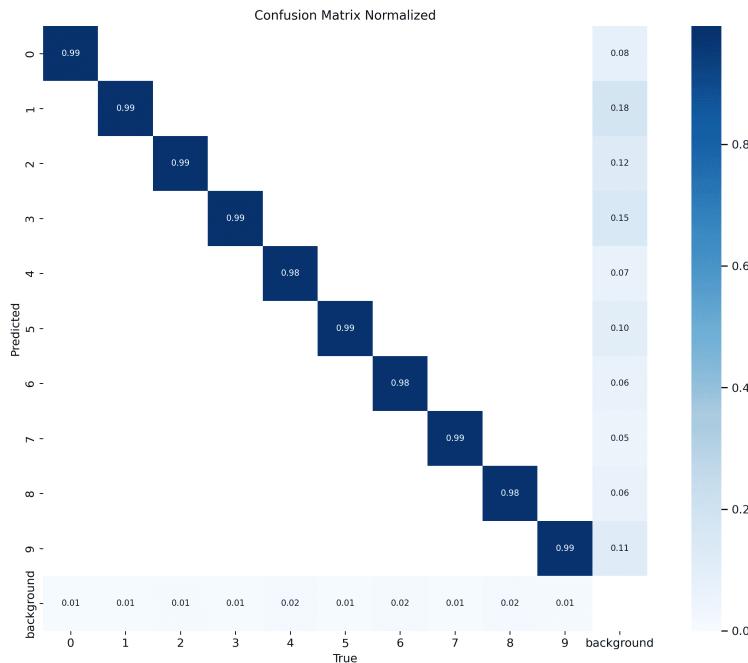
| | P | R | mAP50 | mAP50-95 |
|-------------|-------|-------|-------|----------|
| Box | 0.974 | 0.984 | 0.989 | 0.819 |
| Mask | 0.974 | 0.984 | 0.989 | 0.756 |

Tabuľka 3: Úspešnosť modelu YOLOv8 na testovacej množine číslí datasetu VEGA Cropped

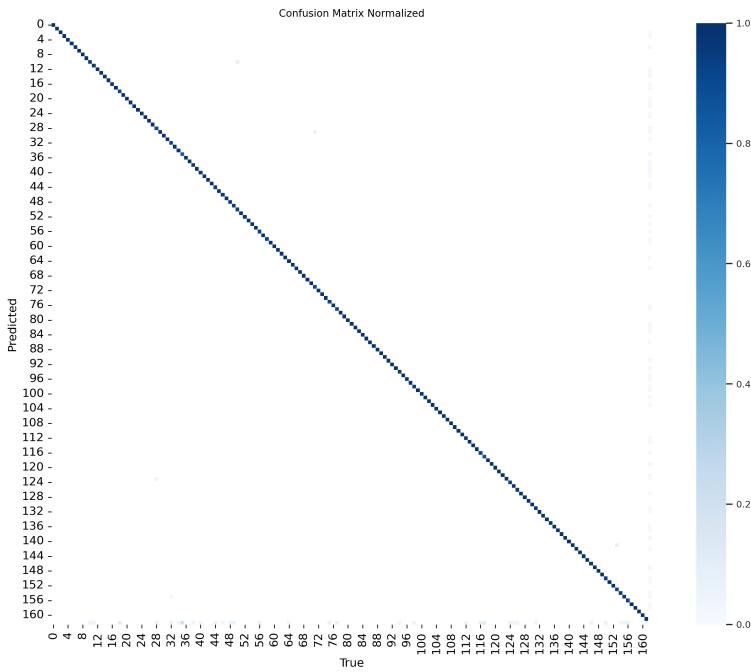
| | P | R | mAP50 | mAP50-95 |
|-------------|-------|-------|-------|----------|
| Box | 0.979 | 0.976 | 0.989 | 0.981 |
| Mask | 0.974 | 0.976 | 0.989 | 0.906 |

Tabuľka 4: Úspešnosť modelu YOLOv8 na testovacej množine glyfov datasetu VEGA Cropped

Úspešnosť modelov pri klasifikácii následne vyhodnocujeme pomocou normalizovanej konfúznej matice, ktorá je automaticky generovaná po dokončení testovania vo validačnom režime [24].



Obr. 41: Konfúzna matica modelu YOLOv8 na testovacej množine číslí datasetu VEGA Cropped



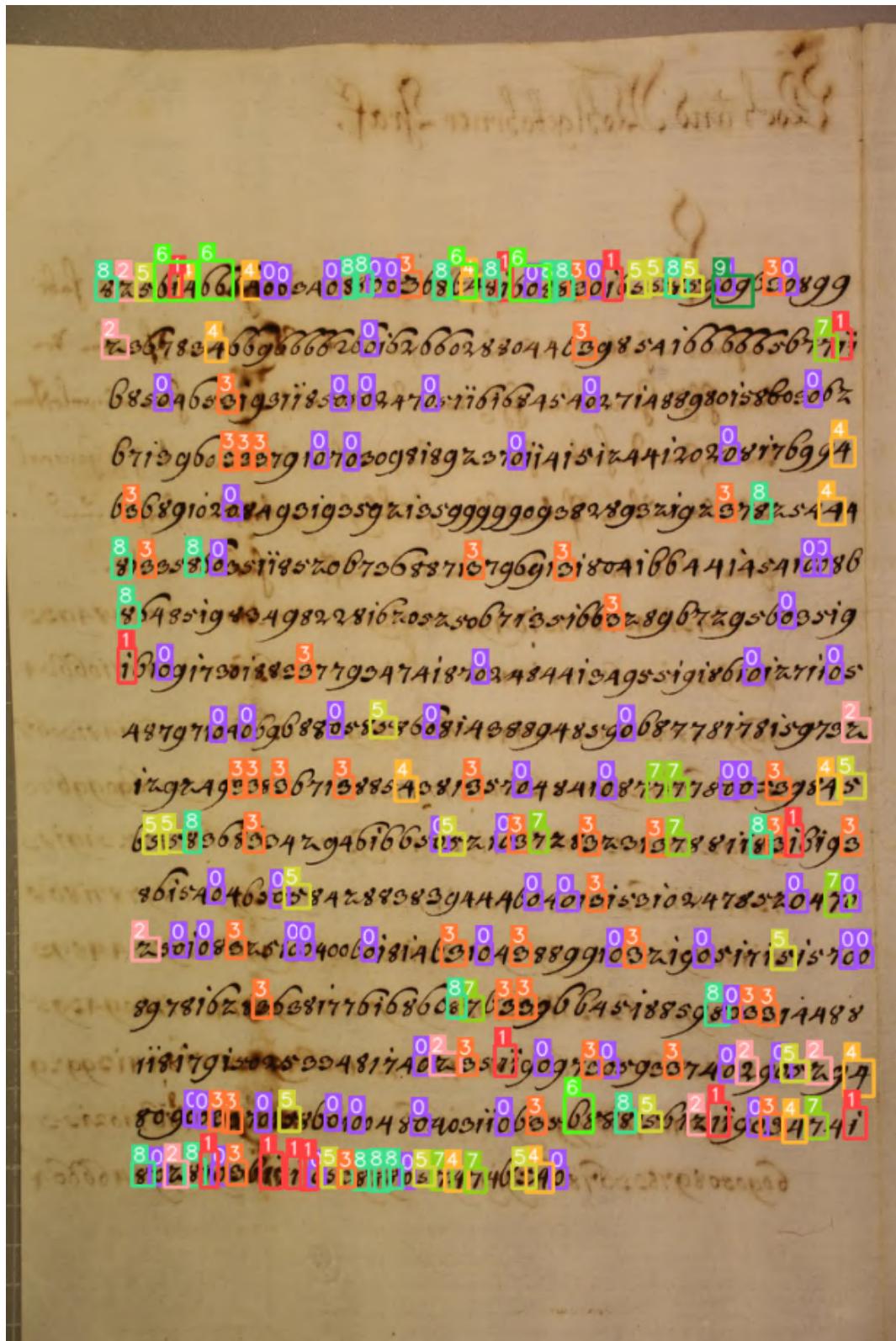
Obr. 42: Konfúzna matica modelu YOLOv8 na testovacej množine glyfov datasetu VEGA Cropped

Z výsledkov inferencie a testovania je zrejmé, že oba modely dosahujú výborné výsledky detekcie, segmentácie aj klasifikácie číslíc a symbolov, napäťo pri porovnaní výsledných hodnôt metrík s pôvodným referenčným modelom trénovanom na datasete COCO dosahujú výrazne lepšie výsledky [24].

Výrazné zlepšenie je tiež zrejmé pri porovnaní so staršími architektúrami modelov, ako napríklad Mask R-CNN, YOLOv5 alebo YOLOv7, ktoré boli natrénované v rámci starších bakalárskych a diplomových prác v rámci projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov na pracovisku ÚIM FEI STU [2, 6, 7, 15].

Vzhľadom na potrebu detekcie a segmentácie objektov na celých historických digitalizovaných rukopisoch sú oba modely testované v inferenčnom móde na vzorkách celých historických dokumentov obsahujúcich číslice alebo glyfy.

Výsledné výstupy ukazujú, že aj napriek pokročilej a komplexnej architektúre modelu YOLOv8 je detekcia alebo segmentácia veľmi malých, husto rozmiestených, objektov problematická, čo tomuto modelu v jeho elementárnej podobe aj napriek výborným výsledkom počas testovania znemožňuje jeho nasadenie v scenároch reálnej transkripcie digitalizovaných historických dokumentov.



Obr. 43: Ukážka detekcie číslí modelom YOLOv8 na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI)

5.1.2 YOLO-NAS

Ďalší detektor, ktorý v rámci našej implementačnej časti využívame, je model YOLO-NAS vyvinutý spoločnosťou Deci AI s dôrazom na detekciu malých objektov. Vzhľadom na potrebu dotrénovania modelu využívame implementáciu YOLO-NAS z knižnice *SuperGradients* od spoločnosti Deci AI. Pri absencii segmentačnej alternatívy pracujeme s detekčnou verziou velkosti l (`yolo_nas_l`) s 66.9 miliónmi parametrov, ktorá na COCO datasete dosahuje nasledujúcu úspešnosť [24]:

| | Size
(px) | Box
(mAP50-95) | Params
(M) |
|-------------------|--------------|-------------------|---------------|
| YOLO-NAS-l | 640 | 0.522 | 66.9 |

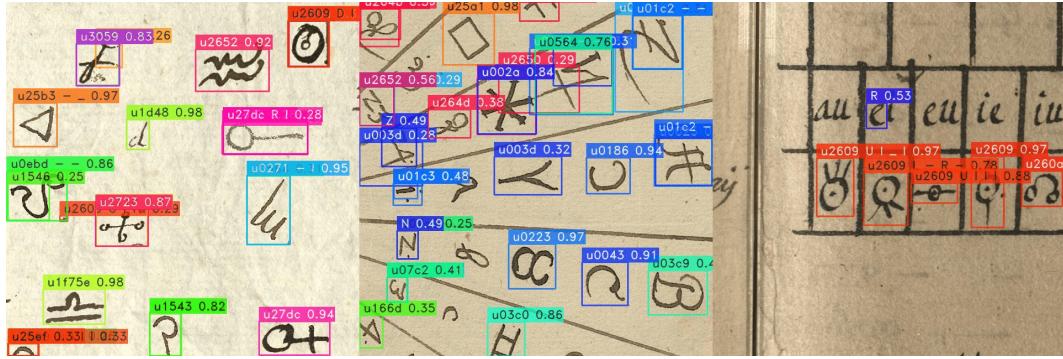
Tabuľka 5: Úspešnosť modelu YOLO-NAS-l na testovacej množine referenčného datasetu COCO [24]

Referenčný model zhodne ako v predošлом prípade dotrénujeme na VEGA Cropped dátových množinách číslíc a glyfov pri použití predvolených hyperparametrov, 400 epochách (*epochs*), veľkosťou vstupu (*imgsz*) 640 pixelov a veľkosťou dávky (*batch*) 8 [24, 25].

Po dotrénovaní modelov spúšťame detekciu na náhodných vzorkách z testovacej množiny pri veľkosti vstupu (*imgsz*) 640 x 640 pixelov a prahovej hodnote (*conf*) 0.25 [24, 25].



Obr. 44: Ukážka detekcie číslíc modelom YOLO-NAS na testovacích vzorkách VEGA Cropped



Obr. 45: Ukážka detekcie glyfov modelom YOLO-NAS na testovacích vzorkách VEGA Cropped

Modely následne spúšťame vo validačnom režime pre evaluáciu a vyčíslenie štandardných detekčných metrík (Box) [24, 25].

| | P | R | mAP50 | mAP50-95 |
|-----|-------|-------|-------|----------|
| Box | 0.968 | 0.976 | 0.985 | 0.806 |

Tabuľka 6: Úspešnosť modelu YOLO-NAS na testovacej množine číslí datasetu VEGA Cropped

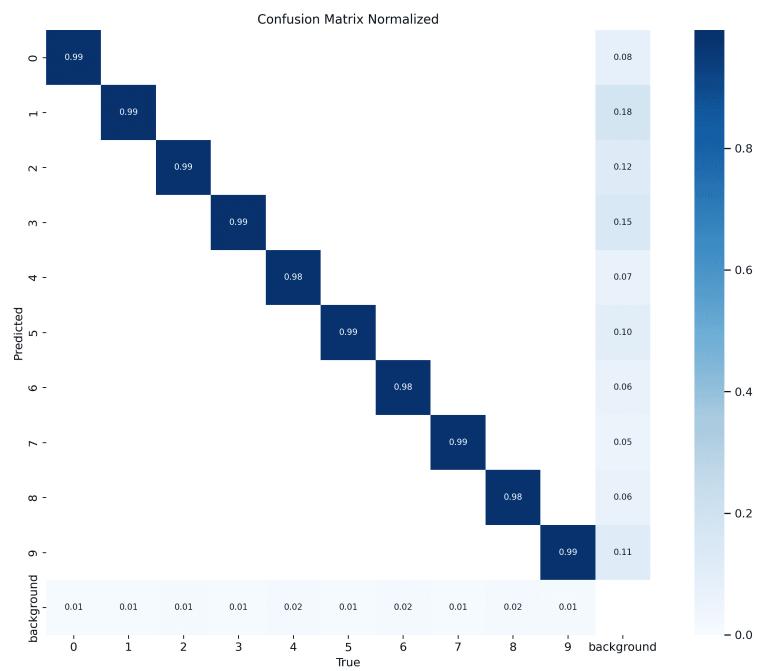
| | P | R | mAP50 | mAP50-95 |
|-----|-------|-------|-------|----------|
| Box | 0.972 | 0.967 | 0.978 | 0.964 |

Tabuľka 7: Úspešnosť modelu YOLO-NAS na testovacej množine glyfov datasetu VEGA Cropped

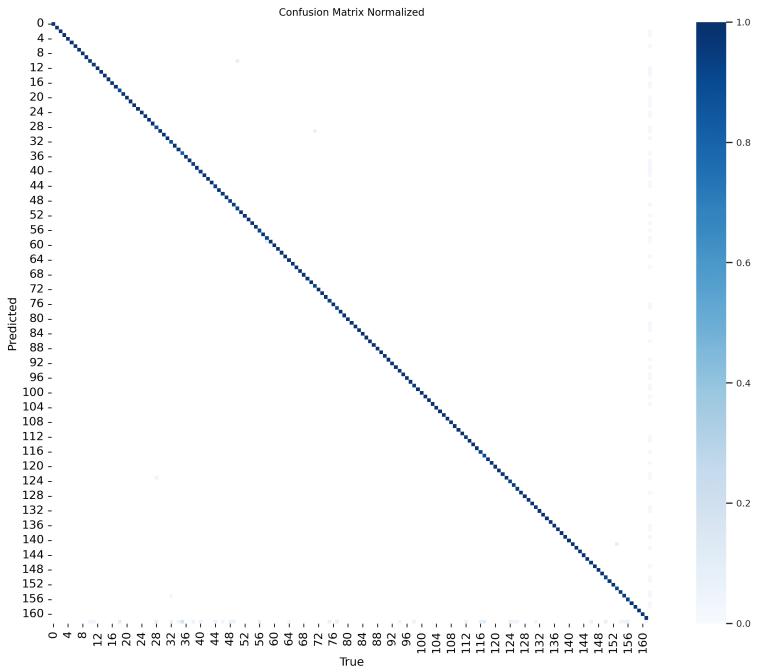
Schopnosť modelov klasifikovať detegované objekty vyhodnocujeme na základe vygenerovaných konfúznych matíc [24, 25].

Modely YOLO-NAS rovnako testujeme aj na celých stranach historických dokumentov [24, 25].

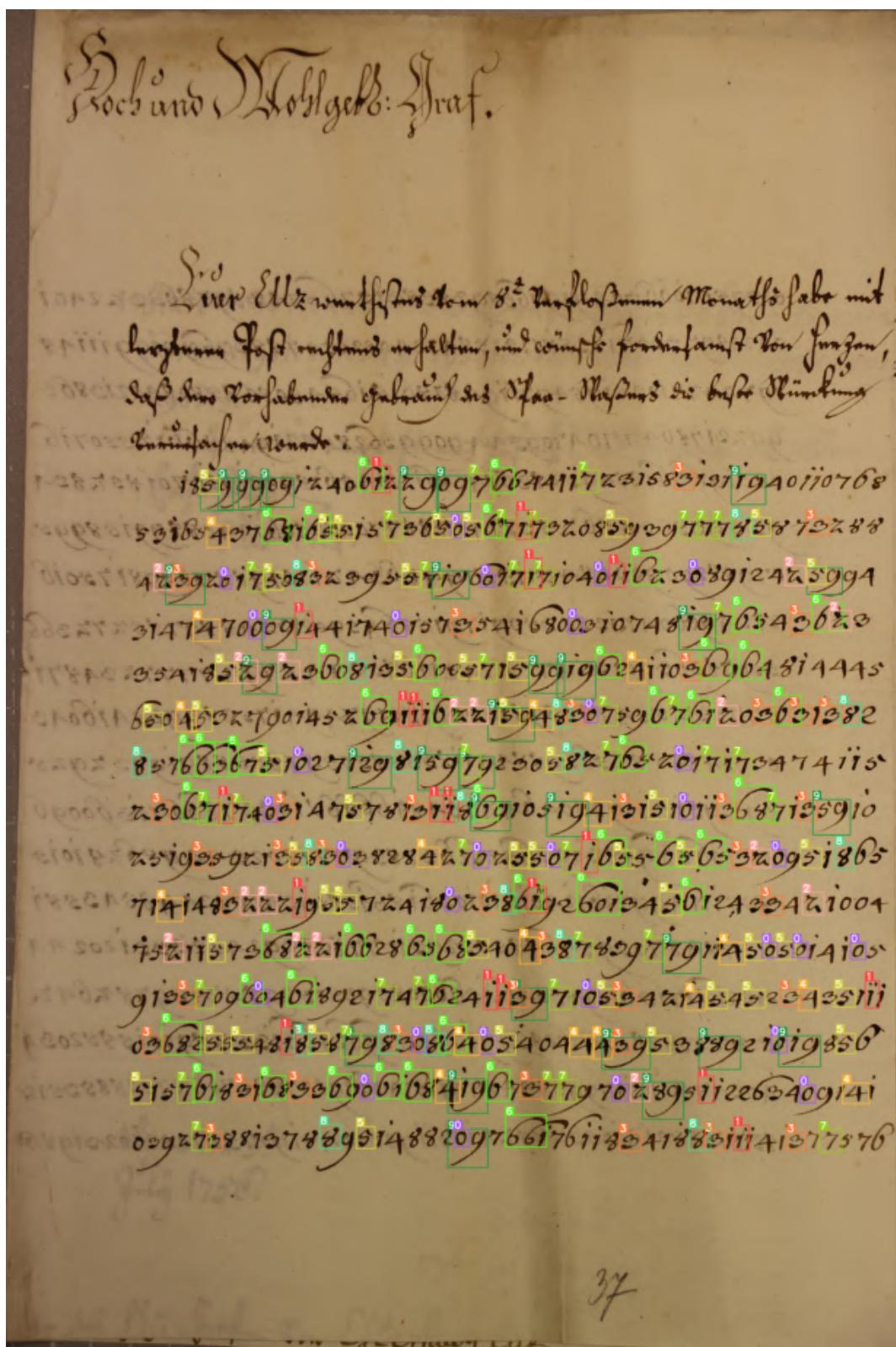
Modely YOLO-NAS skutočne dosahujú lepšie výsledky pri detekcii malých objektov ako modely YOLOv8, no napriek tomu ani zdaleka nedosahujú požadovanú úspešnosť v scenároch detekcie objektov na celých historických dokumentoch. Modely taktiež vykazujú podstatne zvýšenú chybovost vďaka falošným detekciám objektov (FP), čo ich v porovnaní s modelmi architektúry YOLOv8 značne znevýhodňuje. Schopnosť modelov objekty klasifikovať do výsledných tried je opäť na vysokej úrovni.



Obr. 46: Konfúzna matica modelu YOLO-NAS na testovacej množine číslí datasetu VEGA Cropped



Obr. 47: Konfúzna matica modelu YOLO-NAS na testovacej množine glyfov datasetu VEGA Cropped



Obr. 48: Ukážka detekcie číslí modelom YOLO-NAS na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI)

5.1.3 YOLOv9

Ďalším modelom použitým v našej práci je 9. iterácia z rodiny YOLO, YOLOv9. Pracujeme s implementáciou z knižnice *Ultralytics*, konkrétnie využívame segmentačný model veľkosti c (YOLOv9c-seg), ktorý pri evaluácii na datasete COCO dosahuje nasledujúce výsledky [24, 26]:

| | Size
(px) | Box
(mAP50-95) | Mask
(mAP50-95) | Params
(M) | FLOPs
(B) |
|-------------|--------------|-------------------|--------------------|---------------|--------------|
| YOLOv9c-seg | 640 | 0.524 | 0.422 | 27.9 | 159.4 |

Tabuľka 8: Úspešnosť modelu YOLOv9c-seg na testovacej množine referenčného datasetu COCO [24]

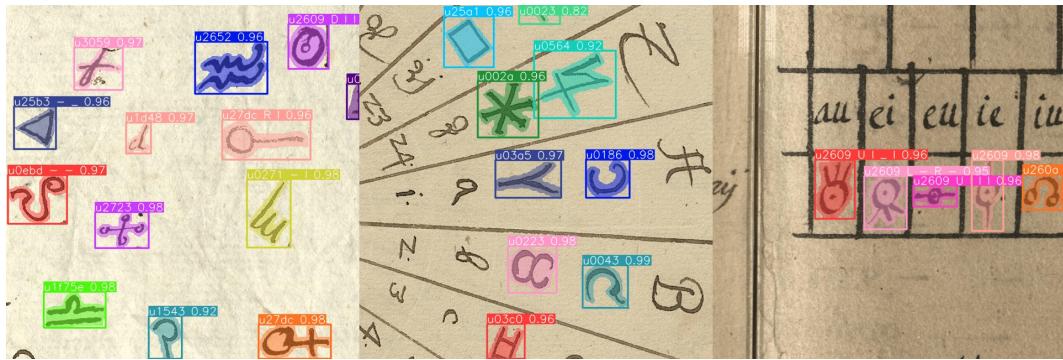
Referenčný model dotrénujeme na VEGA Cropped datasetoch číslíc a glyfov pri zachovaní predvolených hyperparametrov, 300 epochách (*epochs*), veľkosťou vstupu (*imgsz*) 640 pixelov a veľkosťou dávky (*batch*) 4, aktívnu validáciu (*val*) a predčasným zastavením trénovania (early stopping) s trpezlivosťou (*patience*) 50 epoch [24, 26].

Po dotrénovaní oba modely spúšťame v inferenčnom režime na náhodných vstupoch z testovacích množín datasetov číslíc a glyfov pri veľkosti vstupu (*imgsz*) 640 x 640 pixelov a prahovej hodnote (*conf*) 0.25 [24, 26].



Obr. 49: Ukážka detekcie a segmentácie číslíc modelom YOLOv9 na testovacích vzorkách VEGA Cropped

Pre evaluáciu a vyčíslenie úspešnosti detekcie (Box), respektívne segmentácie (Mask), modely spúšťame vo validačnom režime [24, 26].



Obr. 50: Ukážka detekcie a segmentácie glyfov modelom YOLOv9 na testovacích vzorkách VEGA Cropped

Schopnosť klasifikácie detegovaných a segmentovaných objektov vyhodnocujeme pomocou normalizovaných konfúznych matíc [24, 26].

Oba YOLOv9 modely opäťovne testujeme na vstupoch s celými stranami historických rukopisov.

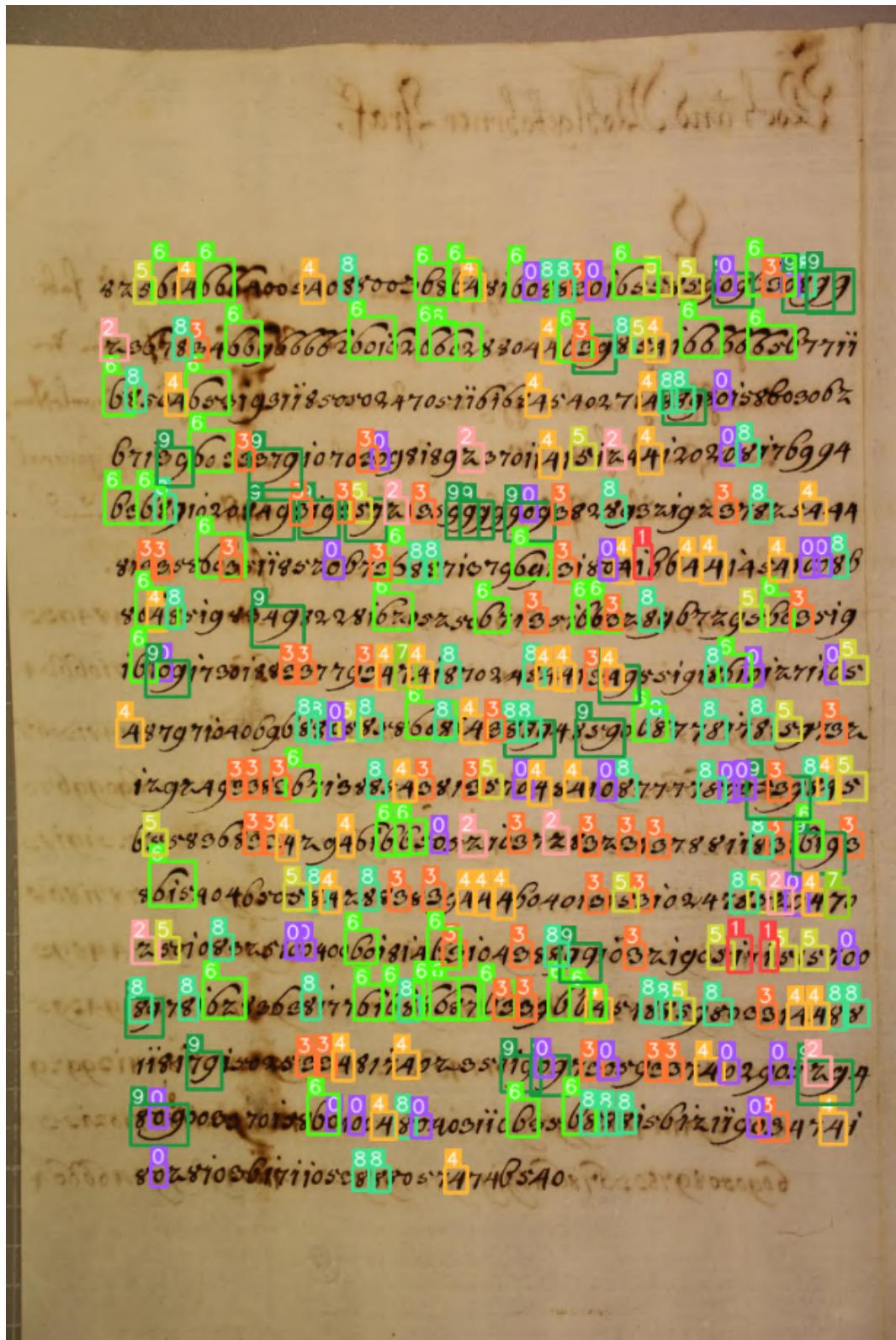
| | P | R | mAP50 | mAP50-95 |
|-------------|-------|-------|-------|----------|
| Box | 0.975 | 0.987 | 0.988 | 0.819 |
| Mask | 0.975 | 0.987 | 0.988 | 0.756 |

Tabuľka 9: Úspešnosť modelu YOLOv9 na testovacej množine číslí datasetu VEGA Cropped

| | P | R | mAP50 | mAP50-95 |
|-------------|-------|-------|-------|----------|
| Box | 0.980 | 0.982 | 0.991 | 0.983 |
| Mask | 0.980 | 0.982 | 0.991 | 0.905 |

Tabuľka 10: Úspešnosť modelu YOLOv9 na testovacej množine glyfov datasetu VEGA Cropped

Modely YOLOv9 dosahujú podobne ako model YOLO-NAS zlepšenie pri detekcii objektov menšej mierky, no opäť nie sú vhodné pri použití na celých historických dokumentoch a trpia vyšším počtom falošných detekcií (FP). Klasifikovanie objektov do tried je však rovnako veľmi presné.



Obr. 53: Ukážka detekcie číslí modelom YOLOv9 na VEGA vzorke celého historického dokumentu (bez použitia techniky SAHI)

5.1.4 RT-DETR

Ďalší detektor používaný v rámci našej práce reprezentuje model architektúry obrazového transformera (ViT), RT-DETR. Pracujeme s implementáciou obsiahnutou v knižnici *Ultralytics* v jej detekčnej verzii vo veľkosti l (rtdetr-l) s nasledujúcimi výsledkami na datasete COCO [21, 24]:

| | Size
(px) | Box
(mAP50-95) | Params
(M) | FLOPs
(B) |
|-----------------|--------------|-------------------|---------------|--------------|
| RT-DETRl | 640 | 0.531 | 42.0 | 108.0 |

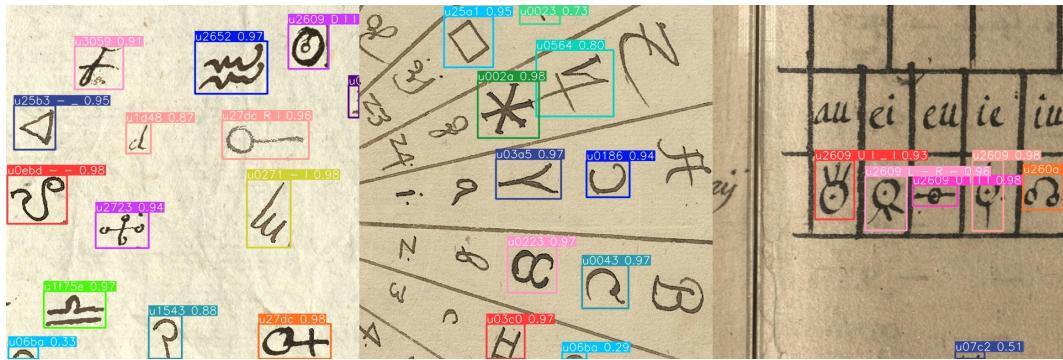
Tabuľka 11: Úspešnosť modelu RT-DETRl na testovacej množine referenčného datasetu COCO [24]

Základný model RT-DETRl predtrénovaný na referenčnom datasete COCO dotrénujeme na dátových množinách číslí a glyfov datasetu VEGA Cropped. Rovnako ako pri modeloch YOLOv8 a YOLOv9 zachovávame predvolené hyperparametre, počet epoch (*epochs*) nastavíme na 300, veľkosť vstupu (*imgsz*) na 640 pixelov, veľkosť dávky (*batch*) na 4 a trpezlivosť (*patience*) 50 epoch pri aktívnom predčasnom zastavení trénovalia pri indikácii pretrénovania (early stopping) [21, 24].

Pre vizuálnu kontrolu detekcie na náhodných obrazových vzorkách testovacej podmnožiny oboch VEGA Cropped datasetov využívame po dotrénovali modelov inferenčný režim s veľkosťou vstupu (*imgsz*) 640 pixelov a prahovou hodnotou detekcie (*conf*) 0.25 [21, 24].



Obr. 54: Ukážka detekcie číslí modelom RT-DETR na testovacích vzorkách VEGA Cropped



Obr. 55: Ukážka detekcie glyfov modelom RT-DETR na testovacích vzorkách VEGA Cropped

Modely opäť spúšťame aj vo validačnom režime na testovacích podmnožinách VEGA Cropped datasetov pre evaluáciu detekčných schopností (Box), klasifikačné schopnosti overujeme pomocou konfúznych matíc [21, 24].

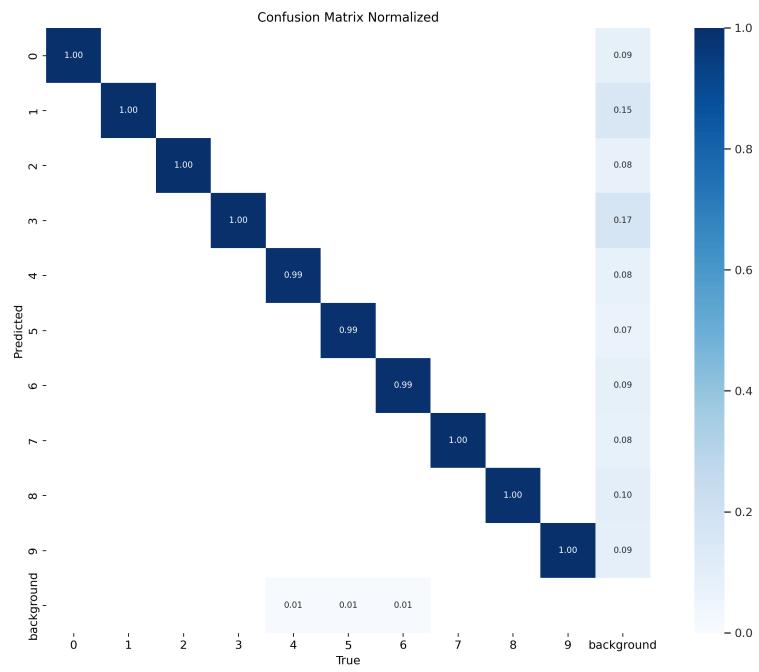
| | P | R | mAP50 | mAP50-95 |
|-----|-------|-------|-------|----------|
| Box | 0.953 | 0.963 | 0.981 | 0.784 |

Tabuľka 12: Úspešnosť modelu RT-DETR na testovacej množine číslic datasetu VEGA Cropped

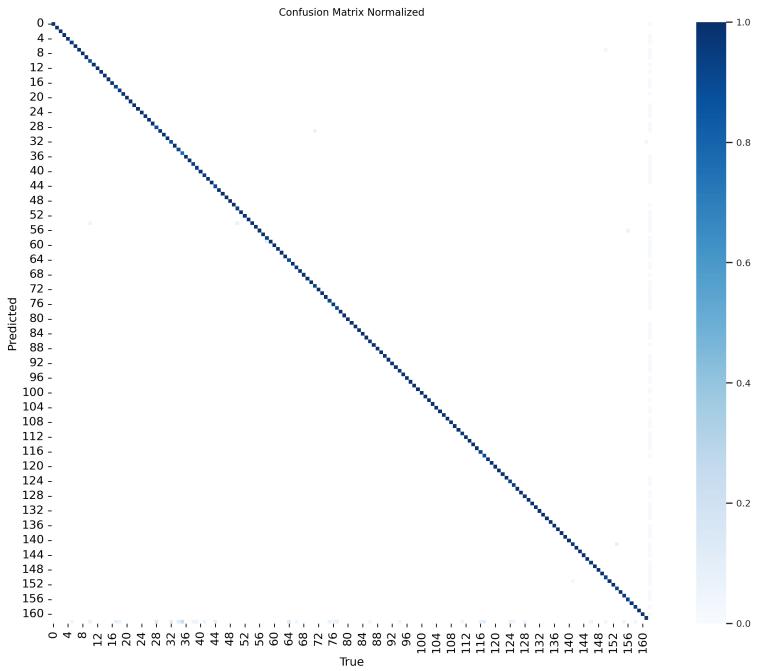
| | P | R | mAP50 | mAP50-95 |
|-----|-------|-------|-------|----------|
| Box | 0.846 | 0.959 | 0.955 | 0.940 |

Tabuľka 13: Úspešnosť modelu RT-DETR na testovacej množine glyfov datasetu VEGA Cropped

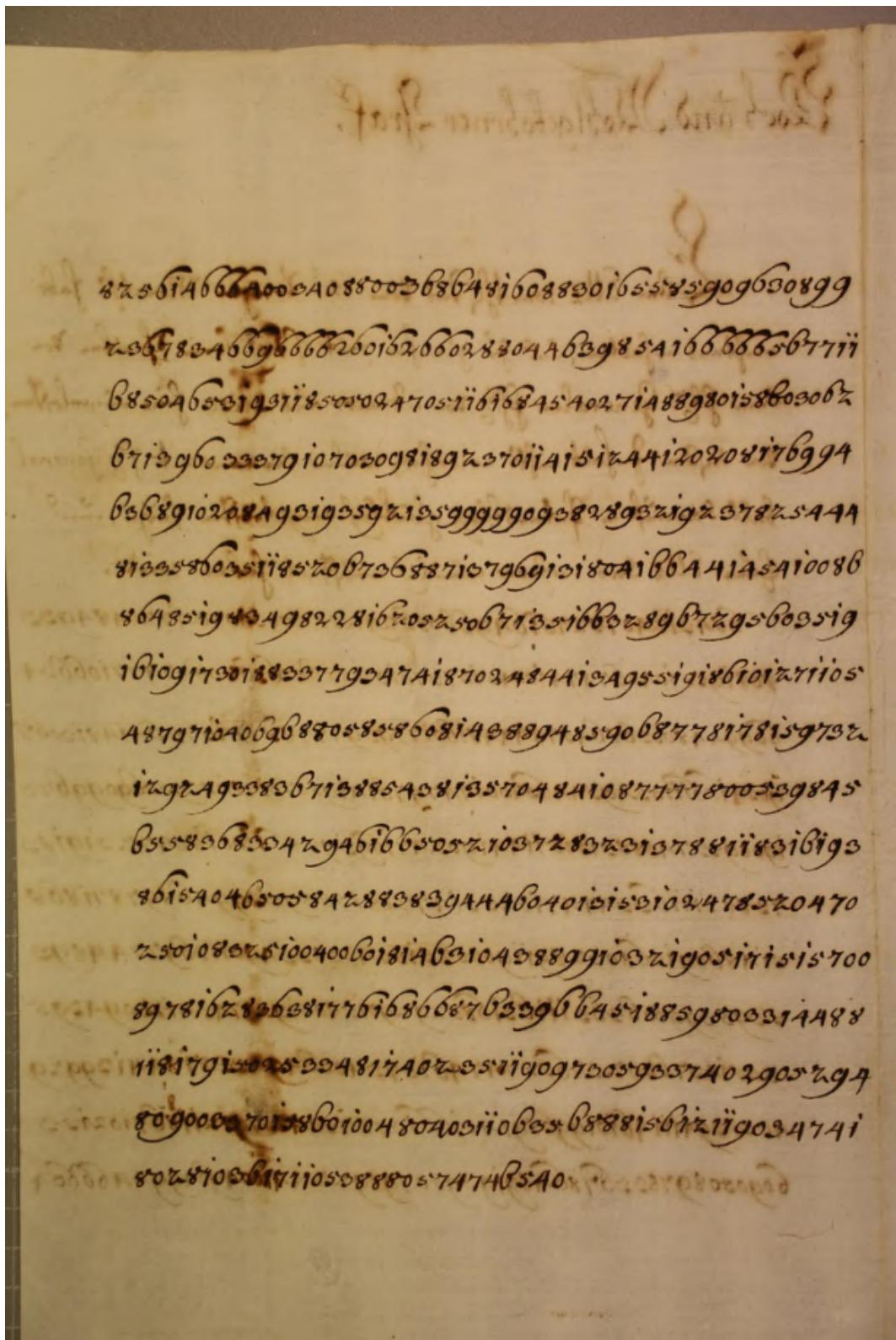
Dotrénované modely architektúry RT-DETR dosahujú horšie výsledky detekcie pri testovaní na výsekokach ako aj celých historických dokumentoch, pri ktorých modely nedetegujú žiadne objekty. Táto skutočnosť je zapríčinená komplikovanou a robustnou architektúrou, ktorá pre relevantnejšie výsledky vyžaduje vyšší počet epoch pri trénovaní a výkonnejší výpočtový hardvér. Klasifikačné schopnosti sú opäť veľmi presné.



Obr. 56: Konfúzna matica modelu RT-DETR na testovacej množine číslic datasetu VEGA Cropped



Obr. 57: Konfúzna matica modelu RT-DETR na testovacej množine glyfov datasetu VEGA Cropped



Obr. 58: Ukážka detekcie číslíc modelom RT-DETR na VEGA vzorke celého historického dokumentu - model nedetegoval žiadnu číslicu (bez použitia techniky SAHI)

5.1.5 YOLO-World

Ďalší využitý model v implementačnej časti našej práce je open-vocabulary detektor (OVD) YOLO-Worldv2. Využitá je implementácia z knižnice *Ultralytics* vo veľkosti l (YOLOv8l-worldv2) s úspešnosťami na referenčnom datasete COCO [24, 33]:

| | Size
(px) | Box
(mAP50-95) | Params
(M) |
|----------------------|--------------|-------------------|---------------|
| YOLO-Worldv2l | 640 | 0.498 | 48.0 |

Tabuľka 14: Úspešnosť modelu YOLO-Worldv2l na testovacej množine referenčného datasetu COCO [24]

Vzhľadom na skutočnosť, že model YOLO-Worldv2 umožňuje príkazovú (prompt) dynamickú špecifikáciu tried bez nutnosti dotrénovania, je model evaluovaný aj v tomto režime. Referenčnému modelu konkrétnie špecifikujeme množinu tried obsiahnutú vo VEGA dátovej množine číslic, respektívne glyfov. Obe konfigurácie testujeme vo validačnom móde na testovacích podmnožinách VEGA Cropped datasetov [24, 33].

| | P | R | mAP50 | mAP50-95 |
|------------|----------|----------|-----------|-----------|
| Box | 0.000127 | 0.000265 | 0.0000638 | 0.0000191 |

Tabuľka 15: Úspešnosť modelu YOLO-Worldv2 pri dynamickej špecifikácii tried na testovacej množine číslic datasetu VEGA Cropped

| | P | R | mAP50 | mAP50-95 |
|------------|--------|------|---------|----------|
| Box | 0.0031 | 0.37 | 0.00446 | 0.0031 |

Tabuľka 16: Úspešnosť modelu YOLO-Worldv2 pri dynamickej špecifikácii tried na testovacej množine glyfov datasetu VEGA Cropped

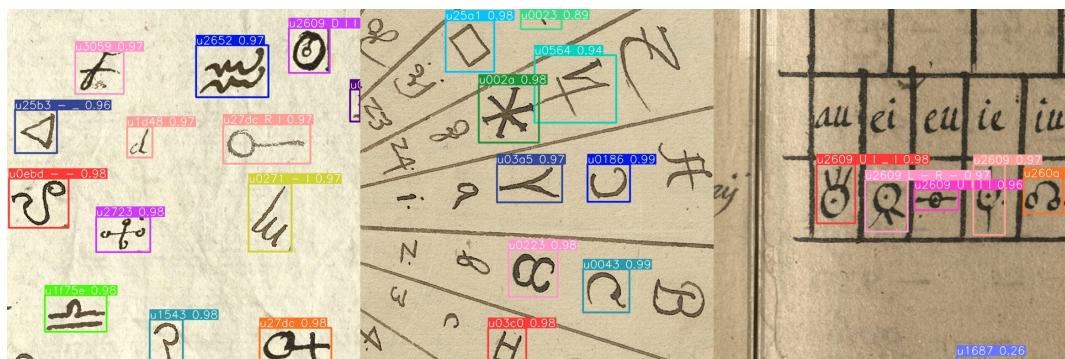
Výsledky dosiahnuté pri evaluácii modelu s dynamicky špecifikovanými triedami číslic a glyfov jasne prezentujú fakt, že problematika, ktorej sa v našej práci venujeme predstavuje výzvu, na ktorú modely vo svojej počiatočnej podobe nie sú pripravené. Zlepšenie neprináša ani SOTA OVD architektúra modelu YOLO-Worldv2 [24, 33].

Ďalší postup pri testovaní schopností YOLO-Worldv2 je analogický s predošlými modelmi. Referenčný model natrénovaný na COCO dataseze dotrénujeme na VEGA Cropped datasetoch obsahujúcich číslice a glyfy. Trénovanie oboch variant modelov prebieha so zachovanými predvolenými hyperparametrami pri nastavení počtu epoch (*epochs*) na hodnotu 300, veľkosti vstupu (*imgsz*) na 640 pixelov, veľkosti dávky (*batch*) na hodnotu 4 a trpezlivosti zastavenia trénovania pri indikácii pretrénovania (early stopping) na 50 epoch [24, 33].

Dotrénované modely pre vizualizáciu detekčných výstupov spúšťame v inferenčnom režime na náhodných vzorkách z VEGA Cropped testovacích dátových množín číslic a glyfov pri veľkosti vstupu (*imgsz*) 640 x 640 pixelov a prahovej hodnote (*conf*) 0.25 [24, 33].



Obr. 59: Ukážka detekcie číslí modelom YOLO-Worldv2 na testovacích vzorkách VEGA Cropped



Obr. 60: Ukážka detekcie glyfov modelom YOLO-Worldv2 na testovacích vzorkách VEGA Cropped

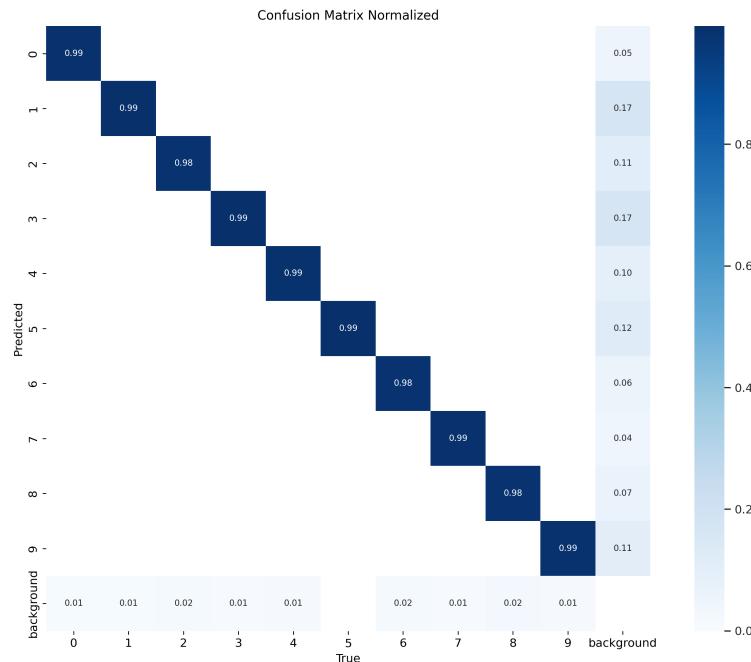
Na overenie schopností detekcie (Box) číslic a glyfov dotrénované modely spustíme aj vo validačnom režime. Klasifikačné schopnosti overíme pomocou konfúznych matíc [24, 33].

| | P | R | mAP50 | mAP50-95 |
|------------|-------|-------|-------|----------|
| Box | 0.977 | 0.984 | 0.989 | 0.843 |

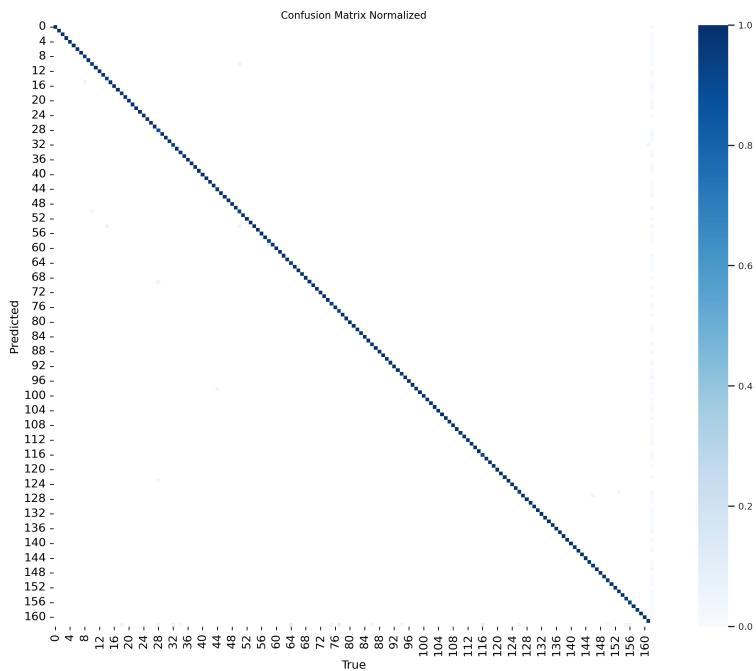
Tabuľka 17: Úspešnosť modelu YOLO-Worldv2 na testovacej množine číslic datasetu VEGA Cropped

| | P | R | mAP50 | mAP50-95 |
|------------|-------|-------|-------|----------|
| Box | 0.972 | 0.983 | 0.989 | 0.980 |

Tabuľka 18: Úspešnosť modelu YOLO-Worldv2 na testovacej množine glyfov datasetu VEGA Cropped



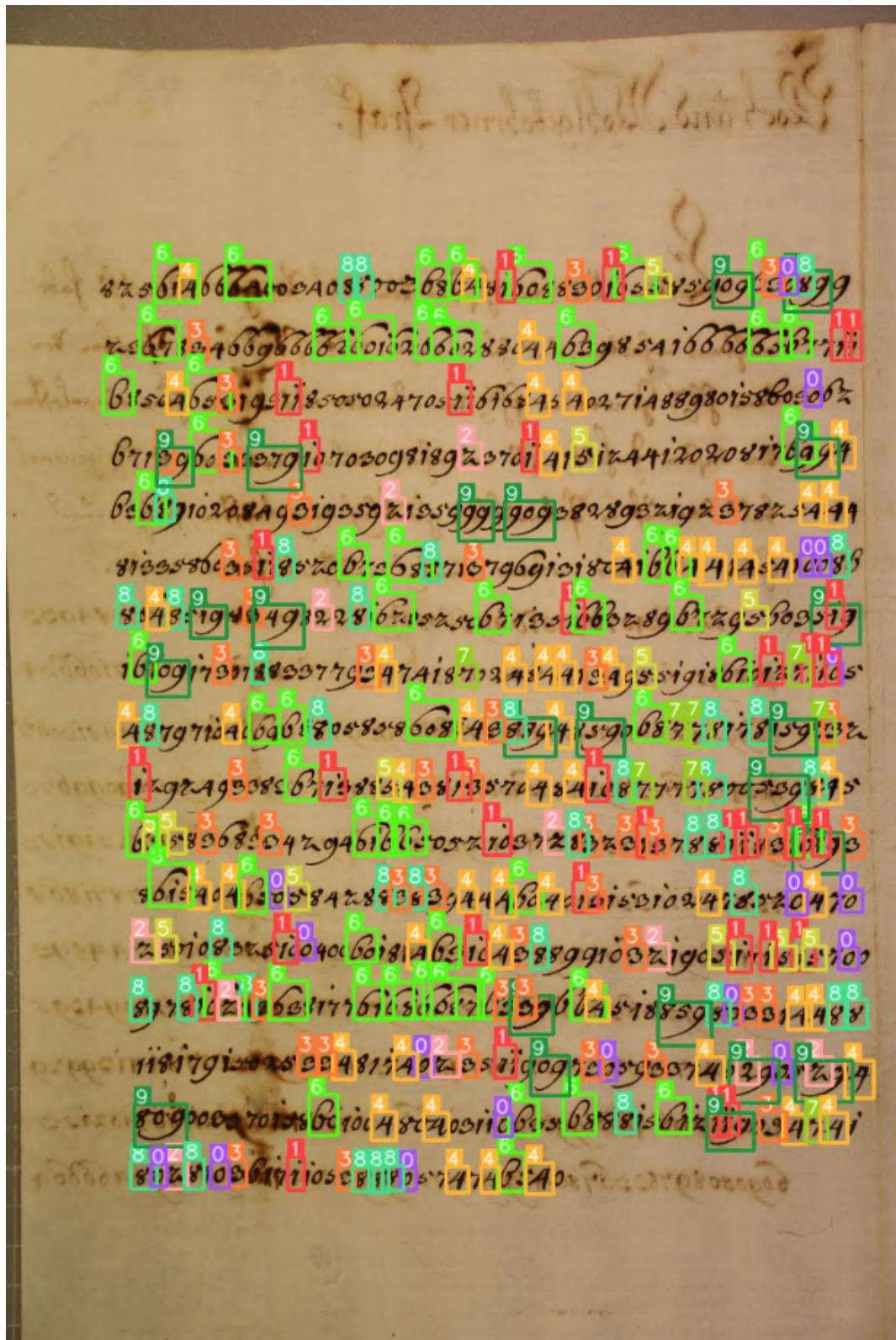
Obr. 61: Konfúzna matica modelu YOLO-Worldv2 na testovacej množine číslic datasetu VEGA Cropped



Obr. 62: Konfúzna matica modelu YOLO-Worldv2 na testovacej množine glyfov datasetu VEGA Cropped

YOLO-Worldv2 modely dosahujú lepšie výsledky pri detekcii malých objektov na vstupoch s celými historickými dokumentami a rovnako zvyšujú presnosť detekcie na testovacích množinách VEGA Cropped datasetov číslíc a glyfov pri zachovaní vysokej úspešnosti klasifikácie. Tento fakt pravdepodobne súvisí s architektúrou modelu, ktorá pracuje s vizuálnymi a textovými príznakmi objektov, čo prirodzene zvyšuje úspešnosť detekcie.

Napriek najlepším výsledkom modelov architektúry YOLO-Worldv2 sú však ich výsledky detekcie v štandardnej forme na celých dokumentoch s malými a husto rozmiestnenými objektami opäťovne nedostatočné.



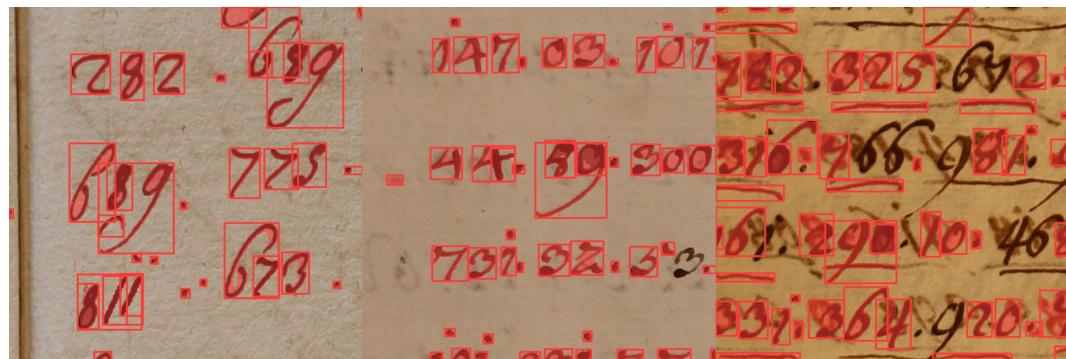
Obr. 63: Ukážka detekcie číslí modelom YOLO-Worldv2 na VEGA vzorke celého historickeho dokumentu (bez použitia techniky SAHI)

5.1.6 FastSAM

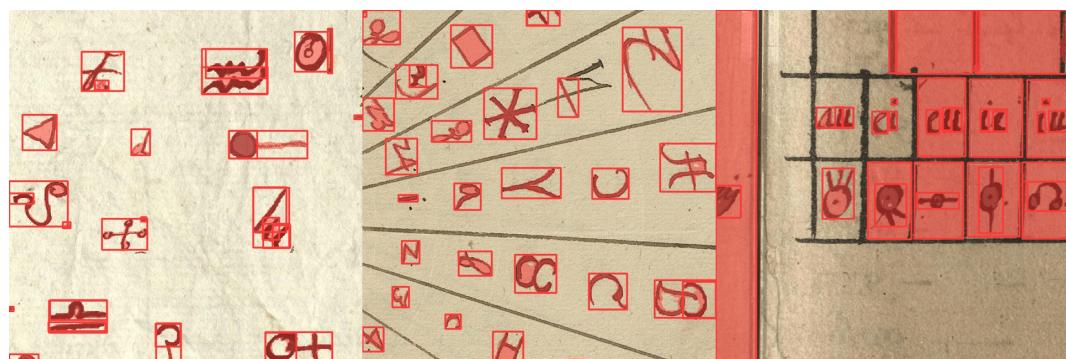
Posledný využitý model v našej práci je zero-shot segmentačný model FastSAM. Z dôvodu vysokých výpočtových a časových nárokov modelu pri segmentácii pracujeme s implementáciou z knižnice *Ultralytics* vo veľkosti s (FastSAMs) [24, 31].

Vzhľadom na odlišnú architektúru, prípady použitia, nemožnosť dotrénovania a fungovanie (zero-shot segmentácia) samotného modelu FastSAM v práci ďalej neuvádzame dosiahnuté výsledky na COCO a VEGA Cropped datasetoch, ani ukážku segmentácie na celých dokumentoch historických rukopisov [24, 31].

Referenčný model vo veľkosti s spúšťame v inferenčnom režime na náhodných vzorkách z VEGA Cropped testovacích dátových množín číslíc a glyfov pri veľkosti vstupu (*imgsz*) 640 x 640 pixelov a prahovej hodnote (*conf*) 0.25 [24, 31].



Obr. 64: Ukážka zero-shot segmentácie číslíc modelom FastSAM na testovacích vzorkách VEGA Cropped



Obr. 65: Ukážka zero-shot segmentácie glyfov modelom FastSAM na testovacích vzorkách VEGA Cropped

Model FastSAM nie je použiteľný v konvenčných prípadoch použitia pri potrebe detekcie alebo segmentácie objektov v historických rukopisoch, napríklad neposkytuje možnosť dotrénovania na špecifickú úlohu a nedisponuje schopnosťou klasifikácie objektov do tried. Prináša však zaujímavú alternatívnu segmentáciu rôznych objektov bez nutnosti trénovania, ktorá je využiteľná napríklad pri generovaní anotácií segmentačných masiek, pri rozširovaní alebo tvorbe nových dátových množín [24, 31].

5.1.7 SAHI

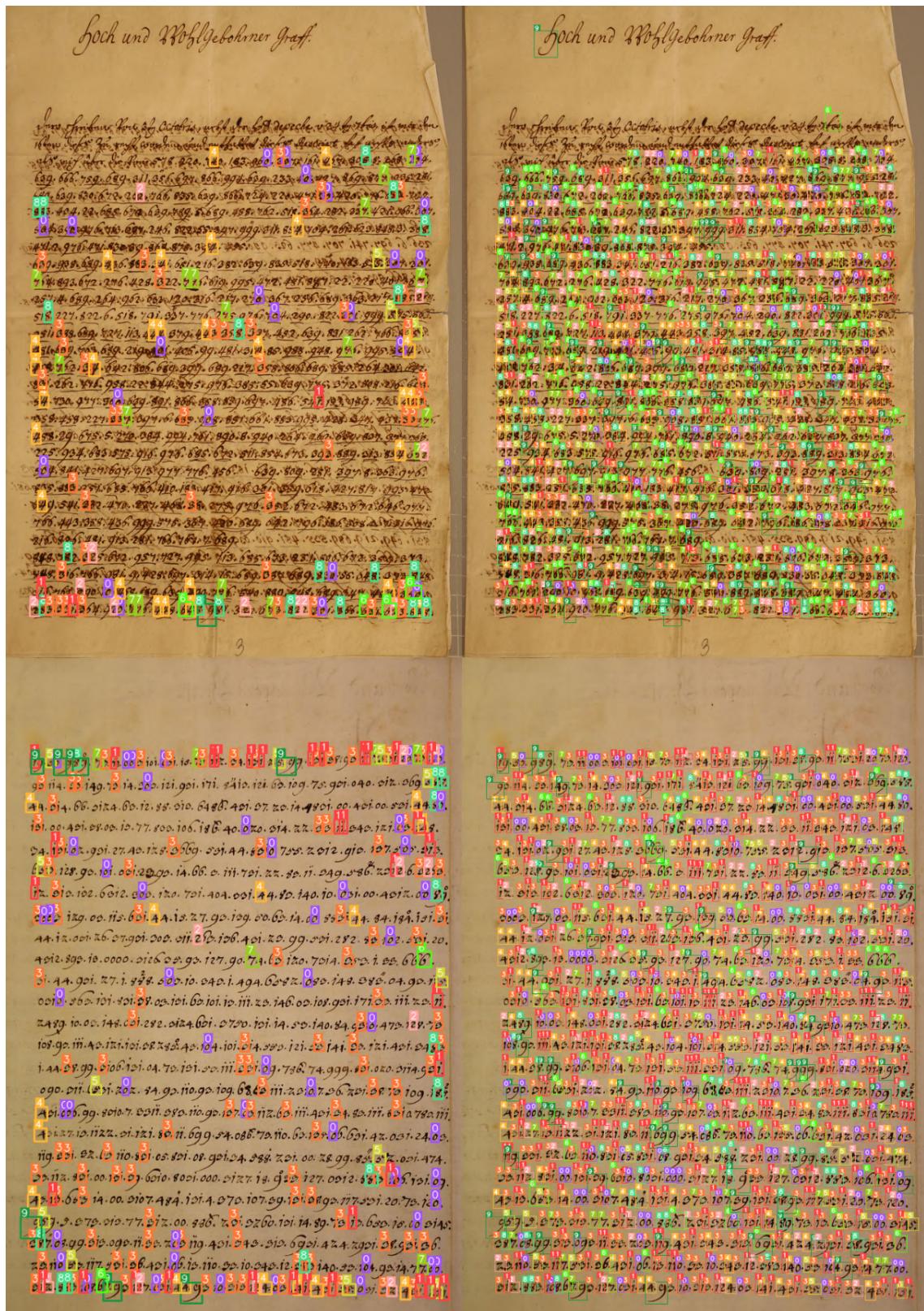
Po vyčíslení úspešnosti a porovnaní modelov na detekciu a segmentáciu rôznych typov a architektúr je preukázateľné, že ani jeden variant nedosahuje požadovanú úspešnosť detekcie alebo segmentácie pri použití na celom rukou písanom historickom dokumente. Z tohto dôvodu využívame pri detekcii na celých historických rukopisoch techniku Slicing Aided Hyper Inference (SAHI). Pracujeme s implementáciou techniky SAHI z knižnice *Supervision* s názvom Inference Slicer. Na základe evaluácie schopností modelov z predošej časti práce využívame na detekciu objektov na blokoch nami dotrénované modely typu YOLOv8. Je tiež vhodné poznamenať, že momentálne technika SAHI poskytuje len detekčnú funkcionality s príslubom podpory segmentácie v blízkej budúcnosti [35, 50].

Technika SAHI v kombinácii s detektorom VFNet a v porovnaní s detekciami samotného modelu VFNet vo veľkosti l dosahuje na testovacej množine datasetu xView, ktorý obsahuje vysokokvalitné snímky zachytené vesmírnymi satelitmi, nasledujúce výsledky [35, 50]:

| | Box (mAP50) |
|----------------------|-------------|
| VFNetl | 0.475 |
| SAHI + VFNetl | 0.633 |

Tabuľka 19: Úspešnosť modelu VFNet a riešenia SAHI + VFNet na testovacej množine referenčného datasetu xView [35]

Pre demonštráciu detekcie objektov na celých dokumentoch historických rukopisov pomocou SAHI spúštame funkcionality v inferenčnom móde na náhodných historických dokumentoch z VEGA testovacej množiny vo veľkosti 720 x 1080 pixelov pri veľkosti bloku ($slice_wh$) 160 x 160 pixelov, pomere prekrytie blokov ($overlap_ratio_wh$) 0.3 x 0.3 a detekčnom modeli YOLOv8 v inferenčnom režime ($callback$) s veľkosťou vstupu ($imgsz$) 640 x 640 pixelov a prahovou hodnotou ($conf$) 0.25. Detailnejšie zobrazenia viacerých vizuálnych detekčných výstupov techniky SAHI uvádzame v prílohe C [35, 50].



Obr. 66: Ukážka detekcie číslíc modelom YOLOv8 (vlavo) a riešenia SAHI + YOLOv8 (vpravo) na VEGA vzorke celého historického dokumentu

Z vizuálnych výstupov techniky SAHI s využitím YOLOv8 je zrejmý masívny nárast úspešnosti pri detekcii malých husto rozmiestnených objektov na vstupoch celých historických rukopisov. Pri využití techniky SAHI s nami dotrénovaným modelom YOLOv8 sa priemerný nárast úspešnosti detekcie objektov na celých historických rukopisov pohybuje na úrovni ~73%, pričom testované výstupy obsahujú minimálny počet falošne negatívnych (FN) a falošne pozitívnych (FP) detekcií.

| | Inferenčný čas (s) |
|----------------------|--------------------|
| YOLOv8 | 0.978 |
| SAHI + YOLOv8 | 3.748 |

Tabuľka 20: Porovnanie inferenčných časov modelu YOLOv8 a riešenia SAHI + YOLOv8 na vzorke celého historického dokumentu datasetu VEGA

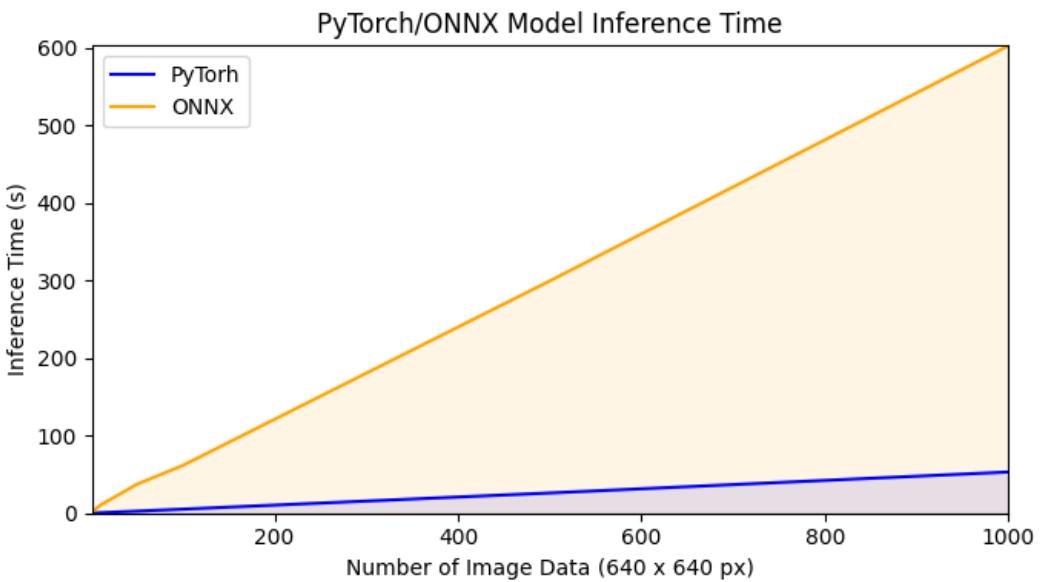
Z výsledkov porovnania inferenčných časov modelu YOLOv8 a riešenia SAHI s využitím YOLOv8 pri detekcii na náhodnej vzorke historického dokumentu z VEGA testovacej množiny vo veľkosti 720 x 1080 pixelov je však zrejmé, že využitie techniky SAHI proces detekcie štvornásobne spomaľuje. Táto skutočnosť je zapríčinená viacnásobným vykonaním detekcie na vzniknutých blokoch pôvodného vstupu a ich následnou agregáciou do výsledného detekčného výstupu.

Využitie techniky SAHI teda predstavuje výrazný posun v problematike detekcie a segmentácie objektov v historických šifrovaných rukopisoch, ktoré je vďaka svojim výsledkom vhodné pri nasadení v reálnych prípadoch použitia.

5.1.8 Porovnanie rýchlosťi inferencie modelov vo formátoch PyTorch a ONNX

Po dotrénovaní a evaluácii z dôvodu lepšej prenositeľnosti modely exportujeme do univerzálneho formátu ONNX (.onnx), nakoľko všetky nami využívané ML knižnice natívne pracujú s formátom PyTorch (.pt), ktorý je priamo závislý od konkrétnej implementácie danej architektúry modelu [24, 40].

Napriek nesporným výhodám trpí formát ONNX nedostatkom dlhších inferenčných časov pri vyššom počte vstupov, čo pre fungovanie v reálnom svete predstavuje zásadný problém, ktorý ho oproti iným exportným formátom modelov strojového učenia znevýhodňuje. Z tohto dôvodu v našej práci testujeme a porovnávame dĺžku inferencie modelu vo formátoch PyTorch a ONNX pri rôznom počte obrazových dátových vstupov [24, 40].



Obr. 67: Graf porovnania inferenčných časov modelu vo formátoch PyTorch a ONNX

Testovanie prebieha na vzdialenom serveri s operačnou pamäťou veľkosti 64 GB, grafickou kartou NVIDIA T4 s veľkosťou pamäte 15 GB a procesorom Intel Xeon s taktom 2 GHz. Testujeme segmentačný model YOLOv8 vo veľkosti l dotrénovaný na VEGA Cropped datasete číslík exportovaný vo formátoch PyTorch a ONNX, pri 1, 5, 10, 50, 100, 500 a 1000 obrazových vstupoch z testovej podmnožiny VEGA Cropped datasetu číslík vo veľkosti 640 x 640 pixelov. Meranie inferenčných časov je uskutočnené v sekundách.

| | Počet dátových vzoriek | | | | | | |
|----------------|------------------------|---------|----------|----------|----------|-----------|-----------|
| | 1 | 5 | 10 | 50 | 100 | 500 | 1000 |
| ONNX | 1.033 s | 5.125 s | 10.174 s | 37.233 s | 61.508 s | 299.407 s | 602.581 s |
| PyTorch | 0.080 s | 0.327 s | 0.573 s | 2.616 s | 5.179 s | 26.054 s | 53.086 s |

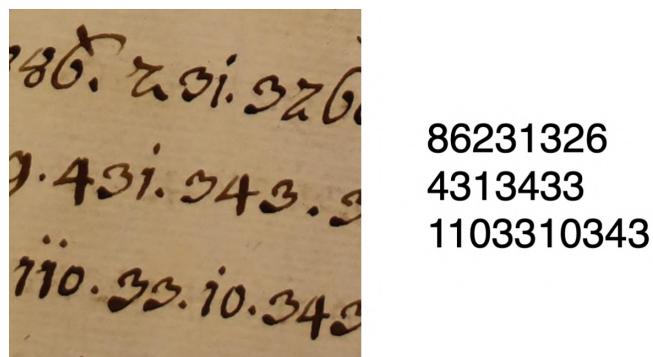
Tabuľka 21: Porovnanie inferenčných časov modelu vo formátoch PyTorch a ONNX

Ako je zrejmé z výsledkov testovania rozdiel inferenčných časov je priepravný a pri zvyšovaní počtu obrazových vstupov násobne rastie. Exportný ML formát ONNX teda súčasťne poskytuje širokú variabilitu pri prenose a nasadzovaní modelov na rôznych hardvérových a softvérových platformách, no daňou je zníženie efektivity a spomalenie inferencie, respektívne detekcie a segmentácie. Formát knižnice PyTorch naopak poskytuje proprietárnu, no vysoko efektívnu a výkonnú alternatívu exportu modelov strojového učenia.

5.2 Nástroj na automatizovanú transkripciu symbolov

V súčasnosti využívaný transkripčný mechanizmus implementovaný v rámci projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov trpí nedostatkom nepresnej transkripcie pri mierne rotovaných riadkoch. Z tohto dôvodu sme sa rozhodli pre implementáciu nami navrhnutého transkripčného mechanizmu s využitím zhlukovacej techniky DBSCAN strojového učenia bez učiteľa (UL) [7].

DBSCAN zhlukovanie (Density-Based Spatial Clustering of Applications with Noise) bude využité pri zoskupovaní detegovaných symbolov do riadkov, ktoré oproti momentál-nemu riešeniu fungujúcemu na báze vyhľadávania lokálnych maxím histogramov stredov ohraničujúcich boxov prináša odolnosť práve voči miernym zakriveniam a rotáciám riadkov. Využitá je implementácia techniky DBSCAN z knižnice *Scikit-Learn* [7, 49].



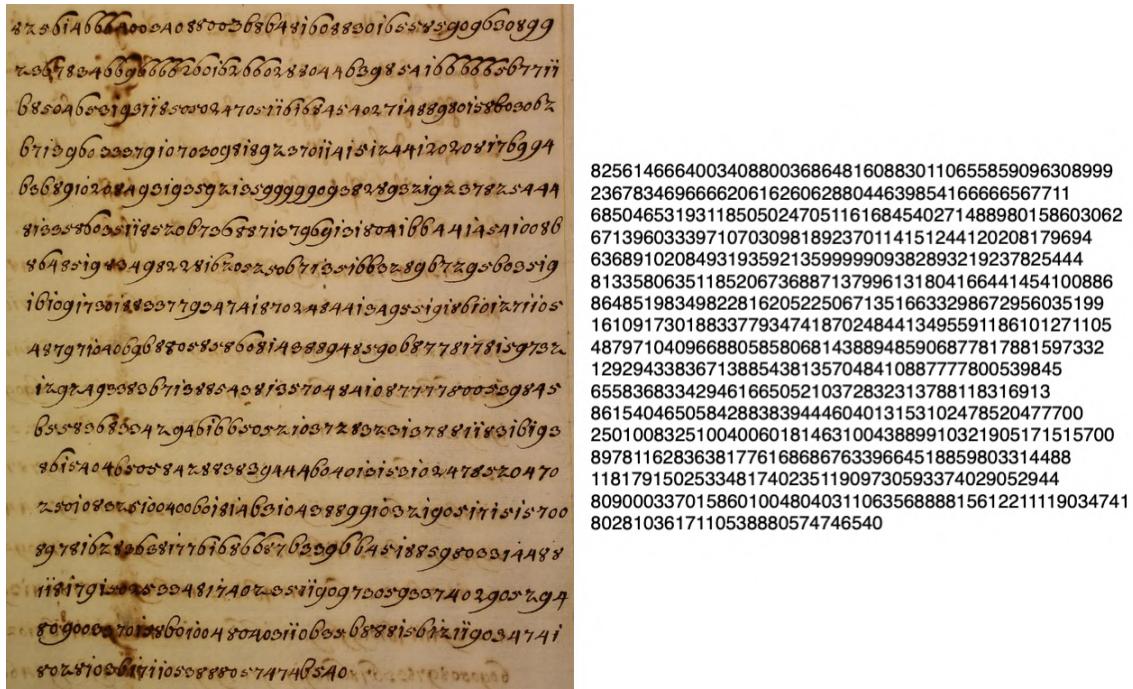
Obr. 68: Ukážka transkripcie číslíc (vpravo) detegovaných modelom YOLOv8 na testovacej vzorke VEGA Cropped s mierne rotovanými riadkami

Nástroj funguje na princípe nami zhrozeného návrhu mechanizmu transkripcie. Po detekcii číslíc alebo glyfov modelom, respektíve modelom s využitím techniky SAHI, prebehne výpočet vertikálnych (y-ových) stredových súradníc detekčných ohraničujúcich boxov (bounding box), na základe ktorých prebehne zhlukovanie technikou DBSCAN. *Scikit-Learn* implementácia techniky DBSCAN pre korektné zhlukovanie vyžaduje nepovinné hyperparametre *min_samples*, ktorý definuje minimálny počet vzoriek vo výslednom zhluku a *eps*, ktorý slúži na definovanie maximálnej vzdialenosť susedných vzoriek. V našej implementácii je hyperparameter *min_samples* vždy nastavený na hodnotu 1, hodnotu hyperparametra *eps* určujeme podľa vzorca [7, 49]:

$$eps = \frac{MaxY\ Bounding\ Boxes\ Coordinate - MinY\ Bounding\ Boxes\ Coordinate}{NumberOfLines * 2} \quad (6)$$

- eps – maximálna vzdialenosť susedných vzoriek,
- $\text{MaxY Bounding Boxes Coordinate}$ – maximálna vertikálna (y-ová) súradnica detekčných ohraničujúcich boxov,
- $\text{MinY Bounding Boxes Coordinate}$ – minimálna vertikálna (y-ová) súradnica detekčných ohraničujúcich boxov,
- NumberOfLines – približný počet riadkov na vstupnom digitalizovanom dokumente.

Po zoskupení detegovaných symbolov do zhľukov (riadkov) prebehne ich vertikálne zoradenie. Následne prebehne horizontálne zoradenie symbolov vo všetkých výsledných zoradených zhľukoch (riadkoch). Posledným krokom je samotný prepis zoskupených a zoradených symbolov do textovej reprezentácie.



Obr. 69: Ukážka transkripcie číslíc (vpravo) detegovaných riešením SAHI + YOLOv8 na testovacej VEGA vzorke celého historického dokumentu

Po otestovaní nami navrhnutého a implementovaného nástroja na vzorke datasetu VEGA Cropped s mierne rotovanými riadkami a na vzorke celého historického dokumentu datasetu VEGA pri použití nami dotrénovaného modelu YOLOv8, respektíve techniky SAHI s modelom YOLOv8, sme dosiahli takmer bezchybného prepisu. Nami navrhnutý a implementovaný mechanizmus transkripcie detegovaných symbolov teda prináša relevantnú alternatívu k súčasnému riešeniu.

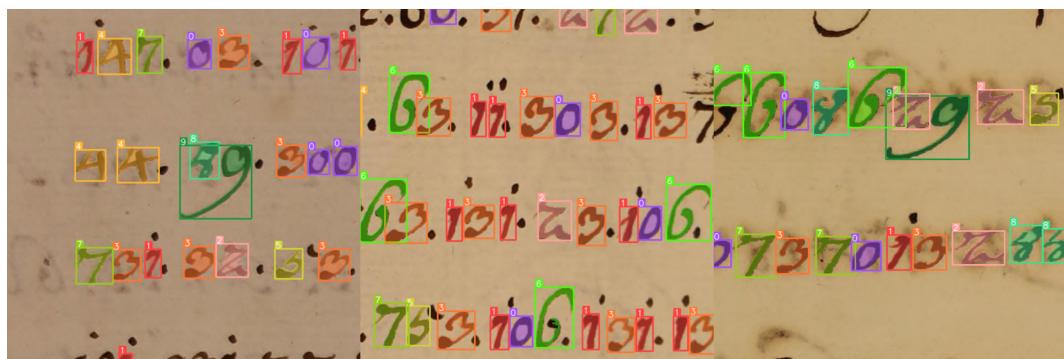
5.3 Nástroj na automatizovanú anotáciu obrázkov

Ako spomíname v predošlých kapitolách, pre optimálne fungovanie detekčných a segmentačných modelov je nutné dotrénovanie na kvalitných a rozsiahlych dátových množinách. Vzhľadom na skutočnosť, že disponujeme kvalitnými modelmi na detekciu číslic, respektíve glyfov (YOLOv8) a modelmi na zero-shot segmentáciu (SAM), využívame tieto modely na automatizované generovanie anotácií vo formáte YOLO [24, 29].

```
auto_annotate(  
    data="new_digits_original_crops/train/images/",  
    det_model="yolo_v8_digits.pt",  
    sam_model="mobile_sam.pt",  
    output_dir="new_digits_original_crops/train/labels/"  
)
```

Obr. 70: Ukážka konfigurácie funkcionality na automatizované generovanie anotácií [24]

Na tento zámer využívame funkcionalitu Annotator z knižnice *Ultralytics*, konkrétnie funkciu *auto_annotate*, ktorá obrazovým vzorkám zahrnutým vo vstupnom priečinku (*data*) automatizované generuje anotácie pomocou ML modelov na detekciu (*det_model*) a zero-shot segmentáciu (*sam_model*), ktoré následne ukladá do výstupného priečinka (*output_dir*) [24].



Obr. 71: Ukážka automatizovane vygenerovaných anotácií funkcionalitou Annotator

Annotator na vstupných obrazových vzorkách deteguje a klasifikuje objekty pomocou detekčného modelu, vo výstupných ohraničujúcich boxoch (bounding box) dochádza ku generovaniu segmentačných polygónov zero-shot segmentačným modelom, ktoré v kombinácii s klasifikačnými triedami vytvárajú výsledné anotácie [24].

5.4 Nástroj na prehľadávanie datasetov s využitím AI

Pre skvalitnenie dátových množín je dôležitá selekcia a filtrovanie dátových vzoriek, ktoré sú v datasetoch obsiahnuté. Na tento účel využívame pokročilé riešenie Explorer z knižnice *Ultralytics*. Explorer na pozadí pracuje s detekčným modelom YOLOv8, ktorý využíva na pokročilé prehľadávanie datasetov [24].

Objekt *Explorer* pomocou detekčného modelu (*model*) na vstupnom datasete (*data*) vo zvolenej dátovej podmnožine (*split*) po zavolaní metódy *create_embeddings_table* generuje tabuľku príznakov, ktorá pozostáva zo stĺpcov [24]:

- *im_file* - cesta k obrazovej dátovej vzorke,
- *labels* - mená tried objektov prítomných na obrazovej dátovej vzorke,
- *cls* - identifikátory tried objektov prítomných na obrazovej dátovej vzorke,
- *bboxes* - súradnice detekčných ohraničujúcich boxov objektov prítomných na obrazovej dátovej vzorke,
- *vector* - vektor príznakových charakteristík obrazovej dátovej vzorky vygenerovaný detekčným modelom.

Tabuľka príznakov je vytvorená na základe dostupnosti obrazových vzoriek, ich anotácií (ohraničujúce boxy/segmentačné polygóny, triedy) a príznakových vektorov, ktoré zvolený detekčný model YOLOv8 generuje z obrazových dátových vzoriek dostupných v datasete [24].

Funkcionalita Explorer po vygenerovaní tabuľky príznakov pracuje v troch režimoch, ktoré umožňujú prehľadávať dataset na základe podobnosti s vloženou obrazovou vzorkou, na základe SQL dopytu a na základe textového dopytu (prompt) [24].

5.4.1 Prehľadávanie na základe podobnosti

Prvým typom vyhľadávania v datasete je filtrovanie na základe podobnosti s vloženou obrazovou dátovou vzorkou [24].

Po vygenerovaní tabuľky príznakov objektom *Explorer* a po zavolaní funkcie *get_similar* dochádza ku extraholovaniu vektora príznakov z vloženého vstupného obrazového vstupu (*img*). Následne prebieha výpočet vzdialenosí medzi vektormi obrazových vzoriek vstupného datasetu z tabuľky príznakov a vektora vstupnej obrazovej vzorky, na základe ktorej sú vzorky vstupného datasetu zoradené, nakoľko vzdialosť príznakových vektorov priamo odzrkadluje podobnosť obrazových dát [24].

```

exp = Explorer(
    data="digits_original_crops/data.yaml",
    model="yolov8n.pt"
)
exp.create_embeddings_table(
    force=True
)
exp.get_similar(
    img="digits_crop_img.jpg"
)

```

Obr. 72: Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe podobnosti [24]



Obr. 73: Ukážka vyhľadaných vzoriek (v strede, vpravo) podobných k pôvodnej vzorke (vľavo) funkctionalitou Explorer

5.4.2 Prehľadávanie na základe SQL dopytu

Druhým typom vyhľadávania v datasete je filtrovanie na základe vloženého SQL dopytu [24].

```

exp = Explorer(
    data="digits_original_crops/data.yaml",
    model="yolov8n.pt"
)
exp.create_embeddings_table(
    force=True
)
exp.sql_query(
    query="WHERE labels LIKE '%0%' AND labels LIKE '%1%' AND labels LIKE '%6%'"
)

```

Obr. 74: Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe SQL dopytu [24]

Po vygenerovaní tabuľky príznakov objektom *Explorer* a po zavolení funkcie *sql_query* dochádza k prehľadávaniu vygenerovanej tabuľky, konkrétnie stĺpcov *labels* a *cls* podľa vloženého dopytu (*query*) [24].

5.4.3 Prehľadávanie na základe textového dopytu

Posledným typom vyhľadávania v datasete je filtrovanie na základe vloženého textového dopytu, respektíve príkazu (prompt) [24].

```
exp = Explorer(  
    data="digits_original_crops/data.yaml",  
    model="yolov8n.pt"  
)  
exp.create_embeddings_table(  
    force=True  
)  
exp.explorer.ask_ai(  
    query="Show me 100 images with exactly one 3 and two 5. " +  
        "There can be other objects too."  
)
```

Obr. 75: Ukážka konfigurácie funkcionality na prehľadávanie datasetu na základe textového dopytu [24]

Po vygenerovaní tabuľky príznakov objektom *Explorer* a po zavolení funkcie *ask_ai* dochádza za pomoci OpenAI API k transformácii textového dopytu (*query*) na SQL dopyt, na základe ktorého sú rovnako ako v predošлом prípade stĺpce tabuľky *labels* a *cls* prehľadávané [24].

5.5 Webové API

Finálnu časť našej práce predstavuje implementácia webového aplikačného rozhrania (API) typu REST. Úlohou webového REST API je jednoduchým spôsobom sprostredkovať funkciuálnitu na:

- detekciu/segmentáciu číslic a glyfov v historických rukopisoch,
- automatizované generovanie anotácií pre datasety číslic a glyfov s využitím AI,
- pokročilé prehľadávanie datasetov s využitím AI.

Na implementáciu webového REST API využívame knižnicu *FastAPI*, na vykonávanie detekcie/segmentácie, automatizované generovanie anotácií datasetov a pokročilé prehľadávanie datasetov knižnicu *Ultralytics*, na prácu s obrazovými dátami a technikou

SAHI knižnicu *Supervision*, a na testovanie rýchlosťi a výkonu celého API aplikačný rámec *Locust*. Riešenie beží na serveri *Uvicorn* [51, 52].

V implementácii využívame nami dotrénované modely YOLOv8, YOLOv9, YOLO-Worldv2, RT-DETR a predtrénovaný foundation model FastSAM exportované vo formáte PyTorch (.pt).

Webové API ponúka pre komfortné nastavenie celého riešenia možnosť konfigurácie pomocou *config.json* súboru, ktorý obsahuje možnosť deklarácie mena (*name*), verzie (*version*), host IP adresy (*host*), portu (*port*) a ciest k váham detekčných/segmentačných modelov (*yolo_v8_digits_source*, *yolo_v8_glyphs_source*, *fast_sam_source*, ...).

API vďaka zvolenej implementácii pomocou *FastAPI* ponúka automatické generovanie interaktívnej *Swagger* dokumentácie koncových bodov REST API a im príslušných tiel požiadaviek (request) a odpovedí (response) [51].

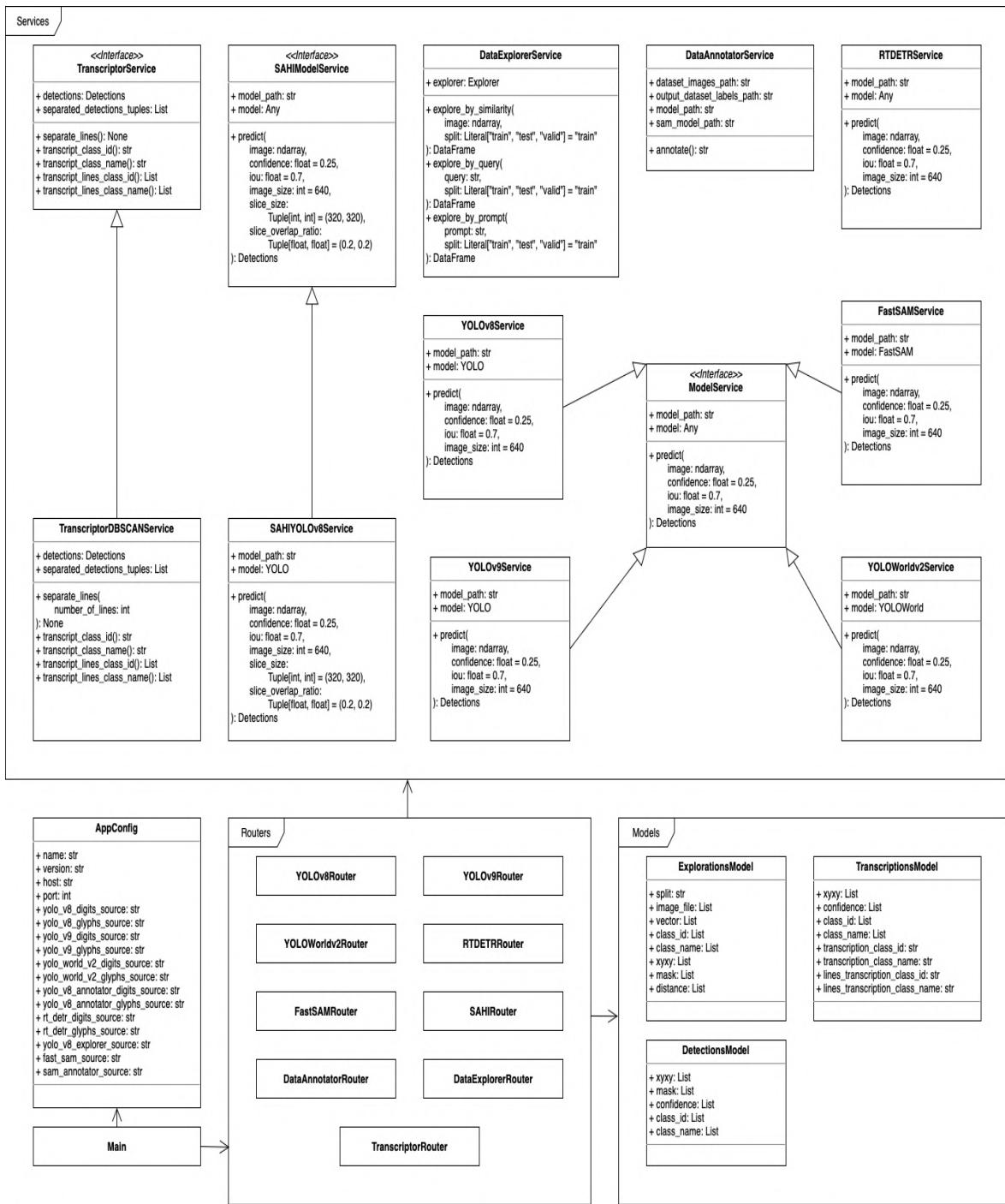
5.5.1 Architektúra API

Ako už bolo v našej práci uvedené, architektúra celého webového aplikačného rozhrania je rozdelená pomocou Three-Tier klient-server architektúry do prezentačnej (presentation tier), logickej (logic tier) a dátovej vrstvy (data tier).

Dátová vrstva (Data tier) je tvorená množinou exportovaných váh dotrénovaných modelov vo formáte PyTorch (.pt) v priečinku */api/resources/weights*.

Logická vrstva (Logic tier) pozostáva z implementácie služieb (services) vykonávajúcich celú logiku a funkcionalitu nášho riešenia. Tieto služby sú implementované ako triedy v priečinku */api/app/service*. Implementované služby poskytujú univerzálne programovacie rozhranie na obsluhu detekčných/segmentačných modelov, ktoré priamo pracujú s exportovanými váhami z dátovej vrstvy, techniky SAHI, nástroju na automatizovanú transkripciu symbolov, automatizované generovanie anotácií a nástroju na pokročilé prehľadávanie datasetov.

Prezentačná vrstva (Presentation tier) obsahuje implementáciu REST API koncových bodov (endpoints). Pri implementácii sú využité triedy *APIRouter* z knižnice *FastAPI* v priečinku */api/app/router* a *BaseModel* z knižnice *Pydantic* v priečinku */api/app/model*. Trieda *APIRouter* reprezentuje abstrakciu API koncového bodu, vďaka ktorej je možné efektívne spravovať jeho URI cestu a enkapsulovať obslužný kód spracovania požiadavky (request), vykonávania funkcionality službou (service) z logickej vrstvy a následného generovania odpovede (response). Abstrakcia tiel požiadaviek a odpovedí je reprezentovaná triedou *BaseModel*, ktorá automaticky overuje formu a obsah konkrétnych požiadaviek a odpovedí [51].



Obr. 76: Diagram tried webového ML REST API

Dodatočné funkcionality na transformáciu dát a spracovanie konfiguračného súboru sú dostupné v priečinku `/api/app/utils`.

5.5.2 Prístupové body API

Pre ľahké použitie celého riešenia je nevyhnutná správna implementácia a štruktúra koncových prístupových bodov REST API. Koncové prístupové body sú rozčlenené do štyroch logických skupín:

- koncové body pre sprostredkovanie **detekčnej a segmentačnej funkcionality**,
 - */yolo_v8/digits/*
 - */yolo_v8/glyphs/*
 - */yolo_v9/digits/*
 - */yolo_v9/glyphs/*
 - */yolo_world_v2/digits/*
 - */yolo_world_v2/glyphs/*
 - */rt_detr/digits/*
 - */rt_detr/glyphs/*
 - */fast_sam*
- koncové body pre sprostredkovanie **SAHI detekčnej funkcionality**,
 - */sahi/yolo_v8/digits/*
 - */sahi/yolo_v8/glyphs/*
- koncové body pre sprostredkovanie **detekčnej a transkripčnej funkcionality**,
 - */transcriptor/yolo_v8/digits/*
 - */transcriptor/yolo_v8/glyphs/*
 - */transcriptor/sahi/yolo_v8/digits/*
 - */transcriptor/sahi/yolo_v8/glyphs/*
- koncové body pre sprostredkovanie **funkcionality automatizovaného generovania anotácií s využitím AI**,
 - */data/annotator/digits/*
 - */data/annotator/glyphs/*

- koncové body pre sprostredkovanie **funkcionality pokročilého prehľadávania datasetov s využitím AI.**
 - */data/explorer/similarity/*
 - */data/explorer/query/*
 - */data/explorer/prompt/*

Koncové prístupové body sú typu *POST*, pracujú s požiadavkami (request) typu *multipart/form-data* a odpovede (response) sú poskytované vo formáte *JSON*.

Kompletná vygenerovaná *Swagger* dokumentácia rozhrania koncových bodov so štruktúrou tiel požiadaviek (request) a odpovedí (response) webového REST API je po spustení celého riešenia dostupná na adrese *<host_ip>:<port>/docs/* [51].

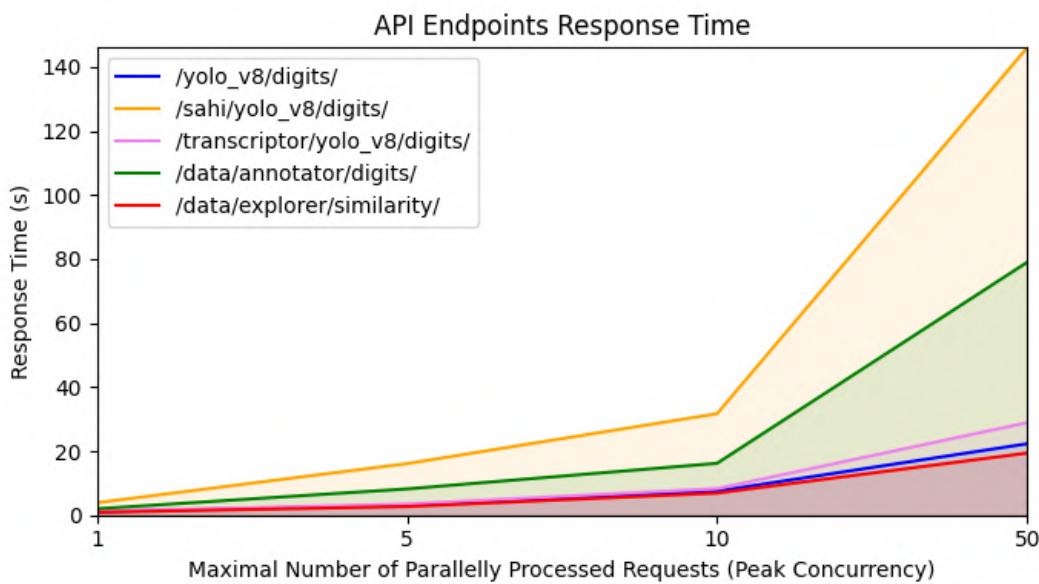
5.5.3 Testovanie API

Finálnu časť implementačnej časti našej práce predstavuje testovanie webového API. Na testovanie sú zhotovené dva druhy testov, jednoduché HTTP testy na overenie funkčnosti prístupových koncových bodov a záťažové (performance) testy na simuláciu správania prístupových koncových bodov pod záťažou.

HTTP testy sú implementované formou jednoduchých testovacích skriptov (.http) v priečinku */api/tests/http_tests*. HTTP testy overujú schopnosť koncových bodov spracovať požiadavky (request) a odosielat odpovede (response). Vzorové testovacie súbory, ktoré sú počas testov využívané (.jpg, .zip) sú zhromaždené v priečinku */api/tests/resources*.

Záťažové (performance) testy sú implementované formou skriptov (.py) využívajúce programovacie rozhranie aplikačného rámca *Locust* v priečinku */api/tests/performance_tests* so vzorovými súbormi potrebnými na testovanie v priečinku */api/tests/resources*. Záťažové testovanie je vykonané na vzdialenom serveri s operačnou pamäťou veľkosti 32 GB, grafickou kartou NVIDIA GeForce GTX 1080 Ti s veľkosťou pamäte 11 GB a procesorom Intel Xeon s taktom 1.8 GHz pri vytvorení 0.5 požiadaviek (request) za sekundu pri maximálnom počte 1, 5, 10 a 50 požiadaviek pri konkurentnom vykonávaní [52].

Pre jednoduchosť záťažovými testami pokrývame z každého typu funkciality jeden koncový bod, konkrétnie */yolo_v8/digits/*, */sahi/yolo_v8/digits/*, */transcriptor/yolo_v8/digits/*, */data/annotator/digits/* a */data/explorer/similarity/*.



Obr. 77: Graf porovnania reakčných časov koncových bodov webového REST API

| | Počet dopytov | | | |
|-------------------------------|---------------|--------|--------|---------|
| | 1 | 5 | 10 | 50 |
| /yolo_v8/digits/ | 1.1 s | 3.1 s | 7.5 s | 22.3 s |
| /sahi/yolo_v8/digits/ | 3.9 s | 16.1 s | 31.7 s | 146.0 s |
| /transcriotor/yolo_v8/digits/ | 1.2 s | 3.5 s | 8.3 s | 28.9 s |
| /data/annotator/digits/ | 2.0 s | 8.2 s | 16.2 s | 79.0 s |
| /data/explorer/similarity/ | 0.9 s | 2.7 s | 6.9 s | 19.4 s |

Tabuľka 22: Porovnanie reakčných časov koncových bodov webového REST API

Zo záťažového testovania je možné vyvodiť niekoľko záverov. Pri počte 50 konkurentne spracovávaných a vykonávaných požiadaviek (request) sa reakčný čas výrazne zvyšuje. Pri nižšom počte požiadaviek je reakčný čas uspokojivý. Najdlhší reakčný čas je nameraný pri koncovom bode detekcie pomocou techniky SAHI v kombinácii s modelom YOLOv8 (*/sahi/yolo_v8/digits/*), čo je však priamo zapríčinené vyšším počtom blokov na ktorých je samotná detekcia vykonávaná. Vzhľadom na skutočnosť, že riešenie je určené na využitie v akademickom prostredí, sú teda dosiahnuté výsledky dostatočné.

Záver

V tejto diplomovej práci sme sa v rámci teoretickej časti oboznámili s podobou a štruktúrou historických šifrovaných rukopisov, respektíve nomenklátorov, a architektúrou systému na ich automatizované spracovanie. V teoretickej časti sme sa ďalej venovali rôzny moderným prístupom detekcie a segmentácie objektov. Prvým prístupom bola elementárna detekcia a segmentácia, konkrétnie modely YOLOv8, YOLO-NAS, YOLOv9 a RT-DETR. Ďalší prístup predstavovala zero-shot segmentácia a modely SAM, respektíve FastSAM. Posledným prístupom zahrnutým v teoretickej časti bola open-vocabulary detekcia reprezentovaná modelom YOLO-World. V práci sme zahrnuli aj techniku SAHI na zvýšenie presnosti detekcie malých, husto rozmiestnených objektov, ktorá v odvetví strojového učenia (ML) a počítačového videnia (CV) predstavuje nezanedbatelný problém.

V časti analýzy problematiky a návrhu riešenia sme analyzovali datasety číslic a symbolov (glyfov) vytvorené v rámci výskumného projektu VEGA 2/0072/20: Moderné metódy spracovania šifrovaných archívnych dokumentov, ktorého súčasťou je aj táto práca. Ďalej sme sa oboznámili so štandardnými evaluačnými detekčnými metrikami a exportným ONNX formátom. Ďalším krokom bolo vypracovanie návrhu mechanizmu na automatizovanú transkripciu symbolov s využitím strojového učenia bez učiteľa UL a návrhu webového aplikačného rozhrania (API), ktoré bude nami implementovanú funkcionality sprostredkúvať bežnému používateľovi. V poslednej časti návrhu riešenia sme vybrali dostupné nástroje, ktoré budú na implementáciu finálneho riešenia využité.

Prvým krokom implementácie bolo natrénovanie a otestovanie nami vybraných modelov na datasetoch VEGA číslic a glyfov. V našej práci boli presnejšie zahrnuté modely YOLOv8, YOLO-NAS, YOLOv9, RT-DETR, YOLO-Worldv2 a FastSAM. Najvyššiu úspešnosť detekcie dosahovali dotrénované modely typu YOLOv8, Precision (P) na úrovni 0.974, Recall (R) na úrovni 0.984, Mean Average Precision 50 (mAP50) na úrovni 0.989 a Mean Average Precision 50-95 (mAP50-95) na úrovni 0.819 na dátovej množine číslic datasetu VEGA Cropped, respektíve P na úrovni 0.979, R na úrovni 0.976, mAP50 na úrovni 0.989 a mAP50-95 na úrovni 0.981 na dátovej množine glyfov datasetu VEGA Cropped. Kompletný prehľad úspešnosti všetkých dotrénovaných detekčných modelov uvádzame v prílohe D.

Všetky natrénované modely boli exportované do štandardizovaného univerzálneho formátu ONNX, ktorého rýchlosť inferencie sme porovnali so štandardným exportným formátom PyTorch.

Ďalším krokom bola implementácia nástroja na automatizovanú transkripciu detegovaných symbolov, využívajúceho na zoskupenie symbolov do riadkov UL zhľukovaciu techniku DBSCAN (Density-Based Spatial Clustering of Applications with Noise).

Najúspešnejšie dotrénované modely YOLOv8 sú pre ich schopnosť detektie s minimálnym počtom falošne pozitívnych detekcií (FP) otestovali aj v kombinácii s technikou na detekciu malých objektov SAHI (Slicing Aided Hyper Inference), ktorú sú využili na vstupoch celých digitalizovaných historických dokumentov datasetu VEGA. Pri využití techniky SAHI v kombinácii s modelom YOLOv8 sú oproti detekcií s modelom YOLOv8 na vzorkách celých digitalizovaných historických dokumentov dosiahli priemerný nárast počtu správne detegovaných objektov na úrovni ~73%.

Následným krokom implementačnej časti bolo zhotovenie samotného webového API typu REST. Na implementáciu bola využitá moderná knižnica pre efektívnu tvorbu webových aplikačných rozhraní v jazyku *Python*, *FastAPI*. Do aplikačného rozhrania boli okrem nami natrénovaných modelov a SAHI techniky na vykonávanie detektie alebo segmentácie zahrnuté aj funkcionality na automatizovanú transkripciu symbolov s využitím AI/UL, na automatizované generovanie anotačných súborov s využitím AI/ML a na pokročilé prehľadávanie datasetov s využitím AI/ML.

Finálne riešenie implementácie webového REST API bolo otestované pomocou HTTP testov na overenie funkčnosti prístupových koncových bodov a zátažových (performance) testov na overenie schopnosti behu pri rôznej miere zataženia.

Z výsledkov testovania a experimentov je zrejmé, že aj napriek využitiu najmodernejších a najpokročilejších modelov na detekciu alebo segmentáciu je problém detektie malých, husto rozmiestnených objektov pri využití samostatných modelov stále prítomný. Masívny nárast úspešnosti pri detekcii malých objektov bol však dosiahnutý pri využití modelu YOLOv8 v kombinácii s technikou SAHI. Toto riešenie sa stalo zároveň využiteľným aj v scenároch reálneho použitia pri detekcii objektov na celých historických šifrovaných rukopisoch.

Z testovania inferenčnej rýchlosťi formátov PyTorch a ONNX je ďalej preukázateľné, že univerzálny formát ONNX súčasťne poskytuje širokú variabilitu pri nasadzovaní modelov na rôznych hardvérových a softvérových platformách, no pri porovnaní s formátom PyTorch dosahuje výrazne vyšších inferenčných časov, čo ho pri využití s dôrazom na rýchlosť značne znevýhodňuje.

Testovanie implementovaného REST API pomocou HTTP a zátažových (performance) testov ukazuje spoľahlivé fungovanie celého rozhrania s ohľadom na funkčnosť aj rýchlosť pri využití v akademických podmienkach, pre ktoré bolo riešenie primárne vyvíjané.

Zoznam použitej literatúry

1. FIRČA, Michal a ANTAL, Eugen. *Detekcia zašifrovaného textu v historických rukopisoch*. Dp. pr. Slovenská technická univerzita v Bratislave, 2023.
2. MIKUŠ, Filip a ANTAL, Eugen. *Porovnanie metód umelej inteligencie na rozpoznanie rukou písaných číslíc*. Bp. pr. Slovenská technická univerzita v Bratislave, 2022.
3. ANTAL, Eugen a GROŠEK, Otokar. *Moderná kryptoanalýza klasických šifier*. Diz. pr. Slovenská technická univerzita v Bratislave, 2017.
4. GONO, Tomáš a ANTAL, Eugen. *Digitalizácia a spracovanie nomenklátorových šifrovacích kľúčov*. Dp. pr. Slovenská technická univerzita v Bratislave, 2021.
5. ANTAL, Eugen a MÍRKA, Jakub. Wrong Design of Cipher Keys: Analysis of Historical Cipher Keys From the Hessisches Staatsarchiv Marburg Used in the Thirty Years' War. 2022. Dostupné tiež z: https://www.researchgate.net/publication/361220017_Wrong_Design_of_Cipher_Keys_Analysis_of_Historical_Cipher_Keys_From_the_Hessisches_Staatsarchiv_Marburg_Used_in_the_Thirty_Years'_War.
6. BUDAI, Jakub a ANTAL, Eugen. *Vytvorenie anotovaných vzoriek šifrovaných symbolov z historických dokumentov*. Dp. pr. Slovenská technická univerzita v Bratislave, 2023.
7. ANTAL, Eugen a MARÁK, Pavol. Automated Transcription of Historical Encrypted Manuscripts. *Tatra Mountains Mathematical Publications*. 2022, č. 82. ISSN 1210-3195. Dostupné tiež z: <https://sciendo.com/article/10.2478/tmmp-2022-0019>.
8. DUNIN, Elonka a SCHMEH, Klaus. *Codebreaking: A Practical Guide*. Robinson, 2020. ISBN 978-1472144218.
9. FISCHER, Andreas, LIWICKI, Marcus a INGOLD, Rolf. Handwritten Historical Document Analysis, Recognition, And Retrieval: State Of The Art And Future Trends. 2020. ISBN 978-9811203237.
10. JOHANSSON, Kajsa a MEGYESI, Beáta. *Transcription of Historical Encrypted Manuscripts: Evaluation of an automatic interactive transcription tool*. Bp. pr. Uppsala University in Sweden, 2019.

11. CHEN, Jialuo, SOUIBGUI, Mohamed Ali, FORNE S, Alicia a MEGYESI, Beáta. A Web-Based Interactive Transcription Tool for Encrypted Manuscripts. *International Conference on Historical Cryptology HistoCrypt*. 2020. Dostupné tiež z: https://www.researchgate.net/publication/341512097_A_Web-based_Interactive_Transcription_Tool_for_Encrypted_Manuscripts.
12. MEGYESI, Beáta, ESSLINGER, Bernhard, FORNÉS, Alicia, KOPAL, Nils, LÁNG, Benedek, LASRY, George, LEEUW, Karl de, PETTERSSON, Eva, WACKER, Arno a WALDISPÜHL, Michelle. Decryption of historical manuscripts: the DECRYPT project. *Cryptologia*. 2020, roč. 22, č. 6. Dostupné tiež z: <https://www.tandfonline.com/doi/full/10.1080/01611194.2020.1716410>.
13. ANTAL, Eugen a ZAJAC, Pavol. HCPortal Overview. *HistoCrypt*. 2020. Dostupné tiež z: https://www.researchgate.net/publication/341515236_HCPortal_Overview.
14. KUREKOVÁ, Gabriela, TIŇO, Radovan a FIKAR, Miroslav. *Digitalizácia historických dokumentov na lignocelulózových nosičoch informácií*. Bp. pr. Slovenská technická univerzita v Bratislave, 2009.
15. BAGAR, Patrik a ANTAL, Eugen. *Vytvorenie vzoriek rukou písaných čísel z historických dokumentov pre potreby HTR metód*. Dp. pr. Slovenská technická univerzita v Bratislave, 2023.
16. KATRENIAK, Martin. Digitalizácia v múzeach/automatická transkripcia. *Medium*. 2022. Dostupné tiež z: <https://novohradskemuzeumagaleria.medium.com/digitalizacia-v-muzeach-automaticka-transkripcia-f25a11d4ce55>.
17. TEAM, Transkribus. Transkribus. [B.r.]. Dostupné tiež z: <https://readcoop.eu/transkribus/>.
18. KUSETOGULLARI, Huseyin, YAVARIABDI, Amir, HALL, Johan a LAVESSON, Niklas. DIGITNET: A Deep Handwritten Digit Detection and Recognition Method Using a New Historical Handwritten Digit Dataset. *Big Data Research*. 2020.
19. KORSTANJE, Joos. What is the difference between Object Detection and Image Segmentation? *Medium*. 2020. Dostupné tiež z: <https://towardsdatascience.com/what-is-the-difference-between-object-detection-and-image-segmentation-ee746a935cc1>.

20. TERVEN, Juan a CÓRDOVA-ESPARZA, Diana Margarita. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. 2024. Dostupné tiež z: <https://arxiv.org/pdf/2304.00501v6.pdf>.
21. LV, Wenyu, ZHAO, Yian, XU, Shangliang, WEI, Jinman, WANG, Guanzhong, CUI, Cheng, DU, Yuning, DANG, Qingqing a LIU, Yi. DETRs Beat YOLOs on Real-time Object Detection. 2023. Dostupné tiež z: <https://arxiv.org/pdf/2304.08069.pdf>.
22. REIS, Dillon, KUPEC, Jordan, HONG, Jacqueline a DAOUDI, Ahmad. Real-Time Flying Object Detection with YOLOv8. 2023. Dostupné tiež z: <https://arxiv.org/pdf/2305.09972.pdf>.
23. TERVEN, Juan, CÓRDOVA-ESPARZA, Diana-Margarita a ROMERO-GONZÁLEZ, Julio-Alejandro. A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS. 2023, roč. 5, č. 4. Dostupné tiež z: <https://www.mdpi.com/2504-4990/5/4/83>.
24. Ultralytics YOLOv8 Docs. 2024. Dostupné tiež z: <https://docs.ultralytics.com>.
25. Deci AI Documentation Hub - SuperGradients. 2024. Dostupné tiež z: <https://docs.deci.ai/super-gradients/latest/documentation/source/welcome.html>.
26. WANG, Chien-Yao, YEH, I-Hau a LIAO, Hong-Yuan Mark. YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information. 2024. Dostupné tiež z: <https://arxiv.org/pdf/2402.13616.pdf>.
27. RATH, Sovit Ranjan. RT-DETR: Paper Explanation and Inference. *Debugger Cafe: Machine Learning and Deep Learning*. 2024. Dostupné tiež z: <https://debuggercafe.com/rt-detr/>.
28. BUCHER, Maxime, CORD, Matthieu, VU, Tuan-Hung a PÉREZ, Patrick. Zero-Shot Semantic Segmentation. 2019. Dostupné tiež z: <https://arxiv.org/pdf/1906.00817.pdf>.
29. KIRILLOV, Alexander, MINTUN, Eric, RAVI, Nikhila, MAO, Hanzi, XIAO, Tete, WHITEHEAD, Spencer, BERG, Alexander C., LO, Wan-Yen, ROLLAND, Chloe, GUSTAFSON, Laura, DOLLAR, Piotr a GIRSHICK, Ross. Segment Anything. 2023. Dostupné tiež z: <https://arxiv.org/pdf/2304.02643.pdf>.
30. ACHARYA, Akruti. Meta AI's Segment Anything Model (SAM) Explained: The Ultimate Guide. 2023. Dostupné tiež z: <https://encord.com/blog/segment-anything-model-explained/>.

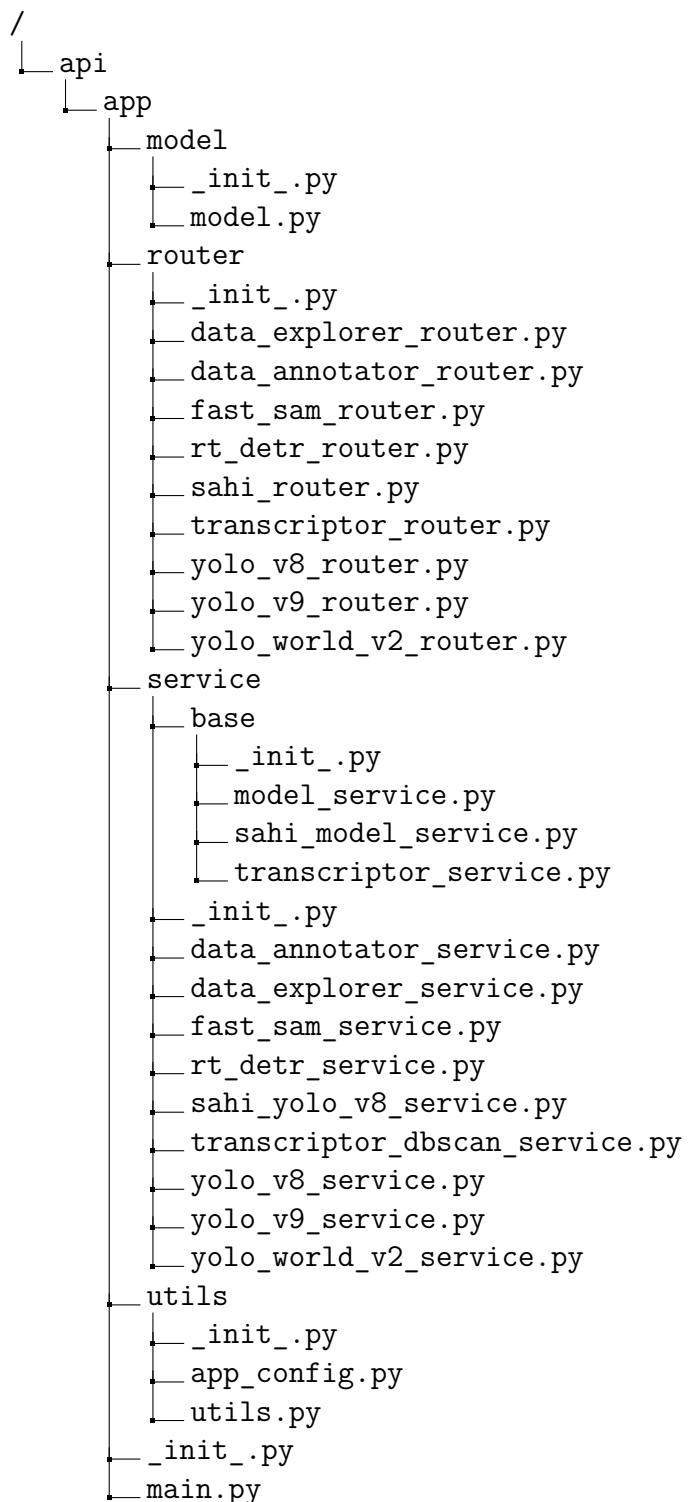
31. ZHAO, Xu, DING, Wenchao, AN, Yongqi, DU, Yinglong, YU, Tao, LI, Min, TANG, Ming a WANG, Jinqiao. Fast Segment Anything. 2023. Dostupné tiež z: <https://arxiv.org/pdf/2306.12156.pdf>.
32. CHENG, Tianheng, SONG, Lin, GE, Yixiao, LIU, Wenyu, WANG, Xinggang a SHAN, Ying. YOLO-World: Real-Time Open-Vocabulary Object Detection. 2024. Dostupné tiež z: <https://arxiv.org/pdf/2401.17270.pdf>.
33. ACHARYA, Akruti. YOLO World Zero-shot Object Detection Model Explained. 2024. Dostupné tiež z: <https://encord.com/blog/yolo-world-object-detection/>.
34. AKYON, Fatih Cagatay, ALTINUC, Sinan Onur a TEMIZEL, Alptekin. Slicing Aided Hyper Inference and Fine-Tuning for Small Object Detection. 2022. Dostupné tiež z: <https://encord.com/blog/yolo-world-object-detection/>.
35. ACHARYA, Akruti. Slicing Aided Hyper Inference (SAHI) for Small Object Detection | Explained. 2023. Dostupné tiež z: <https://encord.com/blog/slicing-aided-hyper-inference-explained/>.
36. VRANIČ, Valentino. Špecifikácia softvéru a prípady použitia. 2013. Dostupné tiež z: <http://www2.fiit.stuba.sk/~vranic/mps/p/p01.pdf>.
37. ANWAR, Aqeel. What is Average Precision in Object Detection Localization Algorithms and how to calculate it? 2022. Dostupné tiež z: <https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b>.
38. RAI, Shiv Pratap. Understanding ONNX: An Open Standard for Deep Learning Model Interoperability. 2023. Dostupné tiež z: <https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b>.
39. CHOUDHARY, Anurag Singh. ONNX Model | Open Neural Network Exchange. 2023. Dostupné tiež z: <https://www.analyticsvidhya.com/blog/2023/07/onnx-model-open-neural-network-exchange/>.
40. ONNX Docs. 2024. Dostupné tiež z: <https://onnx.ai/onnx/>.
41. ESTER, Martin, KRIEGEL, Hans-Peter, SANDER, Jörg a XU, Xiaowei. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. 1996. Dostupné tiež z: <https://www.dbs.ifi.lmu.de/Publikationen/Papers/KDD-96.final.frame.pdf>.

42. KELVIN, Salton do Prado. How DBSCAN works and why should we use it? 2017. Dostupné tiež z: <https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80>.
43. GUPTA, Lokesh. REST API Tutorial. 2023. Dostupné tiež z: <https://restfulapi.net>.
44. MATINA, Shrestha. What is the 3-Tier Architecture? 2023. Dostupné tiež z: <https://medium.com/@shrestha.matina.20/what-is-the-3-tier-architecture-4520522e0720>.
45. Python Docs. 2024. Dostupné tiež z: <https://docs.python.org/3/>.
46. Pip Docs. 2024. Dostupné tiež z: <https://pip.pypa.io/en/stable/>.
47. NumPy Docs. 2024. Dostupné tiež z: <https://numpy.org/doc/>.
48. Pillow Docs. 2024. Dostupné tiež z: <https://pillow.readthedocs.io/en/stable/>.
49. Scikit-Learn Docs. 2024. Dostupné tiež z: <https://scikit-learn.org/stable/>.
50. Supervision Docs. 2024. Dostupné tiež z: <https://supervision.roboflow.com>.
51. FastAPI Docs. 2024. Dostupné tiež z: <https://fastapi.tiangolo.com>.
52. Locust Docs. 2024. Dostupné tiež z: <https://docs.locust.io/en/stable/>.

Prílohy

| | |
|---|----|
| A Štruktúra elektronickej prílohy s praktickým výstupom | II |
| B Používateľská príručka | IV |
| C Ukážky výstupov detekcie na celých historických rukopisoch pri využití
YOLOv8 + SAHI | V |
| D Prehľad úspešnosti detekčných
modelov | XI |

A Štruktúra elektronickej prílohy s praktickým výstupom



```
.../  
    ...api  
    config  
        config.json  
    resources  
        weights  
            exportované váhy modelov vo formáte PyTorch (.pt)  
            a ONNX (.onnx)  
    tests  
        http_tests  
            test_data_annotator.http  
            test_data_explorer.http  
            test_sahi.http  
            test_transcriptor.http  
            test_yolo_rt_detr_sam.http  
        performance_tests  
            performance_test_data_annotator.py  
            performance_test_data_explorer.py  
            performance_test_sahi_yolo_v8.py  
            performance_test_transcriptor_dbSCAN.py  
            performance_test_yolo_v8.py  
        resources  
            súbory (.jpg, .zip) na otestovanie API koncových bodov  
        requirements.txt  
    experiments  
        fast_sam_eval.ipynb  
        rt_detr_eval.ipynb  
        yolo_nas_eval.ipynb  
        yolo_v8_eval.ipynb  
        yolo_v9_eval.ipynb  
        yolo_world_v2_eval.ipynb  
        requirements.txt
```

B Používateľská príručka

- **Experimenty**

- 1. Inštalácia**

- (a) *Python >= 3.9 a Pip >= 24.0*
- (b) *pip install -r experiments/requirements.txt*

- 2. Spustenie**

- (a) *jupyter lab experiments/*

- 3. Používanie**

- (a) spúšťanie buniek v ľubovoľnom Jupyter zošite (.ipynb) z priečinka */experiments/*

- **Webové REST API**

- 1. Inštalácia**

- (a) *Python >= 3.9 a Pip >= 24.0*
- (b) *pip install -r api/requirements.txt*

- 2. Konfigurácia**

- (a) konfigurácia pomocou editácie konfiguračného súboru */api/app/config/config.json*

- 3. Spustenie**

- (a) *python api/app/main.py*

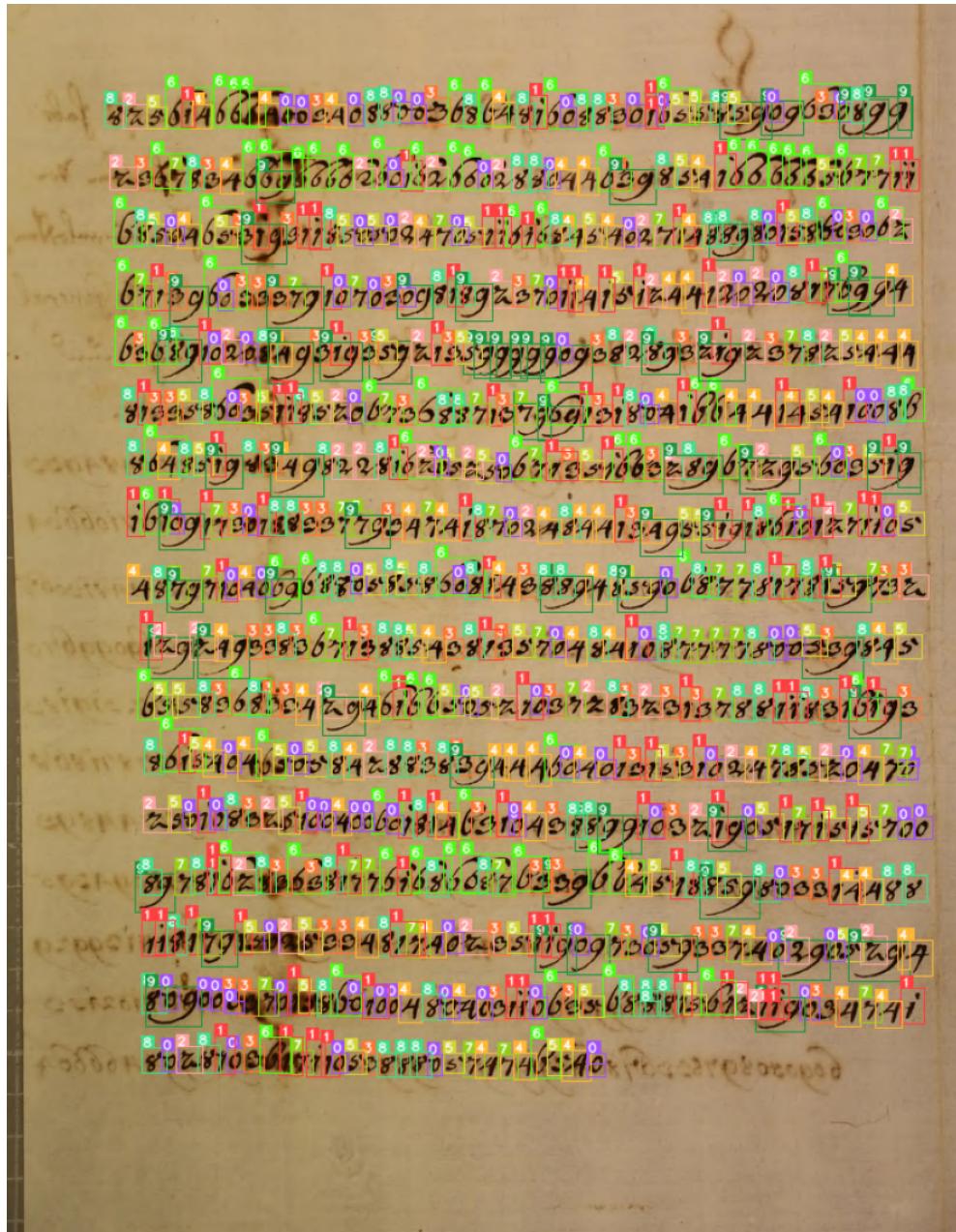
- 4. Používanie**

- (a) dopytovanie na prístupové koncové body REST API zdokumentované na adrese *<host_ip>:<port>/docs/*

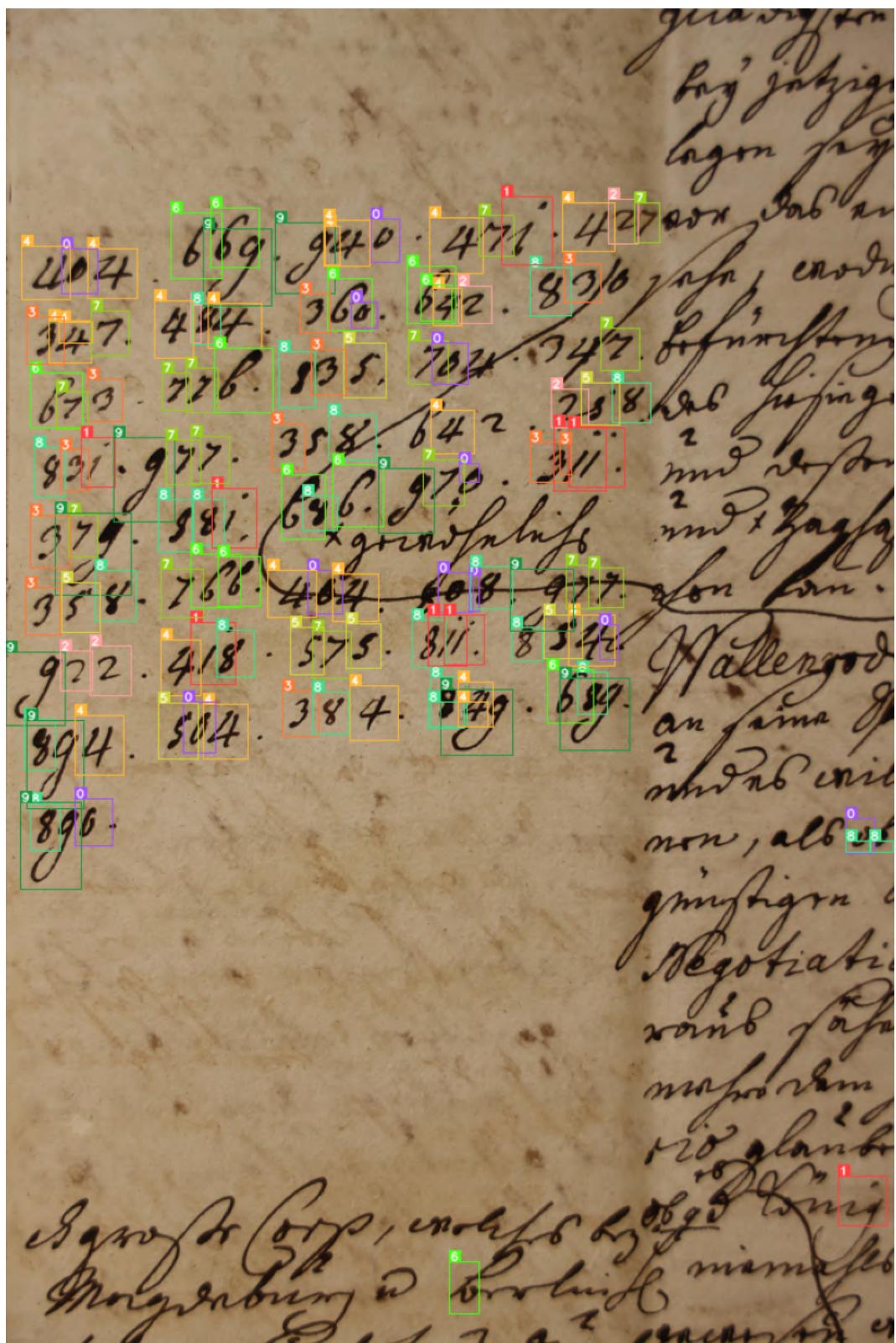
- 5. Testovanie**

- (a) spúšťanie HTTP testov z priečinka */api/tests/http_tests/*
- (b) spúšťanie záťažových (performance) testov z priečinka */api/tests/performance_tests/* pomocou aplikačného rozhrania *locust -f <performance_test_data_annotator.py / performance_test_sahi_yolo_v8.py / performance_test_data_explorer.py / performance_test_yolo_v8.py>*

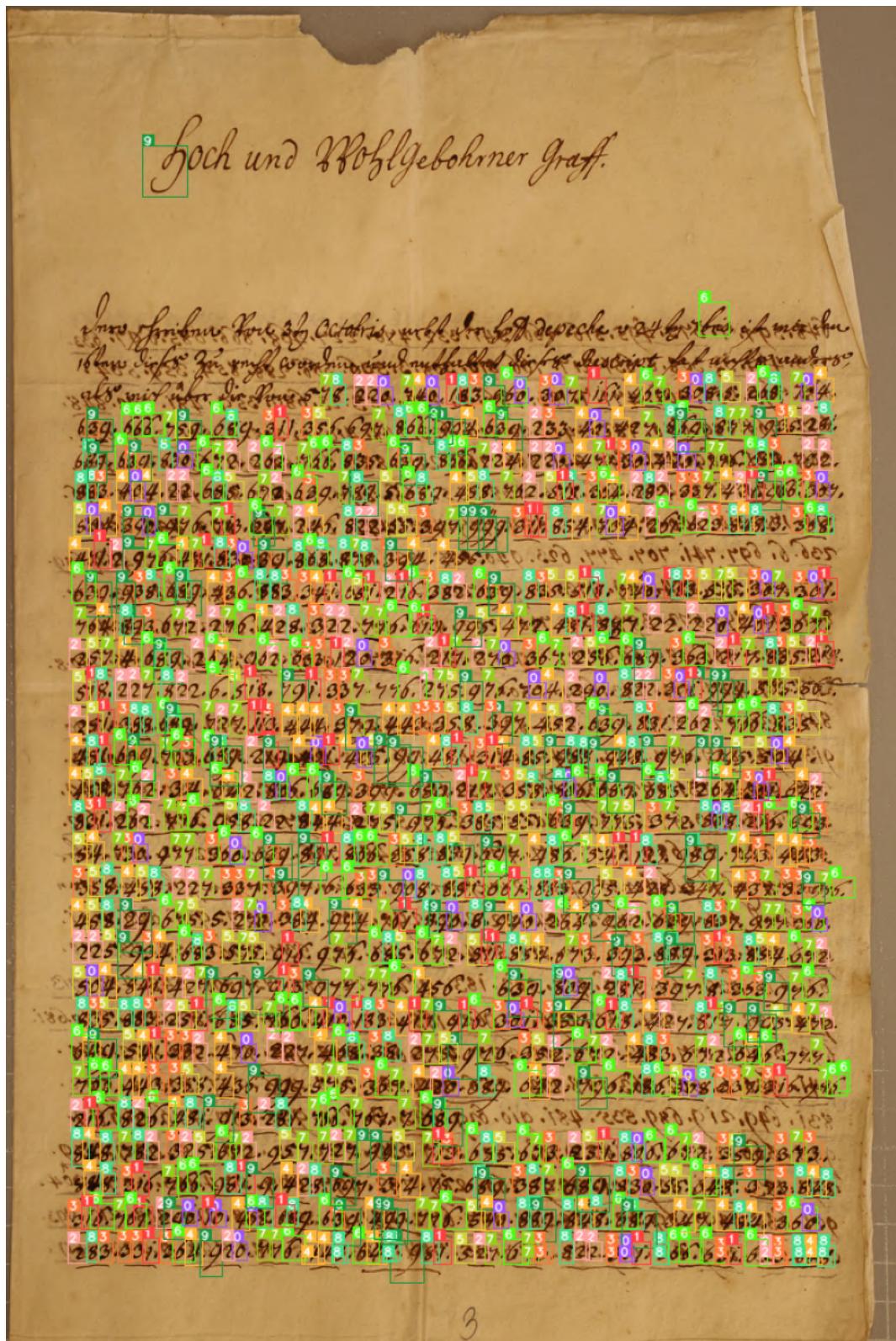
C Ukážky výstupov detekcie na celých historických rukopisoch pri využití YOLOv8 + SAHI



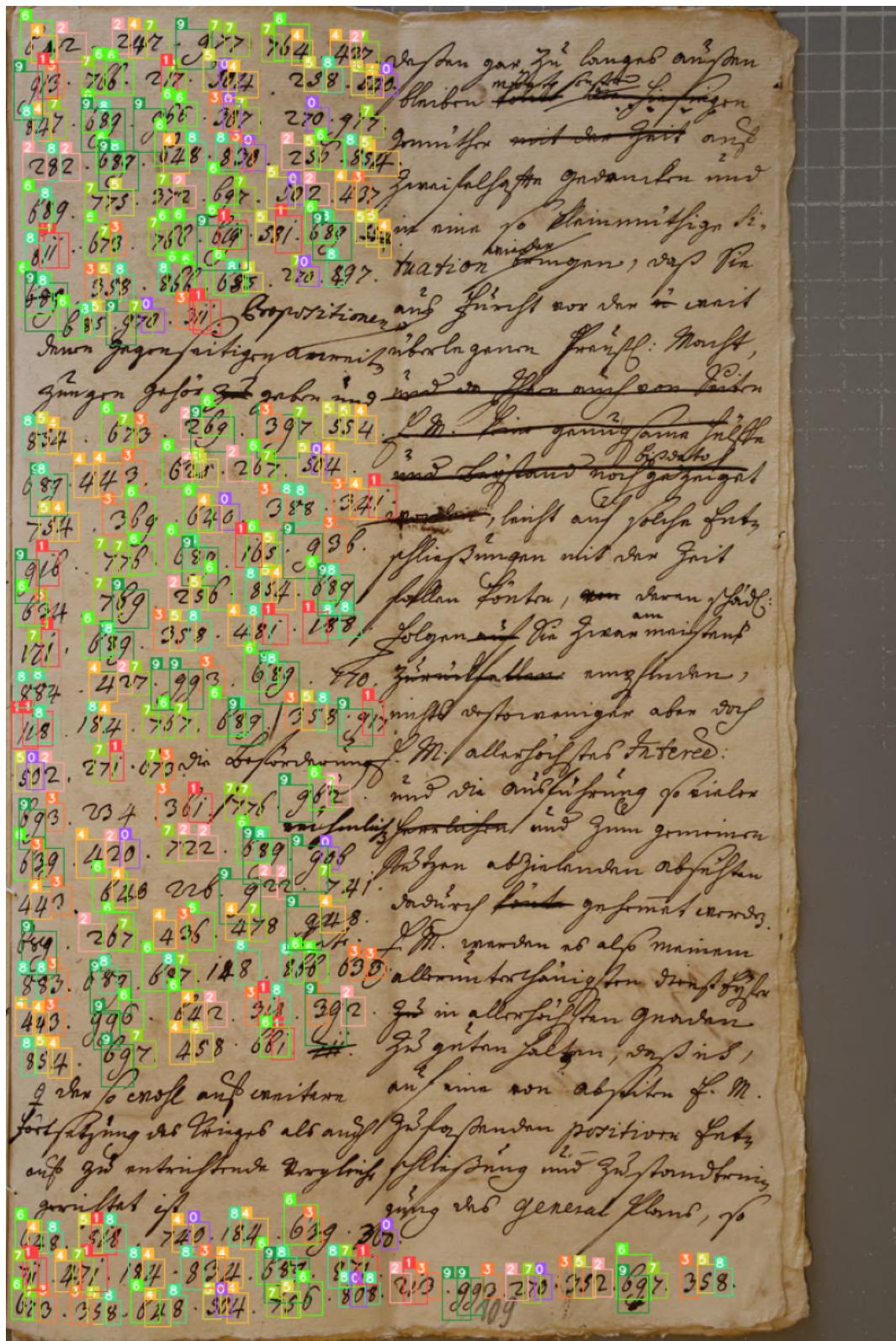
Obr. C.1: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 1 - YOLOv8 + SAHI



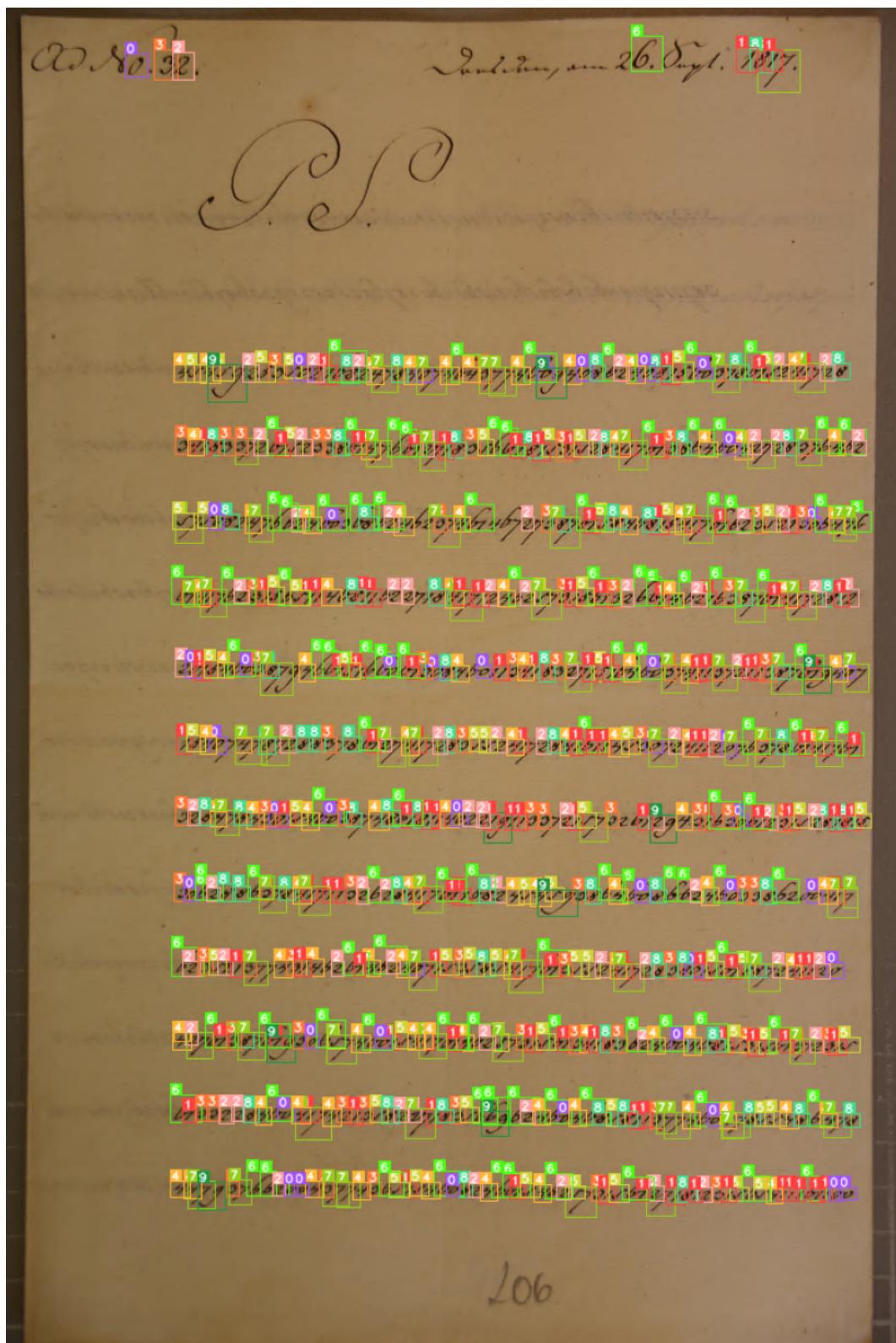
Obr. C.2: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 2 - YOLOv8 + SAHI



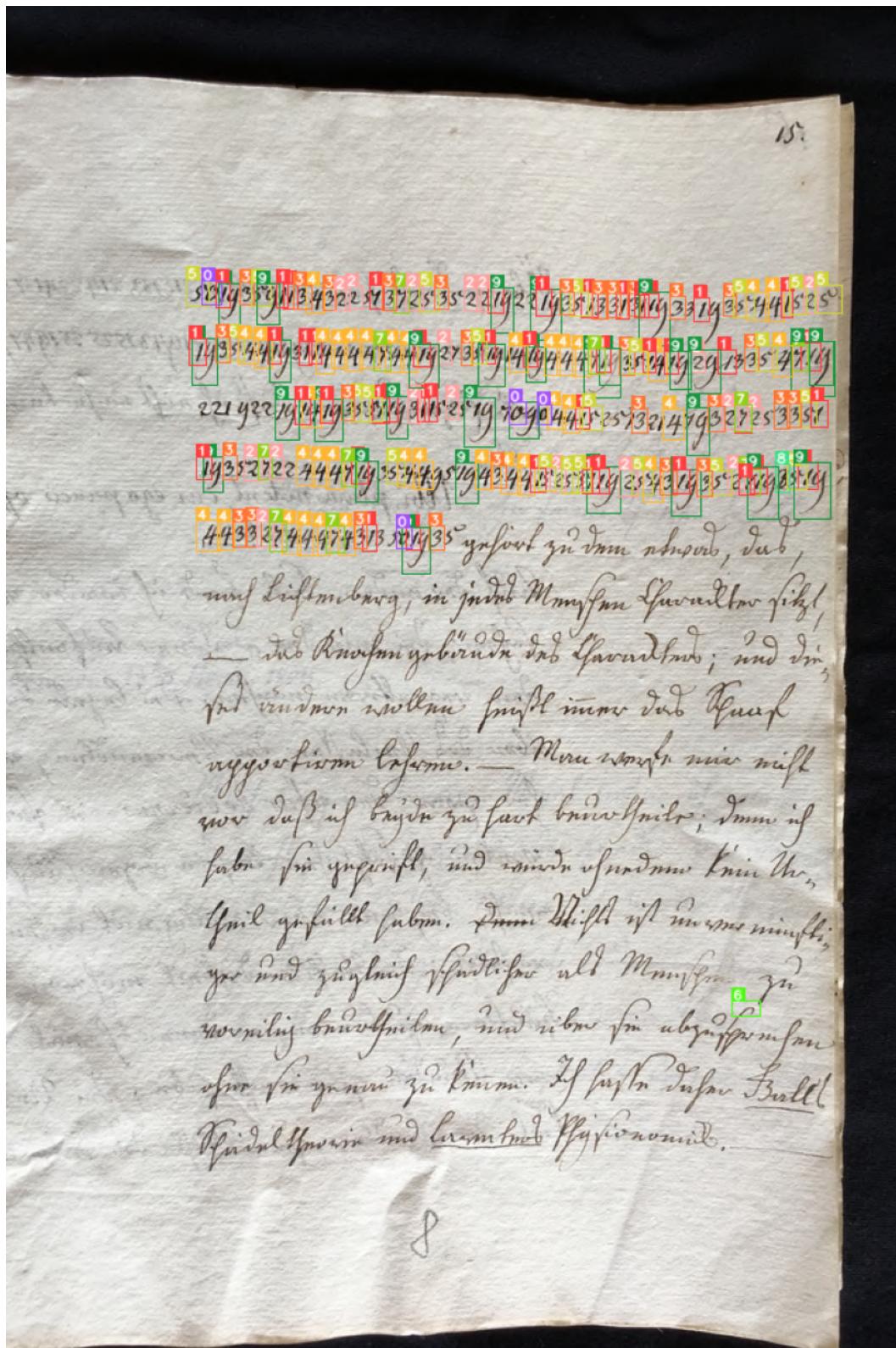
Obr. C.3: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 3 - YOLOv8 + SAHI



Obr. C.4: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 4 - YOLOv8 + SAHI



Obr. C.5: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 5 - YOLOv8 + SAHI



Obr. C.6: Ukážka detekcie malých objektov v historických šifrovaných rukopisoch č. 6 - YOLOv8 + SAHI

D Prehľad úspešnosti detekčných modelov

| | P | R | mAP50 | mAP50-95 |
|-------------------------------|-------|-------|-------|----------|
| VEGA Cropped - Číslice | | | | |
| YOLOv8 | 0.974 | 0.984 | 0.989 | 0.819 |
| YOLO-NAS | 0.968 | 0.976 | 0.985 | 0.806 |
| YOLOv9 | 0.975 | 0.987 | 0.988 | 0.819 |
| YOLO-Worldv2 | 0.977 | 0.984 | 0.989 | 0.843 |
| RT-DETR | 0.953 | 0.963 | 0.981 | 0.784 |
| VEGA Cropped - Glyfy | | | | |
| YOLOv8 | 0.979 | 0.976 | 0.989 | 0.981 |
| YOLO-NAS | 0.972 | 0.967 | 0.978 | 0.964 |
| YOLOv9 | 0.980 | 0.982 | 0.991 | 0.983 |
| YOLO-Worldv2 | 0.972 | 0.983 | 0.989 | 0.980 |
| RT-DETR | 0.846 | 0.959 | 0.955 | 0.940 |