VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Timotej Vančo

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SELF-SUPERVISED UČENÍ V APLIKACÍCH POČÍTAČOVÉHO VIDĚNÍ
SELF-SUPERVISED LEARNING IN COMPUTER VISION APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Timotej Vančo

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Ilona Janáková, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Timotej Vančo ***ID:*** 195460

Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Self-supervised učení v aplikacích počítačového vidění

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je teoreticky zpracovat metody self-supervised (případně semi-supervised) učení - varianta strojového učení bez, případně s omezeným množstvím ručně anotovaných vstupních dat. Součástí zadání je také vybrané metody implementovat a prakticky otestovat. Předpokládá se zaměření na využití v různých aplikacích počítačového vidění.

1. Proveďte a zpracujte literární rešerši dané problematiky. Soustřeďte se především na využití v počítačovém vidění.
2. Zvolte vhodnou testovací úlohu.
3. Pořiďte dostatečně rozsáhlou a pestrou databázi reálných snímků.
4. Na základě předchozích bodů navrhněte vhodné metody pro řešení daného problému.
5. Zvolené metody implementujte a řádně otestujte.
6. Definujte omezující podmínky. Zhodnoťte.

DOPORUČENÁ LITERATURA:

LILIAN, Weng. Self-Supervised Representation Learning. Lil'Log [online]. 10.11.2019 [cit. 2020-09-14]. Dostupné z: https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html

Termín zadání: 8.2.2021 ***Termín odevzdání:*** 17.5.2021

Vedoucí práce: Ing. Ilona Janáková, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Cieľom diplomovej práce je spraviť rešerš problematiky Self-supervised učenia v aplikáciách počítačového videnia, následne vybrať vhodnú testovaciu úlohu s rozsiahlym datasetom, aplikovať Self-supervised metódy a zhodnotiť. Teoretická časť práce je zameraná na popis metód v počítačovom videní, podrobný popis neurónových a konvolučných sietí a rozsiahle vysvetlenie a rozdelenie Self-supervised metód. Záver teoretickej časti je venovaný praktickým aplikáciám Self-supervised metód v praxi. Praktická časť diplomovej práce sa zaoberá popisom tvorby kódu na prácu s datasetmi a aplikáciou metód Rotácie, SimCLR, MoCo a BYOL v úlohe klasifikácie a sémantickej segmentácie. Každá aplikácia metódy je dopodrobna vysvetlená a vyhodnotená pri rôznych parametroch na veľkom datasete STL10. Následne je úspešnosť metód vyhodnotená pri rôznych datasetoch a sú pomenované obmedzujúce podmienky v úlohe klasifikácie. Praktická časť sa uzatvára pri aplikovaní SSL metód na predtrénovanie enkódera v aplikácii sémantickej segmentácie s datasetom Cityscapes.

KĽÚČOVÉ SLOVÁ

Self-supervised učenie, pytorch, konvolučné neurónové siete, klasifikácia, sémantická segmentácia, resnet

ABSTRACT

The aim of the diploma thesis is to make research of the self-supervised learning in computer vision applications, then to choose a suitable test task with an extensive data set, apply self-supervised methods and evaluate. The theoretical part of the work is focused on the description of methods in computer vision, a detailed description of neural and convolution networks and an extensive explanation and division of self-supervised methods. Conclusion of the theoretical part is devoted to practical applications of the Self-supervised methods in practice. The practical part of the diploma thesis deals with the description of the creation of code for working with datasets and the application of the SSL methods Rotation, SimCLR, MoCo and BYOL in the role of classification and semantic segmentation. Each application of the method is explained in detail and evaluated for various parameters on the large STL10 dataset. Subsequently, the success of the methods is evaluated for different datasets and the limiting conditions in the classification task are named. The practical part concludes with the application of SSL methods for pre-training the encoder in the application of semantic segmentation with the Cityscapes dataset.

KEYWORDS

Self-supervised learninig, pytorch, convolution neural networks, classification, semantic segmentation, resnet

VANČO, Timotej. *Self-supervised učení v aplikacích počítačového vidění*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2021, 168 s. Diplomová práca. Vedúci práce: Ing. Ilona Janáková, Ph.D.

Vyhlásenie autora o pôvodnosti diela

Meno a priezvisko autora:	Bc. Timotej Vančo
VUT ID autora:	195460
Typ práce:	Diplomová práca
Akademický rok:	2020/21
Téma závěrečnej práce:	Self-supervised učení v aplikacích počítačového vidění

Vyhlasujem, že svoju záverečnú prácu som vypracoval samostatne pod vedením vedúcej/cého záverečnej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto záverečnej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podpisuje iba v tlačenej verzii.

POĎAKOVANIE

Ďakujem vedúcej diplomovej práce, Ing. Ilone Janákovej, Ph.D. za konzultácie, ochotu vždy mi poradiť a nájsť si čas aj v zložitej dobe pandémie, účinnú metodickú, pedagogickú, odbornú pomoc, trpezlivosť a vedenie a všetky cenné rady pri spracovávaní mojej diplomovej práce. Taktiež by som rád poďakoval za oporu a motiváciu mojej rodine, priateľke a blízkym. Vážim si to.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Umelá inteligencia sa nachádza posledné roky v našich životoch stále viac a viac. Síce spočiatku boli ľudia veľmi skeptickí, ich dôvod bol založený na racionálnych problémoch vo výpočtovej sile vtedy používaných počítačoch. Ten neumožňoval tvorbu niekoľkých vrstiev neurónov a z toho dôvodu sa častejšie používali iné počítačové algoritmy. Už v roku 1997 sa podarilo poraziť šachového majstra počítačom a slová o umelej inteligencii sme počúvali v rôznych filmoch a predpovediach budúcnosti. Dnes sa umelá inteligencia používa v autonómnych autách, ktoré riadi a parkuje bez akéhokoľvek vstupu od vodiča. Používame ju pri navigácii a výpočte vhodnej trasy podľa aktuálnej cestnej premávky. Taktiež ju dennodenne používame pri emailovej komunikácii, kedy nám automaticky navrhne odpoveď alebo ešte predtým pretriedi správy do spamu. Nest termostat dokáže na základe nášho pravidelného fungovania v domácnosti navrhnúť správny model, kedy má zvýšiť alebo znížiť teplotu v dome. Ak si zakúpime produkt cez Amazon alebo iný e-shop, tak nám vďaka umelej inteligencii ich e-shop ponúkne ďalší podobný tovar. Google Translate je vďaka umelej inteligencii dnes už schopný porozumieť hovorenému slovu, pochopiť kontext vety a tak lepšie preložiť danú vetu do iného jazyka. Umelá inteligencia sa bude čoraz viac používať aj v medicíne pri detekovaní nádorov alebo chorôb a vývoji nových liekov. Je to budúcnosť, ktorá nám pomôže žiť kvalitnejší život. V mojej práci sa venujem umelej inteligencii v počítačovom videní.

V prvej kapitole mojej diplomovej práce popisujem všeobecné použitie umelej inteligencie v počítačovom videní a vysvetľujem základné pojmy rozdelenia dát podľa učenia s učiteľom alebo bez. Následne porovnávam spôsoby trénovania medzi sebou a podrobne popisujem metódy v počítačovom videní. Medzi takéto metódy patrí klasifikácia a detekcia obrazu, sledovanie objektov alebo aplikovanie sémantickej segmentácie. Rôzne metódy sa môžu v mnohom odlišovať, ale spájajú ich často konvolučné siete, ktoré popisujem v ďalšej kapitole veľmi podrobne. Taktiež uvádzam príklad reziduálnych spojení v konvolučných sieťach, ktorých princíp neskôr používam na demonštráciu self-supervised metód.

Prirodzene prechádzam ku hlavným kapitolám mojej práce a tými sú teoretický popis Self-supervised metód a ich následné aplikovanie s vyhodnotením v oblasti počítačového videnia. Prvá kapitola, teoretická, je aj najdlhšou kapitolou teórie a podrobne v nej rozoberám ako self-supervised trénovanie funguje, čo je to pretext a downstream úloha, rozdelenie metód na generatívne a kontrastívne. Medzi generatívny patrí rekonštrukcia obrazu (image colorization, super-resolution, inpaiting alebo cross-channel prediction), common sense reasoning (jigsaw, context prediction, geometric transformation recognition), clustering (deep cluster alebo syntetické obrázky). V kontrastívnej oblasti sa nachádzajú presnejšie, ale zložitejšie metódy ako

SimCLR, MoCov2 alebo BYOL, ktoré sa dokážu svojou kvalitou rovnať metódam trénovania s učiteľom ak ich natrénujeme zväčša dlhším tréningom, väčším počtom epoch a taktiež s väčším objemom dát. Ku koncu kapitoly prinášam aj využitie týchto metód v projektoch v praxi napríklad pri predpovedaní hĺbky z 2D obrazu.

Druhá hlavná časť mojej práce je praktická časť, v ktorej aplikujem 4 rôzne self-supervised metódy, konkrétne úloha rotácie a tri presnejšie a zložitejšie metódy z kontrastívnej oblasti a to konkrétne SimCLR, MoCov2 a BYOL. Ako testovaciu úlohu som zvolil úlohu klasifikácie obrázkov do tried. Štyri self-supervised metódy používam v spojení konvolučnej siete s reziduálnymi blokmi, ResNet18 a trénovaní na neoznačených dátach a následným využitím týchto predtrénovaných váh na dotrénovanie poslednej lineárnej vrstvy spôsobom učenia s učiteľom (supervised) pri veľkosti dotrénovávacích dát od 1% po 10%. Jednotlivé metódy presne opisujem a vyhodnocujem každú zvlášť pri rôznych vstupných parametroch. Následne takéto modely porovnávam a vyhodnocujem aj medzi sebou pri rôznych datasetoch ako STL10, CIFAR10, Imagenete, Industry, MNIST, Weather a definujem obmedzujúce podmienky.

Druhá úloha, pri ktorej používam rovnaké 4 self-supervised metódy, je úloha sémantickej segmentácie. V tejto časti rovnako využívam reziduálne konvolučné siete ako enkóder časť a následne dotváram dekóder časť siete, ktorej cieľom je naučiť sa triedy objektov a zakresliť ich do výsledného obrázku. Ako trénovací dataset používam verejne dostupný City-scapes a úspešnosť metód vyhodnocujem metódami celkovej presnosti a prekrytia, takzvané IoU (Intersection over Union).

1 Umelá inteligencia v počítačovom videní

Jedno z hlavných využití umelej inteligencie je aj v počítačovom videní. Algoritmy hlbokého učenia (deep learning) v spolupráci s rýchlym výpočtovým výkonom a prácou s obrazom, sú schopné v reálnom čase detekovať predmety, rozpoznávať tváre, rozlišovať farby a tvary alebo hľadať poruchy a chyby v materiáloch. Tieto algoritmy sa používajú v priemyselnej automatizácii, autonómnych autách, v obchodoch, v medicíne pri skenovaní kardiografov alebo hľadaní nádorových ochorení. Tieto metódy sú ale vždy tak dobré, ako dobré a kvalitné dáme do nich vstupné dáta, respektíve čím kvalitnejšie dáta použijeme o to presnejší výsledok získame. Na základe týchto vstupných dát rozdeľujeme metódy umelej inteligencie na 3 základné - supervised learning (učenie s učiteľom), unsupervised learning (učenie bez učiteľa) a reinforcement learning (učenie posilňovaním).

1.1 Supervised Learning - učenie s učiteľom

Pri učení s učiteľom používame dáta, ktoré obsahujú label (triedu), vďaka ktorej vieme o aké dáta ide. Napríklad fotka psa bude patriť pod triedu psa a fotka s mačkou pod triedu mačky. Takto pripravené dáta následne posúvame do algoritmu, ktorý sa učí priradiť dátam tú správnu triedu. Bez tejto triedy by nebolo možné vytvoriť vzťah medzi obrázkami a ich triedami a následne hľadať vhodnú triedu pre obrázok, ktorý algoritmus vidí po prvý krát. Učenie s učiteľom je najpoužívanejší spôsob tréningu algoritmov umelej inteligencie [1]. Najznámejšie metódy sú - rozhodovací strom (decision tree), lineárna regresia (Linear regression), metódy podporných vektorov (Support vector machines) alebo neurónové siete [1]. V mojej práci budem taktiež využívať aj tento prístup trénovania neurónových sietí s cieľom porovnať ich úspešnosť a kvalitu s inými metódami trénovania.

1.2 Unsupervised Learning - učenie bez učiteľa

Pri učení bez učiteľa, do dát nie je zasahované a neobsahujú žiadny label (triedu) alebo definíciu s popisom, čo sa nachádza na obrázku. Algoritmus sa učí na základe vlastností dát a zvyčajne sa snaží vytvoriť skupiny, do ktorých pretriedi jednotlivé vstupné dáta podľa vzájomnej podobnosti. Algoritmy, ktoré používajú dáta bez tried sa zvyčajne aplikujú na problémy vytvárania skupín/triedenia (clustering) alebo dimenzionálnej redukcie (dimensionality reduction) [2].

1.3 Reinforcement Learning - učenie posilňovaním

Učenie posilňovaním alebo tiež spätnoväzobné učenie je spôsob plánovania a trénovania, v ktorom dochádza počas tréningu k zásahom/interakciám od učiteľa alebo prostredia k modelu, ktorý sa učí. Modelu sa často hovorí aj agent. Cieľom je odmeniť agenta pri vykonaní správnej akcie alebo zvyšovať odmenu pri dosahovaní žiadaného výkonu [3].

Definovanie problému pri trénovaní s učiteľom a bez učiteľa

Keďže v oblasti umelej inteligencie a počítačového videnia, nechceme iba rozpoznávať jednoduché triedy ako napríklad mačky a psov, ale zložitejšie objekty, potrebujeme vytvárať anotácie nových datasetov (anotácia je priradenie triedy novému obrázku a dataset je skupina obrázkov, ktorá bude použitá na trénovanie modelu). Z tohto dôvodu nám vznikajú 2 hlavné problémy:

- Priraďovať triedy alebo anotácie je časovo náročné. Napríklad ak chceme vytvoriť model, ktorý rozpozná typ auta na obrázku, potrebujeme získať niekoľko tisíc až desať tisíc obrázkov áut a každému obrázku priradiť triedu. Čiže človek sa na obrázok musí pozrieť a rozhodnúť o aký typ auta ide.
- Priraďovať triedy alebo anotácie je aj finančne náročné. Vzhľadom na predchádzajúci bod potrebujeme vytvoriť množstvo anotácií a tieto anotácie sú bežne tvorené manuálne ľuďmi. Z toho dôvodu je nutné vždy nájsť človeka, ktorý bude priraďovať triedy jednotlivým obrázkom, čo znamená potrebu daného človeka aj platiť (alebo inak odmeniť za vykonanú prácu) a pravdepodobne nie jedného, čoho následkom je finančná náročnosť tvorby anotácií veľkému množstvu dát. Preto nie je vhodné neustále hľadať nové obrázky a žiadať desiatky ľudí, aby trávili čas prechádzaním každého obrázku. Pri úlohách segmentácie je tento bod ešte znásobený, nakoľko nejde už iba o priradenie obrázku do jednej z niekoľkých tried, ale o časovo náročnú tvorbu anotácií (vyznačenie, kde presne sa hľadaný objekt nachádza) na každom obrázku. Takáto tvorba anotácií napríklad pri obrázkoch röntgenových snímkov mozgu a určovaniu, čo je nádor, vyžaduje aj odborné vzdelanie a prax. Odmena takéhoto experta je teda finančne náročná.

Naskytá sa otázka, aké je potom riešenie tohto problému? Ak máme iba málo času a iba málo peňazí na tvorbu tried alebo anotácií v datasete pre časť obrázkov z veľkého množstva. Tu je priestor, ktorý vypĺňajú metódy semi-supervised učenia (s čiastočným učiteľom), ktoré sa nachádzajú presne medzi skupinami supervised a unsupervised (učenie s a bez učiteľa). Ďalšou skupinou, ktorá patrí medzi nový

spôsob prístupu a tréningu a nachádza sa medzi tými hlavnými je self-supervised učenie (s vlastným učiteľom). Semi-supervised aj Self-supervised metódy budem v mojej práci používať, testovať a vyhodnocovať ich výsledky. Tieto metódy sú si ale veľmi blízke a v nasledujúcom texte ich podrobnejšie vysvetlujem.

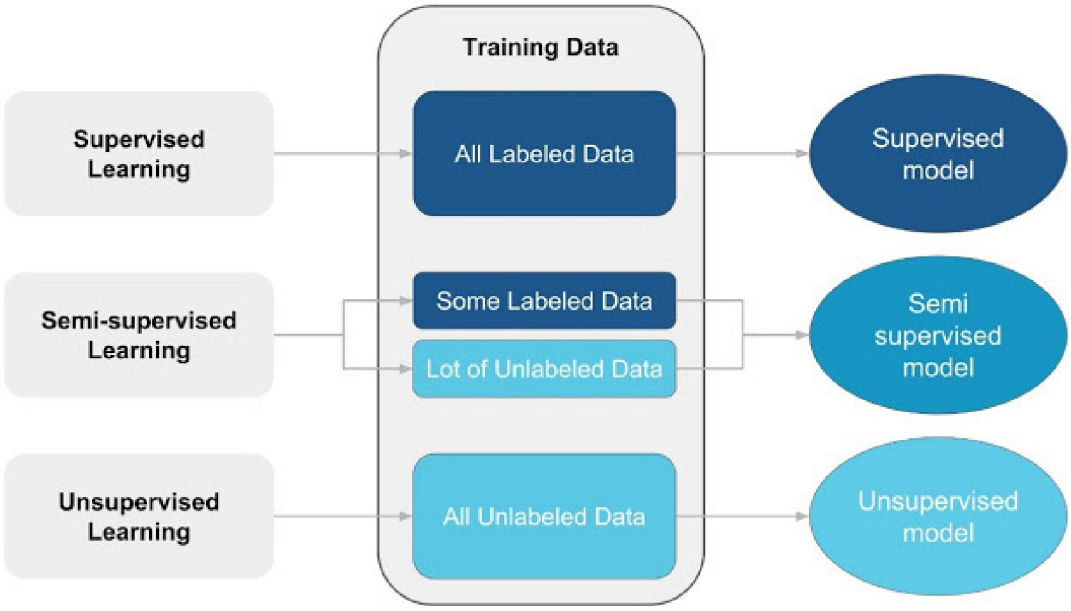
1.4 Semi-Supervised Learning (SmSL) - učenie s čiastočným učiteľom

Semi-supervised spôsob učenia, je učenie s čiastočným dohľadom od učiteľa. Pri tomto prístupe s čiastočným učiteľom môžeme trénovať klasifikátor/model na malom množstve dát, ktoré obsahujú aj označenie tried/labelov a následne predpovedať a hľadať odpoveď na neoznačené dáta. Pretože tieto predpovede sú pravdepodobne lepšie ako náhodné hádanie, predpovede neoznačených údajov je možné v následných iteráciách klasifikátora prijať ako „pseudo triedy“. Aj keď existuje veľa metód učenia sa s učiteľom, táto špecifická technika sa nazýva autotréning alebo semi-supervised učenie.

Trénovanie s čiastočným učiteľom je zbierka techník strojového učenia, kde existujú dva súbory údajov: označený (obsahuje aj triedu/label) a neoznačený (neobsahuje triedu). Existujú dva hlavné problémy, ktoré je možné vyriešiť pomocou výučby s čiastočným učiteľom:

- Transdukčné učenie (vytvoriť triedy pre neoznačené dáta)
- Induktívne učenie (zovšeobecnenie) - nájsť funkciu, ktorá mapuje vstupy na výstupy, napríklad klasifikácia [4]

Semi-supervised štýl učenia pracuje na zdokonalení súboru údajov pridaním nových príkladov. Existujú iteračné algoritmy, kde sa trénuje model na danom datasete a tento model vylepšujeme nasadením do reálneho sveta a konfrontovaním s jeho vstupmi a následným vylepšovaním modelu. Na rozdiel od tréningu s vlastným učiteľom (self-supervised učenie) sa učenie s čiastočným učiteľom zameriava na to, aby sa súčasne použili označené aj neoznačené údaje na zlepšenie výkonu modelu s učiteľom (supervised learning). Príklad použitia tohto prístupu je v štúdii s názvom FixMatch, kde sa trénuje model na obrázkoch s triedami [5]. Následne pre svoje neoznačené obrázky používa rozšírenie (augmentácie) s cieľom vytvorenia dvoch obrázkov pre každý neoznačený obrázok. Tým sa snaží model zaistiť, aby predpovedal rovnaké označenie/triedu pre obe rozšírenia/zmeny obrázkov. Takýto postup sa tiež nazýva, ako hľadanie chyby krížovou entropiou (cross-entropy loss) [5].



Obr. 1.1: Náčrt rôznych prístupov trénovania v strojovom učení [6]

1.4.1 Vzťah medzi semi-supervised a self-supervised metódami:

Self-supervised metóda (na reprezentáciu dát) sa dá považovať za semi-supervised prístup ak sa spraví takzvané doladenie (fine-tuning) reprezentácií naučených údajov pomocou dát, ktoré majú label/triedu/označenie a tým vyriešime problém učenia s učiteľom (supervised learning). Tento prístup sa taktiež nazýva "nasledujúce úloha" - downstream task.

1.4.2 Self-supervised vs. supervised learning

Self-supervised learning (SSL) metóda je z časti učenie s učiteľom, pretože jeho cieľom je nájsť podobnosť alebo funkciu z dvojíc vstupných dát a ich tried. Výslovné použitie párov s označením (labelov) vstupov a výstupov pri trénovaní v SSL nie je potrebné. Namiesto toho sa z údajov implicitne a autonómne extrahujú korelácie, vložené metadáta alebo doménové znalosti dostupné zo vstupu a používajú sa ako signál, ktorými sa dohliada na tréning. Rovnako ako učenie s učiteľom, aj učenie s vlastným učiteľom (SSL) využíva metódy regresie a klasifikácie [7].

1.4.3 Self-supervised vs. unsupervised learning

Učenie s vlastným učiteľom (SSL) je z časti aj učenie bez učiteľa, pretože systém sa učí bez použitia výslovne uvedených labelov/tried/označení. Rozdiel v učení bez učiteľa je v tom, že model sa neučí inherentnú štruktúru údajov. Učenie s vlastným

učiteľom (SSL) sa na rozdiel od učenia bez učiteľa nesústredí a nepoužíva na zhlukovanie a zoskupovanie, znižovanie rozmerov a dimenzií alebo zisťovanie anomálií [7].

1.4.4 Self-Supervised vs. semi-supervised learning

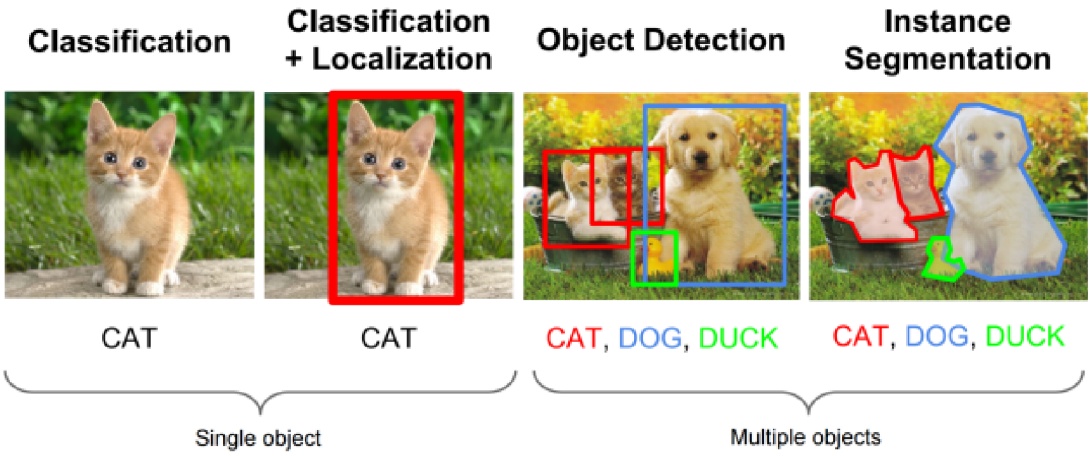
Kombinácia označených a neoznačených údajov sa používa na trénovanie algoritmov s čiastočným učiteľom (semi-supervised), kde menšie množstvo označených údajov v spojení s veľkým množstvom neoznačených údajov môžu urýchliť úlohy učenia. Výučba s vlastným učiteľom (self-supervised) sa líši, pretože systémy sa učia úplne bez použitia výslovne uvedených označení/tried [7].

Používanie modelov s vlastným učiteľom dokáže predvídať prostredníctvom prirodzeného vývoja a dôsledkov jeho konania, podobne ako sa novorodenci môžu naučiť neuveriteľné množstvo informácií v prvých týždňoch / mesiacoch života pozorovaním a zvedavosťou. Učenie s vlastným učiteľom má potenciál rozšíriť trénovanie modelov na nové úrovne vyžadované pri zložitých operáciách v medicíne, autonómnom riadení, robotike, porozumení jazyka alebo rozpoznávaní objektov [7].

2 Metódy používané v počítačovom videní

Počítačové videnie je v súčasnosti jedným z najprogresívnejších výskumných odborov v rámci hlbokého učenia (deep learning). Nachádza sa na križovatke mnohých akademických predmetov, ako sú informatika (grafika, algoritmy, teória, systémy, architektúra), matematika (získavanie informácií, strojové učenie), strojárstvo (robotika, reč, NLP (natural language processing - spracovanie jazyka), spracovanie obrazu), fyzika (optika), biológia (neuroveda) a psychológia (kognitívna veda). Hlavné metódy používané v počítačovom videní sú nasledovné:

- Klasifikácia obrazu
- Detekcia objektov
- Sledovanie objektov
- Sémantická segmentácia
- Segmentácia inštancií

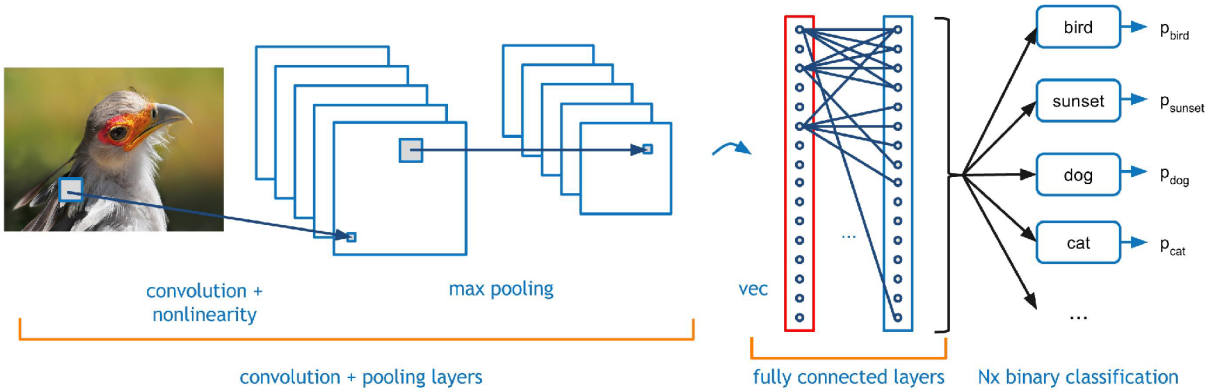


Obr. 2.1: Metódy používané v počítačovom videní [8]

2.1 Klasifikácia obrazu

Problém klasifikácie obrazov je v hľadaní a predpovedaní správnej skupiny pre nový ešte nikdy nevidený obraz a tiež merať presnosť tejto predpovede. S touto úlohou sa spája množstvo výziev vrátane zmeny mierky, triedy, deformácie obrazu alebo zmenám podmienok osvetlenia. Najčastejšie sa na vyriešenie tohto problému používajú konvolučné neurónové siete (Convolution neural network - CNN). Trénovanie takejto siete sa deje nasledovne: Trénovací dataset sa skladá z N obrázkov, z ktorých každý je označený jednou z K rôznych tried [14]. Následne pomocou tejto tréningovej sady

sa model učí, ako každá z tried vyzerá. Nakoniec sa vyhodnotí kvalita modelu tým, že ho požiadame, aby predpovedal triedy pre novú sadu obrázkov, ktoré ešte nikdy predtým nevidel. Potom porovnáme skutočné triedy týchto obrázkov s tými, ktoré predpovedal model. Najznámejšie konvolučné siete s vysokou presnosťou klasifikácie sú AlexNet, VGGNet, ResNet alebo GoogLeNet [14].



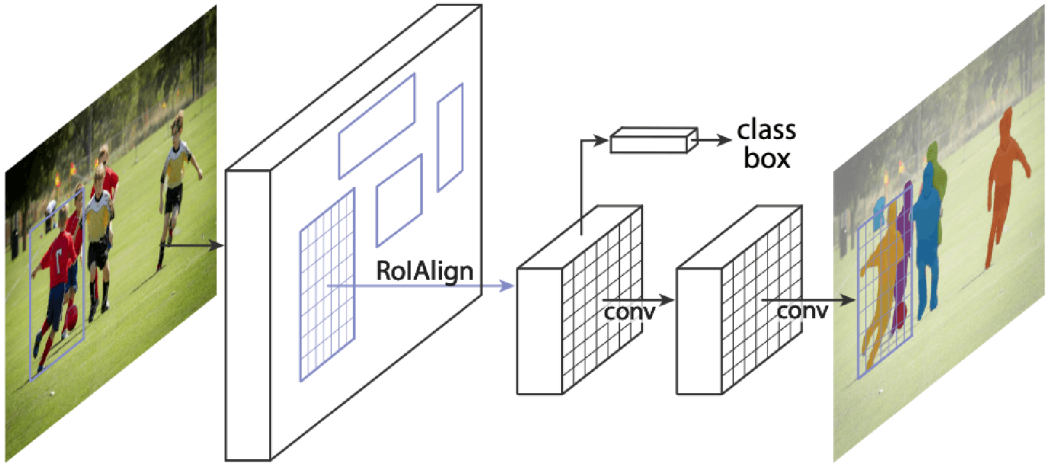
Obr. 2.2: Architektúra konvolučnej neurónovej siete [9]

2.2 Detekcia obrazu

Úloha detekovať objekty v obrázkoch zvyčajne zahŕňa výstup ohraničovacích rámikov a tried pre jednotlivé objekty. Toto je hlavný rozdiel od úlohy klasifikácie alebo lokalizácie použitím klasifikácie a lokalizácie veľa objektov namiesto iba jedného dominantného objektu. Existujú iba 2 triedy klasifikácie objektov a to ohraničenie objektov rámikmi alebo ohraničiť neobjekty. Napríklad pri detekcii automobilov sa musí zabezpečiť detekcia všetkých áut na danom obrázku spolu s ich ohraničením. Pri detekcii obrazu sa používa technika posuvného okna (sliding window), ktoré používa konvolučnú sieť na rozpoznanie predmetu v okne. Takéto využitie konvolučných sietí s posuvným oknom je výpočtovo veľmi náročné. Riešením je detekovať predmety iba v určitých častiach, ktoré pravdepodobne obsahujú hľadané objekty. Prvým modelom s touto metódou bola R-CNN (regionálna konvolučná neurónová sieť). V R-CNN najskôr skenujeme vstupný obraz na možné objekty pomocou algoritmu s názvom Selektívne vyhľadávanie, pričom sa generuje približne 2 000 návrhov regiónov. Následne sa spustí konvolučná sieť nad každý z týchto návrhov. Na záver sa zoberie výstup každej konvolučnej siete a vloží sa do algoritmu SVM (support vector machine) na klasifikáciu oblasti. Týmto krokom sa detekcia obrazu zmenila na klasifikáciu obrazu. Nasledovníkom R-CNN je Fast R-CNN, ktorá zvyšuje rýchlosť detekcie pomocou 2 vylepšení:

- Vykonanie extrakcie vlastností pred navrhnutím oblastí obsahujúcich objekty, teda iba trénovaním jedinej konvolučnej siete pre celý obraz
- Nahradením SVM vrstvou softmax, čím sa namiesto vytvorenia nového modelu rozšírila neurónová sieť o predpoveď výslednej triedy

Fast R-CNN sa podarilo zlepšiť nahradením algoritmu pomalého selektívneho vyhľadávania rýchlou neurónovou sieťou s názvom Region Proposal Network (RPN), ktorá umožňuje predvídať návrhy vlastností obrázku. Týmto krokom vznikla Faster R-CNN. RPN sa používa na rozhodnutie "kde" hľadať, aby sa znížili výpočtové požiadavky celého procesu. RPN rýchlo a efektívne skenuje každé miesto s cieľom posúdiť, či je v danom regióne potrebné vykonať ďalšie operácie na detekciu. Síce Faster R-CNN je rýchla, dnes sa dostávajú do popredia ešte rýchlejšie algoritmy ako You Only Look Once (YOLO), Single Shot MultiBox Detector (SSD) alebo Region-Based Fully Convolutional Networks (R-FCN) [14].

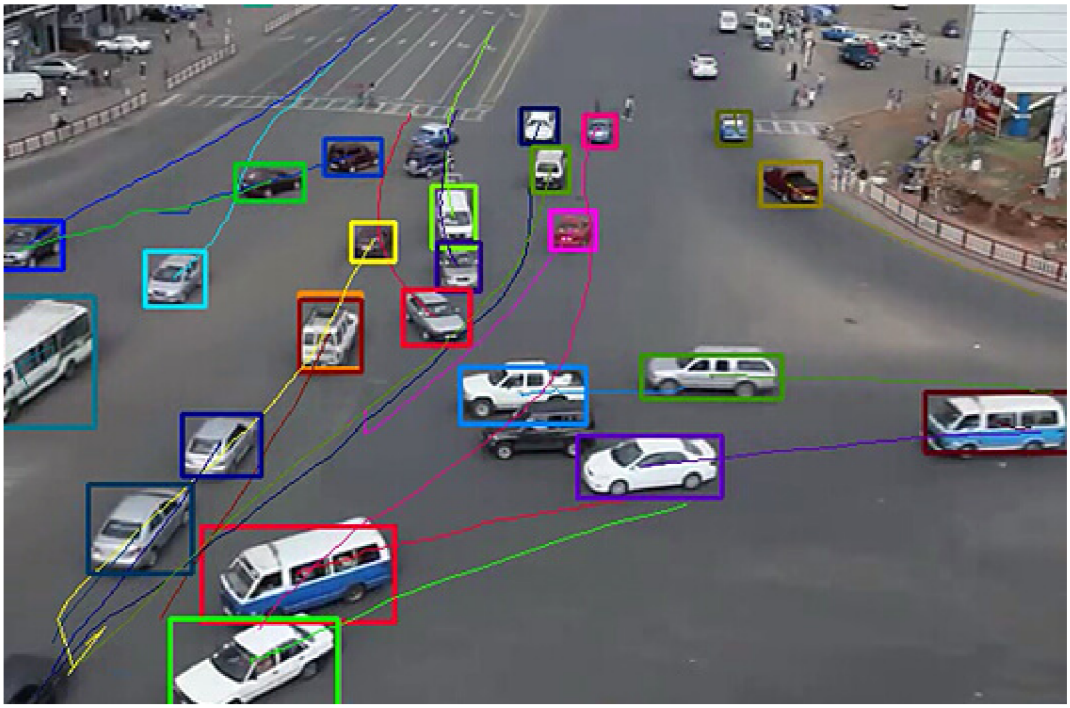


Obr. 2.3: Architektúra Mask R-CNN [10]

2.3 Sledovanie objektov

Za sledovanie objektov sa označuje proces sledovania konkrétneho objektu záujmu alebo viacerých objektov v danej scéne. Tradične má aplikácie vo videách a interakciách v reálnom priestore, kde sa pozorovania uskutočňujú po počiatočnej detekcii objektu. Veľké využitie sa uplatňuje pri autonómnych autách ako Tesla alebo Uber. Metódy sledovania objektov možno podľa modelu pozorovania rozdeliť do 2 kategórií: generatívna metóda a diskriminačná metóda. Generatívna metóda používa generatívny model na opísanie zjavných charakteristík a minimalizuje chybu rekonštrukcie pri vyhľadávaní objektu, napríklad PCA (Principal component analysis -

Analýza hlavných komponentov). Diskriminačnú metódu je možné použiť na rozlíšenie medzi objektom a pozadím, jej výkon je robustnejší a postupne sa stáva hlavnou metódou sledovania. Diskriminačná metóda sa označuje aj ako Tracking-by-Detection a do tejto kategórie patrí hlboké učenie. Aby sme dosiahli sledovanie pomocou detekcie, detekujeme možných kandidátov obsahujúcich hľadaný objekt a v nich všetkých následne pomocou hlbokého učenia rozpoznáme požadovaný objekt od kandidátov. Existujú dva druhy základných modelov, ktoré je možné použiť: stacked auto encoders (SAE) alebo konvolučné neurónové siete (CNN) [14].

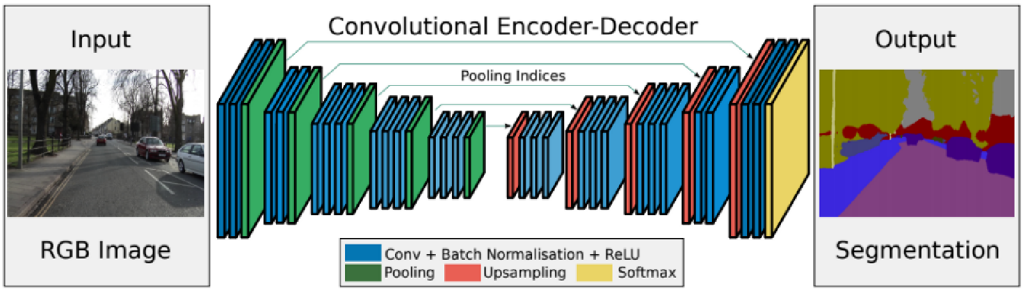


Obr. 2.4: Rozpoznanie a sledovania viacerých objektov [11]

2.4 Sémantická segmentácia

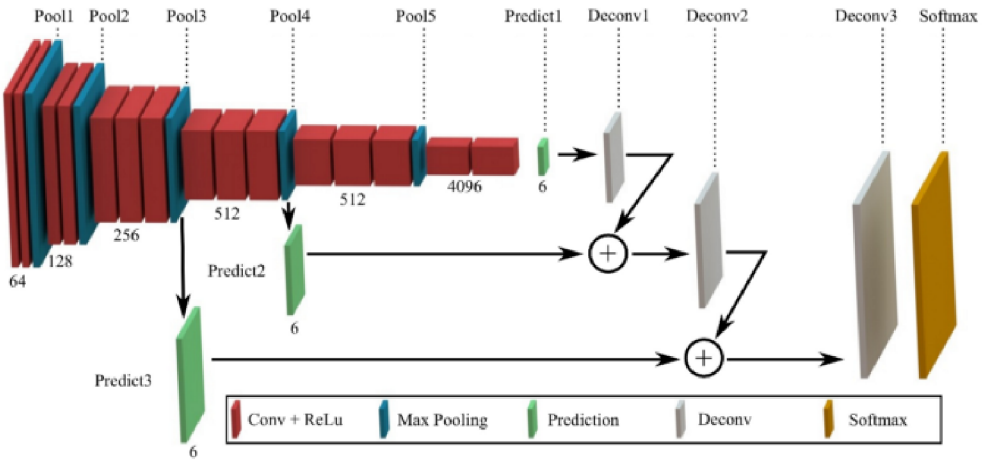
Jedným z hlavných prvkov počítačového videnia je proces segmentácie, ktorá rozdeľuje celé obrázky na zoskupenia pixelov, ktoré je potom možné označiť a klasifikovať. Sémantická segmentácia sa snaží predovšetkým sémanticky pochopiť úlohu každého pixelu v obraze (napríklad je to auto, motorka alebo nejaký iný typ triedy). Napríklad nižšie na obrázku Obr. 2.5 sa musí okrem rozpoznania osoby, cesty, automobilov, stromov atď. vymedziť aj hranice každého objektu. Na rozdiel od klasifikácie preto od našich modelov potrebujeme husté pixelové predpovede. Rovnako ako pri iných úlohách počítačového videnia, aj pri problémoch so segmentáciou majú CNN úspechy.

Jedným z populárnych počiatočných prístupov bola klasifikácia segmentu pomocou posuvného okna, kde sa každý pixel osobitne klasifikoval do tried pomocou segmentov obrázkov okolo neho. Tento prístup je ale výpočtovo veľmi neefektívny, pretože nepoužíva spoločné vlastnosti medzi prekrývajúcimi sa segmentami. Riešením je naopak plná konvolučná sieť (Fully Convolutional Networks (FCN)) na obrázku Obr. 2.6, ktorá popularizovala end-to-end architektúry CNN pre husté predpovede bez akýchkoľvek úplne prepojených vrstiev [14].



Obr. 2.5: Architektúra SegNet [13]

To umožňovalo generovanie segmentačných máp pre obrázky akejkoľvek veľkosti a bolo tiež oveľa rýchlejšie v porovnaní s prístupom ku klasifikácii segmentov. Túto paradigmu si osvojili takmer všetky nasledujúce prístupy k sémantickej segmentácii. Jedným problémom, ale stále zostávajú konvolúcie, ktoré pri tvorbe segmentácií pri pôvodnom rozlíšení obrazu budú veľmi veľké. Na riešenie tohto problému používa FCN podvzorkovanie a nadvzorkovanie [14].



Obr. 2.6: Fully Convolutional Network (FCN) [12]

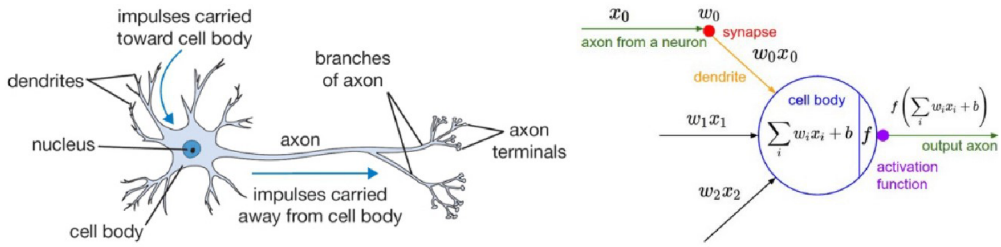
2.5 Segmentácia inštancií

Okrem sémantickej segmentácie sa používa jej vylepšenie, takzvaná segmentácia inštancií, ktorá segmentuje zvlášť aj rôzne inštancie tried. Čiže napríklad označí 5 automobilov 5 rôznymi farbami. Pri segmentácií inštancií je teda dôležitý rozlišovať nielen predmety, ale aj pri rovnakých predmetoch rozlíšiť ich hranice a vzájomné vzťahy. Jednou z konvolučných sietí, ktorá dokáže robiť takýto typ segmentácie je Mask R-CNN, ktorej príklad použitia je možné vidieť na obrázku Obr. 2.7. Tento proces vykonáva pridaním vetvy Faster R-CNN, ktorá produkuje binárnu masku a hovorí, či je alebo nie je daný pixel súčasťou objektu. Výstupom je teda matica obsahujúca 1 na všetkých miestach, kde pixel patrí k objektu a 0 pre pixel, ktorý nepatrí [14].

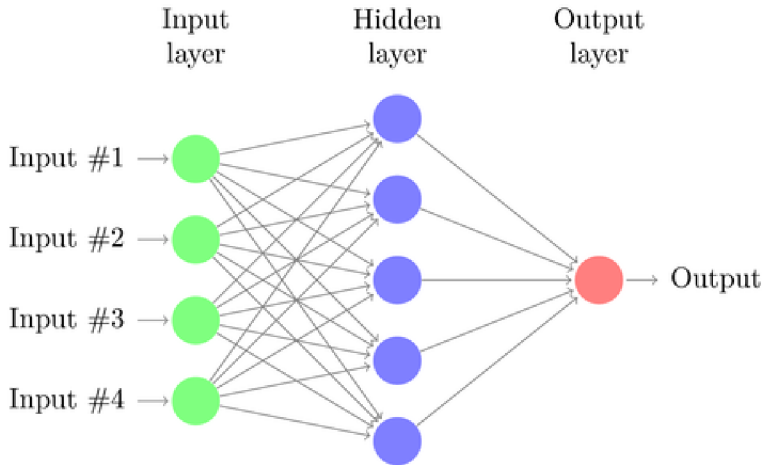
Obr. 2.7: Príklady segmentácie inštancií Mask R-CNN [14]

3 Neurónové siete (NN)

História neurónových sietí sa datuje od roku 1943 keď Warren McCulloch, mladý matematik a neurofyziológ, napísal prvú publikáciu na tému činnosti neurónov a namodeloval prvú neurónovú sieť z elektrických obvodov [15]. Neurónová sieť je vytvorená na model ľudského mozgu a taktiež sa skladá z neurónov. Funkcia neurónov v mozgu a v neurónovej sieti v počítači, si je veľmi podobná. V mozgu sa nachádza niekoľko miliárd neurónov a tieto neuróny sú spojené miliardami synapsií. Obrázok Obr. 3.1 zobrazuje podobnosť neurónu v ľudskom tele a matematický model neurónu, ktorý sa používa ako základ neurónovej siete. Matematický model neurónovej siete sa skladá z vrstiev neurónov, ktoré sa rozdeľujú na 3 hlavné: vstupná vrstva, skrytá vrstva/vrstvy a výstupná vrstva, ktorej počet neurónov býva zväčša rovný počtu tried klasifikácie. Prepojenie neurónov a ich vrstiev je zobrazené na obrázku Obr. 3.2 a toto prepojenie sa tiež nazýva váhy, ktorých hodnoty sa počas tréningu upravujú.



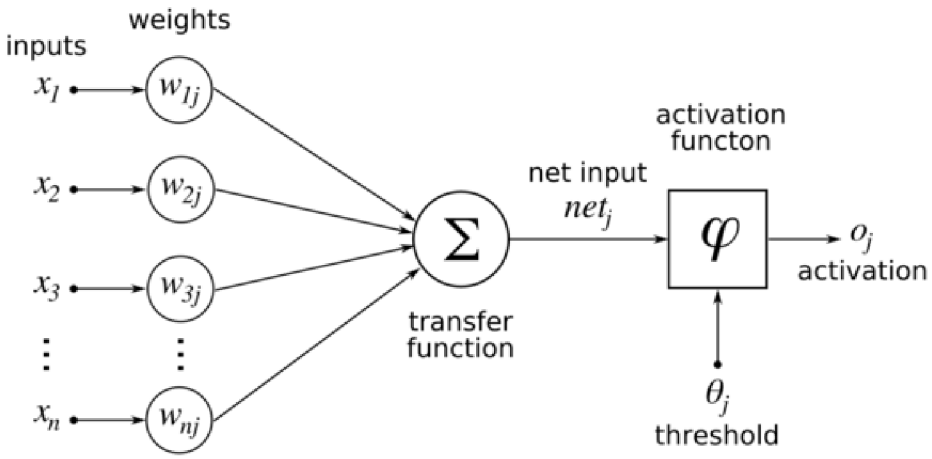
Obr. 3.1: Porovnanie ľudského neurónu a matematického modelu neurónu [16]



Obr. 3.2: Neurónová sieť s vstupnou, skrytou a výstupnou vrstvou [17]

3.1 Aktivačné funkcie

Neurónová sieť pozostáva z neurónov, do ktorých vstupujú numerické dáta. Každý neurón má váhu a vynásobením vstupného čísla s váhou sa získa výstup neurónu, ktorý sa prenesie do ďalšej vrstvy. Aktivačná funkcia je matematická "brána" medzi vstupom napájajúcim aktuálny neurón a jeho výstupom do ďalšej vrstvy. Príklad funkcie neurónovej siete je na Obr. 3.3.



Obr. 3.3: Architektúra neurónovej siete [18]

Tieto funkcie môžu byť rôzne od krokovej funkcie, ktorá zapína a vypína výstup neurónov v závislosti od prahovej hodnoty. Ak je prahová hodnota prekročená, výstup neurónu je v binárnej podobe a to v logickej 1, inak v logickej 0. Týmto sa snaží matematický model neurónu pripodobniť neurónu v ľudskom mozgu, ktorý taktiež buď "zapína" alebo zostáva v kľude. Rozhodnutie je založené na sčítaní vstupných váh, ktoré sú násobené vstupnými dátami. Alebo to môže byť transformácia, ktorá mapuje vstupné signály na výstupné signály, ktoré sú potrebné na fungovanie neurónovej siete. Neurónové siete čoraz častejšie využívajú nelineárne aktivačné funkcie, ktoré môžu sieti pomôcť naučiť sa zložité údaje a takmer akúkoľvek funkciu, ktorá presne rozpozná alebo definuje presné predpovede [30].

Používajú sa tri základné typy aktivačných funkcií:

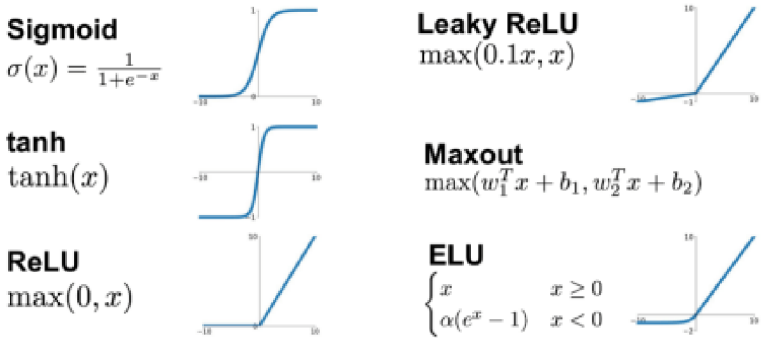
- Binárna skoková funkcia
- Lineárna aktivačná funkcia
- Nelineárna aktivačná funkcia

Najčastejšie sa používajú nelineárne aktivačné funkcie. Umožňujú modelu vytvárať zložité mapovania medzi vstupmi a výstupmi siete, ktoré sú nevyhnutné na učenie a modelovanie komplexných údajov, ako sú obrázky, video, zvuk a súbory údajov, ktoré sú nelineárne alebo majú vysokú dimenzionálnosť. Takmer každý pred-

staviteľný proces je možné predstaviť ako funkčný výpočet v neurónovej sieti za predpokladu, že aktivačná funkcia je nelineárna. Nelineárne funkcie riešia problémy lineárnych aktivačných funkcií:

1. Umožňujú spätné šírenie chyby (takzvaný backpropagation), podľa ktorej upravujú svoje váhy vďaka derivačnej funkcii.
2. Umožňujú "stacking" (stohovanie - dávanie dokopy) viacerých vrstiev neurónov na vytvorenie hlbokej neurónovej siete. Na osvojenie zložitých súborov údajov s vysokou úrovňou presnosti je potrebných viac skrytých vrstiev neurónov.

Príklady aktivačných funkcií sú na nasledujúcom obrázku Obr. 3.4, ich vysvetlenie a popis v nasledujúcom texte.



Obr. 3.4: Príklady aktivačných funkcií [19]

ReLU

Rectified linear unit (ReLU) je najpoužívanejšia aktivačná funkcia, vďaka svojej jednoduchosti v implementácii a zároveň vysokému výkonu v rôznych prediktívnych úlohách. Matematický zápis ReLU funkcie je nasledovný:

$$ReLU(x) = max\{0, x\}$$

Výhody:

- Výpočtovo efektívne - sieť je schopná rýchlo konvergovať (hľadať minimum)
- Nelineárnosť - hoci vyzerá ako lineárna funkcia, ReLU má derivačnú funkciu a umožňuje spätné šírenie chyby (backpropagation)

Nevýhody:

- Problém s "Dying ReLU" - keď sa vstupy blížia k nule alebo sú záporné, gradient funkcie sa stane nulovým, sieť nemôže vykonávať spätné šírenie a nemôže sa učiť [30].

LeakyReLU

Je to podobná funkcia ako ReLU s menším rozdielom, ktorý je jej výhodou. Matematický zápis aktivačnej funkcie je nasledovný:

$$LeakyReLU(x) = max\{0.1x, x\} \tag{3.1}$$

Výhody:

- Zabraňuje "dying ReLU" problému, táto variácia ReLU má malý pozitívny sklon v negatívnej oblasti, čím umožňuje spätné šírenie, a to aj pri negatívnych vstupných hodnotách.
- Zvyšné výhody má rovnaké ako ReLU

Nevýhody:

- Výsledky nie sú konzistentné - LeakyReLU neposkytuje konzistentné predpovede pre záporné vstupné hodnoty [20].

Sigmoid

Sigmoid je široko používaný ako aktivačná funkcia na výstupných jednotkách, keď chceme interpretovať výstupy ako pravdepodobnosť problémov s binárnou klasifikáciou (pre viac dimenzií sa zase používa softmax). Avšak sigmoid sa väčšinou nahrádza jednoduchším a ľahšie trénovateľným ReLU [20].

Výhody:

- Hladký gradient zabraňujúci "skokom" vo výstupných hodnotách.
- Výstupné hodnoty sú medzi 0 a 1 a normalizujú výstup každého neurónu.
- Jasné predpovede - pre X nad 2 alebo pod -2 má tendenciu privádzať hodnotu Y (predikciu) na okraj krivky, veľmi blízko k 1 alebo 0. To umožňuje jasné predikcie.

Nevýhody:

- Miznúci gradient - pre veľmi vysoké alebo veľmi nízke hodnoty X neexistuje takmer žiadna zmena predikcie, čo by spôsobilo problém s miznúcim gradientom. Môže to viesť k tomu, že sa sieť odmietne učiť ďalej alebo bude príliš pomalá na to, aby dosiahla presnú predpoveď.
- Výstupy nie sú vycentrované k nulu.
- Výpočtovo drahé [30].

3.2 Funkcia chyby - Loss function

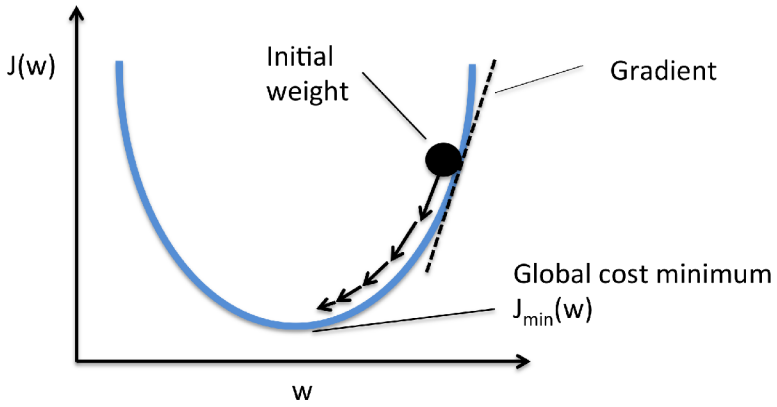
Metóda, ktorá kvantifikuje, ako je "dobrý" aktuálne netrénovaný model. Veľkosť chyby, ktorú funkcia predstavuje sa odvíja od veľkosti rozdielu predikovaného výstupu a skutočnou hodnotou, pravdou. Na základe tejto funkcie sa dokáže neurónová

sieť učiť. Od typu funkcie sa odvíja aj rozdiel chyby. Cieľom je nájsť parametre neurónovej siete také, aby chyba, čiže výstup tejto funkcie bol, čo najmenší, ako sa dá. Bohužiaľ, neexistuje univerzálne najlepšia funkcia, ktorá by perfektne sedela na každý problém [34]. Najznámejšia funkcia pre klasifikáciu sa nazýva Cross Entropy Loss (chyba krížovej entropie). Hodnota výstupnej chyby sa zvyšuje, ak sa hodnota pravdepodobnosti predpovedaného výstupu (p) líši od skutočnej hodnoty (y), ktorá má binárnu formu, podľa toho, či patrí alebo nepatrí k triede. Dôležitým parametrom tejto funkcie je tiež to, že silno trestá predikcie, ktoré majú vysoký predpoklad byť správne, ale sú nesprávne [21]. Vzorec na výpočet tejto chyby je nasledovný:

$$CrossEntropyLoss(x) = -(ylog(p) + (1-y)log(1-p)) \tag{3.2}$$

3.3 Spätná propagácia - Backpropagation

Spätné šírenie chyby je spôsob, ako sa neurónová sieť učí a to vďaka klesaniu gradientu. Je to iteračný proces, ktorého cieľom je znížiť funkciu chyby zmenami váh a tým dosiahnuť každou ďalšou iteráciou nižšiu chybu pri predikcii. Príklad klesania gradientu je na obrázku Obr. 3.5, kde je zobrazená pôvodná hodnota váhy a vďaka hodnotám z funkcie chyby sa gradient posúva nižšie veľkosťou kroku a smeruje ku globálnemu minimu. Veľkosť kroku je určená rýchlosťou učenia, ktoré sa nastavuje pri inicializácii modelu. Ak by bol tento krok veľký, môže prísť k "vyskočeniu z krivky von" a tým nenájdenie správneho minima. Naopak ak je rýchlosť učenia malá, tak model nemusí nikdy nájsť globálne minimum, ale uviazne v lokálnom minime mysliac si, že našiel globálne. Tento hyperparameter je možné vyladiť aj za behu trénovania modelu [22].



Obr. 3.5: Príklad klesania gradientu [22]

4 Konvolučné neurónové siete (CNN)

Konvolučná neurónová sieť (convolution neural network - CNN) je efektívny nástroj používaný v počítačovom videní na rozpoznávanie obrazu. Vznik konvolučných sietí siaha až do rokov 1962 [23]. Napriek tomu, sme o nich počuli viac až po roku 2010 kedy výpočtový výkon bol dostatočne veľký aby dokázal spracovať náročné konvolučné výpočty. V roku 2012 sa konvolučné vrstvy preslávili na každoročnej súťaži Imagenet Visual Recognition Challenge, kde ich metódy využili na stavbu známej Alexnet. Ich metóda úspešne porazila iné algoritmy a s veľkým náskokom vyhrala prvé miesto. Od tohto momentu sa konvolučné siete používajú v rôznych projektoch na rozpoznávanie obrazu, tváre, spracovanie textu, v robotike alebo aj autonómnych autách [23].

Vstupným parametrom konvolučnej siete môže byť obraz, zvuk alebo aj text. Používajú sa v nich nasledovné operácie:

1. Konvolúcia
2. Nelineárna aktivačná funkcia (napríklad ReLU)
3. Pooling
4. Klasifikácia

Tieto operácie sú základným stavebným pilierom takmer každej konvolučnej siete. K týmto operáciám sa taktiež pridávajú metódy ako dropout, padding alebo batch normalization, ktoré podrobne popíšem v nasledujúcom texte. Princíp funkcie konvolučných vrstiev je nasledovný. Ak by sme riešili úlohu klasifikácie, kde chceme určiť aký objekt sa nachádza na obrázku, najskôr sa niekoľkokrát zopakujú operácie 1. až 3., aby zo vstupného obrázku extrahovali potrebné obrazové príznaky, takzvané "features" alebo vlastnosti. Prvotné vrstvy extrakcie vlastností sa učia základné informácie o obrázku, ako hrany a rohy a hlbšie vrstvy sa učia tvary a farby. Vlastnosti sa ukladajú do konvolučných filtrov/máp. Tie sa následne na konci spoja do jednej spoločnej príznakovej mapy v rámci plne prepojených vrstiev a vykoná sa záverečná operácia, ktorou je klasifikácia. Ukážka sledu operácií je znázornená na obrázku Obr. 4.1.

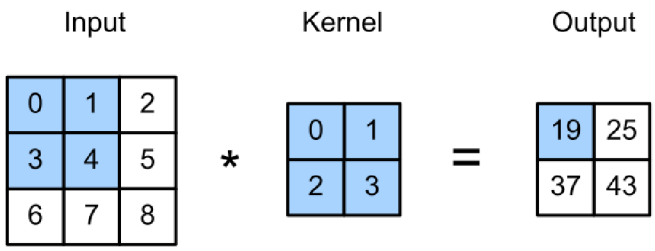
4.1 Konvolúcia

Konvolúcia je špecializovaný druh lineárnej operácie. Presne povedané, konvolučné vrstvy sú trochu nesprávnym pomenovaním, pretože operácia, ktorú vyjadrujú, sa presnejšie označuje ako vzájomná korelácia. V konvolučnej vrstve sa vstupné pole a korelačné jadro takzvaný kernel kombinujú, aby vytvorili výstupné pole prostredníctvom operácie vzájomnej korelácie. Konvolúcia využíva tri dôležité metódy, ktoré

Obr. 4.1: Architektúra konvolučnej siete [24]

pomáhajú zlepšovať systém strojového učenia: riedke interakcie, zdieľanie parametrov a ekvivariantné znázornenie. Konvolúcia navyše poskytuje prostriedky na prácu so vstupmi premenlivej veľkosti. Môžu preto byť konvolúcie aj v jednom, dvoch alebo aj troch rozmeroch [25].

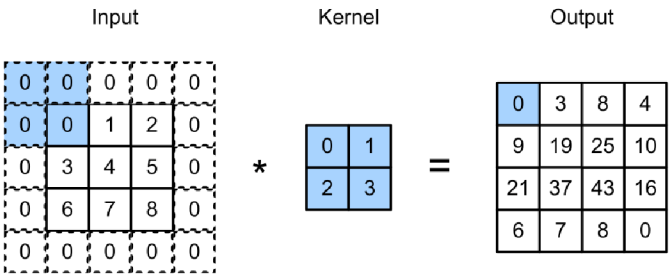
V mojej práci sa budem venovať konvolúciám obrázkov, čiže 2D konvolúciám. Vstupom do konvolučnej vrstvy bude obrázok o rozmeroch výška x šírka x hĺbka, kde pre obyčajný farebný RGB obrázok je podľa počtu farebných kanálov hĺbka 3. Prvou vrstvou je vstupná vrstva, ktorá môže, ale nemusí, mať pevne určené rozmery. Potom nasleduje niekoľko po sebe idúcich konvolučných vrstiev. Ich počet záleží od architektúry a potrieb programátora. Vrstvy sa skladajú z množiny filtrov, ktoré sa používajú na extrahovanie lokálnych obrazových príznakov prostredníctvom operácie konvolúcie. Nie je to nič iné len sčítanie hodnôt niekoľkých pixelov (podľa veľkosti filtra) vynásobených hodnotami filtra. Vypočíta sa tak nová hodnota pixelu na základe hodnôt jeho okolia. Na nasledujúcom obrázku je príklad konvolúcie vypočítanej nasledovne: 0x0 + 1x1 + 3x2 + 4x3 = 19, 1x0 + 2x1 + 4x2 + 5x3 = 25 a tak ďalej. Pri tomto príklade nie je aplikovaný padding, čiže rozšírenie a stride, čiže krok je 1 [24].



Obr. 4.2: Príklad funkcie konvolučnej siete [26]

4.2 Padding

Ako som spomenul vyššie, pri aplikovaní konvolučných vrstiev sa stáva, že konvolúcia je vykonávaná tak, že kernel prechádzajúci po obrázku z časti vytŕča von. Je to vtedy, ak sníma napríklad prvý pixel vľavo hore a kernel má veľkosť 3x3 a chce začať so svojim stredom v lavom hornom rohu obrázku. V tom prípade by určitá časť kernelu nemala žiadnu hodnotu pixelu na násobenie pre určité prvky. Pretože zvyčajne používame malé kernely, pre každú konvolúciu by sme mohli stratiť pár pixelov, ktoré by z dlhodobého pohľadu tvorby niekoľkých konvolučných vrstiev chýbali. Vtedy sa musí rozšíriť obvod obrázka o určité hodnoty. Zväčša je to o nuly a tým vzniká názov "zero padding", čiže rozšírenie obrázka po obvode o nulové hodnoty [27]. Na priloženom obrázku môžete vidieť rozšírenie o nuly a postup pri konvolúcii by bol nasledovný: $0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$.



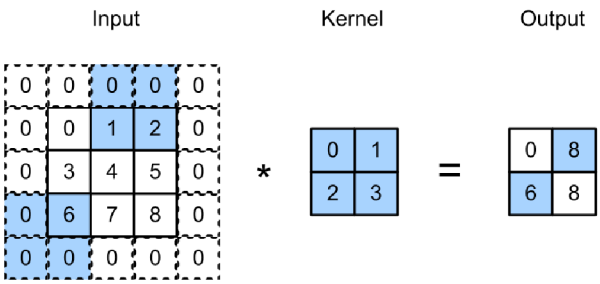
Obr. 4.3: Príklad funkcie padding [26]

4.3 Stride

Stride sa dá nazvať aj krok. Totižto pri výpočte krížovej korelácie začneme konvolučným oknom v ľavom hornom rohu vstupnej matice alebo tenzoru a potom ho posúvame cez všetky miesta doprava a dolu. V predošlom prípade kernel prechádzal vždy jeden pixel po druhom. Niekedy však, buď kvôli výpočtovej efektívnosti, alebo preto, že je potrebné podvzorkovanie, sa kernel posúva o viac ako jedno miesto doprava a dolu. Tento krok sa môže zväčšiť napríklad na 2 alebo 3. Ak sa zvolí 2, bude kernel preskakovať každú druhú hodnotu a tým vytvára nový filter o polovičnej veľkosti. Na nasledujúcom obrázku je ukázané, ako by to vyzeralo, ak by kernel z pôvodnej pozície prešiel o 2 kroky doprava pre výpočet pravej hornej hodnoty výstupu a 3 kroky dolu, pre výpočet hodnoty výstupu ľavo dolu [28].

Výpočet veľkosti výstupného obrázku sa vypočíta nasledovne:

- I - dĺžka hrany vstupného obrázku



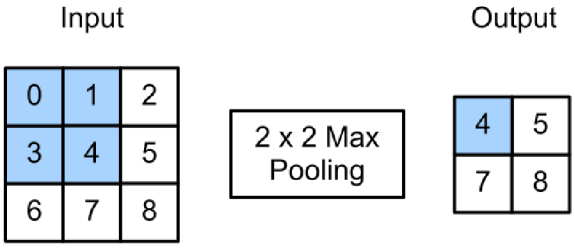
Obr. 4.4: Príklad funkcie stride [26]

- K - dĺžka hrany kernelu
- O - dĺžka hrany výstupného obrázku
- P - veľkosť paddingu
- S - hodnota stride (kroku)

$$O = \frac{I + 2P - K}{S} + 1 \tag{4.1}$$

4.4 Pooling

Funkcia pooling nahrádza výstup siete na určitom mieste štatistickými operáciami. Napríklad operácia max pooling vyberie maximálnu hodnotu v rámci hodnôt, ktoré jej prídu na vstup. Medzi ďalšie často používané funkcie poolingu patrí priemer hodnôt, norma L2, alebo vážený priemer založený na vzdialenosti od stredového pixelu. Tieto funkcie (najčastejšie max pooling) sa aplikujú pri tvorbe filtru, kedy je potrebné zmenšiť počet hodnôt a tým znížiť rozlíšenie. Pre príklad nižšie je zobrazené, ako by sa aplikovaním funkcie max pooling s hodnotou kernelu 2x2 vybrali z čísiel 0, 1, 3, 4 najväčšia 4 a z čísiel 1, 2, 4, 5 najväčšia 5 a takto ďalej [29].



Obr. 4.5: Príklad funkcie pooling [26]

4.5 Normalizácia

Trénovanie hlbokých neurónových sietí je relatívne náročná úloha. Získať konvergenciu neurónovej siete za rozumné množstvo času (alebo vôbec) môže byť naozaj ťažké. Jedným z riešení je využitie takzvanej normalizácie vstupných dát. Táto metóda patrí k novým populárnym technikám, ktorá dôsledne urýchľujú konvergenciu hlbokých sietí. Spolu s reziduálnymi blokmi (popísané v sekcii 4.8.1) dávková normalizácia (batch normalization) umožňuje bežne trénovať siete s viac ako 100 vrstvami [31].

4.5.1 Dávková normalizácia (Batch normalization)

Dávková normalizácia znižuje pravdepodobnosť, že natrénovaná neurónová sieť nebude vedieť správne predikovať na časti datasetu, ktorý by sa veľmi líšil od trénovacieho. Konkrétne ak by sme trénovali iba na obrázkoch čiernych mačiek a pokúsime sa použiť túto sieť na dáta s farebnými mačkami, je zrejmé, že nebude fungovať dobre. Trénovací a testovací dataset sa líši a ak sa algoritmus naučil mapovať X na Y zmenou X nebude algoritmus schopný správne mapovať výstupnú hodnotu Y. Aby sa zvýšila stabilita neurónovej siete, dávková normalizácia normalizuje výstup predchádzajúcej aktivačnej vrstvy odpočítaním priemeru dávky a vydelením smerodajnou odchýlkou dávky. Matematický vzorec je nasledovný:

$$BN(x) = \frac{x - \mu}{\sigma} \odot \gamma + \beta \tag{4.2}$$

Dva parametre μ a σ štatistiky pre dávku sú - priemer a smerodajná odchýlka. Zvyšné symboly ako γ a β sú trénovacie parametre zvyčajne nastavené na $\gamma = 1$ a $\beta = 0$. Vypočítava sa tiež agregovaný priemer a agregovaná štandardná odchýlka spolu so štatistikami dávok použitými v inferenčnom režime, keď šum v μ a σ vznikajúci pri odhadovaní každej z mini-dávok nie je žiaduci po natrénovaní modelu [32].

4.6 Dropout

Technika dropout sa dá preložiť ako vypadávanie alebo odpojenie. Funguje na princípe náhodného odpojenia neurónov medzi vrstvami neurónovej siete. Cieľom je dosiahnuť väčšej presnosti keďže menší počet neurónov sa musí naučiť správne rozpoznávať žiadané parametre [33].

4.7 Optimalizačný algoritmus

Optimalizátor v neurónovej sieti je algoritmická implementácia, ktorá uľahčuje proces gradientného klesania, čiže hľadania globálneho minima v neurónovej sieti. Získa sa minimalizáciou hodnôt chyby poskytovaných prostredníctvom funkcie chyby (loss function). Na zníženie strát je dôležité, aby sa hodnoty váh v sieti vybrali správne. Preto sa neodporúča začať trénovať neurónovú sieť s náhodnými hodnotami, ale použiť už predtrénované vrstvy inej siete [35].

4.8 Typy konvolučných neurónových sietí

Ako som už v sekcii klasifikácie obrazu spomenul, existuje mnoho známych konvolučných neurónových sietí, ktoré dosahujú vysokú presnosť pri úlohách detekcie alebo klasifikácie. Medzi také patria AlexNet, VGGNet, GoogLeNet alebo Resnet. Konvolučná sieť Resnet sa vyznačuje jedinečnými reziduálnymi blokmi, ktoré sú jej výhodou oproti bežným konvolučným sieťam. V mojej práci budem používať CNN typu Resnet a preto ju bližšie popíšem v nasledujúcej kapitole.

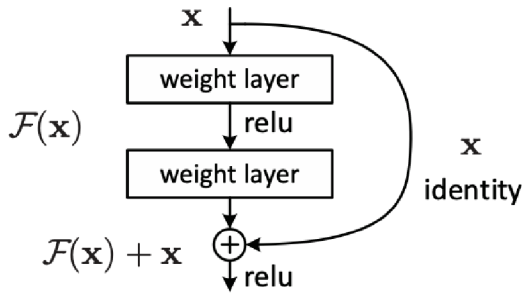
4.8.1 Resnet

Vedci zistili, že pokiaľ ide o konvolučné neurónové siete, má zmysel tvrdenie, že "čím sú hlbšie, tým lepšie", to znamená, čím viac vrstiev, tým viac parametrov. Dáva to zmysel, pretože modely by mali byť kvalitnejšie (ich flexibilita prispôsobenia sa každému priestoru zvyšuje, pretože majú väčší priestor na preskúmanie a natrénovanie parametrov). Bolo však pozorované, že po určitej hĺbke sa výkon zhoršuje. Toto bol aj jeden z problémov známej siete VGG. Nebolo možné tvoriť už tak hlbokú CNN, pretože sa začala strácať schopnosť zovšeobecňovania, generalizácie. Tým vznikol nápad tvorby novej Resnet siete v roku 2015. Pretože neurónové siete sú dobrými aproximátormi funkcií, mali by byť schopné ľahko vyriešiť funkciu identifikácie, kde sa výstupom funkcie stane samotný vstup [36].

$$f(x) = x \tag{4.3}$$

Ak postupujeme podľa rovnakého princípu, tak ak obídeme vstup do prvej vrstvy modelu, aby bol výstupom poslednej vrstvy modelu, sieť by mala byť schopná predpovedať akúkoľvek funkciu, ktorú sa predtým naučila aj so vstupom, ktorý sa k tomu pridá.

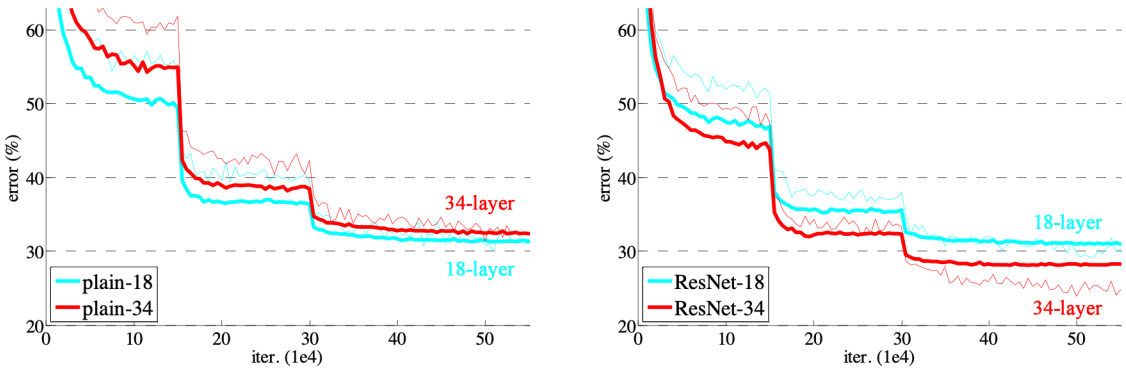
$$f(x) + x = h(x) \tag{4.4}$$



Obr. 4.6: Stavebný reziduálny blok ResNet siete [36]

Jedným z problémov, ktoré Resnet rieši, je známy miznúci gradient. Je to v prípade, keď je sieť príliš hlboká, gradienty, od ktorých sa počíta funkcia chyby, sa po niekoľkých prepočtoch jednoducho zmenšia na nulu. Tento výsledok tým pádom neovplyvní hodnoty váh a tým pádom sa hlboká sieť vôbec neučí. Pomocou Resnet môžu gradienty prúdiť priamo cez spojenia, cez ktoré preskočia hlbšie o niekoľko vrstiev. Príklad reziduálneho bloku je vidieť zobrazený na obrázku Obr. 4.6 a spojením takýchto blokov vznikne sieť Resnet. Oproti klasickej sieti má jediný rozdiel a to reziduálne bloky. Iba vďaka týmto prepojeniam, dokáže byť sieť kvalitnejšia a dosahovať menšiu chybovosť.

Ako som vyššie napísal, prehlbovaním, pridávaním ďalších vrstiev sa od určitej hĺbky začala chyba siete zväčšovať. Čiže hlbšia sieť mala väčšiu chybu. Zapojením reziduálnych blokov, ale došlo k opaku a hlbšia sieť s viacerými reziduálnymi blokmi mala menšiu chybu, ako Resnet sieť s menším počtom blokov. Na obrázku Obr. 4.7 je vidieť porovnanie konvolučnej siete s 18 a 34 vrstvami a taktiež Resnet siete s rovnakým počtom vrstiev. Dôležité je všimnúť si zmenu trénovacej chyby pri väčšom počte vrstiev pri oboch konvolučných sieťach.



Obr. 4.7: Porovnanie konvolučnej a Resnet siete a ich trénovacej chyby [36]

Siete Resnet môžu mať rôznu veľkosť a počet parametrov s ktorými pracuje, v závislosti od toho, aké veľké sú jednotlivé vrstvy modelu, koľko vrstiev má a zároveň každá z vrstiev má rovnaký vzor. Vykonávajú konvolúciu 3x3 s pevným rozmerom mapy funkcií (F) [64, 128, 256, 512], pričom obchádzajú vstup každé dve konvolúcie. Rozmery šírky a výšky zostávajú konštantné počas celej vrstvy. Redukcia medzi vrstvami je dosiahnutá zvýšením počtu krokov (stride) od 1 do 2 pri prvej konvolúcii každej vrstvy (tento prístup sa používa namiesto operácie združovania (pooling), ktorý sa aplikuje pri iných sieťach s cieľom podvzorkovať, zmenšiť počet) [36].

4.9 Evaluácia neurónových sietí

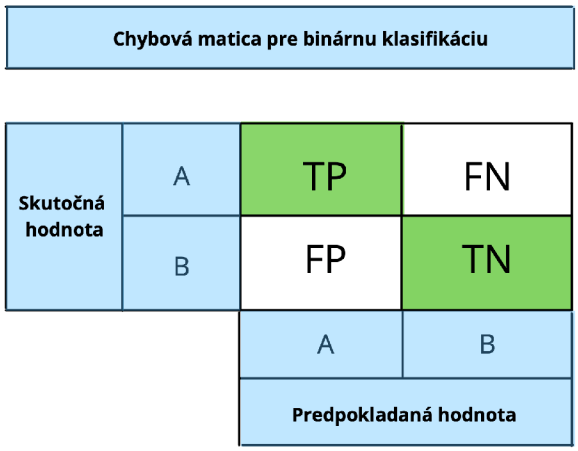
Vyhodnotiť úspešnosť a kvalitu neurónovej siete sa dá niekoľkými spôsobmi. V mojej práci sa budem hlavne venovať problému klasifikácie dát do tried. Klasifikácia je metóda tréningu aplikovaná pri učení s učiteľom, pri ktorom je cieľová, žiadaná premenná diskrétna (alebo kategorická). Vyhodnotenie modelu je rovnako dôležité, ako jeho zostavenie. Vytvárame modely, ktoré fungujú na nových, doteraz nevidených dátach. Preto je na vytvorenie robustného modelu potrebné dôkladné a všestranné vyhodnotenie. Pokiaľ ide o modely klasifikácie, proces hodnotenia môže byť trochu komplikovaný a preto sa úspešnosť modelu dá vyhodnotiť rôznymi metódami:

Hlavné prístupy sú nasledovné:

- Chybová matica / matica zámien (Confusion matrix)
- Presnosť klasifikácie (Classification accuracy)
- Presnosť a vyťaženie (Precision, Recall)
- Skóre F1
- Citlivosť a špecificita
- ROC krivka a AUC
- IoU (Intersection over Union)

4.9.1 Chybová matica

Chybová matica (Confusion matrix) nie je samá o sebe metrikou na vyhodnotenie modelu, poskytuje však vhľad do predpovedí. Je dôležité správne pochopiť a správne interpretovať dáta z matice pre kvalitné interpretovanie úspešnosti modelu. Výsledky matice idú hlbšie ako presnosť klasifikácie tým, že ukazuje aj správne a nesprávne (pravdivé alebo nepravdivé) predpovede pre každú triedu. V prípade úlohy binárnej klasifikácie je matica o veľkosti 2x2. Ak existujú tri rôzne triedy, je to matica 3x3 a tak ďalej. Príklad matice je na obrázku Obr. 4.8.



Obr. 4.8: Chybová matica (confusion matrix)

Matica nám ponúka 4 dôležité definície. Predpokladajme, že trieda A je pozitívna trieda a trieda B je negatívna trieda. Kľúčové pojmy matice sú nasledovné:

- **Skutočne pozitívne (True Positive - TP):** Predikcia pozitívnej triedy ako pozitívnej (správne)
- **Falošne pozitívne (False Positive - FP):** Predpovedanie negatívnej triedy ako pozitívnej (nesprávne)
- **Falošne negatívne (False Negative - FN):** Predpovedanie pozitívnej triedy ako negatívnej (nesprávne)
- **Skutočne negatívne (True Negative - TN):** Predpovedanie negatívnej triedy ako negatívnej (správne)

Požadovaným výsledkom je, aby predpoveď a skutočná trieda boli rovnaké. Pre jednoduchšie pochopenie existuje trik: prvé slovo označuje, či je predpoveď správna a druhé slovo označuje, čo predpovedá model. Falošne pozitívna predikcia sa tiež označuje ako chyba typu 1 a falošne negatívna predikcia ako chyba typu 2 [40]. Cieľom je spraviť model, ktorý bude mať čo najväčšie hodnoty TP a FP. Ako pomôcku prikladám nasledovnú tabuľku na obrázku Obr. 4.9.

	Skutočnosť	Predpoklad	Hodnotenie
TP	Pozitívne	Pozitívne	Dobré
FP	Negatívne	Pozitívne	Nie dobré
FN	Pozitívne	Negatívne	Nie dobré
TN	Negatívne	Negatívne	Dobré

Obr. 4.9: Klasifikácia TP, FP, FN, TN

4.9.2 Presnosť klasifikácie

Presnosť klasifikácie vyhodnocuje, koľko predpovedí je správnych. V niektorých prípadoch vyhodnocuje, aký dobrý je model, existujú však prípady, v ktorých presnosť jednoducho nestačí. Napríklad presnosť 93% pri klasifikácii 2 tried môže znamenať, že sme správne predpovedali 93 zo 100 vzoriek. Toto konštatovanie sa zdá prijateľné, ak nepoznáme podrobnosti úlohy. Ak sme v zadaní mali dataset, ktorý obsahoval 7 obrázkov typu B a 93 obrázkov typu A a model povedal, že všetky obrázky sú typu A, tak jeho presnosť je síce 93%, ale nie je to pravdepodobne kvalitný model. Ak by išlo o dôležité rozhodnutie pri klasifikácii, napríklad detekcií nádoru zo snímky v nemocnici, tak si nemôžeme dovoliť všetkých 100 pacientov poslať domov s rozhodnutím, že sú zdraví, aj keď by len 2 z nich mali nádor a my sme prehlásili 98% presnosť modelu [40]. Spôsob výpočtu presnosti je nasledovný:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.5}$$

4.9.3 Precision, Recall

Pomocou Chybovej matice (Confusion matrix) je možný výpočet presnosti (precision) a vyťaženia (recall). Tieto metriky posúvajú presnosť klasifikácie o krok ďalej a umožňujú nám lepšie porozumieť hodnoteniu modelu. Závisí vždy od úlohy nášho cieľa to, ktorú metriku vyberieme.

Presnosť meria, aký dobrý je náš model, keď je predpoveď pozitívna. Inak povedané, je to počet skutočne pozitívnych predpovedí vydelený súčtom skutočne aj falošne pozitívnych. Čiže koľko pozitívnych predpovedí je pravdivých.

$$precision = \frac{TP}{TP + FP} \tag{4.6}$$

Recall určuje ako úspešný je náš model v správnom predpovedaní pozitívnych tried.

$$recall = \frac{TP}{TP + FN} \tag{4.7}$$

Nie je možné maximalizovať precision aj recall zároveň, pretože medzi nimi existuje priamy vzťah. Zvyšujúca sa precision znižuje recall a naopak. Môžeme sa zamerať na maximalizáciu precision alebo recall v závislosti od úlohy. V prípade modelu detekcie spamu v e-maile sa snažíme maximalizovať precision, pretože chceme byť správni, keď je e-mail detekovaný ako spam. Normálny e-mail nechceme označiť ako spam (ako falošne pozitívny). Na druhej strane pre úlohu detekcie nádoru musíme maximalizovať recall, pretože chceme čo najviac detekovať pozitívne triedy. Existuje

ďalšia miera, ktorá kombinuje precision a recall do jedného čísla, a to je skóre F1 [40].

4.9.4 Skóre F1

Skóre F1 je vážený priemer precision a recall. Je užitočnejším meradlom ako presnosť pri problémoch, ak nie je rovnomerne vyvážené zastúpenie dát v triedach datasetu. Najlepšia hodnota skóre F1 je 1 a najhoršia 0 [40].

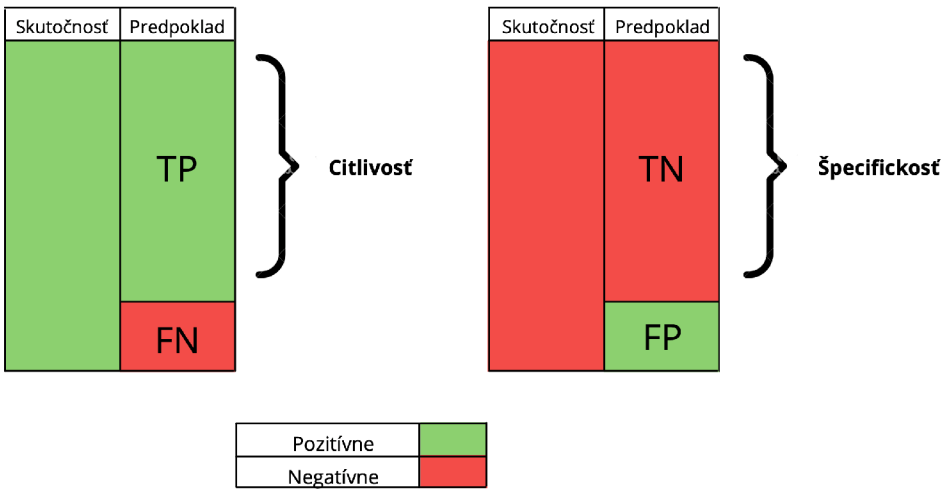
$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.8}$$

4.9.5 Citlivosť a špecifickosť

Citlivosť, známa tiež ako skutočná pozitívna miera (True positive rate - TPR), je rovnaká ako vyťaženie (recall). Preto meria podiel pozitívnej triedy, ktorý je správne predpovedaný ako pozitívny. Špecifickosť je podobná citlivosti, ale zameraná na negatívnu triedu (True negative rate - TNR). Meria podiel zápornej triedy, ktorý je správne predpovedaný, ako záporný [40]. Príklad vyhodnocovania citlivosti a špecifickosti je na obrázku Obr. 4.10.

$$TPR(citlivost) = \frac{TP}{TP + FN} \tag{4.9}$$

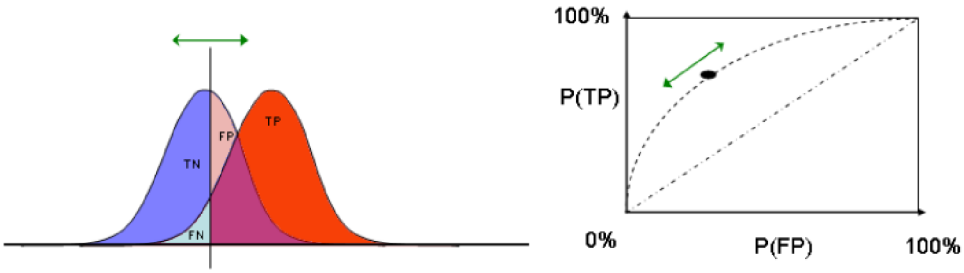
$$TNR(specifickost) = \frac{TN}{TN + FP} \tag{4.10}$$



Obr. 4.10: Vyhodnocovanie citlivosti a špecifickosť

4.9.6 ROC krivka a AUC

ROC krivka je nástroj pre hodnotenie a optimalizáciu klasifikačného modelu, ktorý ukazuje vzťah medzi citlivosťou a špecifickosťou pre dané prípustné hodnoty. Z ROC krivky tiež plynie AOC (Area under curve - plocha pod krivkou). Os Y krivky ROC je skutočná pozitívna miera - TPR (citlivosť) a os X krivky ROC je falošne pozitívna miera - FPR (1 - špecifickosť). Bod na ROC krivke odpovedá hodnote deliaceho kritéria. Ak by sme na grafe na ľavej strane obrázku Obr. 4.11 posunuli deliace kritérium doľava, tak by nám mierne vzrástol podiel TP a výrazne vzrástol podiel FP. Bod na ROC krivke zobrazenej na obrázku Obr. 4.11 vpravo by sa posunul doprava hore. Cieľom je zvyšovať TPR zatiaľ, čo FPR držať na čo najnižších číslach. Z krivky je vidieť, že zvyšovaním TPR automaticky narastá aj FPR. Je preto vždy dôležité si určiť, koľko FP je možné tolerovať. Namiesto hľadania ideálneho pomeru na ROC krivke, existuje AOC. Táto plocha pod krivkou je medzi bodmi [0, 0] a [1, 1] a jej hodnota udáva výkon a kvalitu modelu pri všetkých deliacich hodnotách. Najlepšia hodnota AUC je 1 a symbolizuje dokonalý klasifikátor. Hodnota AUC 0.5 predstavuje náhodnú presnosť modelu pri klasifikácii [40].



Obr. 4.11: Graf s dieliacim kritériom a ROC krivka [41]

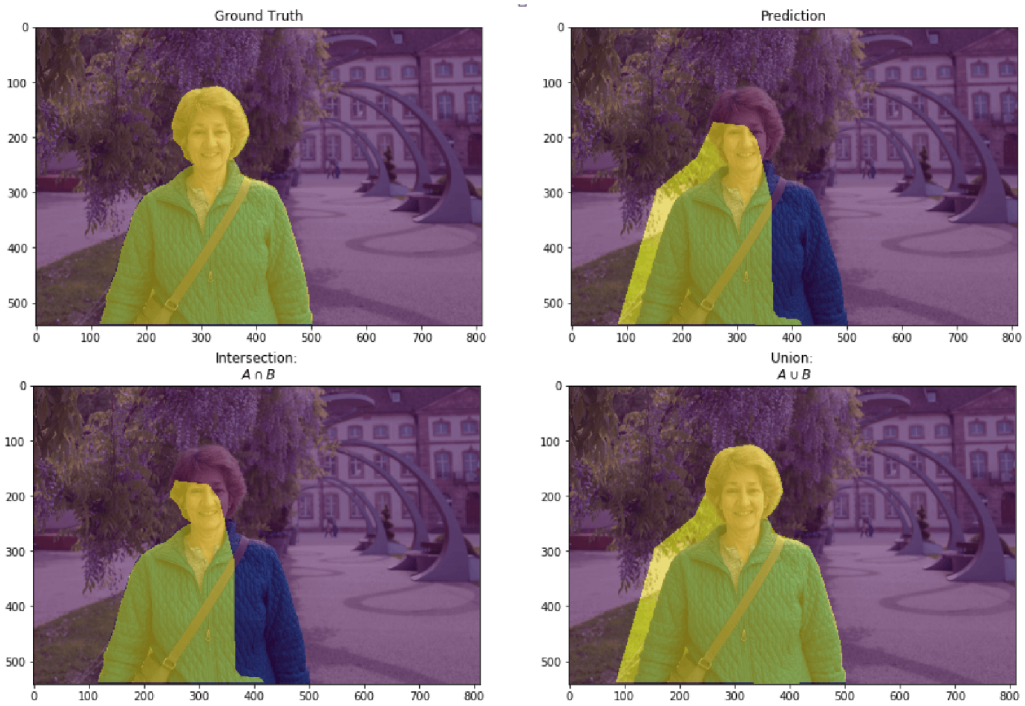
4.9.7 IoU

IoU je skratka pre Intersection over Union, čiže priesečník nad zjednotením. Táto metrika sa taktiež nazýva Jaccard index a je to jedna z hlavných metrík pri vyhodnocovaní úspešnosti prekrytia predikovanej masky a skutočnej hodnoty. IoU meria počet pixelov spoločných medzi predikovanou a skutočnou maskou a delí všetkými pixelmi oboch masiek.

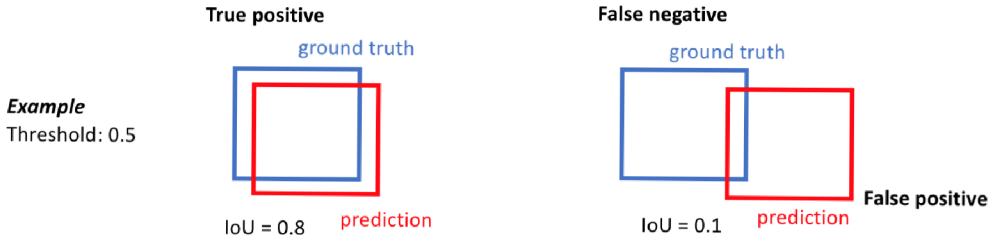
$$IoU = \frac{target \cap prediction}{target \cup prediction} \tag{4.11}$$

$$IoU = \frac{TP}{TP + FP + FN} \tag{4.12}$$

Prienik (target ∩ prediction) pozostáva z pixelov nájdených v predikovanom a skutočnom obrázku na rovnakom mieste. Je to logický AND. Zjednotenie (target ∪ prediction) pozostáva z pixelov nájdených v predikcii a aj skutočnom obrázku, čiže logický OR. Príklad funkcie IoU je znázornený na obrázku Obr. 4.12. Takto vypočítané IoU sa následne môže vyhodnocovať v rôznych porovnaniach. Platí, že IoU väčšie ako 0.5 (50%) sa akceptuje, ako dobrá predikcia.



Obr. 4.12: Príklad evaluácie obrázku pomocou metódy IoU [42]



Obr. 4.13: Príklad evaluácie obrázku pomocou metódy IoU s prahom 0.5 [42]

5 Self-Supervised Learning

V moderných algoritmoch hĺbkového učenia je závislosť na manuálnej anotácii neoznačených údajov jedným z hlavných obmedzení. Aby sme natrénovali dobrý model, zvyčajne musíme pripraviť obrovské množstvo označených dát s triedami alebo anotáciami. V prípade malého počtu tried a údajov môžeme použiť verejne dostupný predtrénovaný model a ňom doladiť niekoľko posledných vrstiev s našimi údajmi. V realite však často môžeme čeliť problému, keď je náš dataset relatívne veľký (priemyselné produkty, produkty v obchode, ľudské tváre...) a model by sa ťažko učil iba pomocou niekoľkých trénovateľných vrstiev. Okrem toho je veľmi veľké množstvo neoznačených obrázkov a celkovo dát na internete. Vytvoriť označenia alebo anotácie pre všetky je takmer nemožné a aj drahé (z časového aj finančného hľadiska, ktoré som popisoval v úvode mojej práce), ale naopak nepoužívať ich je škoda, lebo tieto dáta sú aj veľmi rýchlo generované napríklad na sociálnych sieťach [43].

Ak by sme chceli využiť toto oveľa väčšie množstvo neoznačených dát, jedným zo spôsobov je správne nastavenie učebných cieľov tak, aby sme získali dohľad nad samotnými údajmi. Úloha s vlastným učiteľom (self-supervised) sa skladá z dvoch častí: pretext a downstream task a patria do takzvanej self-supervised learning pipeline. Pomocou metód self-supervised učenia je možné dosiahnuť relatívne podobnej presnosti, ako učením s učiteľom a to s použitím menšieho počtu dát s označením tried/labelov. Práve tejto úlohe sa bude moja práca venovať.

5.1 Self-supervised learning pipeline

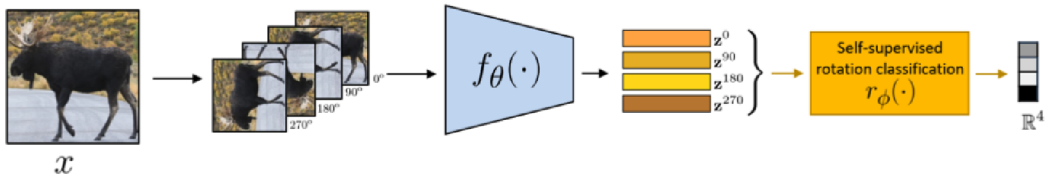
V self-supervised oblasti sa používa množstvo rôznych metód, ktoré nižšie podrobne popíšem. Niektoré sú jednoduchšie a niektoré naopak aj výpočtovo zložitejšie. Zároveň platí, že zložitejšie metódy, hlavne z oblasti kontrastívneho učenia dosahujú vyššiu úspešnosť a presnejšie výsledky v porovnaní s jednoduchšími metódami a zároveň dosahujú skoro podobné výsledky ako metódy učenia s učiteľom, čiže supervised learning. V tejto kapitole sa pokúsim v jednoduchosti vysvetliť, ako celý princíp self-supervised metód funguje v spojení s trénovaním neurónovej siete, použití pretext a následne downstream úlohy. Po použití self-supervised trénovania je bežné následne použiť dotrénovanie poslednej vrstvy (posledných vrstiev) spôsobom učenia s učiteľom a tým zvýšiť presnosť a kvalitu výsledného modelu.

Ako už bolo vyššie napísané, celý proces je možné rozdeliť do 2 fáz:

1. Pretext úloha - Trénovanie neurónovej siete na pretext úlohe
2. Downstream úloha + fine-tuning - Trénovanie modelu naučeného na reprezentáciách na novej úlohe s malým množstvom anotovaných vstupov

5.1.1 Pretext úloha

Pretext úloha nás vedie k funkcii chyby trénovania s učiteľom (supervised loss function). Zvyčajne sa však nestaráme o finálny výsledok tejto pretext úlohy. Skôr nás zaujíma naučená reprezentácia s očakávaním, že táto reprezentácia môže mať dobré sémantické alebo štrukturálne významy a môže byť prospešná pre rôzne typy druhej, čiže downstream úlohy. Napríklad môžeme náhodne otáčať obrázky a trénovať model tak, aby predpovedal, ako sa otáča každý vstupný obrázok, o koľko stupňov. Úloha predikcie rotácie je použitá na naučenie modelu zovšeobecňovať a naučiť sa vlastnosti a reprezentácie v obrázkoch bez anotácií. Očakávame však, že sa model naučí kvalitné latentné premenné pre úlohy v reálnom svete, ako napríklad zostavenie klasifikátora rozpoznávania objektov s veľmi malým počtom označených obrázkov/vzoriek. Pre príklad si môžeme zobrať pretext úlohu rotácie. Na vstupný obrázok použijeme rotáciu o 0, 90, 180, alebo 270 stupňov a takto orotovaný obrázok použijeme ako vstup do konvolučnej siete, ktorá sa naučí vhodné reprezentácie/vlastnosti o vstupnom datasete. Úloha trénovania s učiteľom potom je označenie triedy počet rotácií a vstup je orotovaný obrázok.



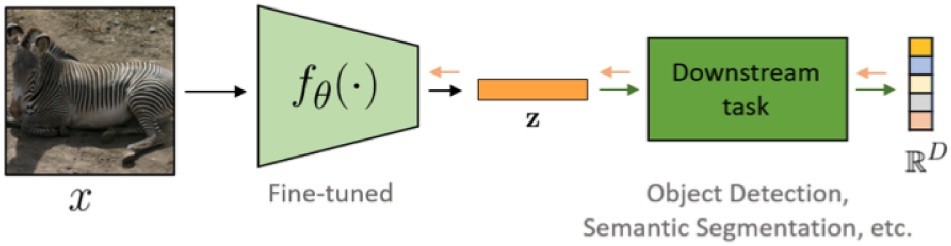
Obr. 5.1: Pretext task Self-supervised pipeline [45]

5.1.2 Downstream úloha

Druhá úloha, downstream task je aplikovanie naučených reprezentácií pre úlohy učenia s učiteľom (supervised) už na dátach, ktoré chceme klasifikovať do skutočných tried. Znamená to, že naučené všeobecné reprezentácie doladíme už s datasetom, ktorý obsahuje anotácie alebo triedy [44].

V druhej fáze trénovania modelu sa deje takzvaná downstream úloha. Ide o trénovanie klasifikátoru naučeného na reprezentáciách z pretext úlohy na novej úlohe s malým množstvom anotovaných vstupov. Inak povedané, je to aplikovanie naučených reprezentácií pre úlohy trénovania s učiteľom (supervised). Znamená to, že naučené všeobecné reprezentácie doladíme už s datasetom, ktorý obsahuje anotácie alebo triedy a následne doladíme parametre tak, aby model vedel vhodne generalizovať a presne identifikovať požadované dáta. Tento prístup sa tiež nazýva fine-tuning.

V jednoduchosti by išlo o postup, kedy najprv použijeme veľké množstvo dát na pretext úlohe, kde sa model učí predikovať správne otočenie obrázku a následne tento model, ktorý videl veľký počet vstupných dát z rovnakého datasetu (ale bez označenia tried) dotrénujeme pridaním jednej alebo dvoch konvolučných vrstiev na jeho koniec. Tieto posledné vrstvy už trénujeme učením s učiteľom a menším počtom dát ako v pretext úlohe. Počas tohto dotrénovania sa už nemenia váhy na predtrénovanom modele z pretext úlohy, váhy sú buď zamrazené alebo extrahované. Cieľom je dosiahnuť presnosť modelu konkurujúcu učeniu s učiteľom s veľkým množstvom dát s triedami.



Obr. 5.2: Downstream task Self-supervised pipeline [45]

Týmto spôsobom je možné dosiahnuť presný a kvalitný model, ktorý bude natrénovaný len s malým objemom anotovaných dát. Na to, aby to bolo úspešné, existujú rôzne metódy s množstvom rôznych aplikácií. V nasledujúcom texte sa ich pokúsim všetky popísať.

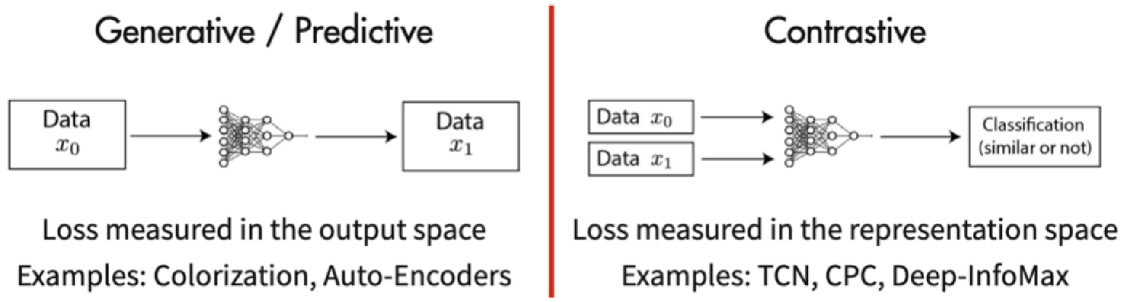
5.2 Metódy self-supervised

Self-supervised učenie sa dá aplikovať v rôznych metódach. Existujú prakticky 2 hlavné skupiny, do ktorých by sa self-supervised metódy dali zhrnúť:

Generatívne alebo prediktívne metódy používajú zväčša upravený vstupný obrázok na naučenie sa vlastností o datasete rôznymi spôsobmi (rotácia, zmena farieb, dopĺňanie farieb alebo dopĺňanie častí obrazu). Kontrastívne (contrastive) metódy, ako už z názvu vyplýva, sa učia reprezentácie na základe kontrastívnych pozitívnych a negatívnych príkladov.

Tie hlavné metódy spadajú do nasledovných skupín:

- Rekonštrukcia (reconstruction)
- Uvažovanie zdravého rozumu (common sense reasoning)
- Zhlukovanie (clustering)
- Kontrastívne učenie (contrastive learning)



Obr. 5.3: Základné rozdelenie SSL metód [46]

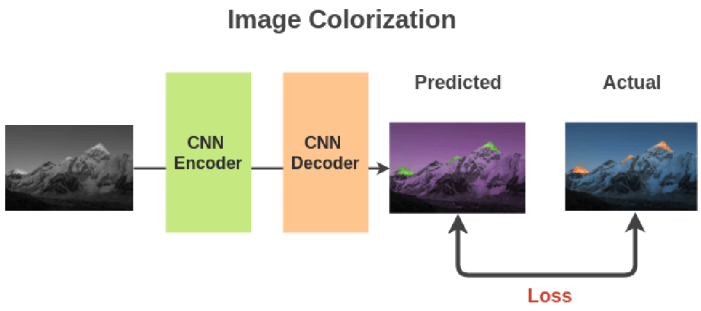
Každá z týchto skupín sa skladá z podskupín alebo konkrétnych metód, ktoré v tejto kapitole podrobne popíšem a vysvetlím. Metódy sa líšia prístupom alebo zložitosťou. Každá metóda, ale ponúka jedinečný prístup trénovania pretext úlohy, vďaka ktorej sa model naučí užitočné vlastnosti o vstupných dátach bez toho, aby obsahovali triedy, alebo ich obsahovali menší počet. Následne sa takýto model upraví a v downstream procese použijú na klasifikáciu obrazu, detekciu objektu alebo sémantickú segmentáciu s malým datasetom obsahujúcim triedu/label.

5.3 Rekonštrukcia

V tejto skupine sa používa ako pretext úloha predpovedanie farebného obrazu zo šedotónového obrazu (Image Colorization - kolorovanie obrazu), predpovedanie obrazu s vysokým rozlíšením z verzie s nízkym rozlíšením (Image Super-resolution - super rozlíšenie obrazu), odstránenie časti obrazu a následným dopĺňaním chýbajúcej časti obrazu (Image Inpainting - maľovanie obrazu) alebo zmeny určitého farebného spektra a následné trénovanie enkodéru na doplnenie farieb a naopak (Cross-channel Prediction - krížna-kanálová predikcia).

5.3.1 Kolorovanie obrazu - image colorization

Pretext úloha v podobe kolorovania obrazu, teda dopĺňania jeho farieb, je relatívne jednoduchá a aj ľahko uskutočniteľná z pohľadu na dostupnosť potrebných dát. Na internete sa nachádza veľké množstvo farebných obrázkov a taktiež je možné si rýchlo takýto dataset vytvoriť. Následne z farebného obrázku sa jednoducho jednou funkciou a aplikovaním filtru vytvorí šedotónový obraz. Pri dopĺňaní chýbajúcich farieb sa používa princíp architektúry enkóder-dekóder založenej na plne prepojených konvolučných sieťach a napríklad rátaní chyby typu L2 medzi predpovedanou farbou a aktuálnou farbou vo farebných obrázkoch [48].



Obr. 5.4: Princíp SSL metódy kolorovania obrazu [47]

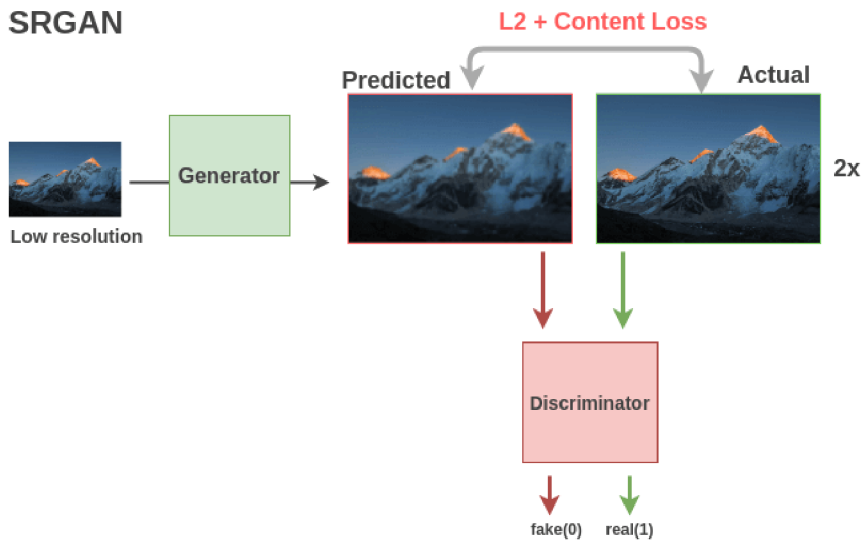
Na vyriešenie tejto úlohy musí model spoznať rôzne objekty nachádzajúce sa na obrázku a pochopiť v akých farbách bývajú vykreslené, aby následne mohol tieto časti namaľovať rovnakou farbou do šedotónového obrázku a vytvoriť tým farebný obraz. Získané vedomosti z dokreslenia farieb sú teda užitočné pre úlohy nadväzujúce v downstream úlohe.

Obr. 5.5: Príklad použitia SSL metódy kolorovania obrazu [49]

5.3.2 Super rozlíšenie obrazu - image super resolution

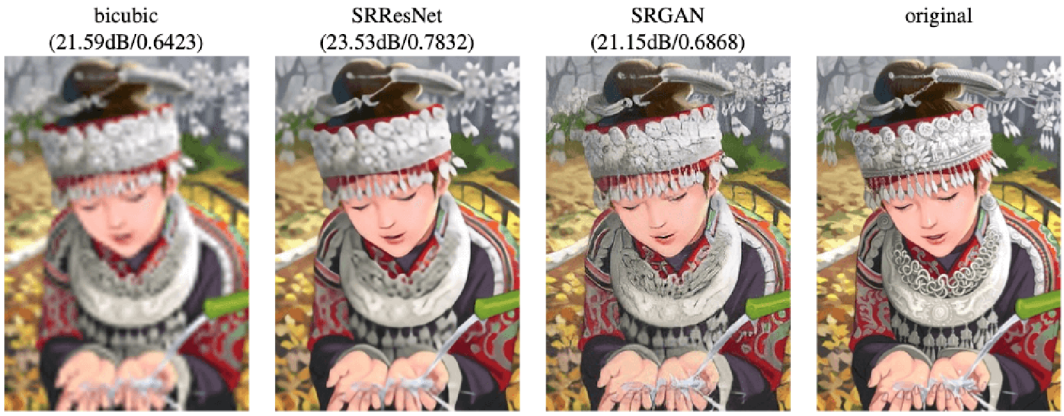
Ďalšou stále relatívne jednoduchou a ľahko aplikovateľnou metódou z pohľadu dostupnosti dát je možnosť upraviť rozlíšenie už existujúceho datasetu a následne ho použiť ako pretext úlohu na trénovanie modelu. Pre túto úlohu sú populárne modely založené na princípe GAN (generative adversarial network), ako napríklad SRGAN.

Generátor sníma obraz s nízkym rozlíšením a použitím plne prepojenej konvolučnej siete vytvorí nový obraz s vysokým rozlíšením. Skutočné a generované obrázky sa porovnávajú napríklad pomocou mocniny priemernej chyby a straty obsahu, aby sa napodobnilo porovnanie kvality, aké by vykonal človek. Diskriminátor s binárnou



Obr. 5.6: Princíp SSL metódy kolorovania obrazu [47]

klasifikáciou zoberie obraz a klasifikuje, či ide o skutočný obrázok s vysokým rozlíšením (1) alebo falošný generovaný obraz s vysokým rozlíšením (0). Táto súhra medzi dvoma modelmi vedie k tomu, že sa generátor učí vyrábať obrázky s čo najmenšími detailmi, na nerozpoznanie od reálneho obrázku. Generátor aj diskriminátor sa učia sémantické vlastnosti, ktoré sa dajú použiť následne na downstream úlohu [50].

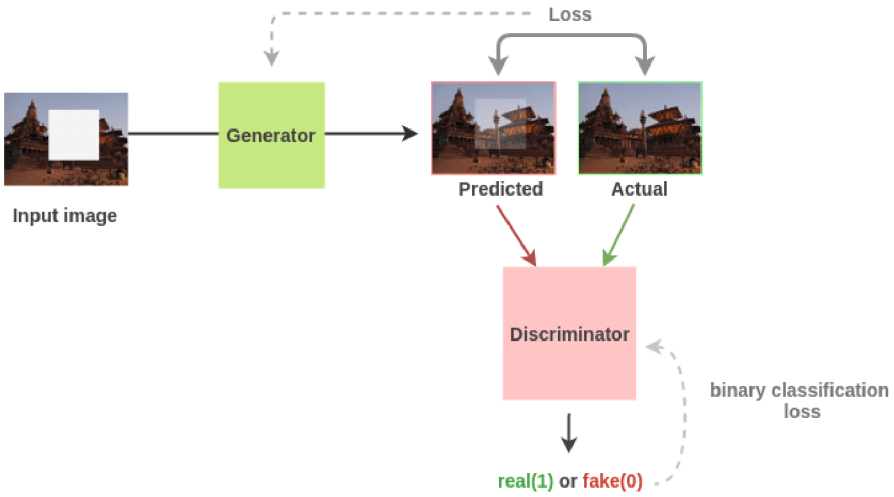


Obr. 5.7: Príklad použitia SSL metódy Super rozlíšenia obrazu [50]

5.3.3 Maľovanie obrazu - image inpaiting

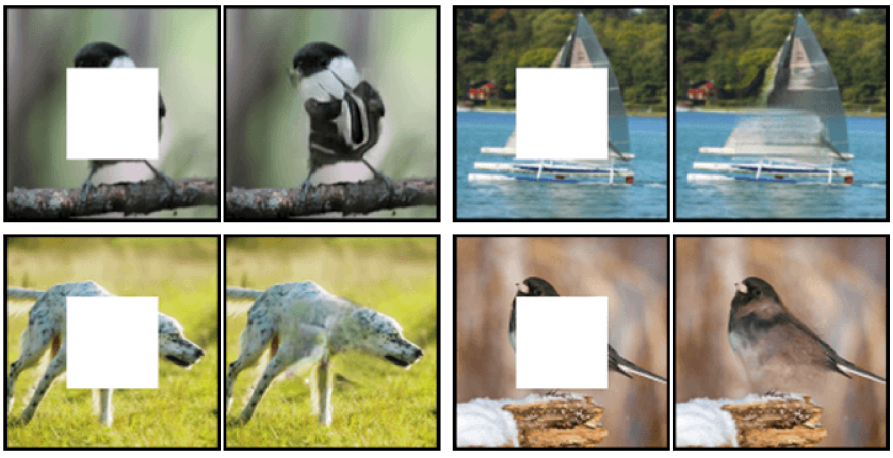
Poslednou jednoduchou metódou z pohľadu tvorby vstupných dát je metóda, ktorá sa snaží dotvoriť a domaľovať poškodený obraz na základe kontextu. Podobne ako

pri super rozlíšení, aj tu sa dá využiť architektúra založenú na princípe GAN, kde sa kontextový generátor musí úspešne naučiť pochopiť obraz, jeho okolie a prostredie aby mohol, čo najkvalitnejšie a najdôveryhodnejšie rekonštruovať chýbajúci obraz, zatiaľ čo diskriminátor oddeľuje skutočné a generované obrazy [51].



Obr. 5.8: Princíp SSL metódy maľovanie obrazu [47]

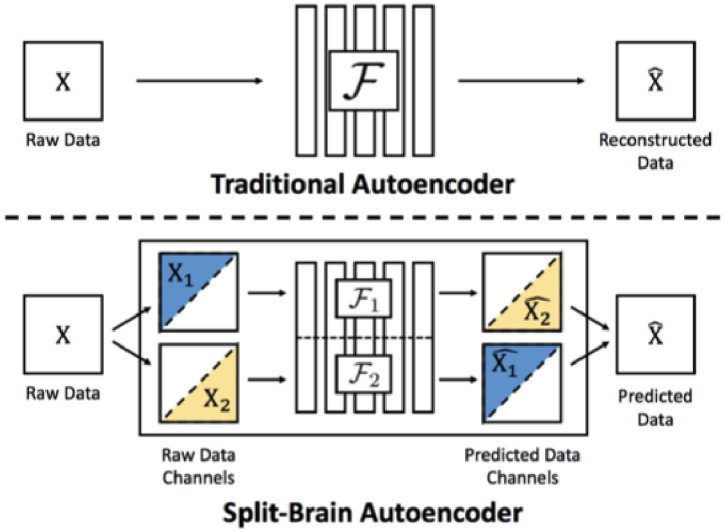
Pokiaľ ide o downstream úlohy, Pathak a kolegovia preukázali [51], že sémantické vlastnosti, ktoré sa takýto generátor naučil, poskytujú 10,2% zlepšenie oproti náhodnej inicializácii pri úlohe sémantickej segmentácie a zároveň poskytujú <4% vylepšenia oproti klasifikácii a detekcii objektov. Výsledky sémantického maľovania na obrázkoch pre kontextový enkóder trénovaný pomocou rekonštrukcie. Fotky sú zo siete ImageNet .



Obr. 5.9: Príklad použitia SSL metódy maľovanie obrazu [51]

5.3.4 Cross-Channel Prediction

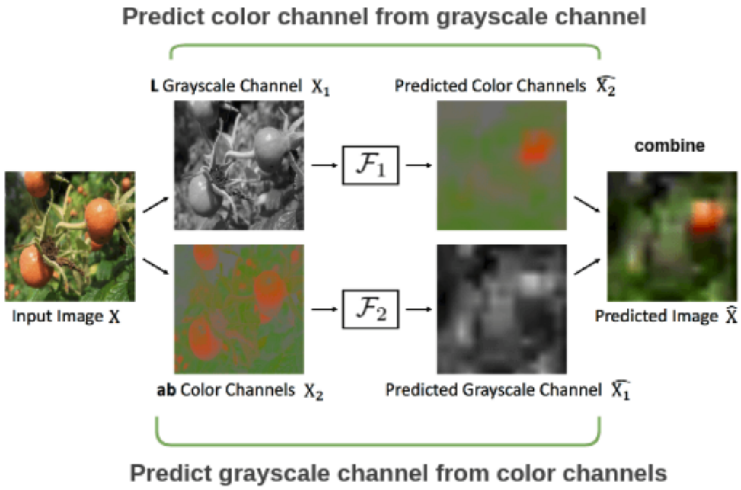
Posledná metóda zo skupiny rekonštrukcie sa nazýva cross-channel, vhodný preklad by mohol byť krížna-kanálová predikcia. Cieľom je, čo najúspešnejšie predpovedať jeden alebo viac kanálov z iného (napríklad šedotónový obraz z farebného) a zároveň trénovať predpovedanie opačným postupom, čiže predpovedať farebný obraz zo šedotónového. Následne dôjde k spojeniu týchto kanálov a rekonštruuje sa celý obraz. Tento prístup bol aplikovaný vo výskume s názvom "Split-Brain Autoencoder". Rozdiel oproti kontextovému enkóderu je v tom, že kontextový dopĺňa celý chýbajúci obraz a nie len určité spektrum farieb, ako split-brain autoenkóder [52].



Obr. 5.10: Ilustrácia Split-Brain Autoencoder [44]

Pre jednoduchšie pochopenie princípu, opíšem príklad. Predstavme si obrázok rajčín. Tento farebný obrázok môžeme rozdeliť do 2 kanálov a to do šedotónového a farebného. Následne pre kanál v šedotónových odtieňov predpovedáme farebný kanál a pre časť farebného kanálu predpovedáme kanál v odtieňoch šedotónovej, ako je vidieť na obrázku Obr. 5.11.

Dva predpovedané kanály X1 a X2 sa skombinujú, aby sa získala rekonštrukcia pôvodného obrazu. Túto rekonštrukciu môžeme porovnať s pôvodným farebným obrázkom, aby sme zistili veľkosť chyby a následne model vylepšili.



Obr. 5.11: Princíp SSL metódy Cross-Channel Prediction [47]

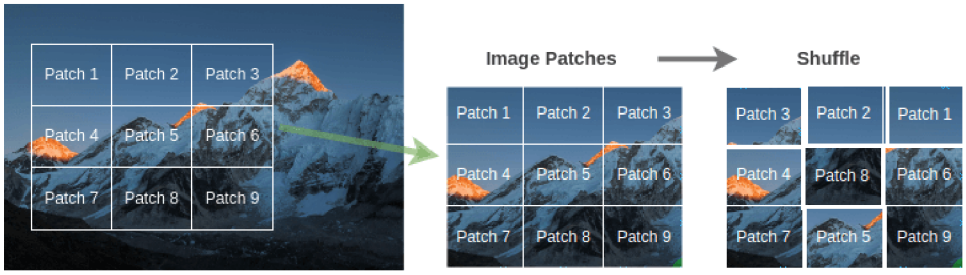
5.4 Uvažovanie zdravého rozumu - Common sense reasoning

Pri metódach "uvažovania zdravého rozumu" je cieľom pochopiť zmysel alebo obsah obrazu. Nachádzajú sa tu metódy, v ktorých sa zoberie obrázok, rozdelí sa na 9 častí, čiže 3x3, pomiešajú sa a následne trénujeme model, aby vedel určiť správne usporiadanie týchto častí obrázku do pôvodného zobrazenia (skladačka - Jigsaw puzzle). Ďalšou podobnou metódou je prístup vybratia stredovej časti obrázku a náhodne inej časti a trénovať model na predpovedanie vzťahu náhodne vybranej časti obrázku voči stredu (Kontextová predikcia - context prediction). Posledná metóda využíva náhodnú rotáciu obrázku o 0, 90, 180, alebo 270 stupňov a následné trénovanie modelu na rozpoznanie o aký typ rotácie išlo (geometrická rotácia - geometric transformation recognition)

5.4.1 Skladačka - Jigsaw puzzle

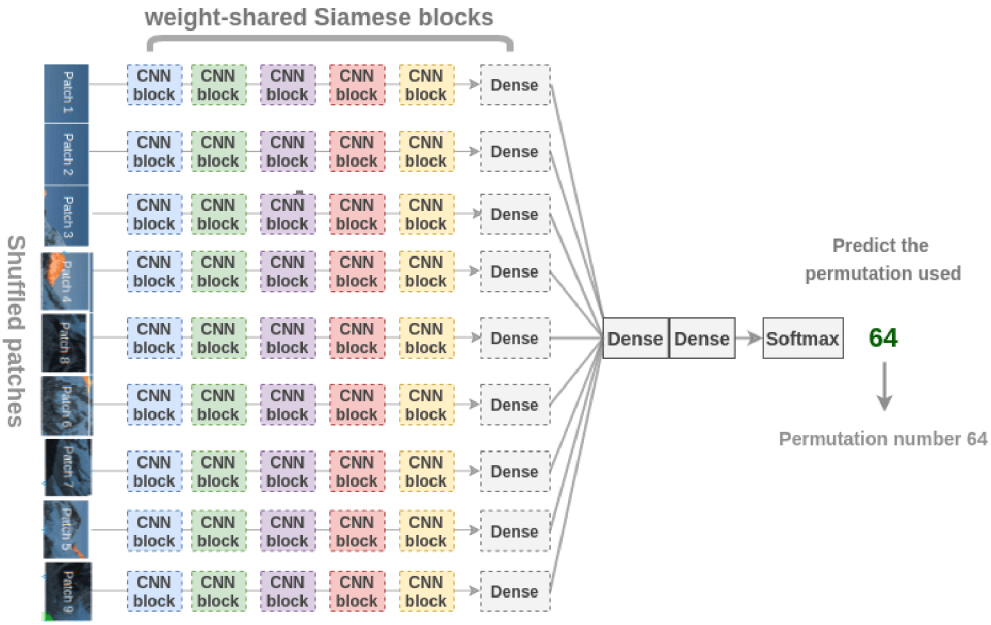
Hlavná myšlienka Jigsaw Puzzle sa zakladá na tvorbe dát v trénovacích pároch, kedy jeden obrázok je originálny a nezmenený a druhý rozdelený na 9 častí, ktoré môžu byť náhodne premiešané. Len pri 9 častiach je možné dosiahnuť až 362880 možných kombinácií hlavolamov. V realite sa používa iba podmnožina týchto permutácií a to 64 permutácií s najväčšou Hammingovou vzdialenosťou.

Tvorca tohto prístupu k self-supervised metóde (Noroozi a kolegia [53]) navrhli neurónovú sieť nazvanú bezkontextová sieť (context-free network - CFN), ako je ukázané nižšie na obrázku Obr. 5.13. Tu jednotlivé časti prechádzajú rovnakými



Obr. 5.12: Princíp SSL metódy Jigsaw puzzle [47]

siamskými konvolučnými vrstvami, ktoré majú spoločné zdieľané váhy. Potom sa funkcie skombinujú do úplne prepojenej vrstvy. Vo výstupe musí model predpovedať, ktorá permutácia bola použitá zo N možných tried. Ak model správne spozná permutáciu, môže úspešne vyriešiť skladačku/zadanie [53].

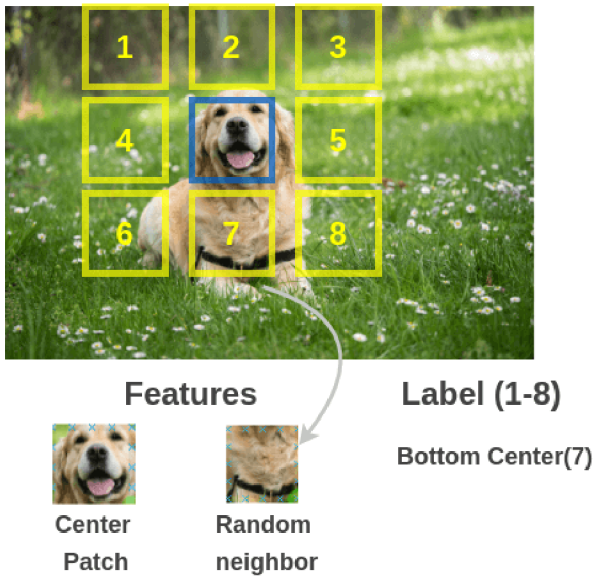


Obr. 5.13: Príklad použitia SSL metódy Jigsaw puzzle [47]

Na úspešné vyriešenie Jigsaw skladačky sa model musí naučiť identifikovať rôzne parametre a časti zhromaždené v objekte, vzájomné polohy rôznych častí predmetov a tvar predmetov. Reprezentácie sú teda užitočné pre následné úlohy pri klasifikácii aj pri detekcii.

5.4.2 Kontextová predikcia - Context prediction

Kontextová predikcia sa podobá prístupu Jigsaw skladačky. Princíp je založený na vybratí tréningových párov, ktoré sú zostrojené z 2 obrázkov, kde prvý je napríklad stredná časť a druhý náhodne vybratý [54].

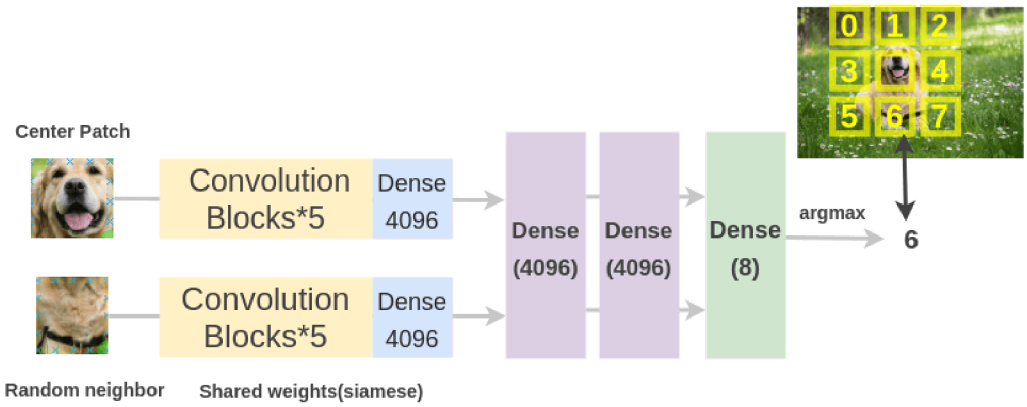


Obr. 5.14: Princíp SSL metódy kontextovej predikcie [47]

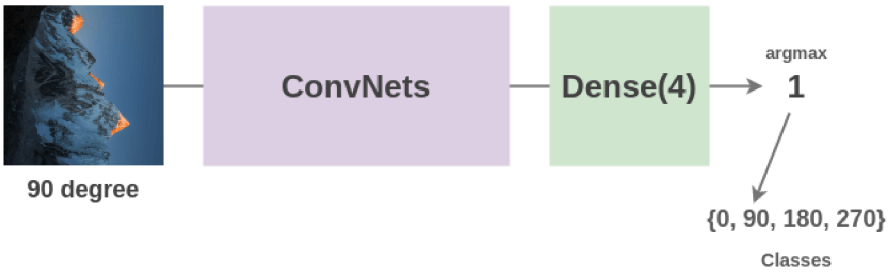
Na vyriešenie tejto pretask úlohy pán Doersch [54] použil architektúru podobnú architektúre Jigsaw skladačky. Časti obrázkov prechádzajú dvoma siamskými konvolučnými sieťami, aby extrahovali funkcie, zreťazili ich a vytvorili klasifikáciu do 8 tried označujúcich 8 možných susedných pozícií. Príklad architektúry CNN pre túto metódu je na obrázku Obr. 5.15.

5.4.3 Geometrická rotácia - Geometric transformation recognition

Tretí prístup z oblasti "uvažovania zdravým rozumom" (common sense reasoning) sa nazýva geometrická rotácia. Jej hlavná myšlienka pozostáva z prípravy tréningových párov, kde vstupný obrázok sa náhodne orotuje o 0, 90, 180 alebo 270 stupňov a priradí sa mu trieda pomenúvajúca počet otočení o 90 stupňov. Úlohou konvolučnej siete je naučiť sa o akú rotáciu ide a tým lepšie pochopiť kontext obrázku pri trénovaní modelu, ktorý sa neskôr použije pri downstream úlohe [55].



Obr. 5.15: Príklad použitia SSL metódy kontextovej predikcie [47]

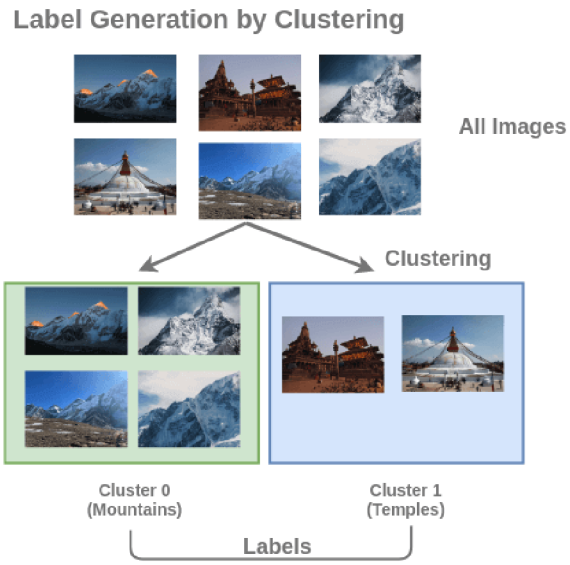


Obr. 5.16: Princíp SSL metódy geometrickej rotácie [47]

5.5 Zhlukovanie (clustering)

Pri zhlukovaní alebo inými slovami združovaní podobných obrázkov do jednotlivých skupín je možné vytvoriť K kategórií pre ktoré bude platiť rovnaký počet tried. Na základe týchto dát sa trénuje model a získavajú sa reprentácie vlastností obrázkov. Do tejto skupiny metód patrí tvorba klastrov spôsobom DeepCluster, syntetické obrázky alebo aj z časti metódy tvorby vlastných tried (Self-Labeling), ktoré ale popisujem v časti Kontrastívneho učenia (Contrastive Learning).

Cieľom tvorby klastrov je použitie tréningových párov (obrázok, číslo klastra) vykonaním zoskupenia na veľkej zbierke obrázkov bez označenia triedy. Na vyriešenie tejto pretext úlohy bola navrhnutá architektúru s názvom DeepCluster. Obrázky najskôr zoskupia a zhluky sa použijú ako triedy. Úlohou CNN je predpovedať označenie klastra pre vstupný obrázok.

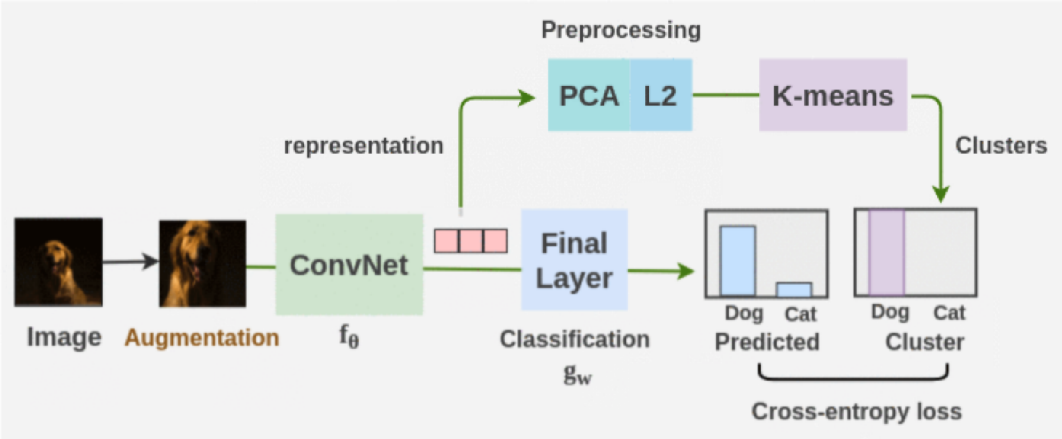


Obr. 5.17: Princíp SSL metódy zhlukovania - clustering [47]

5.5.1 DeepCluster

Mnoho SSL metód využíva pretext úlohy na generovanie náhradných tried a na pretransformovanie problému z učenia bez učiteľa na učenie s učiteľom. Niektoré metódy zahŕňajú predikciu rotácie, vyfarbenie obrazu, Jigsaw puzzle atď. Takéto úlohy však závisia od domény a na ich vytvorenie je potrebné určité znalosti. DeepCluster je metóda s vlastným učiteľom, ktorú navrhol vedec Caron z výskumného tímu vo Facebook AI. Táto metóda nevyžaduje znalosti špecifické pre doménu a dá sa použiť na osvojenie hĺbkových reprezentácií pre scenároch, v ktorých sú anotované dáta v trénovacom datasete obmedzené. DeepCluster kombinuje dva prvky: klastrovanie bez učiteľa a hlboké neurónové siete. Navrhuje end-to-end metódu spoločného učenia sa parametrov hlbokej neurónovej siete a klastrových priradení jej reprezentácií. Funkcie sa generujú a klastrujú iteratívne, aby dostali natrénovaný model a triedy ako výstupné artefakty [57].

Ako je vidieť na obrázku Obr. 5.18, najprv sa zoberú obrázky bez označenia tried a aplikujú sa na nich augmentácie, ako zmenšenie obrázku, orezanie, rotácia alebo zrkadlenie. Následne sa ako extraktor funkcií použije CNN, ako napríklad AlexNet alebo VGG-16. Spočiatku je CNN inicializovaná s náhodnými váhami, z vrstvy pred ukončením záverečnej klasifikácie sa vyberie vektor funkcií. Potom sa pomocou PCA a L2 normalizáciou zmenší dimenzia vektora. Na záver sa spracované funkcie posunú do metód K-means, aby sa získalo priradenie klastra pre každý obrázok. Tieto priradenia klastrov sa používajú ako pseudooznačenia a systém CNN je trénovaný na predpovedanie týchto klastrov. Chyba získaná z krížovej entropie sa



Obr. 5.18: Deep Cluster Pipeline [56]

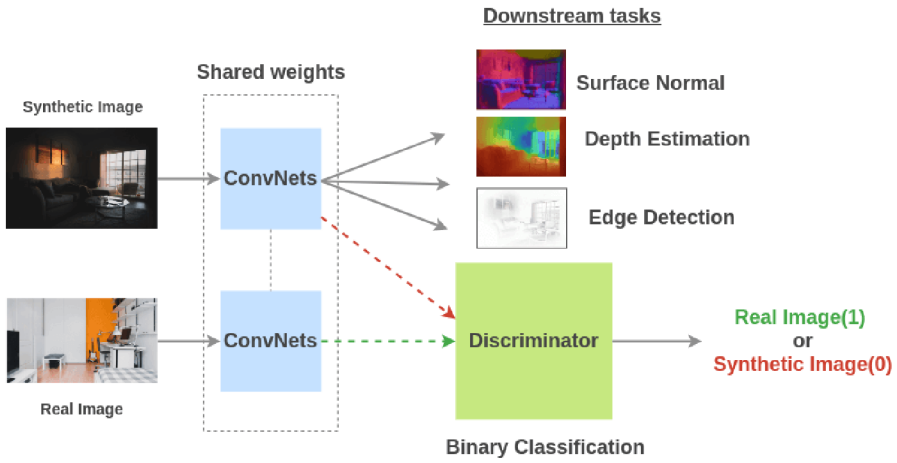
používa na meranie výkonu a presnosti modelu. Model môže byť trénovaný napríklad na 100 epochách, pričom klastrovanie nastáva raz za epochu. Po natrénovaní modelu sa zoberú naučené reprezentácie a použijú na downstream task [56].

5.5.2 Syntetické obrázky

Jedna z posledných metód používa herné nástroje na tvorbu syntetických obrázkov. Cieľom je pripraviť tréningové páry (obrázok, vlastnosti) generovaním syntetických obrázkov pomocou herných nástrojov a ich prispôsobenie skutočným obrázkom. Na vyriešenie tejto pretext úlohy je navrhnutá architektúra, v ktorej sa zdieľané váhy z konvolučnej siete trénujú na syntetických aj reálnych obrázkoch, a následne sa diskriminátor naučí klasifikovať, či funkcie konvolučnej siete, ktoré sú doň privádzané, majú syntetický alebo skutočný obraz. Vďaka kontradiktórnosti sa zdieľané zobrazenia medzi skutočnými a syntetickými obrázkami zlepšuje (Obr. 5.19) [58].

5.6 Kontrastívne učenie (contrastive learning)

Kontrastívne učenie sa s vlastným učiteľom (contrastive self-supervised learning) je sľubnou metódou, ktorá vytvára reprezentácie učením sa zoskupovania toho, čo robí dve veci podobné alebo odlišné. Pri obrázkoch model môže byť trénovaný aby vedel správne klasifikovať medzi podobnými a rozdielnymi obrázkami, konkrétne pracuje s augmentáciami vstupných obrázkov a následným porovnaním s originálom a cieľom je priblížiť dvojicu obrázkov z pôvodného originálu k sebe kým rozdielne dvojice alebo zvyšné obrázky pôjdu od seba. Kontrastívne metódy trénované na neoznačených dátach z ImageNet a vyhodnotené lineárnym klasifikátorom dokážu prekonať



Obr. 5.19: Princíp SSL metódy syntetických obrázkov [47]

úspešnosť siete AlexNet. Vykazujú taktiež významnú efektívnosť a úspešnosť pri použití na učení sa reprezentácií dát bez označenia v porovnaní s učením čisto s učiteľom. Použitie kontrastívneho učenia na predtrénovanie ImageNet dokáže posunúť naučené reprezentácie cez downstream úlohu a tým prekonať úspešnosť iných metód, ktoré na predtrénovanie používajú označené dáta s triedami/anotáciami. V tejto oblasti sa aplikujú metódy ako SimCLR, PIRL, SwAV, MoCov2 alebo BYOL a väčšinu z nich popíšem v nasledujúcom texte.
Hlavné prístupy sú nasledovné:

- **Architektúra enkódera:** Konvertuje obraz na reprezentácie
- **Meranie podobnosti medzi dvoma obrázkami:** stredná štvorcová chyba, kosínová podobnosť, strata obsahu
- **Generovanie tréningových párov:** manuálna anotácia, metódy s vlastným učiteľom

5.6.1 Kontrolované kontrastívne učenie (supervised contrastive learning)

Pri trénovaní dochádza k posunu rovnakých tried bližšie k sebe a rozdielnych ďalej od seba. Následne sa zmrazia vrstvy vo fáze 1 a trénuje sa iba klasifikátor/model bežným prístupom s využitím krížovej entropie a softmax metódy. V jednoduchosti by sa celý postup dal popísať nasledovne:

1. Zobrať obrázok a vyrobiť augmentácie
2. Zobrať ďalšie obrázky z tej istej triedy a urobiť podobné augmentácie. Týmto vzniknú takzvané pozitívne obrázky a tie sa priradia do rovnakej triedy spolu s augmentovanými obrázkami

3. Zobrať iné obrázky rozdielnych tried a urobiť augmentácie. Tým vzniknú takzvané negatívne obrázky, ktoré chceme oddeliť od zvyšku [59]

5.6.2 Self-supervised kontrastívne učenie (self-supervised contrastive learning)

Pri metóde s vlastným učiteľom vylepšujeme neurónovú sieť predtrénovaním modelu a vrstiev dátami bez tried, čiže trénovaním bez učiteľa. Postup je jednoduchší:

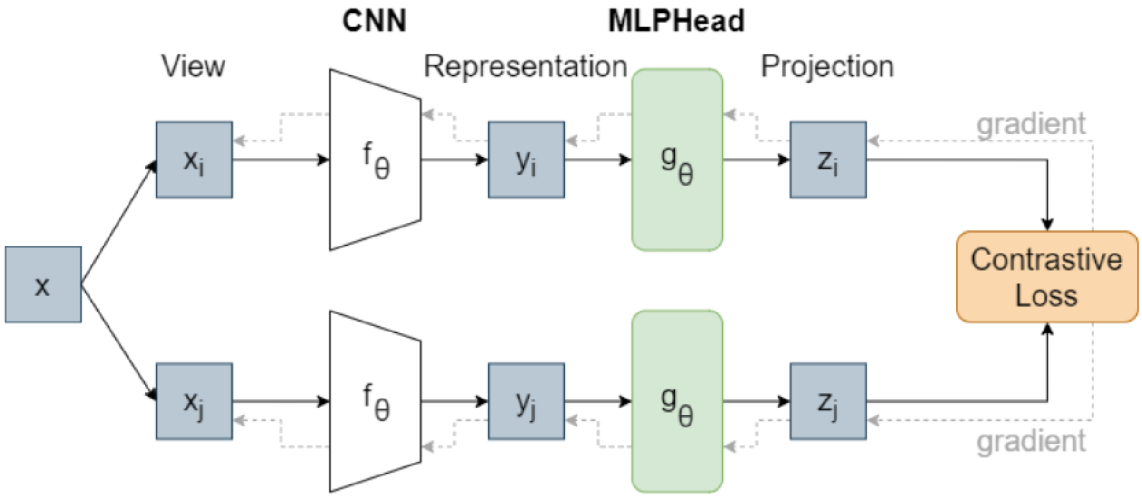
1. Zobrať obrázok a urobiť augmentácie
2. Cieľom je dosiahnuť to isté ako pri učení s učiteľom (podobné k sebe, rozdielne od seba) ale s tým, že program sám priradí podobné obrázky k sebe a rozdielne od seba s využitím augmentácii. Podobnosť obrázkov sa počíta napríklad z kosínovej vzdalenosti vektorov reprezentácií/vlastností dvoch obrázkov alebo konkrétnej funkcie na výpočet chyby v danej kontrastívnej metóde [59].

5.6.3 A Simple Framework for Contrastive Learning of Visual Representations (SimCLR)

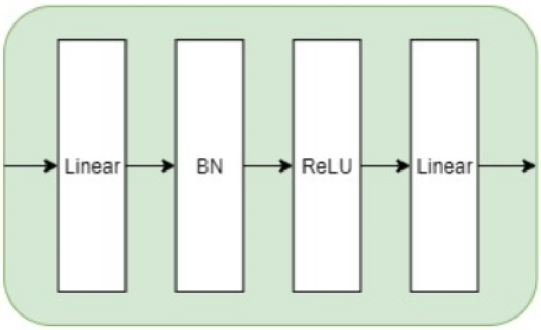
Google v apríli 2020 predstavil SimCLR, spôsob pre zdokonaľovanie metód s čiastočným (semi) a vlastným učiteľ (self-supervised) pre analýzu obrázkov. Na základe princípov kontrastívneho učenia poskytuje SimCLR model, ktorý sa učí reprezentácie (vlastnosti obrázkov) maximalizáciou zhody medzi rôzne upravenými obrázkami toho istého dátového príkladu prostredníctvom kontrastívnej chyby v latentnom priestore. Koncepčne je SimCLR založený na nasledujúcich komponentoch:

1. **Modul stochastických augmentácií:** Táto zložka transformuje akýkoľvek dátový príklad (obrázok - x) a generuje dva korelované pohľady (x_i, x_j) toho istého príkladu, ktorý nazývame pozitívny pár. Konkrétne tento modul aplikuje na daný vstup rôzne transformácie údajov: náhodné orezanie, náhodné skreslenie farieb, náhodnú rotáciu a náhodné Gaussovo rozostrenie. Príklad použitých augmentácií je na obrázku Obr. 5.23.
2. **Enkodér základne f(.):** Táto zložka extrahuje reprezentačné vektory zo vzoriek augmentovaných údajov. Výstup je z poslednej vrstvy použitej na extrakciu reprezentácií (vlastností).
3. **Projekčná časť/hlava g(.):** Táto časť sa po anglicka nazýva multilayer projection head (MLPHead) a mapuje reprezentácie do priestoru, kde sa aplikuje kontrastívna chyba. Ako projekčná časť sa používa neurónová sieť o veľkosti dvoch alebo troch vrstiev (1 vstupná, 1 skrytá a 1 výstupná) spolu s dávkovou normalizáciou. Príklad MLPHead zloženej z 2 vrstiev, dávkovej normalizácie a ReLU aktivačnej funkcie, je na obrázku Obr. 5.21.

4. **Funkcia kontrastívnej chyby:** Táto zložka berie ako vstup množinu príkladov vrátane pozitívnej dvojice príkladov (z_i, z_j) a má za cieľ identifikovať z_j v danej množine pre dané z_i. Čiže priblížiť podobnosť vektorov obrázkov z jedného rovnakého pôvodného obrázku a oddialiť podobnosť vektorov od iných reprezentácií ostatných obrázkov. Chyba je počítaná zo vzorcu funkcie s názvom NT-Xent (normalized temperature-scaled cross-entropy loss) a v porovnaní s inými typmi výpočtu chyby mala najlepšie výsledky v aplikovaní v SimCLR metóde [60].



Obr. 5.20: Architektúra SSL metódy SimCLR



Obr. 5.21: Projekčná hlava (MLPHead)

Ak by sme použili NT-Xent funkciu na 25 vstupných obrázkov (napr. jedna dávka (batch)) tak z 25 obrázkov aplikovaním augmentácií spravíme 50. V týchto

50 obrázkoch sú navzájom pozitívne a negatívne páry. Pozitívny pár obrázkov je ten, ktorý pochádza z rovnakého pôvodného obrázku. Všeobecne pre dávku o veľkosti N, dostaneme 2N augmentovaných obrázkov. Z toho vyplýva, že konkrétny pozitívny pár má voči sebe počet negatívnych obrázkov vo veľkosti 2(N-1).

Takýto pozitívny pár obrázkov je vstupom do konvolučnej neurónovej siete, z ktorej výstupom sú vektory reprezentácií z_i a z_j. Následne je cieľom maximalizovať podobnosť/zhodu týchto dvoch vektorov, čiže aby vektory reprezentujúce dva rôzne augmentované obrázky mali čo najviac podobné reprezentácie. Táto podobnosť je maximalizovaná minimalizovaním kontrastívnej chyby, ktorá sa počíta pomocou NT-Xent funkcie medzi vektormi.

Funkcia NT-Xent chyby [60] pre pozitívny pár obrázkov reprezentovaných vektormi z_i a z_j je nasledovná:

$$L_{i,j} = -log\frac{exp(sim(z_i, z_j)/\tau)}{\sum_{k=0}^{2N} exp(sim(z_i, z_k)/\tau)} \tag{5.1}$$

Vo vzorci je použitá kosínová podobnosť vektorov z nasledujúcej funkcie:

$$sim(u, v) = \frac{u^T v}{\|u\|\|v\|} = \hat{u}^T \hat{v} \tag{5.2}$$

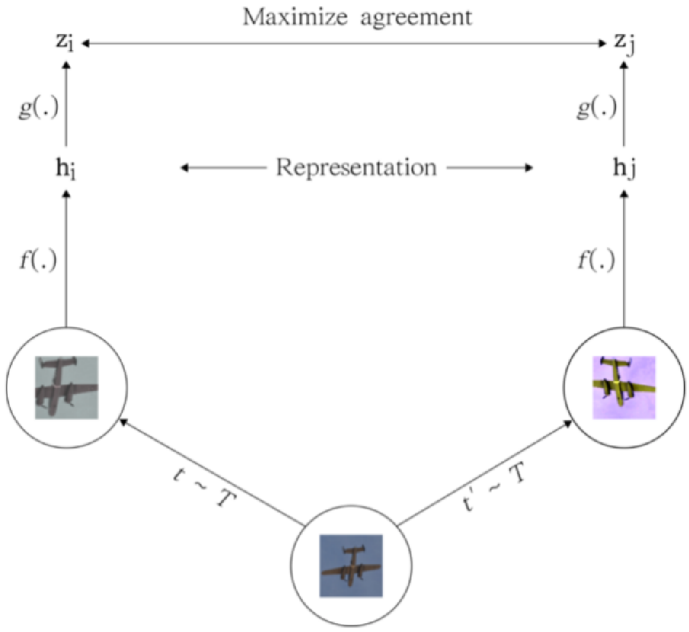
Zjednodušením využitím vlastností logaritmu a aplikovaním algebraických postupov sa dá dopočítať k nasledujúcemu vzorcu:

$$\downarrow L_{i,j} = log(1 + \frac{\sum_{k=1}^{2N} exp(\downarrow sim(z_i, z_k)/\tau)}{exp(\uparrow sim(z_i, z_j)/\tau)}) \tag{5.3}$$

Zo vzorca vidíme, že minimalizovaním NT-Xent chyby pre pozitívny pár (i, j) neurobíme nie len vektory z_i a z_j viac podobné sebe navzájom, ale zároveň aj menej podobné iným vektorom. Výpočet chyby je asymetrický ($L_{i,j}$!= $L_{j,i}$) kvôli výpočtu sumy v čitateli, ktorá by bola iná, pri prehodení poradia. Z toho dôvodu budem počítať finálnu chybu pre všetky obrázky v oboch poradiach a následne vypočítam priemer. Hodnotu τ je možné rôzne meniť a tým ovplyvniť výsledok NT-Xent chyby.

SimCLR mapuje predchádzajúce koncepčné komponenty na inteligentnú architektúru hlbokej neurónovej siete. Základná architektúra je inšpirovaná reziduálnymi neurónovými sieťami (Resnet). SimCLR spočiatku náhodne čerpá príklady z pôvodného súboru údajov a každý príklad dvakrát transformuje pomocou kombinácie jednoduchých augmentácií (náhodné orezanie, náhodné skreslenie farieb, náhodná rotácia a Gaussovo rozostrenie), čím vytvorí dve sady zodpovedajúcich zobrazení. Následne metóda vypočíta obrazovú reprezentáciu pomocou CNN založenej na architektúre ResNet. SimCLR dopĺňa tento krok výpočtom nelineárnej projekcie obrazu pomocou viacvrstvového perceptrónu v MLPHead, ktorý tvorí hlavu celej siete.

Tento krok zosilňuje invariantné vlastnosti a maximalizuje schopnosť siete identifikovať rôzne transformácie toho istého obrazu. Výstup CNN aj MLPHead je optimalizovaný pomocou stochastického gradientu, aby sa minimalizovala stratová funkcia kontrastívneho učenia [60]. Príklad celej architektúry pre metódu SimCLR je na obrázku Obr. 5.20. Na obrázku Obr. 5.22 je ilustrovaný vzťah medzi komponentami augmentácií a postupu ich aplikovania:

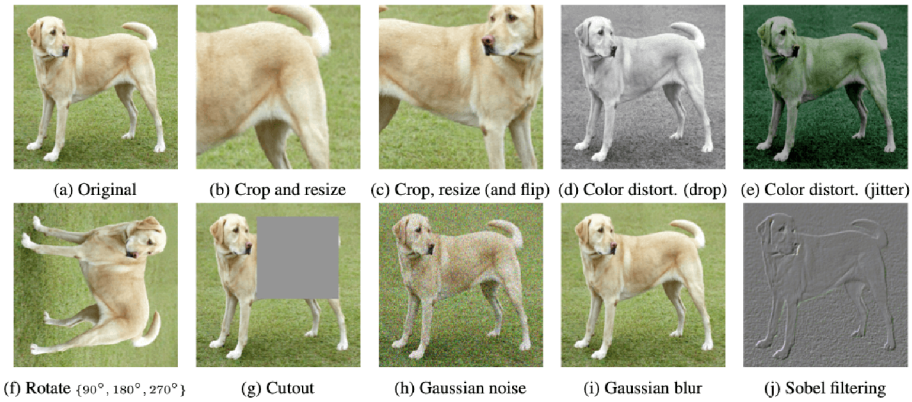


Obr. 5.22: Princíp SSL metódy SimCLR [62]

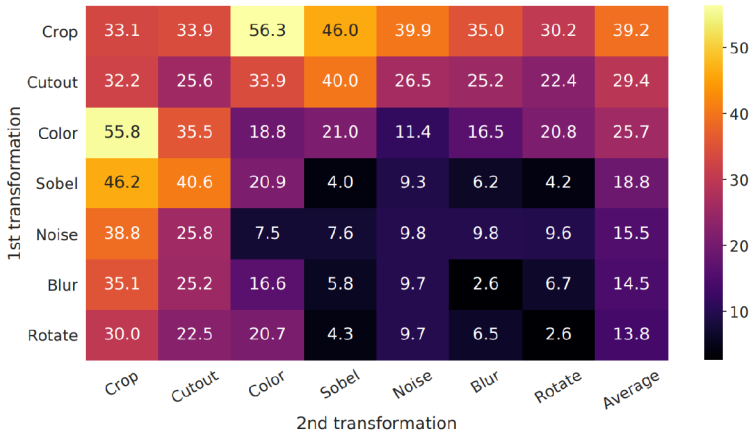
Implementácia SimCLR obsahuje niektoré dôležité osvedčené postupy, ktoré ju odlišujú od iných metód. Toto sú niektoré z nich:

1) Viac augmentácií vytvorí lepšie reprezentácie

SimCLR obsahuje rôzne augmentácie (transformácie obrázku), ktoré je možné použiť na množinu vstupných údajov. Experimenty so SimCLR preukázali, že žiadna konkrétna augmentácia nestačí na definovanie predikčnej úlohy, ktorá by priniesla najväčší úspech, ale využitie dvoch konkrétnych transformácií za sebou vyniká: náhodné orezanie a náhodné skreslenie farieb. Aj keď ani orezanie, ani skreslenie farieb nevedie samo osebe k vysokej úspešnosti, zloženie týchto dvoch transformácií vedie k takzvaným state-of-the-art výsledkom, čiže najlepším momentálne možným [60]. Výsledky aplikovania rôznych augmentácií v porovnaní s inými, je na obrázku Obr. 5.24, z ktorého je zrejmé, že použitie náhodného orezania a náhodného skreslenia farieb dokopy umožňuje dosiahnuť najväčšiu presnosť (výsledky sú z lineárnej evaluácie predtrénovaného SSL modelu).



Obr. 5.23: Príklady augmentácií na použitie v SimCLR [60]



Obr. 5.24: Výsledky aplkácie rôznych augmentácií v metóde SimCLR [60]

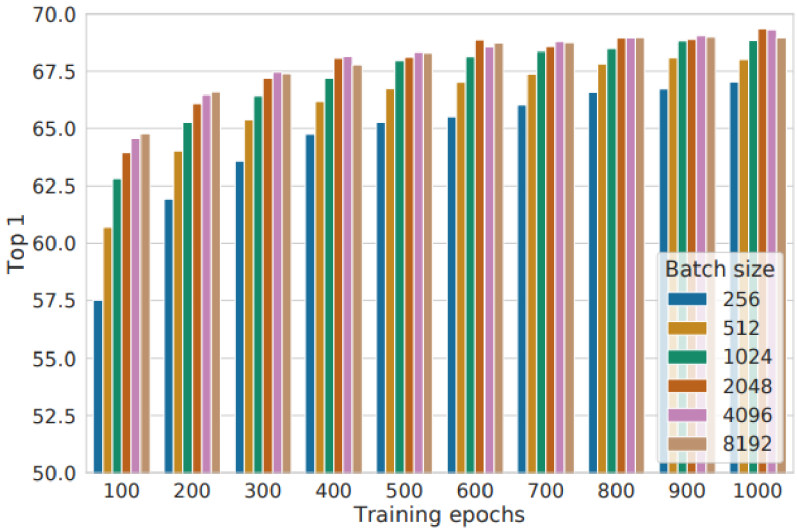
2) Nelineárne projekcie sú dôležité

SimCLR využíva vrstvu MLPHead na implementáciu nelineárnej projekcie do obrazu pred výpočtom chyby. Myšlienka je taká, že nelineárna projekcia môže pomôcť identifikovať invariantné vlastnosti každého vstupného obrazu a maximalizovať schopnosť siete identifikovať rôzne transformácie toho istého obrazu. Počiatočné experimenty ukázali, že ak MLPHead obsahuje nelineárnu funkciu pomôže to zlepšiť učenie reprezentácií (vlastností) o viac ako 10% [63].

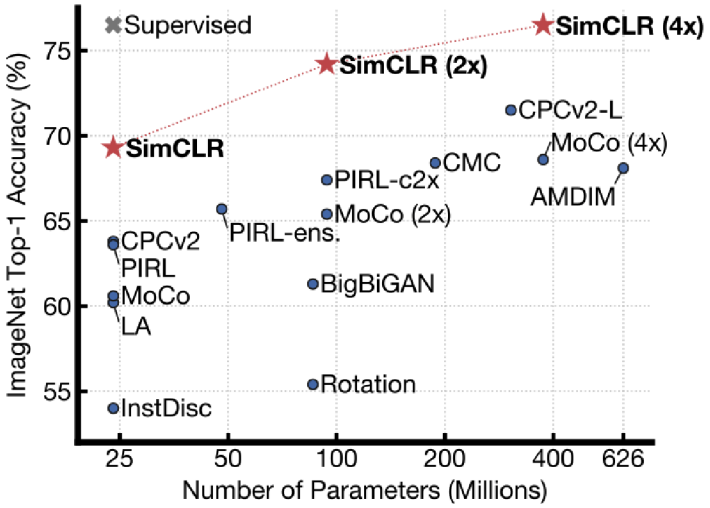
3) Škálovanie nahor výrazne zlepšuje výkon

Výsledky experimentov metódy SimCLR dokazujú, že rozšírenie parametrov, ako napríklad spracovanie viacerých príkladov v jednej dávke, použitie väčších sietí a dlhšie učenie (väčší počet epoch), vedú k významným zlepšeniam. Z tohto pohľadu je pri používaní SimCLR dôležitá optimalizácia pre väčší tréning. Výsledky tohto

experimentu sú zobrazené na obrázku Obr. 5.25. SimCLR ukázalo pozoruhodné zlepšenia výkonu v porovnaní s inými modelmi s vlastným a čiastočným učiteľom pre klasifikáciu obrázkov. Väčšie verzie SimCLR z pohľadu veľkosti dávok obsahujúcich vstupné obrázky (batchsize) alebo dĺžky tréningu (počet epoch) dosahujú dokonca úrovne presnosti porovnateľné s metódami, ktoré využívajú trénovanie s učiteľom (supervised) a toto porovnanie je na obrázku Obr. 5.26. [63].



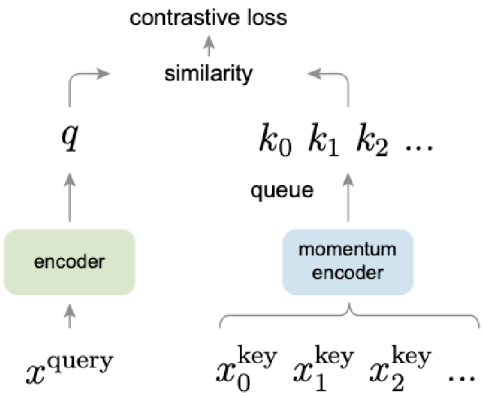
Obr. 5.25: Porovnanie veľkosti vstupných dát (batchsize) a dĺžky tréningu (epoch) v SSL metóde SimCLR [60]



Obr. 5.26: Porovnanie presnosti SSL metód so SimCLR [60]

5.6.4 Momentum Contrastive Learning (MoCo)

Predošlá metóda SimCLR je relatívne náročná na výpočtový výkon. Z tohto dôvodu sa podarilo vytvoriť metódu, ktorá je jednoduchšia na výpočtový výkon a tou je Momentum Contrastive Learning (MoCo), ktorá pochádza z vývojového tímu od firmy Facebook. Cieľom je metóda, ktorá rozlíši vzájomnú podobnosť a rozdielnosť augmentovaných párov kontrastívnym učením (rovnako ako iné kontrastívne metódy). MoCo sa na tento problém používa dynamický slovník. Takýto slovník sa skladá z kľúčov (tokenov - keys), ktoré sú porovnávané so žiadosťami (query). "Kľúče" sú v slovníku vzorkované z obrázkov a sú reprezentované sieťou enkóderu. Učenie bez učiteľa trénuje enkódery tak, aby vykonávali vyhľadávanie v slovníku: zakódovaná "žiadosť" by mala byť podobná zodpovedajúcemu "kľúču" a zároveň by mala byť odlišná od ostatných kľúčov. Učenie sa uskutočňuje, ako minimalizácia kontrastívnej chyby. Metóda sa dá aplikovať na počítačové videnie, ale aj spracovanie jazyka alebo iné úlohy strojového učenia. Jednoduchý popis algoritmu tejto metódy je zobrazený na obrázku Obr. 5.27 [69].



Obr. 5.27: Jednoduchý popis algoritmu MOCO [66]

MoCo trénuje enkodér vizuálnej reprezentácie priraďovaním kódovanej žiadosti "query - q" k slovníku kódovaných kľúčov "keys - k" pomocou kontrastívnej chyby. Kľúče slovníka k_0, k_1, k_2,..., sú definované za behu množinou vzoriek údajov. Kľúče sú kódované pomaly postupujúcim enkodérom, ktorý je poháňaný aktualizáciou enkodéru "žiadostí". Na základe tejto perspektívy sa predpokladá, že je potrebné zostaviť slovníky, ktoré sú: veľké a konzistentné. Veľké aby lepšie vzorkovali viacrozmerný vizuálny priestor a konzistentné, aby kľúče k slovníku boli reprezentované rovnakým alebo podobným enkodérom, aby ich porovnania so "žiadosťou" bolo konzistentné počas celého trénovania [68].

MoCo je možné použiť s rôznymi pre-text úlohami. Tvorcovia tejto vedeckej

publikácie použili jednoduchú pre-text úlohu a to rozlišovanie inštancií, na základe zhody "ziadostí" a "kľúča", ak ide o zakódované zobrazenia toho istého obrázka. Pomocou takéhoto spracovania metódy MoCo sa podarilo konkurovať iným modelom trénovaných s učiteľom s rovnakou úlohou klasifikácie na ImageNet datasete. Hlavným účelom učenia bez učiteľa je predtrénovať reprezentácie (vlastnosti), ktoré je možné preniesť do downstream časti doladením. Tvorcovia ďalej tvrdia, že v siedmich downstream úlohách týkajúcich sa detekcie alebo segmentácie môže MoCo s predtrénovaním bez učiteľa prekonať sieť ImageNet trénovú s učiteľom [66].

MoCo do veľkej miery odstraňuje medzeru medzi učením bez učiteľa a s učiteľom v mnohých úlohách počítačového videnia a môže slúžiť, ako alternatíva k predtrénovaniu s učiteľom siete ImageNet. Princíp funkcie MoCo je nasledovný. Zoberie sa zakódovaná "žiadosť - q" a množina zakódovaných vzoriek k_0, k_1, k_2,..., ktoré sú kľúčmi slovníka. Predpokladá sa, že v slovníku je jeden jediný kľúč (označený ako "k+"), ktorý priamo patrí ku žiadosti "q". Kontrastívna chyba je výstup kontrastívnej funkcie v metóde MoCo, ktorej hodnota je nízka, keď "q" je podobné kladnému kľúču "k+" a odlišné od všetkých ostatných kľúčov (čiže kľúče "k-" sú záporné pre "q") [70].

Tvorba pozitívnych párov je podobná ako pri SimCLR. Využije sa odskúšaný postup (na vstupný obrázok x) náhodného orezania a následne zmeny sfarbenia (to sú 2 najdôležitejšie augmentácie a ku ním je možné pridať ďalšie) a takto augmentované obrázky (x_i, x_j) sa použijú na trénovanie konvolučnej siete a tým sa model učí vlastnosti vstupných obrázkov. Príklad postupu aplikovania 2 základných augmentácií je na obrázku Obr. 5.28. MoCo patrí do kontrastívnej oblasti SSL metód a aplikovanie kontrastívneho učenia je v "hľadaní správnej žiadosti ku správnemu kľúču". Namiesto toho, aby bol použitý jeden enkodér, existujú dva - jeden pre "žiadosti" (CNN - Q) a druhý pre "kľúče" (CNN - K). Navyše, aby bolo možné aplikovať kontrastívny princíp s väčším množstvom negatívnych párov, je k dispozícii veľký dynamický slovník kódovaných kľúčov.

Pozitívny pár v tomto kontexte znamená, že "žiadosť" sa zhoduje s "kľúčom". Zhodujú sa, ak "žiadosť" aj "kľúč" pochádzajú z rovnakého obrázka. Zakódovaná "žiadosť" (vektor vlastností) by mala byť podobná jej zodpovedajúcemu "kľúču" a odlišná od všetkých ostatných. Pre negatívne páry sa udržiava veľký slovník, ktorý obsahuje zakódované "kľúče" z predchádzajúcich dávok. Slúžia ako negatívne vzorky k danej "žiadosti". Slovník je udržiavaný vo forme fronty (FIFO), čiže aktuálna mini-dávka príde do radu a najstaršia mini-dávka sa vyradí z dôvodu dlhého veku dát a predpokladu jej nepotrebnosti. Zmenou veľkosti tohto poradia sa mení počet negatívnych vzoriek. Keď sa zmení enkóder kľúčov, kľúče, ktoré sú neskôr zaradené do fronty, sa môžu stať nekonzistentné s kľúčmi, ktoré boli do fronty zaradené pomerne skoro. Aby prístup kontrastívneho učenia fungoval, všetky kľúče, ktoré sa porovná-

Obr. 5.28: Princíp aplikácie augmentácií v MOCO [68]

vajú so "žiadosťami", musia pochádzať od rovnakých alebo podobných enkodérov, aby porovnania boli zmysluplné a konzistentné [68].

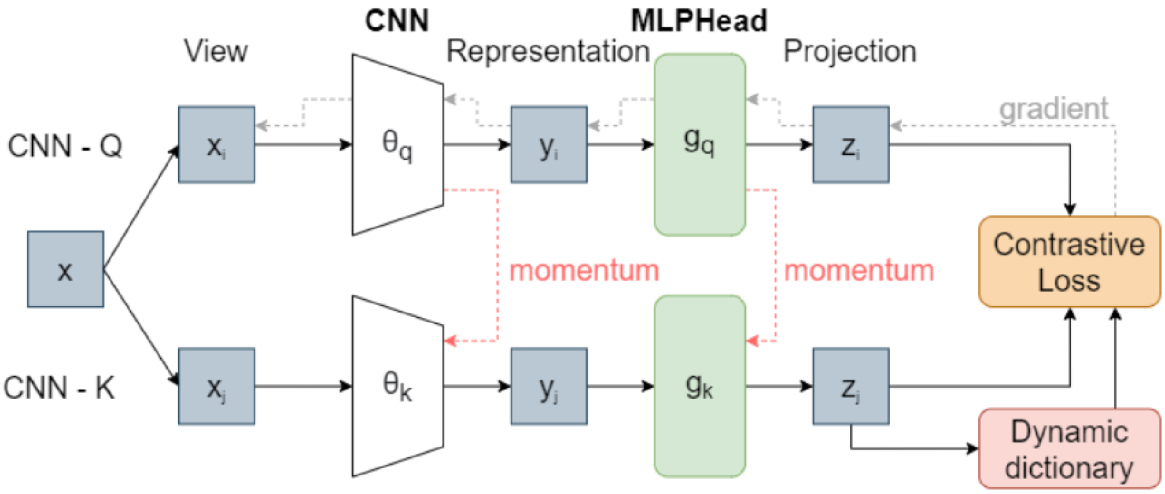
Ďalšou výzvou je, že nie je možné naučiť sa kľúčové parametre enkóderu pomocou spätného šírenia (backpropagation), ako je to aplikované pri metóde SimCLR, pretože by to vyžadovalo výpočet gradientov pre všetky vzorky vo fronte (čo by malo za následok veľmi veľký výpočtový graf). Keďže pri učení sa kontrastívnym spôsobom ide o spájanie podobných a odďaľovanie sa od rozdielnych obrázkov, v tejto metóde je to rovnaké. V MoCo je cieľom urobiť "žiadosť - q" podobnú pozitívnemu "kľúču", a rozdielnu všetkým "negatívnym kľúčom". Funkcia výpočtu kontrastívnej chyby, ktorá je použitá pri metóde MoCo sa nazýva InfoNCE (Information Noise Contrastive Estimation) [66] a jej rovnica je nasledovná:

$$L_q = -log\frac{exp(q \cdot k_+/\tau)}{exp(q \cdot k_+/\tau) + \sum_{i=0}^{K} exp(q \cdot k_-/\tau)} \tag{5.4}$$

Na riešenie oboch týchto problémov implementuje MoCo kľúčový enkóder, ako hybný kĺzavý priemer enkóderu "žiadostí" nazvaný "momentum update". Je to uľahčenie problému, ktorý by vznikol ak by sme museli spätne šíriť gradient aj do enkóderu žiadostí a aj enkóderu kľúčov. Váhy enkodéru žiadostí sa trénujú pomocou klasického gradient descent a váhy enkodéru kľúčov pomocou spomenutého "momentum update". Aktualizujú sa tým iba kľúčové parametre enkóderu a to nasledujúcim spôsobom:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \tag{5.5}$$

kde m (koeficient moment) sa udržuje blízko 1 (napr. typická hodnota je 0,999), čo zaisťuje, že spracovávané kľúče sa získajú v rôznych časoch od podobných enkóderov. Pri experimentoch vedeckého tímu skúšali hodnoty od 0 a limitne sa približujúc k 1, dosahovali vyššiu presnosť. Spätným šírením sa aktualizujú iba váhy enkóderu θ_q. Následne o krok neskôr sa pomocou vyššie spomenutého vzorca plynulo upravujú váhy aj enkóderu kľúčov. Aktualizácia cez momentum umožňuje, aby sa θ_k vyvíjal plynulejšie ako θ_q. Výsledkom je, že aj keď sú kľúče vo fronte kódované rôznymi enkódermi (v rôznych dávkach), rozdiel medzi týmito enkódermi môže byť malý. Tento prístup umožňuje efektívnejšiu prácu s pamäťou počítača a využitia GPU pri trénovaní, keďže si nepotrebuje zapamätať gradient pre všetky obrázky. Náčrt architektúry týchto dvoch sietí (CNN - Q, CNN - K), zapojenia dynamického slovníku a spätné šírenie gradientu a úprava váh cez momentum, je zobrazená na obrázku Obr. 5.29.



Obr. 5.29: Architektúra SSL metódy MoCo

Výsledky experimentálneho testovania metódy MoCo na sieti ResNet50 dosahovali 60.6% presnosť, čo je viac ako konkurenčné metódy pri rovnakej veľkosti modelov (24 miliónov parametrov). Pri testovaní využitia väčšieho počtu parametrov (375 miliónov) sa podarilo dosiahnuť až 68.6% presnosť. Vedci trénovali sieť Resnet50 na dataset ImageNet-1M s veľkosťou mini-batch 256 a tréning uskutočnili na 8GPU. Takýto experiment im trval 53 hodín. Pre porovnanie s predošlou metódou SimCLR, dosahuje MoCo mierne lepšiu úspešnosť (69.3% vs 71.1%) pri kratšom tréningu (800 vs 1000 epoch) a menšej vstupnej dávke z datasetu (256 vs 4096 batchsize) [66].

V mojej diplomovej práci plánujem aplikovať mierne vylepšenú metódu MoCov2 (verzia 2), ktorá podľa publikovaných informácií vďaka použitiu projekčnej hlavy

case	unsup. pre-train MLP	aug+	cos	epochs	batch	ImageNet acc.
MoCo v1 [6]				200	256	60.6
SimCLR [2]	✓	✓	✓	200	256	61.9
SimCLR [2]	✓	✓	✓	200	8192	66.6
MoCo v2	✓	✓	✓	200	256	**67.5**
*results of **longer** unsupervised training follow:*						
SimCLR [2]	✓	✓	✓	1000	4096	69.3
MoCo v2	✓	✓	✓	800	256	**71.1**

Obr. 5.30: Porovnanie self-supervised metód SimCLR a MoCo [66]

(MLPHead) a aplikovaním Gaussovho rozostrenia (obidve vylepšenia rovnako, ako pri SimCLR) má dosiahnuť mierne lepšie výsledky, ako pôvodná verzia MoCov1 [67]. Hlavné rozdiely medzi metódou MoCov1 (aj MoCov2) voči SimCLR sú použitie dvoch neurónových sietí (pre kľúče a pre žiadosti), ukladanie reprezentácií jedného z dvoch augmentovaných obrázkov (kľúču) do dynamického zoznamu a úprava váh druhej CNN (pre kľúče) pomocou hybného kĺzavého priemeru "momentum update".

5.6.5 Bootstrap Your Own Latent (BYOL)

Jednou z najnovších (jún 2020) a najúspešnejších metód oblasti SSL je Bootstrap Your Own Latent (BYOL), ktorú vytvorili v DeepMind v spolupráci s Imperial Collage. Je to ďalšia z kontrastívnych metód, ktoré sú menej náročné na výpočtový výkon v pomere úspešnosti a potrebného výkonu, ako generatívne/prediktívne metódy. Autori BYOL metódy to popísali nasledovne: "Kontrastívne prístupy zabraňujú nákladnému kroku generovania v pixelovom priestore tým, že približujú zastúpenie rôznych pohľadov na ten istý obrázok ("pozitívne páry") a posúvajú od seba reprezentácie pohľadov z iných obrázkov ("negatívne páry") od seba." [71].
Aby takýto prístup dobre fungoval, musíme každú vzorku/obrázok porovnávať s mnohými ďalšími negatívnymi vzorkami/obrázkami. To môže byť problematické, pretože to môže vniesť nestabilitu do tréningu a zväčšiť predsudky alebo ovplyvnenie modelu na základe datasetu. Autori BYOL na to upozorňujú nasledovným textom: "Kontrastívne metódy sú citlivé na výber augmentácií. Napríklad SimCLR nefunguje dobre pri odstraňovaní farebného skreslenia z augumentácie obrazu. SimCLR ukazuje, že výrezy z toho istého obrázka väčšinou zdieľajú rovnaké farebné histogramy. Zároveň sa farebné histogramy líšia medzi obrázkami. Preto, keď sa kontrastívna úloha spolieha iba na náhodné výrezy z typu augmentácií, dá sa to väčšinou vyriešiť zameraním sa iba na farebné histogramy. Výsledkom je, že reprezentácia nie je motivovaná k uchovávaniu informácií nad rámec farebných histogramov." [71].
Tento prístup sa objavuje aj pri iných augmentáciách a preto je dôležité ich kombi-

novať a nepoužívať jednotlivo. Vo všeobecnosti, kontrastívne učenie bude citlivé na systematické predsudky v datasete. Predpojatosť voči dátam (takzvaný bias) je v strojovom učení veľmi rozšíreným problémom (napriklad pri rozpoznávaní tváre pri ženách a mužoch alebo etnických menšinách) a pre kontrastívne metódy je to veľmi vážny problém. BYOL našťastie nezávisí od negatívneho vzorkovania, čo poskytuje únik od tohto problému [72].

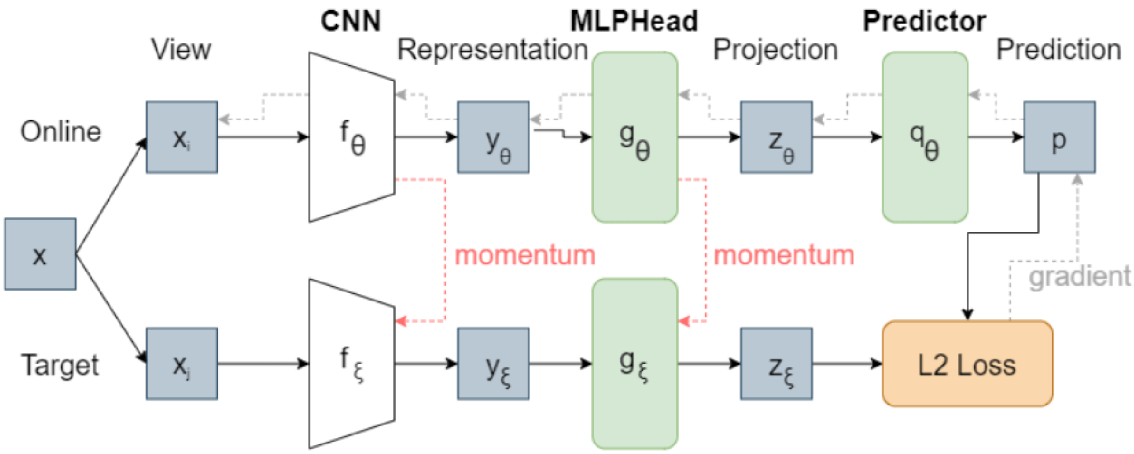
Cieľ BYOL je podobný kontrastívnemu učeniu, ale s jedným veľkým rozdielom. BYOL si nerobí starosti s tým, či majú rozdielne vzorky rozdielne reprezentácie (kontrastívna časť kontrastívneho učenia). Záleží iba na tom, aby podobné vzorky mali podobné reprezentácie. Môže sa to javiť ako jemný rozdiel, ale má to veľký dôsledok na efektívnosť a zovšeobecnenie tréningu v nasledujúcich bodoch:

1. Trénovanie je efektívnejšie, pretože BYOL nevyžaduje negatívne vzorkovanie. Vzorkuje sa každý tréningový príklad iba raz za epochu. Negatívne náprotivky možno úplne ignorovať.
2. Model je menej citlivý na systematické predsudky z tréningového datasetu. Zvyčajne to znamená, že lepšie zovšeobecňuje na príkladoch, ktoré nevidel.

BYOL minimalizuje vzdialenosť medzi reprezentáciami augmentácií vzoriek. Medzi príklady augmentácií patria: translácia, rotácia, rozmazanie, inverzia farieb, kolísanie farieb, gaussov šum atď. Väčšinou sa trénuje pomocou niekoľkých rôznych typov augmentácií, ktoré je možné aplikovať naraz alebo nezávisle. Všeobecne platí, že ak by bolo potrebné, aby bol model pri konkrétnej augmentácií invariantný, je potrebné ho zahrnúť do svojho tréningu.

Metóda BYOL obsahuje dve identické CNN, takzvané enkódery. Úloha enkóderu je extrakcia vlastností a reprezentácii z obrázku. Prvá sieť, nazvaná ako Online sieť je trénovaná bežným spôsobom a jej váhy sa aktualizujú s každou tréningovou dávkou (batch). Druhá, označovaná ako "cieľová" (Target) sieť, sa aktualizuje pomocou exponenciálneho kĺzavého priemeru váh prvej enkóderovej siete (podobný princíp trénovania druhého enkóderu je aj v predošlej metóde MoCo). Počas tréningu je cieľovej sieti poskytnutá jedna verzia augmentovaných obrázkov a druhej sieti je poskytnutá druhá verzia augmentovaných obrázkov z tej istej dávky. Každá sieť generuje nízkodimenzionálne, latentné znázornenie svojich príslušných údajov, čiže tvorí vektory reprezentácií. Následne sa online sieť pokúša predpovedať výstup cieľovej siete pomocou viacvrstvového perceptronu (takzvaný prediktor). BYOL maximalizuje podobnosť medzi touto predpoveďou a výstupom cieľovej siete [72]. Príklad architektúry metódy BYOL s aplikovaním augmentácií, online siete, cieľovej siete, prediktoru a výpočtu gradientu a následného momentu je na obrázku Obr. 5.31.

Cieľom BYOL je naučiť sa reprezentáciu y_θ, ktorá sa potom dá použiť na downstream úlohy. Ako už bolo opísané, BYOL používa na učenie dve neurónové siete:



Obr. 5.31: Architektúra SSL metódy BYOL

online a cieľovú sieť. Online sieť je definovaná súpravou váh θ a skladá sa z troch stupňov: enkodér f_θ, projektor (MLPHead) g_θ a prediktor q_θ, ako je znázornené na obrázku Obr. 5.31 a 5.32. Cieľová sieť má rovnakú architektúru ako online sieť, ale používa inú sadu váh ξ a neobsahuje prediktor. **Úlohou prediktoru je predpovedať výslednú reprezentáciu obrázku, na ktorý bola aplikovaná iná augmentácia a prešiel cez cieľovú sieť.** Cieľová sieť poskytuje regresné ciele na trénovanie online siete a jej parametre ξ sú exponenciálnym kĺzavým priemerom online parametrov θ. Podľa úspešnosti predikcie z prediktoru sa následne upravia váhy oboch sietí [71].

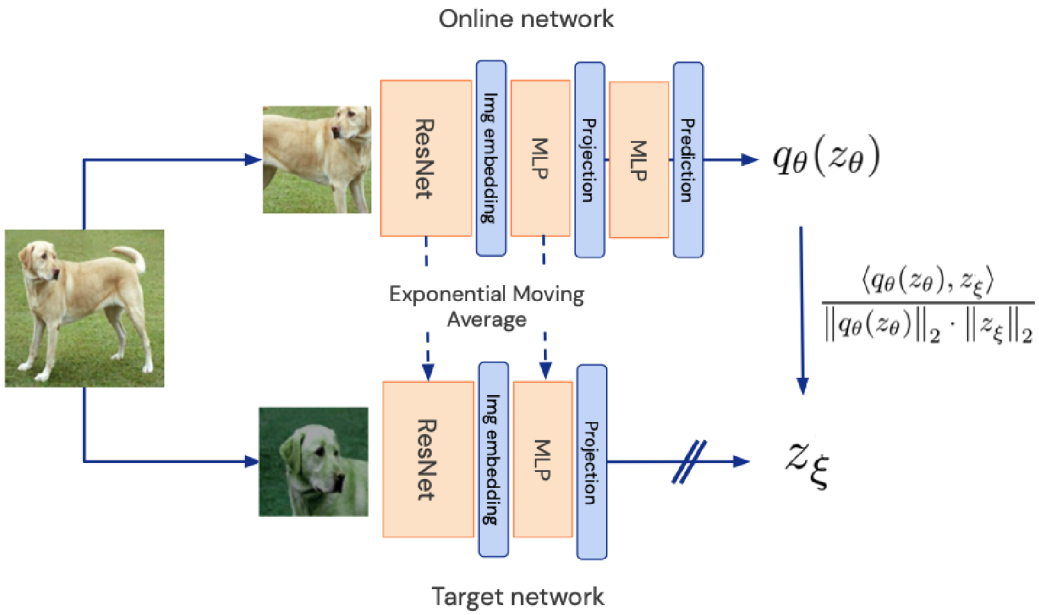
Výpočet chyby v BYOL sa nepočíta ako kontrastívna chyba medzi pozitívnymi a negatívnymi pármi augmentovaných obrázkov, ale ako stredná kvadratická chyba medzi normalizovanými predikciami z prediktora a cieľovej (target) siete [71]. Cieľom je priblížiť predikciu $q_\theta(z_\theta)$ ku reprezentácii z_ξ. Jej vzorec je nasledovný:

$$L_{\theta,\xi} = \|\overline{q_\theta}(z_\theta) - \overline{z_\xi}\|_2^2 = 2 - 2\frac{(q_\theta(z_\theta), z_\xi)}{\|q_\theta(z_\theta)\|_2 \cdot \|z_\xi\|_2} \tag{5.6}$$

Úprava váh cieľovej siete sa deje podobným spôsobom, ako pri metóde MoCo a to každým krokom postupne optimalizovať váhy cieľovej siete na základe váh online siete [71]. Vzorec sa počíta podľa nasledovného postupu:

$$\xi \leftarrow \tau\xi + (1 - \tau)\theta \tag{5.7}$$

Veľkosť τ môže priamo ovplyvniť optimalizáciu váh cieľovej sieti a tým aj presnosť celého trénovaného modelu online siete. Príklad aplikácie rôznych veľkostí τ je na obrázku Obr. 5.33 [71].



Obr. 5.32: Vizuálny príklad BYOL postupu [71]

Target	τ_{base}	Top-1
Constant random network	1	18.8±0.7
Moving average of online	0.999	69.8
Moving average of online	0.99	**72.5**
Moving average of online	0.9	68.4
Stop gradient of online†	0	0.3

Obr. 5.33: Porovnanie presnosti modelu predtrénovaného pri rôznych veľkostiach τ [71]

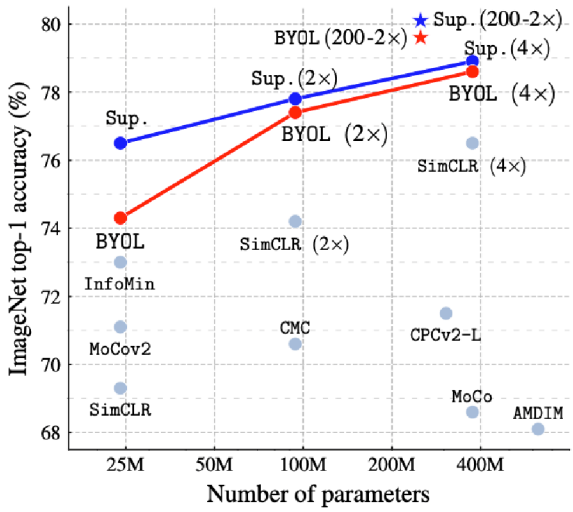
Na konci tréningu sa cieľová sieť zahodí. Tým pádom zostáva iba jediný enkóder, ktorý bol natrénovaný na generovanie konzistentných reprezentácií pre všetky príklady z tréningových dát a tým je enkóder f_θ z online siete. To je presne dôvod, prečo BYOL funguje veľmi dobre v oblasti trénovania pod vlastným učiteľom (self-supervised). Pretože naučené reprezentácie sú konzistentné, sú (väčšinou) invariantné pri rôznych transformáciách dát. Z toho vychádza, že podobné príklady majú podobné reprezentácie aj v trénovanom modely.

Metóda BYOL je jedna z najúčinnejších metód v oblasti učenia sa s vlastným učiteľom, čoho dôkazom sú aj porovnania s inými metódami, ktorých výsledky sú znázornené na Obr. 5.34. Na testovaní s ImageNet prekonáva SimCLR alebo MoCo

Method	Architecture	Param.	Top-1		Top-5	
			1%	10%	1%	10%
CPC v2 [32]	ResNet-161	305M	-	-	77.9	91.2
SimCLR [8]	ResNet-50 (2×)	94M	58.5	71.7	83.0	91.2
BYOL (ours)	ResNet-50 (2×)	94M	62.2	73.5	84.1	91.7
SimCLR [8]	ResNet-50 (4×)	375M	63.0	74.4	85.8	92.6
BYOL (ours)	ResNet-50 (4×)	375M	69.1	75.7	87.9	92.5
BYOL (ours)	ResNet-200 (2×)	250M	**71.2**	**77.7**	**89.5**	**93.7**

Obr. 5.34: Porovnanie metódy BYOL, SimCLR a CPC pri dotrénovaní na 1% a 10% označených dát [71]

a ďalšie metódy. BYOL dosahuje skoro rovnakú presnosť pri testovaní, ako učenie sa s učiteľom (supervised). Pri väčšom počte parametrov sa k tomuto učeniu skoro úplne rovná, čo môžeme vidieť na zobrazení Obr. 5.35 [71].



Obr. 5.35: Porovnanie SSL metódy BYOL s inými metódami a učením s učiteľom [71]

V skratke, BYOL sa snaží zbaviť negatívnych príkladov pri počítaní kontrastívnej chyby tým, že prediktor sa snaží predpovedať druhý augmentovaný obrázok na základe poznania prvého augmentovaného obrázku z ich dvojice. Je to kombinácia Momentum Contrast a SimCLR metódy s následným zmazaním negatívnych vzoriek. Rozdiel oproti metóde SimCLR je použitie troch CNN (online, cieľová a prediktor), aplikovanie kĺzavého priemeru na úpravu váh cieľovej siete a rozdiel oproti MoCo (aj SimCLR) je v nepoužití negatívnych párov pri kontrastívnom učení, ale aplikovanie strednej kvadratickej chyby.

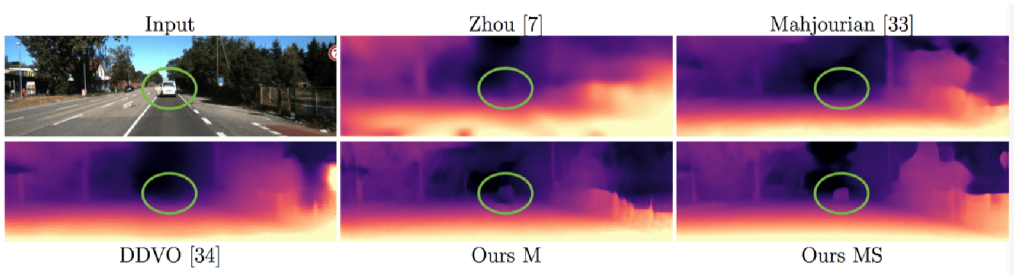
5.7 Projekty v praxi

Self-supervised učenie je relatívne mladá oblasť učenia, ktorej metódy sa iba posledné mesiace a roky dostávajú do výskumu a praxe. V tejto sekcii sa pokúsim priblížiť už praktické aplikovanie rôznych metód a ich úspešné výsledky v rôznych oblastiach.

5.7.1 Odhad hĺbky

Moderné autonómne autá a autonómne mobilné roboty potrebujú na to, aby mohli bezpečne a efektívne pracovať, dokonale porozumieť svojmu prostrediu. Pomocou senzorov je však vnímané iba obmedzené množstvo informácií pokiaľ ide o ich schopnosti, ako zorné pole a druh údajov, ktoré poskytujú. Zatiaľ čo snímače ako LIDAR, Radar a Kinect poskytujú 3D údaje vrátane všetkých priestorových rozmerov, kamery naopak poskytujú iba 2D pohľad na okolie. 3D snímače sú ale násobne drahšie ako kamery zachytávajúce 2D obraz. V minulosti bolo urobených veľa pokusov o skutočnú extrakciu 3D údajov z 2D obrázkov. Ľudský vizuálny systém je pri riešení tejto úlohy pozoruhodne úspešný, zatiaľ čo algoritmy veľmi často nedokážu rekonštruovať a odvodiť z obrazu hĺbkovú mapu.

Problém s odhadom 3D priestoru z 2D obrázku vyriešili nasledovne. Vedci z University College v Londýne trénovali hlbokú neurónovú sieť (deep learning) s vlastným učiteľom (self-supervised) z dôvodu nedostatku veľkého množstva obrázkov, ktoré by obsahovali hĺbku. Problém formujú ako syntézu pohľadu, kde sa sieť učí predpovedať cieľový obraz z pohľadu iného obrazu. Navrhovaná metóda je schopná poskytnúť odhad hĺbky, ak je daný iba jeden farebný obrázok na vstup. Princíp metódy pozostáva z enkodéru na základe ResNet architektúry, do ktorého vstupuje farebný obrázok, nasleduje dekóder, ktorý na výstupe generuje priradenú hĺbku, ktorá postupuje do ďalšej časti, ktorá odhaduje polohu predmetu, čiže jeho hĺbku. Táto metóda je schopná poraziť dnešné state-of-the-art metódy. Ich nový prístup ukazuje sľubné výsledky v úspešnom odhade hĺbky z 2D obrázkov. Porovnanie ich metódy (pravo dolu) s konkurenčnými je na obrázku Obr. 5.36 pre vstupný obrázok auta na ceste (ľavo hore). Dôležitou úlohou v rámci budúcnosti autonómnych robotov a áut je dokonalé porozumenie hĺbke prostredia, čo by vďaka takýmto metódam mohlo byť nahradené jednoduchými a relatívne lacnými kamerami v porovnaní s drahými 3D snímačmi. V neposlednom rade tento prístup opäť potvrdzuje silu hlbokého učenia a self-supervised metód [73].



Obr. 5.36: Porovnanie metód na odhad hĺbky [73]

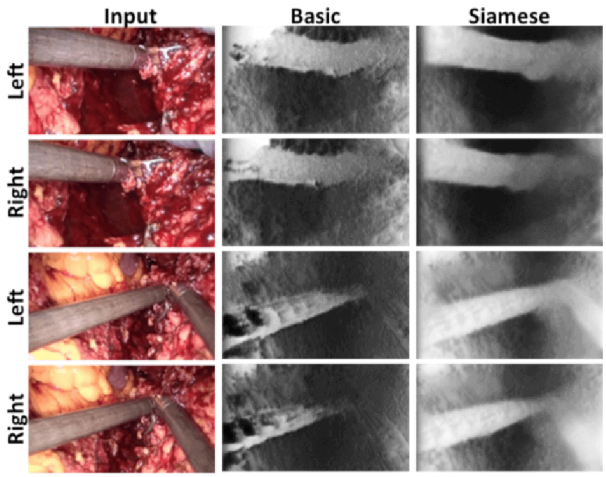
5.7.2 Medicína - robotická operácia a odhad hĺbky

Robotická chirurgia sa stala výkonným nástrojom na vykonávanie minimálne invazívnych procedúr a oproti tradičnej chirurgii poskytuje výhody v obratnosti, presnosti a 3D videní. Jedným z populárnych robotických systémov je chirurgická platforma da Vinci, ktorá umožňuje začleniť predoperačné informácie do živých procedúr pomocou rozšírenej reality (Augmented Reality (AR)). Odhad hĺbky scény je nevyhnutným predpokladom AR. Pokroky hlbokého učenia v poslednej dobe umožnili odhad hĺbky pomocou technológie Konvolučných neurónových sietí (CNN), ale trénovanie vyžadovalo veľký súbor obrazových údajov s hĺbkou ako základnou pravdou (ground-truth). Vedcom sa v tomto projekte podarilo využiť SSL trénovanie pre presnejší odhad hĺbky chirurgickej scény. Rovnako ako v predošlom projekte, aj tu použili autoenkodér na predikciu hĺbky a diferencovateľného priestorového transformátora na výcvik autoenkóderu na dvojiciach obrázkov bez základnej pravdy [74]. V aplikácii metódy ako hlavnú CNN využili architektúru siamskej siete a príklad aplikácie ich metódy je na obrázku Obr. 5.37 v pravo.

V ich projekte použili knižnicu Torch, dataset obsahujúci 20 000 obrázkov, ktoré mali rozlíšenie 196x196 a model natrénovali na 12 GB NVidia Titan X GPU. Experimenty uskutočňované s natrénovaným modelom na videách zhromaždených počas robotickej chirurgie ukázali, že ich prístup predpovedá presný odhad hĺbky a prekonáva štandardné stereofónne prístupy. Ich prístup nevyžaduje počas trénovania poznať anotácie/label hĺbky v obrázku a poskytuje tak vynikajúcu použiteľnosť na rozsiahle spracovanie videa, kde známe hĺbky nie sú k dispozícii [74].

5.7.3 Pohyb robotickej ruky na základe od pozerania pohybov ľudí z videí

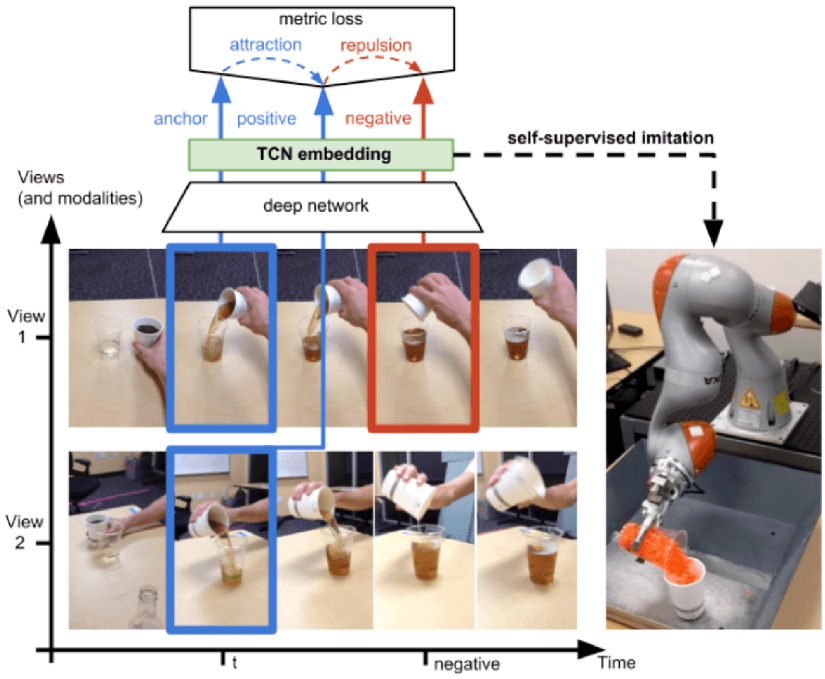
Vedci v tomto projekte navrhujú využitie self-supervised učenia na učenie reprezentácií a robotického správania výlučne z neoznačených videí (unsupervised) zaznamenaných z viacerých pohľadov. Následne sa model z týchto videí učí, ako možné



Obr. 5.37: Ukážka a porovnanie robotickej operácie [74]

použiť jeho ruku v dvoch nastaveniach robotickej imitácie: napodobňovanie interakcií objektov z videí ľudí a napodobňovanie ľudského správania. Imitácia ľudského správania si vyžaduje hľadisko, ktoré zachytáva vzťahy medzi koncovými efektormi (rukami alebo uchopovačmi robotov) a prostredím, atribútmi objektov a pózami tela. Ich model sa súčasne učí rozpoznávať, čo je spoločné medzi rozdielne vyzerajúcimi obrázkami a čo sa líši medzi podobne vyzerajúcimi obrázkami. Takáto metóda umožňuje, že ich model objaví atribúty, ktoré sa nemenia naprieč uhlom pohľadu, ale menia sa v priebehu času, pričom ignoruje premenné, ako sú rozmazanie pohybom, osvetlenie a pozadie [75].

Ako hlavnú sieť na trénovanie použili Časovo kontrastné siete (Time-Contrastive Networks - TCN). Spôsob učenia je podobný kontrastívnemu. V tomto prípade sa vytvárajú fotky predmetov a aktivít, ktoré sa má robot naučiť v rôzne časе a z rôznych uhlov. Pozitívny pár obrázkov je ten, ktorý je vytvorený v rovnakom čase, ale z rôznych pohľadov a negatívny pár voči tomuto obrázku je ten, ktorý je vytvorený z rovnakého uhľa pohľadu (rovnakou kamerou) ale v inom čase. Model sa sám učí, keď má vyriešiť úlohu (pochopiť podobnosť): “Čo je rovnaké medzi dvoma (modrými) obrázkami, ktoré vyzerajú inak? Čo je rozdielne medzi podobne vyzerajúcimi (červenými) obrázkami?” Príklad zobrazenie tohto princípu učenia je na obrázku Obr. 5.38. Ukazuje sa, že tento prístup môže robot použiť na priamu imitáciu ľudských pozícií bez priameho ovládania a že ho možno použiť aj ako funkciu odmeny v rámci reinforcement algoritmov. Projekt ponúka úspešné závery v napodobňovaní pohybu človeka robotom napríklad pri nalievaní, kedy sa robot naučil vykonávať pohyb iba z pozorovania tretej osoby. S pridaním funkcií odmeňovania z metód reinforcement learning je možné dokázať ešte lepšie výsledky [75].



Obr. 5.38: Ukážka princípu fungovania SSL algoritmu a robotickej ruky [75]

6 Praktická časť

Vzhľadom na širokú ponuku self-supervised metód, rozhodol som sa v mojej práci vyskúšať 4 metódy. Pri výbere metód som sa prioritne pozeral na úspešnosť a potenciál jednotlivých metód s cieľom dosiahnuť, čo najlepší výsledok. Rozhodol som sa konkrétne pre 1 metódu z oblasti uvažovania zdravého rozumu (commom sense reasoning), ktorou je pretext úloha rotácie a následne 3 kvalitnejšie a zložitejšie metódy z oblasti kontrastívneho učenie (contrastive learning), ktorými sú SimCLR, MoCov2 a BYOL. Ako vhodnú úlohu na test funkčnosti týchto metód, som sa rozhodol hlavne venovať úlohe klasifikácie a to z nasledujúcich dôvodov:

Po prvé úloha klasifikácie je jedna zo základných a hlavných úloh, pre ktorú jednotlivé metódy boli vytvorené a taktiež vedecky testované. Môžem teda vďaka tomuto testu následne porovnať výsledky v mojej práci s výsledkami tvorcov metód.

Po druhé úloha klasifikácie umožňuje univerzálne použitie pri rôznych datasetoch a taktiež s rôznymi metódami, či už z SSL oblasti alebo učenia s učiteľom.

Po tretie vďaka univerzálnosti je možné vyskúšať test pri rôznych parametroch trénovania, testovania a vstupných dátach s rôznym počtom obrázkov alebo tried a to v relatívne krátkom čase vzhľadom na obmedzený výpočtový výkon a čas venovaný tejto práci (týždne/mesiace).

Ako druhej úlohe, v ktorej som aplikoval SSL metódy, je úloha sémantickej segmentácie. V nej používam predtrénovanú Resnet sieť pomocou SSL metód, ako enkóder a následne trénujem spolu s dekóderom s cieľom naučiť sa segmentačné masky podľa vstupného datasetu. Úloha segmentácia je bližšie vysvetlená a popísaná v kapitole 8.

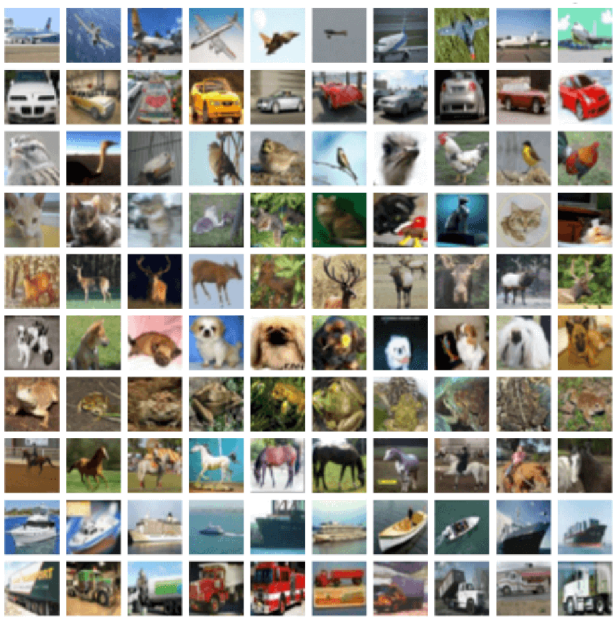
V tejto kapitole popisujem a odôvodňujem výber datasetov, ich parametre, veľkosť a zdroj.

6.1 Datasety

Na overenie správnosti a funkčnosti self-supervised metód, ktoré som popísal a následne naprogramoval je dôležité aj vybrať vhodné datasety. Keďže jedna z hlavných potrieb pri trénovaní self-supervised modelov je veľkosť datasetu, čo sa týka počtu obrázkov, rozhodol som sa v prvom rade pre datasety STL10, CIFAR10 a MNIST, keďže všetky obsahujú tisíce obrázkov. Zároveň SSL metódy chcem otestovať aj na menších datasetoch s objemom stoviek obrázkov v datasetoch. Pre oblasť sémantickej segmentácie som zvolil dataset CityScapes. Všetky datasety, ktoré používam v mojej diplomovej práci sú verejne dostupné. Aplikáciu augmentácií a prípravu datasetov pre prácu s CNN popisujem v nasledujúcej kapitole a v popísaní jednotlivých SSL metódach.

6.1.1 CIFAR 10

CIFAR10 je jeden z najznámejších datasetov na svete. Dá sa použiť ako súčasť knižnice Pytorch ale taktiež stiahnuť a pracovať s jednotlivými obrázkami. Dataset pozostáva zo 60 000 obrázkov, ktoré majú rozlíšenie 32x32 pixelov a skladajú sa z 10 tried, čiže 6 000 obrázkov na jednu triedu. Dataset sa skladá z 50 000 obrázkov na tréning CNN a 10 000 obrázkov na test natrénovaného modelu CNN. Dataset sa skladá z nasledujúcich 10 tried: lietadlo, automobil, vták, mačka, jeleň, pes, žaba kôň, loď a nákladné auto [76]. Takto vyzerá príklad obrázkov z datasetu:



Obr. 6.1: Zobrazenie príkladov z datasetu CIFAR10 [76]

6.1.2 STL10

Dataset STL10 taktiež patrí ku datasetom vhodným na rozpoznanie obrazu a klasifikačnú úlohu. Je ideálnym datasetom pre self-supervised tréning lebo obsahuje až 100 000 obrázkov ktoré nemajú triedu/label. Následne obsahuje 5000 obrázkov s označením triedy na tréning a 8000 obrázkov na test natrénovaného modelu v rovnomernom rozložení pre každú triedu. Pri dotrénovaní modelov lineárnou klasifikáciou aplikujem dotrénovanie za použitia 1%, 2%, 3%, 4%, 5% a 10% dát s označením triedou (label). Tieto dáta sú pomer k všetkým neoznačeným dátam. Čiže dotrénovanie lineárnej klasifikácie 1% dát s labelom je 1 000 obrázkov, keďže počet neoznačených dát je 100 000. Dataset STL10 bohužiaľ neponúka viac ako 5 000 obrázkov s

označením na tréning a z toho dôvodu som sa rozhodol dataset trénovací a testovací spojiť, vytvoriť tak 13 000 obrázkov s označením a následne prerozdeliť podľa potreby v danej trénovacej/testovacej úlohe. Dataset sa skladá z 10 tried: lietadlo, automobil, vták, mačka, jeleň, pes, kôň, opica, loď a nákladné auto. Veľkosť obrázkov je 96x96 pixelov [77]. Príklad datasetu je veľmi podobný ako CIFAR10 a z toho dôvodu nepridávam fotky z datasetu STL10.

6.1.3 MNIST

Dataset MNIST (Modified National Institute of Standards and Technology database) je veľký verejne dostupný dataset pozostávajúci z ručné písaných čísiel, ktorý sa často používa na úlohy v spracovaní obrazu a iných úloh v strojovom učení. Tento dataset som si stiahol vo verzii obrázkov vo formáte png a skladá sa z 10 tried (pre každé číslo 0-9 jedna trieda). Dáta som rozdelil do dvoch skupín na trénovacie a testovacie. Trénovacia skupina obsahuje 60 000 obrázkov (6000 na triedu) a testovacia 10 000 obrázkov (1 000 na triedu). Veľkosť obrázkov je 28x28 pixelov. Príklad výzoru obrázkov v datasete je na obrázku Obr. 6.2.

Obr. 6.2: Zobrazenie príkladov z datasetu MNIST [78]

6.1.4 Imagenete

Ďalším z datasetov, ktorý som použil na trénovanie modelov je jeden z najväčších a najznámejších na svete a to dataset Imagenet. Tento dataset bežne obsahuje 1 281 167 obrázkov na tréning, 50 000 na validáciu a 100 000 na test modelu pri 1000

triedach. Ja som sa pre obtiažnosť práce a obmedzenom výpočtovom výkone na takto veľkom datasete rozhodol pracovať s 10 triedami (kostol, pes, ryba, golfové palice, parašutista, pumpa, kazeta, píla, odpadky a trúbka). Veľkosť trénovacieho datasetu som zvolil 10 000 obrázkov a testovacieho 4 000 obrázkov. Veľkosť obrázkov je 224x224 pixelov (v prípade niektorých s inou veľkosťou, ich na túto veľkosť transformujem).

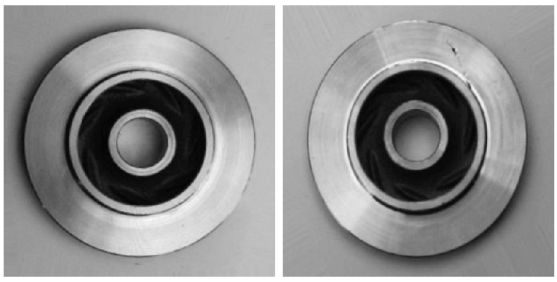
6.1.5 Weather

Dataset Weather je ďalším datasetom, s ktorým som trénoval SSL modely. Nazval som ho takým spôsobom vzhľadom na to, že obsahuje typy počasí. Dataset sa skladá z piatich tried/počasí (zamračené, hmla, dážď, slnečno a východ slnka). Počet obrázkov na tréning je 1200 a na testovanie 250. Dataset je verejne dostupný na stránke kaggle [79]. Obrázky s ktorými pracujem majú veľkosť 300x200 pixelov. Príklad výzoru obrázkov je na obrázku Obr. 6.3.

Obr. 6.3: Zobrazenie príkladov z datasetu Weather

6.1.6 Industry circles

Posledným datasetom použitým na úlohu klasifikácie je technicky zameraný dataset, ktorý obsahuje 2 typy krúžkov, podľa toho, či krúžok je poškodený alebo nie. Krúžky pochádzajú z fotiek odlievania, kde pri tvorbe môže prísť k poškodenému alebo zle vytvorenému odliatku, medzi poškodenia patria dierky, praskliny, metalurgické chyby atď. Dataset obsahuje dokopy 7348 obrázkov, ktoré sú rozdelené na 2 hlavné skupiny pre tréning (6633) a test (715). Obrázky sú vo veľkosti 300x300 pixelov. Príklad oboch typov obrázkov je na obrázku Obr. 6.4. Dataset je verejne dostupný na stránke kaggle [80].



Obr. 6.4: Zobrazenie príkladov z datasetu Industry circles (na ľavo správne, na pravo poškodené)

6.1.7 CityScapes

Dataset s názvom CityScapes používam na úlohu sémantickej segmentácie v spojení so Self-supervised tréningom modelov. Tento dataset obsahuje fotky z 50 Nemeckých miest. Na fotkách je zvyčajne vidieť cesta, chodci, cyklisti, stromy, autá, budovy, značky a semafór. Dokopy má tento dataset 8 tried, ktoré obsahujú dokopy 30 podtried. V úlohe sémantickej segmentácie budem pracovať iba s 8 triedami (nerozlišujem typ vozidla alebo typ budovy). Fotky boli vytvorené počas dňa a za dobrých podmienok počasia. Dataset obsahuje 2975 obrázkov na tréning a 500 obrázkov na test modelu. Každý obrázok pozostáva z originálnej fotky a obrázku, ktorý obsahuje label/triedy. Finálne rozmery sú teda 256x512px.

Obr. 6.5: Príklad obrázku z datasetu Cityscapes [81]

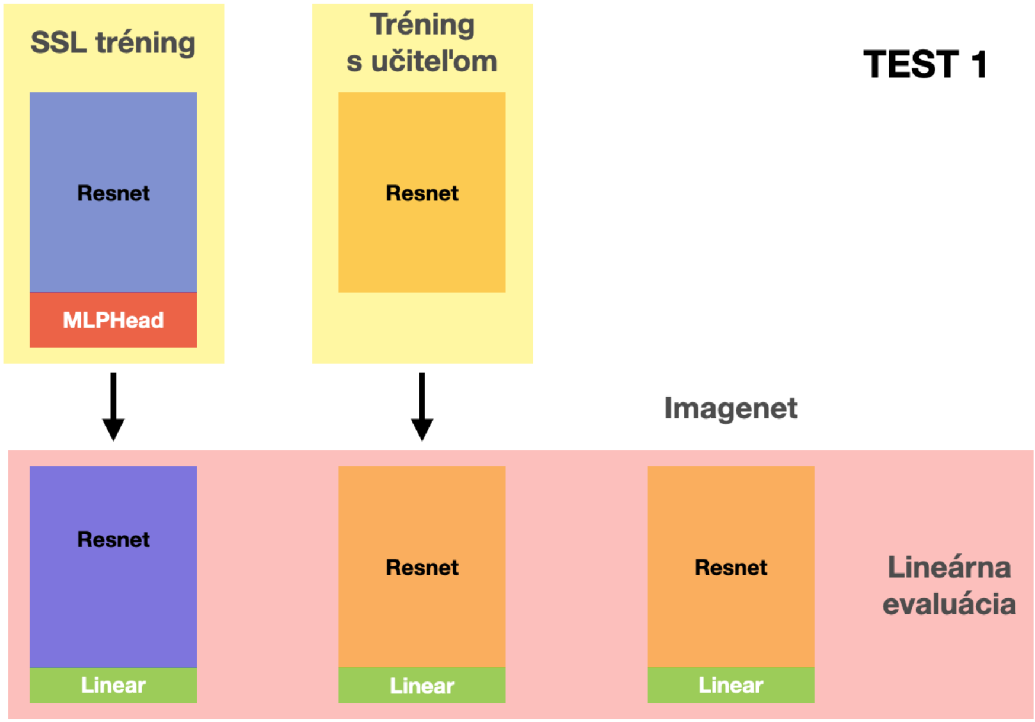
7 Klasifikácia

V kapitole klasifikácie píšem o spôsobe aplikovania jednotlivých SSL metód, vyhodnotení jednotlivých metód zvlášť a následne aj medzi sebou a zároveň aj medzi rôznymi datasetmi. Kapitola je štruktúrovaná do ôsmich častí, kedy v prvej píšem všeobecný postup, ktorý sa aplikuje pri všetkých metódach z dôvodu univerzálnosti kódu, ktorý som vytvoril. Rovnako v prvej časti popíšem testy, ktorými vyhodnocujem metódy. V druhej až piatej podkapitole vysvetľujem rozdiely pri aplikovaní SSL metód v kóde a tiež ich jednotlivo vyhodnocujem. Šiestu podkapitole venujem trénovaniu s učiteľom. V siedmej kapitole vyhodnocujem SSL metódy na datasete STL10 pri rôznych parametroch a v poslednej ôsmej podkapitole porovnávam SSL metódy pri úlohe klasifikácie na rôznych datasetoch.

7.1 Postup tvorby kódu, tréningu a vyhodnotenia metód

Počas tvorby mojej diplomovej práce som naprogramoval niekoľko funkcií a spôsobov ako dosiahnuť požadované výsledky aplikácie SSL metód. Spočiatku som každú metódu programoval zvlášť a jedinečným postupom držania sa vedomostí a výsledkov z odborných výskumov daných metód. Keď boli metódy úspešne naprogramované a odtestované, rozhodol som sa vytvoriť, čo najviac univerzálny kód, ktorý je v určitých častiach trénovania týchto metód zhodný. Týmto prístupom nemusím popisovať rovnaký/podobný kód niekoľko krát znovu v každej metóde a preto v tejto kapitole popíšem celkový postup tvorby kódu, prípravy datasetov, trénovania modelov a následného vyhodnotenia, ak ide o časť kódu, ktorá je identická naprieč SSL metódami. Tento postup som do finálnej časti diplomovej práce aplikoval na všetky 4 metódy a rozdiely, ktoré metódy v určitých častiach obsahujú (architektúra modelov, spôsob výpočtu kontrastívnej chyby...) popíšem bližšie v ďalších podkapitolách venovaných vždy konkrétnej metóde. Celý postup trénovania a vyhodnotenie metód sa dá rozdeliť na 3 typy testov, ktoré som stručne pomenoval TEST1, TEST2, TEST3 a TEST4. Prvý TEST1 zobrazený na obrázku Obr. 7.1. pozostáva z počiatočného trénovania Resnet architektúry s projekčnou hlavou (Multilayer projection head - MLPHead) SSL metódou, paralelné trénovanie rovnakej architektúry s rovnakým datasetom ako pri SSL metóde, ale aj s označenými triedami metódou trénovania s učiteľom. Po týchto tréningoch dochádza k lineárnej evaluácii, v ktorej sa vyhodnocuje úspešnosť a porovnanie modelov vzatím pôvodnej predtrénovanej architektúry a pridaním lineárnej vrstvy na jej koniec s tým, že iba táto záverečná lineárna časť môže byť trénovaná. Porovnávam 3 modely a to predtrénovaný SSL metódou, predtrénovaný

s učiteľom a predtrénovaný s učiteľom na datasete Imagenet (tento posledný model je možné stiahnuť priamo z knižnice pytorch a rozhodol som sa ho zahrnúť do porovnania z dôvodu použitia Imagenet modelu pri vedeckých publikáciách SSL metód).



Obr. 7.1: Príklad postupu trénovania a vyhodnotenia metód spôsobom TEST1

Kód som sa rozhodol písať v programovacom jazyku Python, ktorý ponúka celosvetovo používané knižnice v oblasti strojového a hlbokého učenia. Python taktiež umožňuje rýchle a efektívne skriptovanie a jednoduchšiu prácu s dátami a tvorby neurónových sietí. Zároveň je to rýchlo rastúci jazyk, čo sa týka veľkosti komunity programátorov na celom svete.

Celkovo je kód náročný na výpočtový výkon a neurónové a konvolučné siete sa zvyčajne trénujú na grafických kartách (GPU), ktoré umožňujú rýchlejšie trénovať model ako robiť výpočet na procesore (CPU). Z tohto dôvodu kód programujem a následne model spúšťam na platforme Google Colab, kde Google ponúka zadarmo trénovanie na Tesla K80 GPU na 12 hodín. Ja som sa rozhodol pre platenú verziu Google Colab Pro za 9,99$ mesačne, ktorá ponúka GPU Nvidia Tesla T4 a P100 na dobu trénovania 24 hodín. Kód sa programuje priamo v platforme a píše sa do blokov za sebou. Výhodou je, že programátor môže rôzne časti kódu po menšej úprave v rôznom poradí spúšťať a testovať, čo som mnohokrát využil. Takto napísaný kód má koncovku ".ipynb". Google priamo ponúka prepojenie platformy Colab s cloud úložiskom Google Drive. Túto možnosť som využil a tak kód aj dáta trénujem a

ukladám online na cloud, čo mi umožňuje trénovať v rôznych časoch, z rôznych miest a zároveň si byť istý, že dáta aj kód sú v bezpečí na cloude [82].

Univerzálny postup aplikovaný vo všetkých 4 metódach je nasledovný:

1. Import knižníc
2. Definícia premenných, tvorba súborov na uloženie modelov a grafov
3. Úprava datasetu, aplikácia augmentácií a tvorba dataloadru
4. Tvorba modelu, výber GPU, kontrola načítania predtrénovaného modelu
5. Trénovanie SSL modelu
6. Tvorba funkcií na vyhodnotenie presnosti a úspešnosti modelov
7. Trénovanie modelu s učiteľom (Supervised)
8. Načítanie predtrénovaného SSL modelu a tvorba lineárnej vrstvy nad SSL modelom
9. Tréning a test SSL modelu s lineárnou vrstvou
10. Lineárna evaluácia a vyhodnotenie úspešnosti modelov pri rôzne veľkom datasete na dotrénovanie
11. Porovnanie trénovania s učiteľom SSL modelu a nepredtrénovaného modelu

7.1.1 Import knižníc:

Ako jednu z hlavných knižníc používam *torch*. Knižnica Torch alebo tiež PyTorch patrí k najpoužívanejším verejne dostupným knižniciam na prácu s neurónovými sieťami, počítačovým videním a spracovaním jazyka na svete. Torch bola prvý krát predstavená v 2016 a vytvorili ju vo vedeckom tíme firmy Facebook. Z tejto knižnice používam funkcie na tvorbu siete Resnet a lineárneho modelu, aplikovanie augmentácií na dataset a prácu s dátami, komunikáciu s GPU, načítavanie predtrénovaného modelu, jeho trénovanie a mnohé ďalšie [83].

Ďalšou veľmi potrebnou knižnicou je *sklearn*. Pomocou tejto knižnice vyhodnocujem kvalitu modelu a jeho presnosť hlavne vďaka funkciám na tvorbu Precision Recall a ROC kriviek a Chybovej matice (Confusion Matrix).

Dôležitou knižnicou na zobrazenie grafov a obrázkov je *matplotlib*. Na prácu s maticami používam knižnice *numpy*, *pandas* a *seaborn*. Priebežné informácie o tréningu a úspešnosti modelov neustále zaznamenávam do textového súboru pomocou knižnice *logging*. Prácu s uloženými dátami uskutočňujem efektívne a ľahko vďaka knižniciam *os*, *pathlib* a *glob*. Na zmeranie rýchlosti tréningu jednej epochy a celého trénovania modelu používam knižnicu *time* a pre priebežný výpočet zostávajúceho času trénovania a zobrazenie percenta natrénovaných dát používam knižnicu *tqdm*.

7.1.2 Definícia premenných, tvorba súborov na uloženie modelov a grafov:

Vzhľadom na dĺžku času tvorby, trénovania a vyhodnotenia modelov, ktorá môže byť v rádoch desiatok minút až hodín, som sa rozhodol v úvode kódu vytvoriť originálny názov pre súbor, ktorý bude obsahovať všetky potrebné dáta po natrénovaní a vyhodnotení modelov pri konkrétnych vstupných parametroch. Takýto názov súboru pozostáva z názvu použitého datasetu, typu použitej hlavnej architektúry, počtu trénovacích epoch a veľkosti dávky použitej na trénovanie SSL modelu. Následne takýto súbor vytvorím použitím knižnice *pathlib* a doňho ukladám všetky súbory z daného tréningu. Pri tvorbe súboru sa kontroluje, či už v tejto ceste neexistuje rovnaký súbor, a ak áno, nevytvára sa nový, ale použijú sa dáta už z predvytvoreného na dotrénovanie modelov.

Ďalšie premenné, ktoré treba v úvode definovať sú:

- názov použitej architektúry
- názov datasetu
- veľkosť dávky (batch) použitej na: trénovanie SSL modelu, evaluáciu a modelu na trénovanie s učiteľom
- dĺžka tréningu v epochách na trénovanie: SSL modelu, evaluácie a modelu na trénovanie s učiteľom
- veľkosť a počet neurónov pre vrchnú časť neurónovej siete SSL modelu a počet neurónov pre MLPHead
- parametre na definíciu optimizer (veľkosť učiaceho kroku, momentu a úbytku (weight_decay))
- názov a počet tried použitého datasetu

7.1.3 Úprava datasetu, aplikácia augmentácií a tvorba dataloadru:

Môj finálny kód diplomovej práce som sa pokúsil vytvoriť tak, aby bolo možné použiť akýkoľvek dataset určený na klasifikáciu dát. Stačí určiť cestu k datasetu a následne definovať meno datasetu a program už načítanie trénovacích/testovacích dát, rozdelenie dát na trénovanie pre SSL model a zvlášť pre model s učiteľom a taktiež prípravu dát vo veľkosti iba niekoľkých percent na dotrénovanie SSL modelu pri evaluácii spraví sám. Prácu s datasetom robím dvomi metódami, prvá je ak je dataset na disku v jednotlivých súboroch a druhá metóda je, ak sťahujem dataset z knižnice *torchvision.datasets*. Je to z toho dôvodu, že ideálne pripravený dataset STL10 sa mi nepodarilo nájsť na verejnom úložisku s možnosťou stiahnuť obrázky ako JPEG, JPG alebo PNG, rovnako ako napr. CIFAR10 alebo iné data-

sety. Naopak táto druhá metóda otvára dvere pre všetky verejne dostupné datasety z knižnice *torchvision.datasets*. Prakticky je teda možné pracovať s akýmkoľvek datasetom určeným na klasifikáciu, či už z disku alebo z knižnice. V nasledujúcich riadkoch popíšem prácu s datasetom CIFAR10, ktorý sa nachádza na disku.

Na načítanie určitého zlomku zo všetkých dát, napríklad 10% používam funkciu *random.sample* a počet vstupných parametrov je podiel všetkých názvov obrázkov daným číslom, čiže 10.

```
datapath_own = "/content/drive/MyDrive ... datasets/classification"
datapath = os.path.join(datapath_own, dataset_name)

if dataset_name == "CIFAR10":
    train_names = sorted(glob.glob(datapath+'/train/*/*.png',
        recursive=True))
    test_names = sorted(glob.glob(datapath+'/test/*/*.png',
        recursive=True))
    NUM_OUTPUT_CLASS = 10
    shape = 32
    class_names = ["airplane", "automobile", "bird", "cat" ... "truck"]

random.seed(0)
names_train = random.sample(train_names, len(train_names))
names_test = random.sample(test_names, len(test_names))
names_train_10_percent = random.sample(train_names, len(train_names) // 10)

# getting labels based on filenames
labels_train = [x.split('/')[10] for x in names_train]
labels_test = [x.split('/')[10] for x in names_test]

# these X % labels will be used for training the linear classifer
labels_train_10_percent = [x.split('/')[10] for x in names_train_10_percent]

# DATASETS
training_dataset_rotated = TrainDatasetFiles(names_train,
                                labels_train, rotate=True)
training_dataset_10      = TrainDatasetFiles(names_train_10_percent,
                                labels_train_10_percent, rotate=False)

# DATALOADERS
train_loader_rotated  = DataLoader(training_dataset_rotated,
                                   batch_size=Rotation_BATCH,
                                   num_workers=num_workers,
```

```
                                    drop_last = False,
                                    shuffle=True)
train_loader_10       = DataLoader(training_dataset_10,
                                    batch_size=Rotation_BATCH,
                                    num_workers=num_workers,
                                    drop_last = False,
                                    shuffle=True)
```

Keď mám vybrané aké názvy obrázkov do daného modelu chcem načítať, použijem triedu *TrainDatasetFiles*, ktorej inicializačné parametre sú mená obrázkov, ich label a parameter typu bool, či sa má spraviť na obrázkoch augmentácia. Na základe posledného parametra sa aj pripravené datasety neskôr použijú. Ak bude aplikovaná augmentácia, budú použité na trénovanie SSL modelu, ak nebude použitá augmentácia, budú použité na trénovanie modelu s učiteľom alebo finálnu evaluáciu a dotrénovanie zlomkom dát s označením. Trieda *TrainDatasetFiles* dedí z triedy *Dataset*, ktorá je priamo importovaná z knižnice *torch*. Z toho dôvodu je nutné prepísať 3 dôležité funkcie: `__init__`, `__len__` a `__getitem__`. Vo funkcii `__init__` inicializujem názvy premenných, ktoré v triede *TrainDatasetFiles* budem používať. Vo funkcii `__len__` vypočítam dĺžku datasetu s ktorým trieda a funkcie budú pracovať a funkcia `__getitem__` umožňuje načítať postupne každý obrázok a aplikovať naňho žiadanú augmentáciu alebo transformáciu do tensoru a následne vrátiť upravený obrázok. Príklad kódu pre jednotlivé funkcie uvediem pri bližšom popísaní SSL metód, keďže prístupy k augmentácii sú rôzne.

Pri práci s datasetom stiahnutým z knižnice *torchvision.datasets*, konkrétne ide o dataset STL10, používam 2 rôzne prístupy aplikovania augmentácií. Prvý je pri metóde Rotácie, kedy používam mierne upravenú triedu *TrainDatasetFiles*. Tým, že túto triedu použijem iba na augmentáciu obrázkov bez označenia tried, tak nepotrebujem vytvárať možnosť práce s dátami pre následný tréning s učiteľom, ako v predošlom príklade. Rovnako ale musím definovať 3 dôležité funkcie: `__init__`, `__len__` a `__getitem__`. Bližší popis použitia tejto funkcie na tvorbu augmentácií popíšem v nasledujúcich kapitolách venovaných práci s datasetom v SSL metóde rotácie. Pri tvorbe augmentácií u kontrastívnych metód, aplikujem augmentácie priamo pri stiahnutí datasetu. Tento rozdiel oproti metóde rotácie je v tom, že pri rotácii potrebujem každý obrázok uchopiť, orotovať a vrátiť orotovaný obrázok aj s označením počtu rotácii, čiže triedou pre úlohu rotácie, zatiaľ čo pri kontrastívnych metódach mi stačí aplikovať augmentácie na vstupný obrázok a vytvoriť pozitívny augmentovaný pár.

Výstupom predošlej funkcie sú datasety, ktoré použijem ako vstupný parameter do importovanej funkcie z knižnice *torch*, ktorou je *Dataloader*. Táto funkcia rozdelí

daný dataset do dávok (batch), premieša dáta a pripraví ich na použitie trénovania modelu CNN.

Prípravu trénovacích a testovacích dát s označením zo stiahnutého datasetu STL10 robím nasledovne. Na stiahnuté dáta aplikujem iba transformáciu do tensoru a následne pomocou funkcie *random_split()* rozdelím dataset v určitom pomere. Napríklad pri neskoršom dotrénovaní pomocou lineárnej vrstvy budem používať dáta vo veľkosti 1% z pôvodných trénovacích dát, čiže v tejto funkcii musím, ako vstupný parameter nastaviť dataset a druhý parameter pomer rozdelenia. Pri trénovacom datasete, ktorý obsahuje 5000 obrázkov s triedami by upravený dataset s 1% ku pôvodným 100 000 trénovacím dátam vyzeral nasledovne (zobrazujem stiahnutie datasetu, rozdelenie na pomer 1% a príprava dataloaderu):

```
# STL10 - TRAIN - 5000 images (500 per 1 class)
train_dataset = datasets.STL10("/content/drive/MyDrive ... datasets..",
        split='train', download=True,
        transform=data_transforms_supervised)

# 1% labeled images, 100 per class
train_dataset_1, _ = torch.utils.data.random_split(train_dataset, [1000, 4000])

train_loader_1 = DataLoader(train_dataset_1,
        batch_size=LINEAR_CLASSIFIER_BATCH,
        num_workers=4, drop_last=False, shuffle=True)
```

Takto pripravený `train_loader` ukladám do poľa a pri dotrénovaní SSL modelu postupne aplikujem rôzne veľké datasety a pozorujem zmeny a nárast presnosti pri evaluácii.

7.1.4 Tvorba modelu, výber GPU, kontrola načítania predtrénovaného modelu:

Ako som už v teoretickej časti napísal, ako hlavnú kostru modelu, ktorý je trénovaný s augmentovanými obrázkami používam architekrúru Resnet vďaka možnosti práce s reziduálnymi blokmi. Knižnica *torch* ponúka mnoho architektúr už postavených a tým pádom uľahčuje proces iba jednoduchým stiahnutím architektúry. Tiež umožňuje rovno stiahnuť predtrénovanú architektúru na datasete Imagenet. To ale nie je môj prípad a ja sťahujem architektúru s nepredtrénovanými váhami. Takáto architektúra na svojom konci obsahuje plne spojenú neurónovú vrstvu, ktorú nepotrebujem, keďže plánujem model zväčšiť o projekčnú hlavu. Túto projekčnú hlavu, čiže ďalšiu CNN vytvorím pomocou mojej triedy *MLPHead*. Táto trieda vytvorí 2

konvolučné vrstvy s aktivačnou funkciou ReLU a dávkovou normalizáciou po prvej vrstve. Hlavnú triedu, cez ktorú sa sťahuje architektúra Resnet a následne pridáva projekčná hlava som nazval *ResNet* a spolu s triedou *MLPHead* dedia z knižnice *torch* funkciu *Module* a z toho dôvodu nutne musia obsahovať aj mnou napísanú funkciu *forward()*, pomocou ktorej definujem prepojenia vrstiev konvolučnej siete. Na následnom príklade ukazujem časť kódu triedy ResNet a v nej volanie tvorby projekčnej hlavy.

```
class ResNet(torch.nn.Module):
    def __init__(self, network_name):
        super(ResNet, self).__init__()
        if network_name == 'resnet18':
            resnet = models.resnet18(pretrained=False)
        elif network_name == 'resnet34':
            resnet = models.resnet34(pretrained=False)
        elif network_name == 'resnet50':
            resnet = models.resnet50(pretrained=False)

        self.encoder = torch.nn.Sequential(*list(resnet.children())[:-1])
        # build Projection head with input = output of resnet
        self.projection = MLPHead(
                IN_channels = resnet.fc.in_features,
                mlp_HIDDEN_size  = projection_head_mlp_hidden_size,
                OUT_channels     = projection_head_projection_size)

    # Defines the computation performed at every layer
    def forward(self, x):
        h = self.encoder(x)                  # input to Resnet
        h = h.view(h.shape[0], h.shape[1])  # transform it
        return self.projection(h)            # output to projection head
```

V celom kóde a jednotlivých metódach som sa rozhodol pracovať s Resnet18 z dôvodu dostatočnej veľkosti architektúry a zároveň z dôvodu dĺžky tréningu. Väčšie CNN obsahujú viac parametrov a tréning by netrval niekoľko hodín, ale desiatky hodín až dní a pri teste štyroch metód spolu s rôznymi vstupnými parametrami, by nebolo možné otestovať všetko. CNN Resnet18 zároveň ponúka dostatočnú presnosť a kvalitu. Po vytvorení modelu kontrolujem, či je možné trénovať na GPU, aby sa tréning uskutočnil rýchlejšie, ako na CPU. Výberom zariadenia a uložením do premennej *device* presuniem aj vytvorený model do tohto zariadenia. Následne vytvorím *optimizer*, ktorý je zodpovedný za vyladenie parametrov CNN s cieľom minimalizovať funkciu chyby (smerovať váhy takým smerom, aby funkcia prišla do

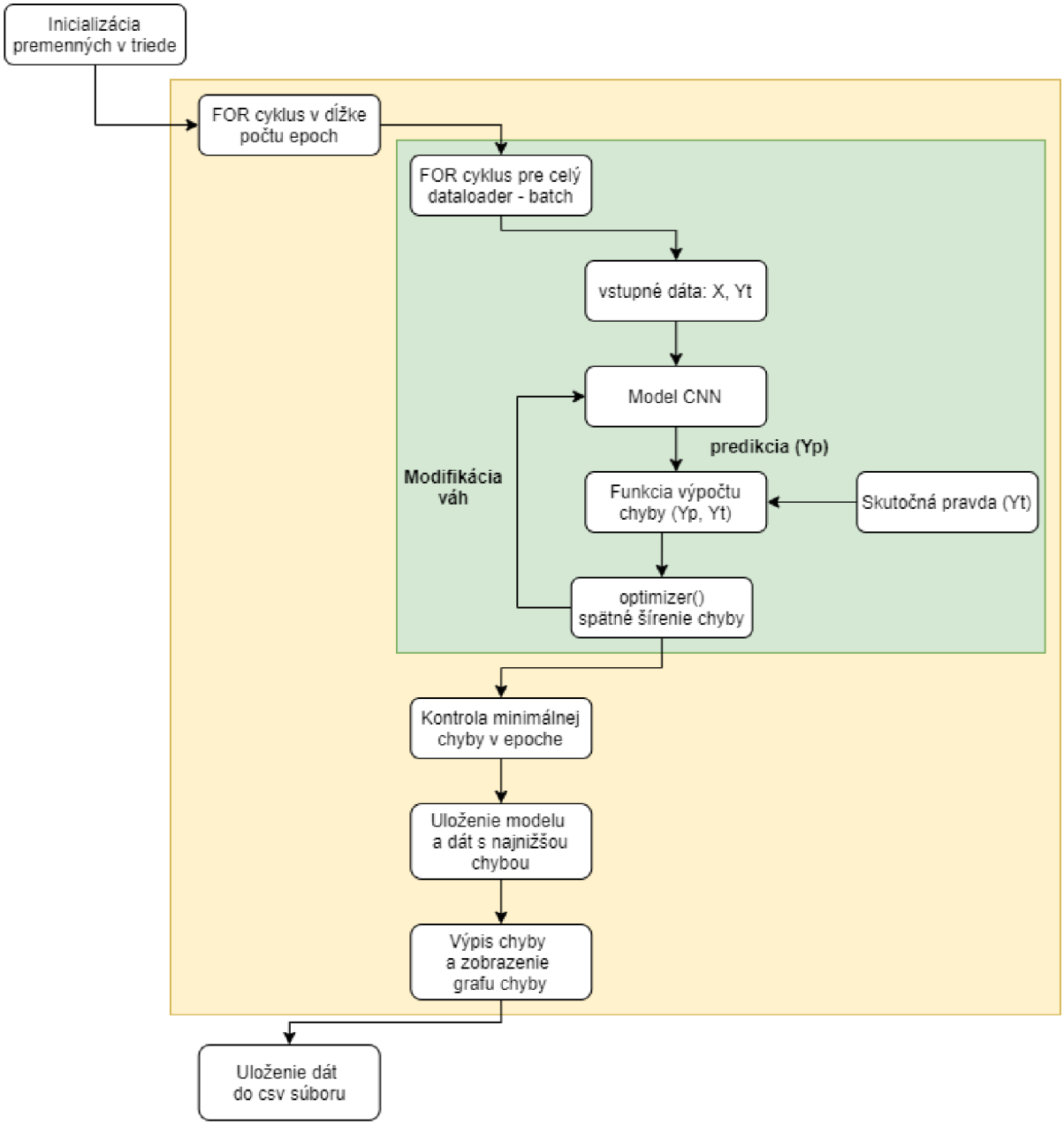
globálneho minima). Ja som sa rozhodol používať *optimizer SGD with momentum* (stochastic gradient descent with momentum). Tento optimizer aplikuje momentum, ktoré pomáha pri odšumení gradientov pri derivovaní a tým urýchľuje čas potrebný na nájdenie globálneho minima. Príklad kódu vykonávajúceho vyššie popísane kroky je nasledovný:

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
Rotation_parameters = list(rotation_network.parameters())
optimizer = torch.optim.SGD(Rotation_parameters,
                lr = Rotation_opt_lr,
                momentum = Rotation_opt_momentum,
                weight_decay = Rotation_opt_weight_decay)
```

Vzhľadom na to, že trénovanie veľkých modelov (veľký dataset, dĺžka trénovania a veľký počet epoch) môže trvať niekoľko hodín, pridal som možnosť priebežného ukladania modelov počas trénovania a následne, ak by došlo k akémukoľvek problému (odpojenie od internetu, reštart počítača...) model bude v nasledujúcom tréningu pokračovať z kroku, kde v predošlom tréningu skončil. Ak sa nájde takýto čiastočne trénovaný model v cieľovom súbore, načítam jeho váhy, optimizer, epochu v ktorej skončil a hodnoty chyby do poľa. Následne už budem pracovať s týmito predtrénovanými hodnotami.

7.1.5 Trénovanie SSL modelu:

Trénovanie modelu sa uskutočňuje vo funkcii, ktorá patrí triede modelu. Každému SSL modelu vytváram objekt, ktorý obsahuje inicializáciu, funkciu výpočtu chyby predikovanej hodnoty od skutočnej pravdy alebo kontrastívnu chybu, funkciu na trénovanie modelu a funkciu na ukladanie modelu s optimizerom a poslednou trénovanou epochou. V princípe celý algoritmus trénovania je všeobecne popísaný na obrázku Obr. 7.2 (zelená farba značí priebeh jednej epochy, žltá farba priebeh trénovania všetkých epoch). Počas trénovania priebežne po každej epoche vypisujem veľkosť chyby a dĺžku času trénovania danej epochy. Každých N epoch (N je možné vybrať ľubovoľne veľké) sa vykreslí graf so zmenou chyby počas trénovania. Po natrénovaní modelu sa uloží model s najnižšou chybou, vykreslí sa graf chyby celého tréningu a najnižšia dosiahnutá chyba trénovania s príslušnou epochou danej chyby. Bližší a presnejší popis tréningu každej metódy popíšem v kapitolách venovaných metódam.



Obr. 7.2: Všeobecný algoritmus trénovania SSL modelu

7.1.6 Tvorba funkcií na vyhodnotenie presnosti a úspešnosti modelov:

Vyhodnocovanie presnosti vykonávam podľa metód spomenutých v teoretickej časti tejto diplomovej práce. Funkcie na vyhodnotenie presnosti a úspešnosti modelov sa v mojom kóde nazývajú: *report_matrix_curves* a *plot_ROC_PreRec*. Vyhodnocujem presnosť modelu počas trénovania a finálny model následne vyhodnocujem s confusion matrix, precision, recall, skóre F1, ROC krivkou a plochou pod

touto krivkou (AUC) a krivkou precision_recall a tiež plochou pod touto krivkou. Keďže zväčša vykonávam úlohu klasifikácie niekoľkých tried (pri datasete STL10, CIFAR10 a MNIST je to 10 tried), musím na výsledok z modelu aplikovať funkciu *label_binarize* z knižnice *sklearn*. Pomocou tejto funkcie dostanem z predikovanej triedy pole o dĺžke počtu tried a číslom 1 na indexe prislúchajúce danej triede (príklad - ak je vstupná predikovaná trieda číslo 5, výstup funkcie bude [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] pre dataset s 10 triedami). Tento proces vykonám pre každú predikciu triedy a následne z dát (predikcií a konvertovaných predikcií funkciou *label_binarize*) vypočítam ROC krivku, FPR (false positive rate), TPR (true positive rate) a AUC (area under curve) pre každú triedu datasetu. Tieto dáta ukladám do poľa a následne všetky krivky vykreslím v jednom grafe. Identický proces aplikujem aj na výpočet krivky precision_recall a hodnôt precision a recall. Z nich následne vypočítam priemernú presnosť daného modelu funkciou *average_precision_score*. Výsledky pre každú triedu rovnako zobrazím do spoločného grafu.

7.1.7 Trénovanie modelu s učiteľom (Supervised):

Napriek tomu, že moja diplomová práca sa zameriava na metódy SSL (self-supervised learning, trénovanie s vlastným učiteľom) a pri následnom dotrénovaní posledných vrstiev pri evaluácií sa dá hovoriť o SmSL (semi-supervised learning, trénovanie s čiastočným učiteľom) robím tréning modelu aj pri učení s učiteľom (supervised learning). Je to z toho dôvodu, aby som pri vyhodnocovaní úspešnosti SSL modelov a ich dotrénovaní pri rôznych veľkostiach dát s označeniami tried mohol porovnať aj s modelmi trénovanými s učiteľom (obdobne to robili aj tvorcovia metód pri ich testovaní). Pre takýto model rovnako vytváram dataset a následne dataloader, ako som popísal pri príprave dát pre trénovanie SSL modelov. V tomto prípade robím minimálne úpravy vstupného datasetu (orezanie na rovnakú veľkosť, transformovanie do tensoru a normalizovanie). Ak model trénujem na datasete STL10, ktorý obsahuje 13 000 obrázkov s označením, rozdelím ich na trénovacie (10 000) a testovacie (3 000). Pre tento typ trénovania vytváram zvlášť súbor, do ktorého sa uloží model, grafy a informácie ku trénovaniu s učiteľom. Ako vhodný trénovací krok som zvolil po experimentoch veľkosť 0.01 s veľkosťou momentu 0.9 pre optimizer. Ako funkciu výpočtu chyby som zvolil *CrossEntropyLoss()*. Taktiež som aplikoval *lr_scheduler()*, ktorý zabezpečuje zmenšovanie veľkosti tréningového kroku vynásobením 0.1 každých N epoch (zníženie na 10% z predošlej hodnoty, vďaka čomu môže začať pri väčšej hodnote a pomaly klesať ku globálnemu minimu). Na trénovanie modelu s učiteľom som vytvoril podobnú triedu, ktorá obsahuje inicializáciu premenných, trénovanie a testovanie modelu a ukladanie všetkých dát ako pri trénovaní SSL modelu. Jediný rozdiel je, že pri tomto type trénovania poznám aj označenia vstupných dát

triedami a môžem po každej epoche testovať daný model. Z toho dôvodu po každej epoche vyhodnocujem chyby aj úspešnosť modelu, ktoré priebežne vypisujem spolu s dĺžkou času trénovania. Model s najlepšou úspešnosťou ukladám do určeného súboru a po ukončení trénovania najlepší model vrátim z funkcie von.

7.1.8 Načítanie predtrénovaného SSL modelu a tvorba lineárnej vrstvy nad SSL modelom:

Po natrénovaní SSL modelu, treba otestovať jeho natrénované vlastnosti. Podľa postupu vedcov a programátorov v ich publikáciách postupovali rovnako a to načítaním takéhoto modelu a nad ním postavenie lineárnej/lineárnych vrstiev s cielom klasifikovať dáta do správnych tried. V úvode trénovania vytváram potrebnú architektúru, v mojom prípade Resnet. Do tejto architektúry následne načítam už predtrénované váhy z SSL modelu s rovnakou architektúrou. Odstránim MLPHead a pridám novú hlavu - lineárnu vrstvu/vrstvy novej CNN. V mojom testovaní som sa rozhodol spraviť aj pokus a to zväčšiť počet lineárnych vrstiev novej hlavy CNN a porovnanie ich úspešnosti pri aplikovaní jednej, dvoch alebo troch vrstiev. Tento test som nazval TEST2 a jeho príklad aplikácie je na obrázku Obr. 7.3.

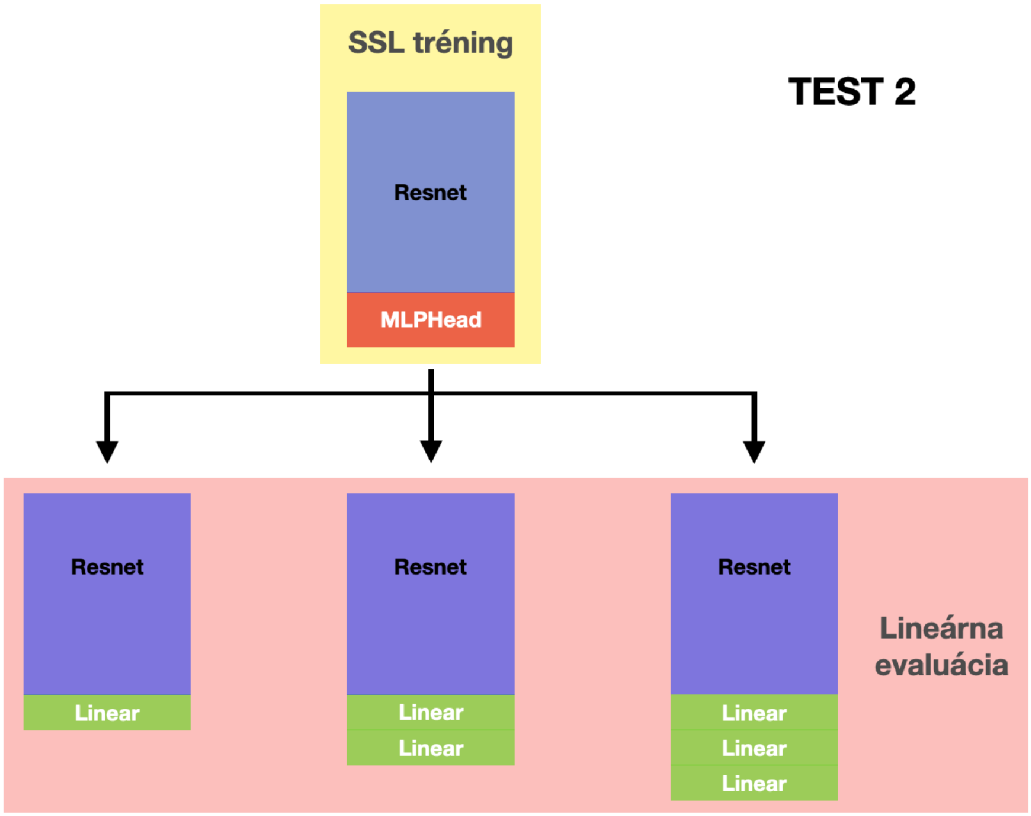
Tvorbu novej hlavy, lineárnej vrstvy na predtrénovaný SSL model tvorím pomocou triedy *LinearNet*. Vstupný počet neurónov tejto siete je rovný počtu výstupných neurónov siete SSL modelu a výstupný počet neurónov lineárnej CNN je závislý od počtu tried datasetu. Ak používam dve alebo tri lineárne vrstvy (*LinearNet2, LinearNet3*), tak počet neurónov vo vnútri tejto menšej CNN je vždy polovica predošlého počtu neurónov v predošlej vrstve. Aktivačná funkcia v jednotlivých vrstvách týchto väčších CNN je *ReLU*.

```
class LinearNet(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearNet, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        return self.linear(x)

class LinearNet3(torch.nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(LinearNet3, self).__init__()

        self.linear1 = torch.nn.Linear(input_dim, hidden_dim)
        self.linear2 = torch.nn.Linear(hidden_dim, int(hidden_dim/2))
```



Obr. 7.3: Zobrazenie postupu aplikovania rôznej veľkosti lineárnej hlavy pri evaluácii

```
        self.linear3 = torch.nn.Linear(int(hidden_dim/2), output_dim)

    def forward(self, x):
        x = self.linear1(x)
        x = F.relu(x)
        x = self.linear2(x)
        x = F.relu(x)
        x = self.linear3(x)
        return x
```

Ako optimizer na tréning siete som zvolil *Adam* vzhľadom na jeho možnosť adaptovať sa a upraviť parametre počas trénovania - hľadania globálneho minima. Z toho dôvodu som mohol použiť aj menší trénovací krok o veľkosti 0.0003. Funkciu výpočtu chyby medzi predikciou a skutočnou pravdou som rovnako ako v predošlom prípade zvolil *CrossEntropyLoss()*. Príklad časti kódu z načítania modelu, odstránení vrchnej časti architektúry, tvorby novej konvolučnej vrstvy a nového optimizer s funkciou chyby je nasledovný:

```
encoder = ResNet(network_name)
```

```
output_feature_dim = encoder.projection.net[0].in_features

if (rotation_model):
    print("[INFO] Loading pretrained model...")
    load_params = torch.load(os.path.join(results_path_training_of_model,
            'Rotation_model.pth'),
            map_location = torch.device(torch.device(device)))
    if 'Rotation_Net_state_dict' in load_params:
        encoder.load_state_dict(load_params['Rotation_Net_state_dict'])
        print("[INFO] Parameters successfully loaded.")
    loss_name = str(index) + "_train_loss"
    acc_name = str(index) + "_test_acc"

encoder = torch.nn.Sequential(*list(encoder.children())[:-1])
encoder = encoder.to(device)

linear_classifier = LinearNet(output_feature_dim, NUM_OUTPUT_CLASS)
linear_classifier = linear_classifier.to(device)

optimizer = torch.optim.Adam(linear_classifier.parameters(), lr=3e-4)
criterion = torch.nn.CrossEntropyLoss()
```

7.1.9 Tréning a test SSL modelu s lineárnou vrstvou:

Keďže pri tréningu a teste modelu s lineárnou vrstvou už nebudem trénovať a upravovať váhy predtrénovaného SSL modelu, rozhodol som sa jeho predikcie vytiahnuť a použiť. Pri tomto kroku vychádzam z princípu funkcie CNN, kedy na vstup príde obrázok, postupuje cez jednotlivé vrstvy a neuróny až na záver, kde posledná vrstva na základe odporúčaní od predošlých neurónov generuje výstupný vektor reprezentácií alebo definuje typ obrázku pre danú triedu v závislosti od počtu neurónov a typu poslednej vrstvy. Keďže určovanie konkrétnej triedy obrázku má robiť nová lineárna vrstva, ktorej váhy sa ešte dotrénujú, tak váhy už natrénovaného SSL modelu môžu svoju predpoveď k obrázku vopred generovať. Vytvoril som funkciu *get_features_from_encoder()*, ktorej parametre sú predtrénovaný model (encoder) a dataset pripravený na tréning a test (dataloader). Tento pripravený dataloader obsahuje dáta v jednotlivých dávkach zložených z obrázku (image) a triedy (label). Keďže SSL model nebudem trénovať, chcem od neho zistiť, "čo si myslí o danom obrázku", inými slovami, obrázok prejde cez celý SSL model až na poslednú vrstvu, kde sa vytvorí vektor vlastností, ktorý neskôr bude vstupným vektorom pre lineárnu časť, ktorej váhy sa ešte môžu upraviť. Tento vektor ukladám do poľa *X_train* a

na rovnaký index do poľa *y_train* aj jeho skutočnú triedu. Pole jednotlivých vektorov ešte štandardizujem pomocou funkcie *preprocessing.StandardScaler()*, ktorou normalizujem hodnoty vektoru (odstránim priemer, vycentrujem hodnoty na 0 a vydelím smerodatnou odchýlkou). Je to zaužívaný postup pri strojovom učení s cieľom dosiahnuť lepšiu úspešnosť modelu. Z takto predspracovaných dát vytvorím dataloader, ktorý dáta premieša a vytvorí dávky s veľkosťou 64.

```
def get_features_from_encoder(encoder, loader):
    x_train = []
    y_train = []

    # get the features from the pre-trained model
    for (_, sample_batched) in enumerate(loader):
        x = sample_batched['image']
        y = sample_batched['label']
        with torch.no_grad():
            feature_vector = encoder(x)
            x_train.extend(feature_vector)
            y = [int(i) for i in y]
            y = torch.Tensor(y)
            y_train.extend(y.numpy())

    x_train = torch.stack(x_train)
    y_train = torch.tensor(y_train)
    return x_train, y_train

x_train, y_train  = get_features_from_encoder(encoder, train_loader)
x_test, y_test    = get_features_from_encoder(encoder, test_loader)
```

Takto pripravené dáta obsahujúce vektory vlastností obrázkov môžem použiť na dotrénovanie lineárnej vrstvy modelu. Trénovanie sa deje podľa bežného princípu, ktorý je aj na obrázku Obr. 7.2 načrtnutý s miernym rozdielom. Lineárny model na svoj vstup nedostáva čisto obrázok, ale už vektor vlastností, ktorý prešiel cez model predtrénovaný SSL metódou. Tento spôsob mi ponúka veľkú výhodu a to použiť akúkoľvek SSL metódu na predtrénovanie aj s iným datasetom a následne dotrénovať lineárnu vrstvu/vrstvy s cieľom klasifikácie obrazov do tried.

7.1.10 Evaluácia a vyhodnotenie úspešnosti modelu:

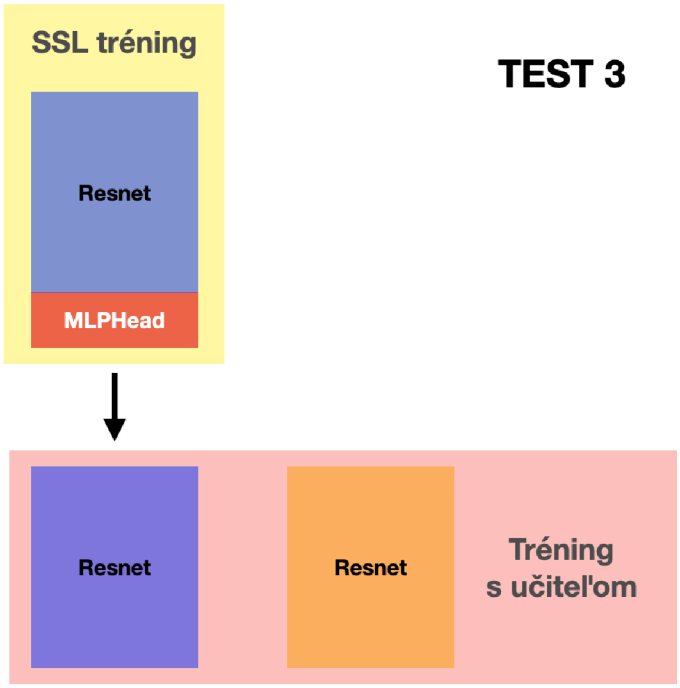
Úspešnosť SSL modelu spolu s dotrénovanou lineárnou hlavou vyhodnocujem rovnakými spôsobmi, ako model trénovaný učiteľom - priebežná úspešnosť porovnania

predpovedaných tried so skutočnými, confusion matrix, precision, recall, skóre F1, ROC krivkou a plochou pod touto krivkou (AUC). Bližšie vysvetlené v predošlej podkapitole. Výsledky ukladám zvlášť do súboru patriaceho iba lineárnej evaluácii. Keďže cieľom tejto práce je skúsiť dotrénovať model s rôzne veľkým datasetom obsahujúcim dáta s triedami, tak na dotrénovanie lineárnej vrstvy používam niekoľko datasetov a to zväčša datasety obsahujúce správne označenia tried pre 1%, 2%, 3%, 4%, 5% alebo 10% dát. Úspešnosť všetkých takto dotrénovaných modelov ukladám do zoznamu a po dokončení zobrazím v jednom grafe pre porovnanie. Pri vyhodnotení modelov používam nasledovné vstupné parametre pri trénovaní a vyhodnocovaní: datasety rôzne, architektúra Resnet18, dĺžka SSL tréningu o veľkosti 50, 100, 300 a 500 epoch s veľkosťou dávky 50, 100, 300 a 500. Maximálny počet epoch 500 som zvolil vzhľadom na dĺžku trénovania. Pri niektorých tréningoch trvá výpočet jednej epochy 5 minút. Čas, ktorý môžem využiť na trénovanie v platforme Colab je obmedzený. Veľkosť dávky som zvolil na 500 vzhľadom na obmedzený výpočtový výkon. Pri niektorých datasetoch (s menšími rozmermi obrázkov) bolo možné trénovať aj na 600 a pri iných už nie. Rozhodol som sa pre 500, aby bolo možné porovnať rôzne datasety medzi sebou. Ak som sledoval zmenu presnosti pri zmene veľkosti epoch, tak som nastavil veľkosť dávky na 200 a naopak pri sledovaní zmeny presnosti pri zmene veľkosti dávky som epoch nastavil na 50.

7.1.11 Porovnanie trénovania s učiteľom SSL modelu a nepredtrénovaného modelu:

Jedna z možností ako preukázať efektivitu SSL metód je dotrénovanie jednej (alebo viacerých) lineárnych vrstiev, pri zamrazení predtrénovaných váh/alebo ich extrakcii a nezmenení pri spätnej propagácii gradientu. Ďalšia možnosť je využitie predtrénovaných váh na následné trénovanie s učiteľom v celom rozsahu architektúry. V tomto prípade je použitý celý model a nie je aplikovaná žiadna extrakcia vlastností, ako v predošlom prípade. Tento test som nazval ako TEST3 a aplikujem v ňom tréning s učiteľom na model, ktorý bol predtrénovaný SSL metódou. Príklad postupu je zobrazený na obrázku Obr. 7.4.

Vytvorím nový model s rovnakou architektúrou, ako pri SSL tréningu, do tohto nového modelu načítam uložené váhy z SSL tréningu (100 epoch, 200 dávka) a odstránim nepotrebnú projekčnú hlavu. Takýto model trénujem s učiteľom, funkciou chyby *CrossEntropyLoss()*, trénovacím krokom o veľkosti 0.01 a rovnakým optimizerom, ako pri trénovaní s učiteľom. Dataset a dataloader v tomto type tréningu je rovnaký, ako pri tréningu modelu s učiteľom (čiže vo veľkosti 10 000 trénovacích a 3 000 testovacích obrázkov s označením v prípade datasetu STL10). Vlastnosti na trénovanie modelu sú teda rovnaké, ako pri trénovaní prázdneho modelu s učiteľom.



Obr. 7.4: Zobrazenie postupu trénovania s učiteľom SSL model a prázdny model

Každú zo štyroch SSL metód, ktorých princíp aplikujem v tejto práci som otestoval aj takýmto spôsobom a výsledky vykreslené v grafoch zobrazujem pri vyhodnotení každej z metód v ich kapitolách. Z výsledkov sa dá zistiť, či model, ktorý už podobné dáta videl, ale nepoznal ich triedy, bude pri následnom trénovaní lepší a presnejší, ako model, ktorý dáta nevidel a je trénovaný od počiatku.

Posledný TEST4 je tiež trénovanie s učiteľom predtrénovaného SSL modelu a nepredtrénovaného, ale s použitím iba 1% alebo 5% dát s označením. Cieľom testu je poukázať na efektivitu SSL metód, ktoré vďaka predtrénovaniu a následnému identickému tréningu, ako konkurenčný model, dosiahnu vyššie výsledky presnosti. Výsledky pre TEST4 sú zobrazené v tabuľke v danej kapitole venovanej konkrétnej SSL metóde.

7.1.12 Výsledky testovania modelov:

Predtrénované SSL modely na rôznych veľkostiach dávok a epoch (50, 100, 300, 500) následne dotrénujem s 1 lineárnou vrstvou a výsledky zapisujem do tabuliek Tab. 7.1, Tab. 7.4, Tab. 7.8, Tab. 7.12 vyhodnotenia jednotlivých metód (TEST1). Príklad vývoja presnosti SSL modelu predtrénovaného na 100 epochách a dávke 200 je na grafoch Obr. 7.5, Obr. 7.9, Obr. 7.12, Obr. 7.15. V tabuľkách Tab. 7.2, Tab. 7.5, Tab. 7.9, Tab. 7.13 porovnávam presnosť SSL modelov dotrénovaných s 1 lineárnou

vrstvou (50 epoch, 500 dávka) a vyhodnocujem ich úspešnosť pomocou presnosti, AUC, PR plochy a F1 skóre. Výsledky sú v komparácii medzi modelmi trénovanými na 1% a 5% dát s označením. Ako hodnotu AUC a plochy PR som sa rozhodol spraviť priemer hodnôt pre všetky triedy v danom trénovaní, nakoľko výsledok AUC a plochy PR sú vždy pre každú triedu zvlášť, čiže pri jednom teste dostanem výsledok pre 10 tried. Aritmetický priemer som sa rozhodol použiť aj preto, lebo počet obrázkov v triedach je rovnomerný. SSL model s lineárnym modelom dotrénovaným 1 až 3 vrstvami, 50 epochami a veľkosťou dávky 500 je zobrazený na grafoch Obr. 7.6, Obr. 7.10, Obr. 7.13, Obr. 7.16 a výsledky patria do testu TEST2. Model s učiteľom je trénovaný vždy časťou datasetu obsahujúcou označenia tried, dĺžkou 50 epoch a veľkosťou dávky 32. Porovnanie trénovania SSL modelu a nepredtrénovaného modelu touto metódou je TEST3 a výsledky sú v tabuľkách Tab. 7.1, Tab. 7.4, Tab. 7.8, Tab. 7.12 v posledných dvoch stĺpoch. Vývoj presnosti trénovania týchto modelov je na grafoch Obr. 7.7, Obr. 7.11, Obr. 7.14, Obr. 7.17. Posledný spôsob otestovania modelov (TEST4), čiže trénovanie s učiteľom SSL model a nepredtrénovaný model na 1% a 5% dát je v tabuľkách Tab. 7.3, Tab. 7.6, Tab. 7.10, Tab. 7.14.

7.2 SSL metóda Rotácie:

Ako prvú metódu z oblasti self-supervised učenia som sa rozhodol aplikovať princíp rotácie na predtrénovanie modelu siete Resnet18. Ide o metódu, kde na vstup CNN príde orotovaný obrázok o 0, 90, 180 alebo 270 stupňov (poprípade aj iný uhol, ktorý môžem definovať) a model má predikovať o koľko stupňov je daný obrázok otočený. Následne sa podľa rozdielu predikcie uhlu rotácie a skutočnej pravdy uhlu rotácie vypočíta chyba a podľa nej sa upravia váhy modelu.

7.2.1 Príprava datasetu:

Pri príprave datasetu postupujem zhodne s krokmi, ktoré som všeobecne popísal v úvode kapitoly praktickej časti diplomovej práce. Po načítaní dát z disku alebo stiahnutí dát z knižnice používam triedy *RotationalTransform* a *TrainDatasetFiles*. Prvá spomenutá pri jej zavolaní má za úlohu otočiť tensor o počet stupňov, ktorý jej príde na vstup pri inicializácii. V mojej práci som sa rozhodol pre 4 typy rotácie a to o rotáciu 0, 90, 180 alebo 270 stupňov. Objekty tejto triedy so vstupmi 0, 90, 180 a 270 stupňov si uložím do poľa *class_transforms* pri inicializácii triedy *TrainDatasetFiles*. V tejto triede inicializujem aj dataset a počet tried datasetu, ktorý vyplýva z dĺžky poľa obsahujúceho objekty triedy *RotationalTransform*. Je to z toho dôvodu, lebo triedy pri tomto SSL tréningu nie sú skutočné triedy z datasetu (auto, pes, mačka...) ale počet rotácií do ľava, čiže 90 stupňov je 1 rotácia, 180 stupňov 2

rotácie a 270 stupňov 3 rotácie. Vo funkcii `__getitem__` triedy *TrainDatasetFiles* načítam obrázok funkciou *Image.open()* (patriacej do knižnice *PIL*) a prekonvertujem ho do RGB. Následne program náhodne vyberie počet rotácií o 90 stupňov funkciou *random.choice*, ktorej vstupný parameter je dĺžka poľa (N) obsahujúceho všetky možné rotácie a funkcia vráti číslo 0 až N do premennej *angle*. Pri práci s datasetom STL10, rovno pri jeho stiahnutí aplikujem transformáciu do tensorov. Pri práci s datasetom z disku musím túto transformáciu aplikovať v triede *TrainDatasetFiles*. Augmentáciu rotácie v oboch prípadoch robím rovnako a to zavolaním tej triedy zodpovednej za rotáciu, ktorá je na indexe rovnom počtu rotácií v premennej *angle*. Funkcia vracia augmentovaný obrázok, ktorý spolu s premennou *angle* je výstupným prvkom ukladaným do zoznamu z funkcie `__getitem__`. Zoznam je tvorený dvojicami, v prípade aplikovania rotácie [“image”, “angle”] a v inom prípade, ak ide iba o transformáciu do tensoru (prípad týkajúci sa datasetu načítaného z disku) vráti [“image”, “label”]. Ak pracujem s datasetom z disku, tak label je názov súboru, v ktorom je obrázok uložený. Príklad aplikovania náhodnej rotácie (v tomto prípade 180 stupňov, čiže angle = 2) na vstupný obrázok z datasetu STL10 je na obrázku Obr. A.1.

7.2.2 Tréning SSL modelu Rotácie:

Pred tréningom SSL modelu spravím inicializáciu objektu *RotationTrainer* s inicializovaním modelu, ktorý sa bude trénovať (*Rotation_Net*), optimizeru, typu zariadenia (väčšinou GPU), počiatočnej epochy *start_epoch* (táto premenná záleží od toho, či pokračujem v trénovaní už predtým trénovaného modelu a došlo k zastaveniu trénovania), dataloadru obsahujúceho pripravený dataset na tréning, funkcie na výpočet chyby (*loss_function*), ktorá je v tejto SSL metóde *CrossEntropyLoss()*.

Po inicializácií nasleduje zavolanie trénovacej funkcie *train()*. V jej úvode zapíšem do textového súboru obsahujúceho všetky priebežné informácie o programe správu o začatí trénovania, definujem potrebné premenné pre výpočet chyby, počítania dĺžky času a spustím FOR cyklus, ktorého rozsah začína úvodnou epochou *start_epoch* a končí epochou *max_epochs*, ktorej hodnotu definujem v úplnom úvode celého programu spolu s definíciou veľkosti dávok a epoch pre tréning SSL, s učiteľom a dotrénovanie pre lineárnu klasifikáciu. Keďže dataset je veľký a v jednej epoche nedokážem modelu ukázať všetky dáta, musím použiť druhý vnorený FOR cyklus, ktorý postupne dávku po dávke vloží do vstupu modelu a očakáva predikciu s následným výpočtom chyby od skutočnej pravdy a upravením váh. Pred vložením dát do modelu zapnem spätné šírenie gradientu. Po výpočte chyby a spätnému šíreniu volám optimizer na zefektívnenie hľadania globálneho minima a následne iteratívne ukladám chybu do premennej na neskorší výpočet chyby za poslednú epochu. Po

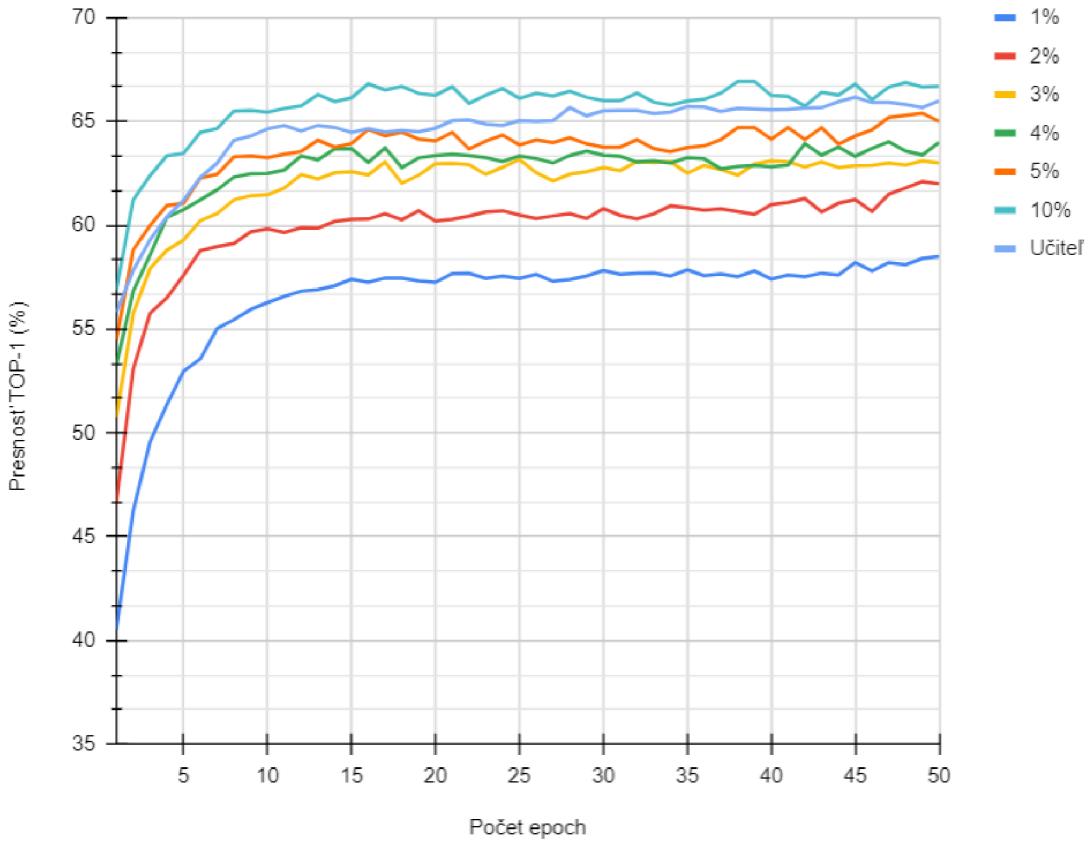
načítaní všetkých dát z dataloadru prejdem k výpočtu chyby za poslednú epochu a kontrole, či neprišlo k výpočtu nového minima, ak áno, tento model priebežne ukladám do súboru. Následne vypíšem informácie do konzoly aj do textového súboru obsahujúce číslo trénovanej epochy, chybu na danej epoche a čas trénovania epochy. Ak prišlo k trénovaniu N-tej epochy, vykreslím graf všetkých zatiaľ vypočítaných chýb a tým aj vizuálne sledujem, či dochádza ku klesaniu chyby. Dáta zozbierané z trénovania v závere uložím do CSV súboru na prípadnú neskoršiu prácu s dátami alebo vykresľovanie v iných programoch. Po dotrénovaní zapíšem do konzoly a textového súboru informáciu o konci trénovania a najmenšiu vypočítanú chybu s príslušnou epochou. Celý kód aplikovania SSL metódy Rotácie je v prílohe na CD.

7.2.3 Vyhodnotenie modelu:

SSL metóda - Rotácia										
SSL		TEST1 - Lineárna evaluácia (presnosť) [%]							TEST3 - Porovnanie trénovania s učiteľom	
epoch	dávka	1%	2%	3%	4%	5%	10%	Učiteľ	SSL [%]	Učiteľ [%]
50	200	57,3	59,9	62,1	63,0	64,1	65,1	66,0	67,4	63,0
100	200	58,5	62,3	63,2	64,1	65,0	66,7	66,0	70,5	63,0
300	200	58,0	62,4	63,3	64,2	65,1	67,1	66,0	70,3	63,0
500	200	58,2	60,9	62,2	64,1	64,6	67,2	66,0	71,5	63,0
50	50	56,1	60,2	62,3	62,4	63,9	66,8	66,0	66,5	63,0
50	100	56,7	62,0	62,9	64,2	64,0	66,7	66,0	69,0	63,0
50	300	55,1	57,8	60,1	61,5	62,4	66,5	66,0	70,6	63,0
50	500	55,4	59,2	61,0	62,3	63,8	68,3	66,0	71,2	63,0

Tab. 7.1: Porovnanie výsledkov tréningu SSL modelu Rotácie na testoch TEST1 a TEST3

Z výsledkov v tabuľke Tab. 7.1 je zrejmé, že postupné zvyšovanie počtu dát s označením zvyšuje presnosť testovaného modelu. SSL metóda rotácie bola na datasete STL10 schopná dosiahnuť najväčšiu presnosť 65,1% pri dotrénovaní s 1 lineárnou vrstvou pri 5% označených dát a 67,2% presnosť pri 10% označených dát. Z výsledkov môžem konštatovať, že či už zvýšením veľkosti dávky alebo počtu epoch, nedochádza k veľkým spomenutia hodným zmenám testovacej presnosti. Viac rozhodujúce je množstvo dát určených na dotrénovanie už takto predtrénovaného modelu. Z výsledkov v tabuľke Tab 7.2 môžem konštatovať, že najväčšia presnosť (65,1%)

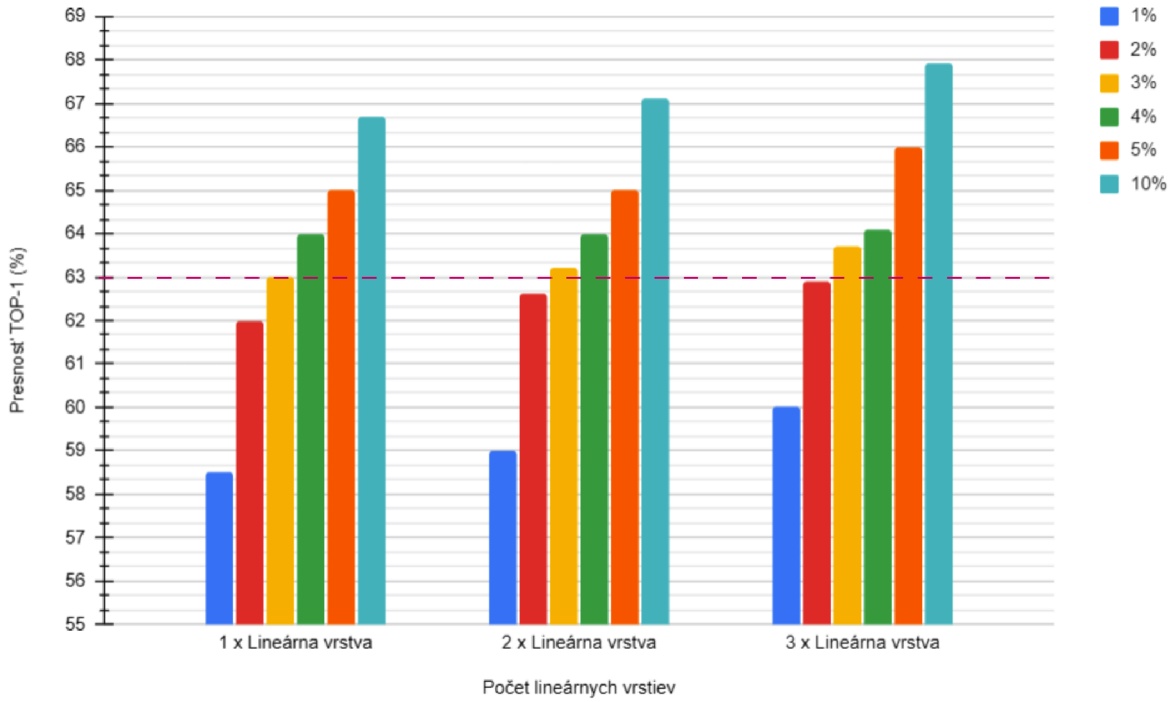


Obr. 7.5: Graf porovnania rôznych veľkostí označených dát na dotrénovanie SSL modelu Rotácie (TEST1)

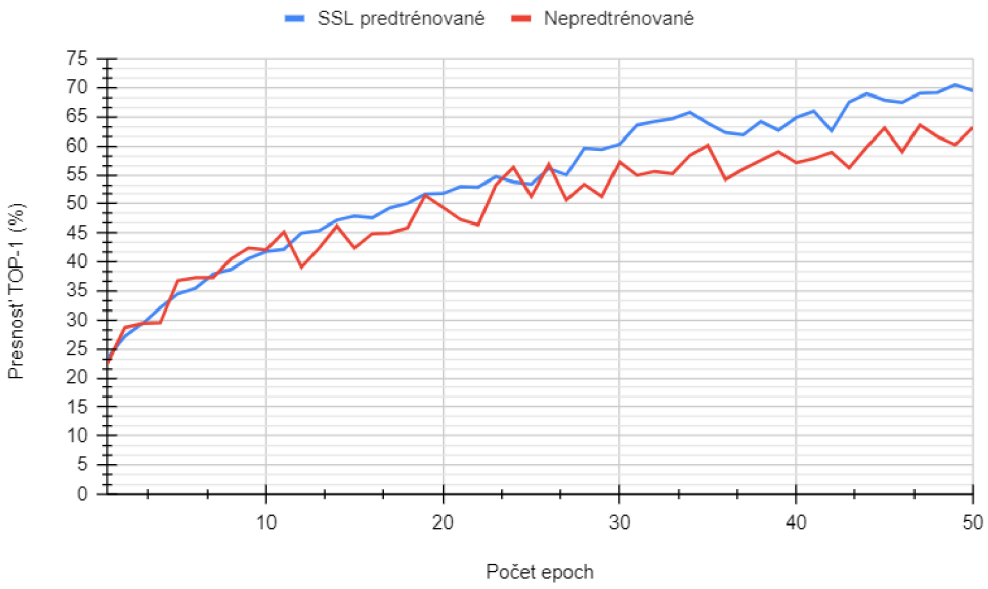
SSL		Vyhodnotenie SSL metódy Rotácie, dotrénovanie 1%/5% dát s označením			
epoch	dávka	Presnosť [%]	AUC	PR plocha	F1 skóre
50	200	57,3 / 64,1	0,896 / 0,912	0,537 / 0,611	0,53 / 0,60
100	200	58,5 / 65,0	0,898 / 0,914	0,549 / 0,625	0,55 / 0,62
300	200	58,0 / 65,1	0,899 / 0,914	0,554 / 0,627	0,56 / 0,63
500	200	58,2 / 64,6	0,893 / 0,912	0,538 / 0,599	0,55 / 0,62
50	50	56,1 / 63,9	0,890 / 0,900	0,531 / 0,601	0,53 / 0,59
50	100	56,7 / 64,0	0,888 / 0,895	0,533 / 0,604	0,54 / 0,61
50	300	55,1 / 62,4	0,891 / 0,907	0,535 / 0,605	0,52 / 0,60
50	500	55,4 / 63,8	0,893 / 0,910	0,537 / 0,610	0,52 / 0,61

Tab. 7.2: Tabuľka porovnaní presností pre dotrénovanie s 1% a 5% dátach (SSL model rotácie, TEST1)

modelu spolu s najväčšou plochou (AUC) pod krivkou ROC (0,914) bolo pri trénovaní SSL modelu s 300 epochami a veľkosťou dávky 200 pri dotrénovaní na 5% dátach. Príklad krivky ROC s výpisom veľkosti plochy pod krivkou (AUC) na takto



Obr. 7.6: Porovnanie počtu lineárnych vrstiev pri dotrénovaní SSL modelu s rôznou veľkosťou dát (TEST2)



Obr. 7.7: Porovnanie trénovania SSL modelu Rotácie a prázdneho modelu trénovaného s učiteľom (TEST3)

Presnosť modelov	Množstvo označených dát	
	1%	5%
Tréning s učiteľom	40%	63%
SSL + tréning s učiteľom	58,5%	65,0%

Tab. 7.3: Porovnanie predtrénovaného SSL modelu Rotácie a nepredtrénovaného modelu pri trénovaní s učiteľom na 1% a 5% trénovacích dát (TEST4)

definovaných parametroch, je na obrázku Obr. B.1. Z grafu na Obr. 7.5 je vidieť, že predtrénovaný model dotrénovaný s 5% dát s označením, dokáže konkurovať modelu trénovaného s učiteľom a pri 10% dát s označením ho prekonať. Graf na obrázku Obr. 7.6 zobrazuje TEST2 a vyplýva z neho, že zvýšenie lineárnej vrstvy síce zvýši presnosť modelu, ale minimálne. Najväčší pozitívny rozdiel je pri 1% dát s označením a to +1.0% presnosti zvýšením z 2 na 3 lineárne vrstvy. Fialová prerušovaná čiara symbolizuje presnosť 63%, ktoré dosahuje model trénovaný s učiteľom. Pri zvýšení počtu vrstiev z 1 na 3 dosiahne aj model dotrénovaný na 3% dát presnosť, ako model trénovaný s učiteľom a pri 3% dátach jasne prekoná tento model. Z výsledkov TEST3 na obrázku Obr. 7.7. je zrejmé, že model, ktorý už s dátami pracoval spočiatku dosahuje rovnakú presnosť a následne sa jeho náskok vo veľkosti niekoľko jednotiek percent začne prejavovať približne od 30 epochy a predtrénovaný model si ho udrží až do konca. Výsledky TEST4 z tabuľky Tab. 7.3 jasne značia, že už pri 1% dát dokáže predtrénovaný model jasne poraziť model trénovaný s učiteľom na rovnakom množstve dát. Pri 5% dátach sa presnosť modelov začne blížiť spoločnej hodnote. Metódu hodnotím ako praktickú a jednoduchú vzhľadom na pomer obtiažnosti aplikovania voči zvýšeniu prenosti, ktorú metóda zabezpečí. Záverom konštatujem, že metóda rotácia síce zlepšuje presnosť pri následnom dotrénovaní, ale kontrastívne metódy popísané v ďalších kapitolách sú v tomto ešte úspešnejšie.

7.3 SSL metóda SimCLR:

V poradí druhá úspešne aplikovaná a otestovaná metóda je SimCLR. Táto a nasledujúce metódy už patria do skupiny kontrastívneho učenia. Ich výsledky sú presnejšie, ako pri SSL metóde rotácie. V nasledujúcich podkapitolách sa venujem príprave datasetu, popisu trénovania a vyhodnotenie modelu predtrénovaného na SimCLR metóde.

7.3.1 Príprava datasetu:

Rozdiel kontrastívnych metód oproti iným metódam z oblasti SSL je v tom, že ich cieľom je zo vstupného obrázku spraviť dvojicu obrázkov, na ktoré budú aplikované rôzne augmentácie. Tieto obrázky putujú do CNN a následne už podľa danej metódy je cieľom tieto obrázky priblížiť, čo najviac k sebe (pozitívny pár) a naopak oddialiť všetky ostatné (negatívne páry). V metóde SimCLR (a ďalších kontrastívnych metód) sa na vstupný obrázok aplikujú nasledovné augmentácie: náhodné orezanie (veľkosť si môžem definovať, ja som vybral polovicu dĺžky obrazu), náhodné otočenie (s pravdepodobnosťou 50%), náhodná zmena farebného spektra (pravdepodobnosť 80% na zmenu jasu, kontrastu, saturácie a odtieňu), náhodné aplikovanie šedotónového spektra (pravdepodobnosť 40%), rozmazanie gaussovým filtrom (veľkosť kernelu 13 pixelov) a normalizácia (hodnota priemeru a odchýlky 0.5 čoho výsledkom je normalizácia na rozsah [-1, 1]). Na všetky tieto augmentácie používam funkciu *SimCLR_data_transforms()*, ktorej vstupné parametre sú veľkosť obrázku a hodnota premennej použitej na veľkosť aplikovania augmentácie zmeny farebného spektra pri úprave jasu, kontrastu, saturácie a odtieňu. Jednotlivé úpravy obrázku aplikujem použitím príkazov z knižnice *torchvision.transforms*. Funkcia spojí všetky vyššie vymenované augmentácie dokopy príkazom *Compose()* z knižnice *torchvision*. Príklad vynechania augmentácií (náhodného orezania, náhodnej úpravy farebného spektra, vynechanie oboch augmentácií a naopak aplikovanie všetkých augmentácií) zobrazujem na obrázku Obr. 7.8. Na štvrtom obrázku je zmena sfarbenia spôsobená aplikáciou zmeny náhodného šedotónového spektra.

```
from torchvision import transforms as T
class ApplyAugmentationsToImages(object):
    def __init__(self, *args):
        self.transforms = args[0]
        self.random_flip = T.RandomHorizontalFlip()

    def __call__(self, sample, *with_consistent_flipping):
        if with_consistent_flipping:
            sample = self.random_flip(sample)

        output = [transform(sample) for transform in self.transforms]
        return output

def SimCLR_data_transforms(size, s=1):
    color_jitter = T.ColorJitter(0.8 * s, 0.8 * s, 0.8 * s, 0.2 * s)

    data_transforms = T.Compose([
```

```
            T.RandomResizedCrop(size=int(size/2)),
            T.RandomHorizontalFlip(p=0.5),
            T.RandomApply([color_jitter], p=0.8),
            T.RandomGrayscale(p=0.4),
            T.GaussianBlur(13),
            T.ToTensor(),
            T.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])])
    return data_transforms
```

Obr. 7.8: Príklad neaplikovania augmentácii na obrázku z datasetu STL10

Keďže augmentácie s náhodnými parametrami musím aplikovať 2 krát na jeden obrázok, čiže vytvoriť dvojicu augmentovaných obrázkov z jedného, vytvoril som ešte jednu triedu a to *ApplyAugmentationsToImages()*, ktorá aplikuje jednotlivé augmentácie a vytvorí dva obrázky. Príklad aplikovania augmentácií na vstupný obrázok z datasetu STL10 je na obrázku Obr. A.2. Takto pripravený dataset postupuje do dataloadru, ktorý rozdelí dáta do dávok a následne je používaný pri tréningu modelu SSL metódou SimCLR.

7.3.2 Tréning SSL modelu:

Pred tréningom rovnako ako pri ostatných metódach definujem zariadenie prioritne GPU (ak je možné), definujem sieť spolu s projekčnou hlavou (o veľkosti 2 vrstiev), tvorím SGD optimizer na základe parametrov definovanej siete a kontrolujem, či už sieť s parametrami veľkosti dávky a počtu epoch nebola trénovaná. Ak áno, kontrolujem poslednú trénovanú epochu a pokiaľ tréning neskončil budem v ňom pokračovať od danej epochy a predtrénovaných váh modelu.

Po kontrole inicializujem objekt triedy *SimCLRTrainer()*, ktorý očakáva na vstup definovanú sieť, optimizer, počiatočnú epochu, dataloader a zariadenie na trénovanie.

Po úspešnej inicializácii volám trénovaciu funkciu *SimCLRTrainer.train()*. V tejto funkcii dochádza k tréningu v dvoch vnorených FOR cykloch. Prvý FOR cyklus

postupne iteruje cez počet epoch a druhý FOR cyklus otvára každú dávku z dataloadru. V každej dávke sa nachádzajú 2 augmentované obrázky, ktoré pri vytiahnutí z dataloadru ukladám do zariadenia, na ktorom sa trénuje. Následne oba posielam do CNN a výstupy predikcie ukladám do premenných *y1, y2*. Tieto premenné použijem, ako vstupné parametre funkcie *ContrastiveLoss(y1, y2)*. Táto funkcia má za úlohu vypočítať kontrastnú stratu podľa vzorca bližšie vysvetleného a popísaného v teoretickej časti SimCLR metódy (kapitola 5.6.3). Aplikácia výpočtu kontrastívnej chyby v kóde je zobrazená pod týmto textom. Podobnosť dvoch vektorov sa počíta, ako výsledok násobku jednotkových vektorov. Tým, že CNN pracuje s dávkami obsahujúcimi viac obrázkov, tak v premenných *y1, y2* sú augmentované obrázky danej dávky. Na začiatku normalizujem a transformujem na jednotkové vektory všetky vektory v riadkoch a výsledok uložím do *a_unit, b_unit*. Následne aplikujem zreťazenie pomocou príkazu *torch.cat* v nultej dimenzii, čiže pozdĺž osy y a prvú maticu transponujem, aby som dosiahol požadovaný tvar vo vzorci 5.2. Pokračujem prenásobením týchto matíc, čoho výsledkom je matica podobností reprezentujúca podobnosť vektorov a pre každý prvok i, j je výsledok podobnosti na diagonále, kde sa indexi prvkov i, j pretnú. Následne je výsledok podobnosti vydelený hyperparametrom τ a dopočítaný čitateľ a menovateľ, ktorých podiel je argument pre výpočet záporného logaritmu. Z výsledku urobím priemer a vrátim kontrastívnu chybu.

```
def ContrastiveLoss(a, b):
        a_norm = torch.norm(a, dim=1).reshape(-1, 1)
        b_norm = torch.norm(b, dim=1).reshape(-1, 1)
        a_unit = torch.div(a, a_norm)
        b_unit = torch.div(b, b_norm)

        a_unit_b_unit = torch.cat([a_unit, b_unit], dim=0)
        a_unit_b_unit_transpose = torch.t(a_unit_b_unit)
        b_unit_a_unit = torch.cat([b_unit, a_unit], dim=0)

        sim = torch.mm(a_unit_b_unit, a_unit_b_unit_transpose)
        sim_by_tau = torch.div(sim, tau)
        exp_sim_by_tau = torch.exp(sim_by_tau)
        sum_of_rows = torch.sum(exp_sim_by_tau, dim=1)
        exp_sim_by_tau_diag = torch.diag(exp_sim_by_tau)

        numerators = torch.exp(torch.div(torch.nn.CosineSimilarity()
                            (a_unit_b_unit, b_unit_a_unit), tau))
        denominators = sum_of_rows - exp_sim_by_tau_diag
        num_by_den = torch.div(numerators, denominators)
        neglog_num_by_den = -torch.log(num_by_den)
```

```
loss = torch.mean(neglog_num_by_den)
return loss
```

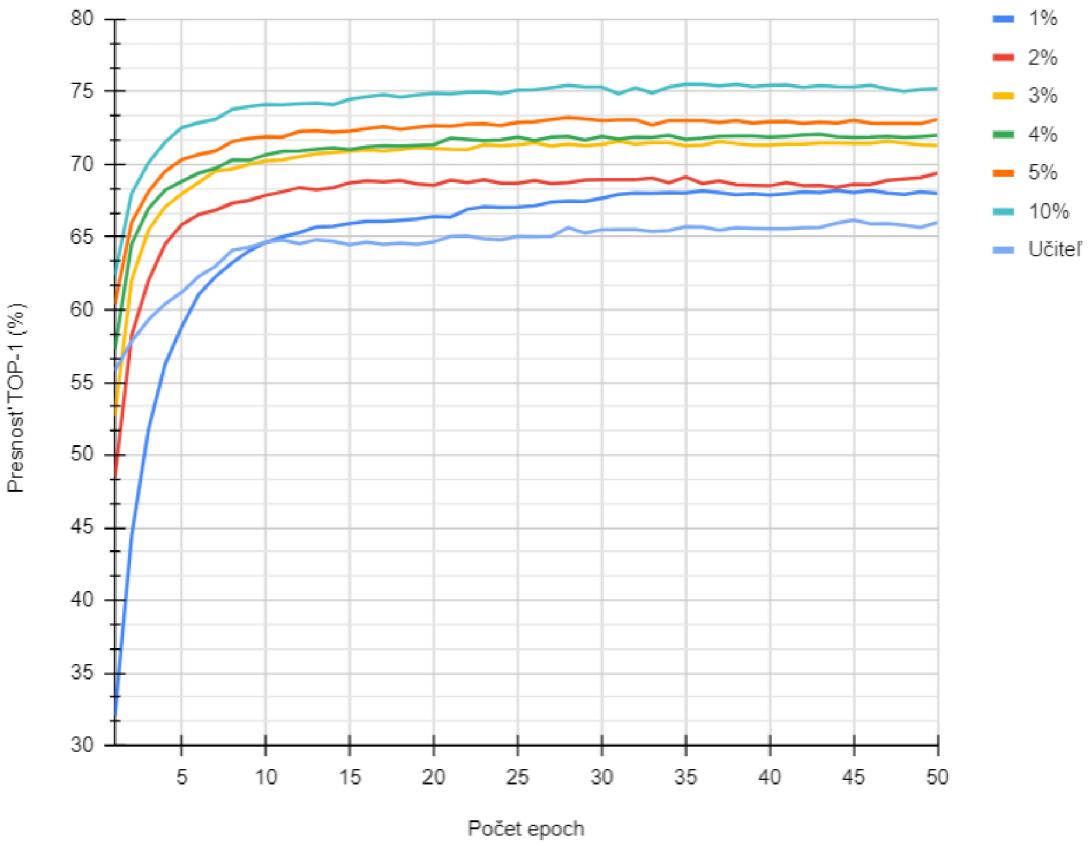
Následne chybu príkazom *loss.backward()* posielam späť cez všetky váhy a volám optimizer na úpravu váh. Priebežne počítam celkovú chybu za danú epochu. Po načítaní všetkých dávok a ukončení jednej epochy vypočítam priemernú chybu všetkých dávok na danú epochu a kontrolujem, či v danej epoche neprišlo k novému minimu. Ak áno, model ukladám a vypisujem do konzoly aj textového súboru informácie o danej epoche, veľkosti chyby a dĺžky trénovania poslednej epochy. Po dotrénovaní modelu vypočítané dáta chyby ukladám do csv súboru a vypíšem dĺžku celkového tréningu spolu s informáciou o najnižšom minime chyby s danou epochou.

7.3.3 Vyhodnotenie modelu:

SSL metóda - SimCLR										
SSL		TEST1 - Lineárna evaluácia (presnosť) [%]							TEST3 - Porovnanie trénovania s učiteľom	
epoch	dávka	1%	2%	3%	4%	5%	10%	Učiteľ	SSL [%]	Učiteľ [%]
50	200	67,1	69,2	71,2	71,4	71,5	73,5	66,0	70,4	63,0
100	200	68,2	69,4	71,3	72,6	73,1	75,2	66,0	78,7	63,0
300	200	69,1	71,2	72,3	73,8	75,0	76,8	66,0	79,1	63,0
500	200	69,9	72,3	72,9	74,1	75,5	77,1	66,0	80,2	63,0
50	50	60,4	65,3	66,1	66,5	67,0	70,5	66,0	68,4	63,0
50	100	63,3	65,5	68,5	69,2	70,2	73,4	66,0	71,3	63,0
50	300	65,5	67,2	70,2	71,3	72,1	74,3	66,0	73,2	63,0
50	500	65,6	68,1	70,3	71,4	72,2	75,2	66,0	75,1	63,0

Tab. 7.4: Porovnanie výsledkov tréningu SSL modelu SimCLR na testoch TEST1 a TEST3

Na výsledkoch z tabuľky Tab. 7.4 (TEST1) je zaujímavé, že zvyšovaním počtu epoch pri SSL trénovaní dochádza k vyššej presnosti pri dotrénovaní, ako zvyšovaním veľkosti dávky. Maximálna presnosť bola dosiahnutá pri SSL trénovaní na 500 epochách, 200 veľkosti dávky a dotrénovaní na 10% dát. Bohužiaľ, ani toto stále nestačí na porazenie modelu predtrénovaného na Imagenet datasete, ktorý dosiahol 80% presnosť (z dôvodu veľkosti tabuľky a konštantnej hodnoty presnosti Imagenet, som túto hodnotu uviedol iba do textu a grafu na obrázku Obr. 7.24 pre dataset STL10). Prekonanie modelu trénovaného učiteľom je možné už pri 1% dát. Z grafu

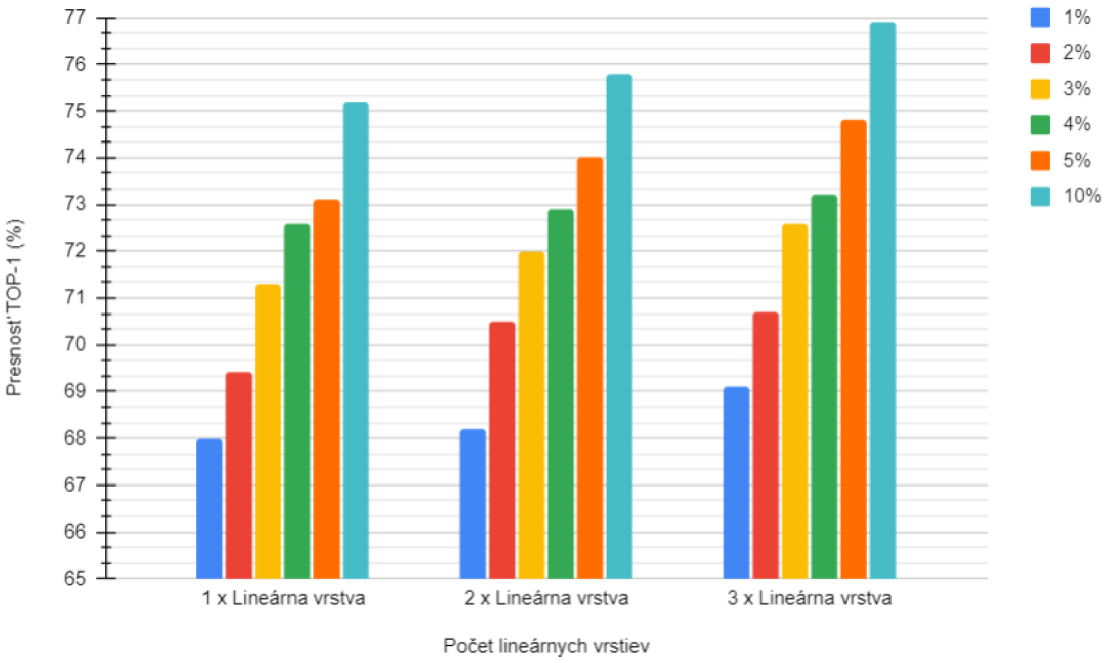


Obr. 7.9: Graf porovnania rôznych veľkostí označených dát na dotrénovanie SSL modelu SimCLR (TEST1)

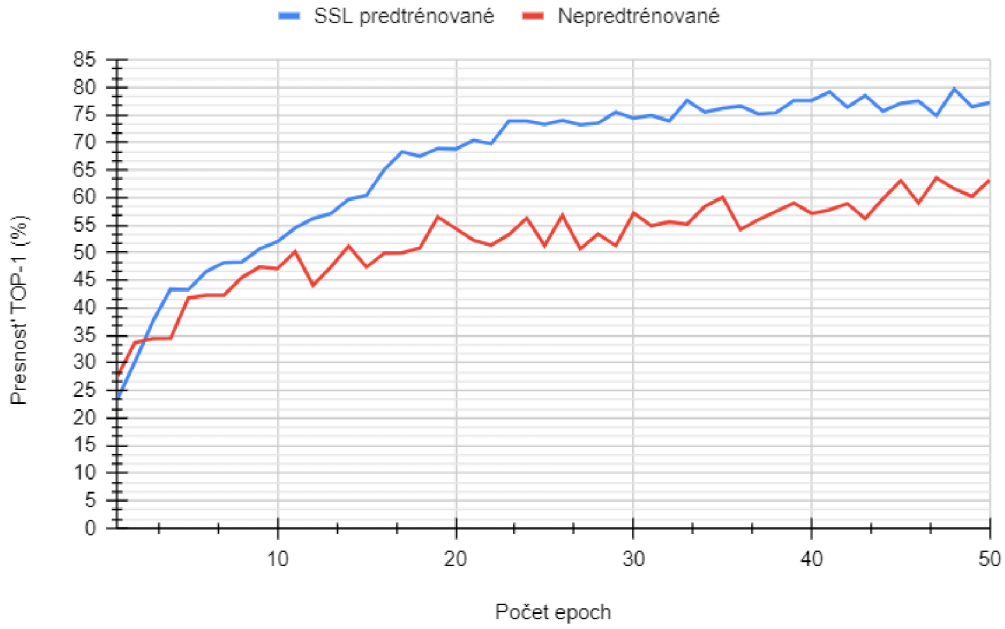
SSL		Vyhodnotenie SSL metódy SimCLR, dotrénovanie 1%/5% dát s označením			
epoch	dávka	Presnosť [%]	AUC	PR plocha	F1 skóre
50	200	67,1 / 71,5	0,899 / 0,914	0,539 / 0,613	0,65 / 0,69
100	200	68,2 / 73,1	0,921 / 0,918	0,551 / 0,628	0,67 / 0,71
300	200	69,1 / 75,0	0,923 / 0,918	0,552 / 0,631	0,68 / 0,73
500	200	69,9 / 75,5	0,927 / 0,919	0,540 / 0,601	0,69 / 0,73
50	50	60,4 / 67,0	0,882 / 0,899	0,533 / 0,602	0,59 / 0,66
50	100	63,3 / 70,2	0,891 / 0,901	0,545 / 0,612	0,61 / 0,69
50	300	65,5 / 72,1	0,896 / 0,905	0,551 / 0,623	0,62 / 0,71
50	500	65,6 / 72,2	0,897 / 0,912	0,557 / 0,637	0,63 / 0,70

Tab. 7.5: Tabuľka porovnaní presností pre dotrénovanie s 1% a 5% dátach (SSL model SimCLR, TEST1)

na obrázku Obr. 7.9 je zrejmé, že už 1% dát na dotrénovanie modelu postačí na prekonanie modelu trénovaného s učiteľom už po viac ako 12 epochách dotrénovania. Pri dokončení lineárnej evaluácie dosahujú modely s 3%, 4% 5% a 10% dátami



Obr. 7.10: Porovnanie počtu lineárnych vrstiev pri dotrénovaní SSL modelu s rôznou veľkosťou dát (TEST2)



Obr. 7.11: Porovnanie trénovania SSL modelu SimCLR a prázdneho modelu trénovaného s učiteľom (TEST3)

Presnosť modelov	Množstvo označených dát	
	1%	5%
Tréning s učiteľom	40%	63%
SSL + tréning s učiteľom	68,2%	73,1%

Tab. 7.6: Porovnanie predtrénovaného SSL modelu SimCLR a nepredtrénovaného modelu pri trénovaní s učiteľom na 1% a 5% trénovacích dát (TEST4)

		Lineárna evaluácia (presnosť) [%]					
Zmena farebného spektra	Náhodné vyrezanie z obrázku	1%	2%	3%	4%	5%	10%
nie	nie	31,5	35,2	36,8	37,1	37,3	40,3
nie	áno	56,0	60,1	62,3	63,5	64,2	65,0
áno	nie	30,2	35,4	36,3	37,4	37,5	41,2
áno	áno	67,1	69,2	71,2	71,4	71,5	73,5

Tab. 7.7: Porovnanie výsledkov tréningu SSL modelu SimCLR pri rôznych aplikáciách dvoch najdôležitejších augmentácií zmeny farebného spektra a náhodného orezania

vyššiu presnosť ako 70%. Na obrázku Obr. 7.10 poukazujem na zvýšenie presnosti zväčšením počtu lineárnych vrstiev. K najväčšiemu rastu presnosti dochádza pri 2% a 10% dotrénovávacích dát zvýšením z 1 na 2 respektíve 2 na 3 lineárne vrstvy a to pri oboch +1,1%. Použitie metódy SimCLR na predtrénovanie modelu ponúka jasnú výhodu pri následnom spôsobe trénovania s učiteľom. Model, ktorý už videl dáta, síce bez označení a s augmentáciami, dokáže od samého začiatku dosahovať vyššiu presnosť (TEST3). Po trénovaní na 50 epochách s veľkosťou dávky 32 dosahuje presnosť 79% oproti modelu trénovaného od nuly rovnakým spôsobom a s rovnakými vstupnými parametrami. Vzájomný náskok je +16% pre model predtrénovaný SSL metódou SimCLR. Zobrazenie grafu je na obrázku Obr. 7.11. Z výsledkov TEST4 (Tab. 7.6) vyplýva, že metóda SimCLR dokáže pri predtrénovaní a iba pri 1% označených dát zabezpečiť presnosť až 68,2%. Pri 5% označených dát je presnosť 73,1%. V oboch prípadoch ide o vyššiu hodnotu (hlavne pri prvom prípade) ako pri trénovaní s učiteľom na rovnako veľkom datasete bez predtrénovania. V teoretickej časti vysvetlujúcej metódu SimCLR spomínam, že využitie hlavne dvoch typov augmentácií (zmena farebného spektra a náhodné vyrezanie, ktorých príklady sú na Obr. 7.8) najviac pomáha pri naučení dôležitých vlastností z obrázku (Obr. 5.24). Spravil som pokus, ktorého výsledky sú v tabuľke Tab. 7.7, kde som skúsil nepoužiť tieto augmentácie alebo iba jednu z nich (zvyšné augmentácie boli použité, ako som vyššie

popísal). Výsledky potvrdzujú tvrdenia tvorcov metódy a tiež, že použitie náhodného vyrezania pomôže pri kontrastívnom učení viac, ako náhodná zmena farebného spektra (výsledky pre SLL model predtrénovaný na 50 epochách a dávke 200).

7.4 SSL metóda MoCov2:

Metóda MoCov2 (momentum contrastive learning version 2) umožňuje aplikovať mierne odlišný prístup k spracovaniu augmentovaných dát, ako SimCLR, využitím dynamického slovníka (bližšie vysvetlené v teoretickej časti 5.6.4). Z toho dôvodu je možné predpokladať lepšie výsledky pri práci s väčšou dávkou alebo datasetom pri trénovaní modelu a následnom vyhodnotení.

Metóda MoCov2 patrí do skupiny kontrastívnych metód a z toho dôvodu je aj jej cieľom minimalizovať kontrastnú chybu, čiže priblížiť dva augmentované obrázky z jedného spoločného originálu a odlíšiť ich od ostatných. MoCov2 tento krok nerobí iba v danej dávke, ale rozširuje svoju kapacitu vďaka dynamickému slovníku. Týmto postupom je možné porovnávať vstupné obrázky s viacerými, ako pri jednej dávke v SimCLR metóde a pravdepodobne dosiahnuť aj lepšie výsledky.

7.4.1 Príprava datasetu:

Tvorba datasetu je obdobná predošlej kontrastívnej metóde. Vzhľadom na to, že som sa inšpiroval novšou metódou MoCov2, ktorá obsahuje aj gaussovo rozostrenie, aplikácia augmentácii aj ich zoznam aj hodnoty je rovnaký ako pri SimCLR. Augmentácie aplikujem po stiahnutí datasetu pomocou objektu triedy *ApplyAugmentationsToImages*, ktorej vstupné parametre sú augmentácie vytvorené funkciou *MOCOv2_data_transforms()*. Príklad aplikácie týchto augmentácii je na obrázku Obr. A.3. Po aplikovaní augmentácii pripravím datasety pre trénovanie s učiteľom a taktiež pre lineárnu evaluáciu. Pri týchto datasetoch už nie sú aplikované takto zložité augmentácie a naopak ku obrázkom sú priradené označenia tried. Datasety s označením rozdelím na 1% až 10% rovnako, ako pri predošlom príklade SSL metódy. Zo všetkých datasetov následne vytvorím dataloader.

7.4.2 Tréning SSL modelu:

Pred trénovaním SSL modelu, najprv vytvorím 2 CNN s projekčnou hlavou pre žiadosť (Q_network) a kľúče (K_network). Obe siete sú postavené na architektúre Resnet18 a ich projekčné hlavy sa skladajú z 2 vrstiev. Po tvorbe sietí pristupujem k tvorbe dynamického slovníka na uloženie kľúčov. Ide o frontu (queue), ktorej veľkosť definujem v úvode celého kódu. Ja som sa rozhodol pre veľkosť 8192 kľúčov,

ale v závere vyhodnotenia robím test pri rôznych veľkostiach fronty a následne vyhodnocujem presnosť. Tvorba fronty sa deje v prípade, že nie je vopred definovaná alebo načítaná iná fronta, následne vstupuje do WHILE cyklu a postupne otvára augmentované obrázky z dataloaderu. Keďže so žiadosťou "Q" reprezentovanou v prvom augmentovanom obrázku budem neskôr pracovať, v tomto cykle používam iba druhý augmentovaný obrázok, ktorý bude reprezentovať kľúč "K". Tento kľúč pošlem na vstup do siete (K_network) a jej výstup po normalizácii pridám do fronty. Tento proces robím do momentu naplnenia fronty kľúčmi, čiže reprezentáciami augmentovaných obrázkov. Následný postup spočíva v kontrole existencie predtrénovaných sietí a zoznamu kontrastívnej chyby, v prípade existencie pokračuje ich načítanie a trénovanie od momentu posledného zastavenia.

V prípade trénovania od začiatku, inicializujem nový objekt *MoCov2Trainer*, ktorého parametre sú dve CNN pre žiadosti a kľúče, optimizer, zariadenie na trénovanie (GPU), fronta, dataloader, veľkosť momentu a počiatočná epocha tréningu. Následne volám funkciu triedy *MoCov2Trainer.train()*. Vo vnútri funkcie spočiatku dochádza k nastaveniu potrebných premenných na výpočet chyby a sputenie počítania času tréningu a nastavenie CNN Q do tréningového režimu. Tréning sa deje v dvoch FOR cykloch. V druhom vnorenom FOR cykle dochádza k prechádzaniu cez dataloader a otváranie jednotlivých dávok s augmentovanými obrázkami. Ďalším krokom je vynulovanie optimizeru a poslanie augmentovaných obrázkov do konvolučných sietí Q a K. Ich výstupy sú normalizované do jednotkového vektoru a poslané do funkcie na výpočet kontrastívnej chyby spolu s frontou. Výpočet chyby sa deje vo funkcii *Loss(self, q, k, queue)* podľa vzorca zobrazeného v teoretickej časti MoCov2 metódy v kapitole 5.6.4 a moja aplikácia v kóde spolu s normalizáciou po z výstupu CNN je pod nasledujúcim textom. Na začiatku funkcie si zoberiem hodnotu veľkosti dávky a rozmeru reprezentácií dávky. Postup pri prenásobení matíc je identický ako pri metóde SimCLR a preto v tomto prípade už použijem jednoduchší zápis v jednom riadku za použitia funkcie *torch.bmm* (batch matrix multiplication). Výsledok ukladám do premennej *posit* reprezentujúcu pozitívny pár a pokračujem maticovým násobením, exponenciálou a sumou príkazmi *torch.mm*, *torch.exp* a *torch.sum*, ktorých výsledok ukladám do premennej *negat* reprezentujúcej negatívne páry. Sčítaním týchto premenných dostávam menovateľ a ako čitateľ zostáva hodnota *posit*. Výsledok po delení je argument záporného logaritmu a následná aplikácia priemeru, ktorých výsledok vraciam z funkcie von.

```
q = torch.div(q, torch.norm(q,dim=1).reshape(-1,1))
k = torch.div(k, torch.norm(k,dim=1).reshape(-1,1))

def Loss(self, q, k, queue):
    S = q.shape[0]    # the batch size
```

```
    D = q.shape[1]    # dimensionality of the representations

    posit = torch.exp(torch.div(torch.bmm(
                    q.view(S,1,D), k.view(S,D,1)).view(S, 1), T))
    negat = torch.sum(torch.exp(torch.div(
                    torch.mm(q.view(S,D), torch.t(queue)), T)), dim=1)

    denominator = negat + posit
    loss = torch.mean(-torch.log(torch.div(posit, denominator)))
    return loss
```

Po vypočítaní chyby dochádza k pridaniu nových kľúčov reprezentujúcich obrázky z poslednej dávky do fronty. Ak je fronta väčšia, ako jej kapacita, najstaršie kľúče o veľkosti dávky sa vyhodia jednoduchým indexovaním poľa. Ako jeden z posledných krokov je aktualizovanie enkóderu - CNN pre kľúče, ktorej váhy sú kĺzavý priemer ovplyvnení premennou momentu. Postup a vzorec je vysvetlený v kapitole 5.6.4 a príklad aplikácie kódom je v nasledujúcom texte.

```
self.queue = torch.cat((self.queue, k), 0)

if self.queue.shape[0] > queue_max_size:
    self.queue = self.queue[MOCOv2_BATCH:, :]

for K_CNN, Q_CNN in zip(self.MOCOv2_K.parameters(),
                        self.MOCOv2_Q.parameters()):
    K_CNN.data.copy_(self.momentum * K_CNN.data
                + Q_CNN.data * (1.0 - self.momentum))
```

Po skončení epochy dochádza k výpočtu chyby za epochu, výpisu informácii do textového súboru a konzoly a tiež vykreslenie grafu vývoja chyby ak došlo k trénovaniu N epoch, kde N je možné vopred definovať napr. na 10 a tým každých 10 epoch vidieť vývoj kontrastívnej chyby. Tento cyklus sa opakuje dokým nedôjde k trénovaniu v počte epoch definovaných v úplnom úvode. Následne dôjde k zapamätaniu si modelov pre žiadosti aj kľúče a optimizer v epoche, kde bola dosiahnutá najmenšia chyba. V závere tréningu sa vypíše dĺžka trénovania celého modelu a najnižšia vypočítaná chyba s príslušnou epochou.

7.4.3 Vyhodnotenie modelu:

Metódu MoCov2 som využil na trénovanie modelu siete Resnet18, s veľkosťou fronty 8192, τ 0.05, momentu 0.999, učiaceho kroku 0.01, veľkosti projekčnej hlavy skrytej

SSL metóda - MoCov2										
SSL		TEST1 - Lineárna evaluácia (presnosť) [%]							TEST3 - Porovnanie trénovania s učiteľom	
epoch	dávka	1%	2%	3%	4%	5%	10%	Učiteľ	SSL [%]	Učiteľ [%]
50	200	64,3	68,2	70,1	70,8	71,5	73,9	66,0	74,5	63,0
100	200	63,1	67,1	68,6	69,8	72,2	74,3	66,0	80,9	63,0
300	200	68,4	71,2	72,4	74,1	74,5	76,8	66,0	81,2	63,0
500	200	70,2	71,3	72,5	73,5	74,2	77,2	66,0	82,3	63,0
50	50	60,0	64,8	66,1	68,3	70,1	74,2	66,0	75,1	63,0
50	100	62,4	64,6	67,8	69,2	71,9	75,1	66,0	79,6	63,0
50	300	67,1	69,5	71,3	73,2	74,3	77,4	66,0	80,1	63,0
50	500	66,5	69,1	72,4	74,3	76,2	79,3	66,0	81,7	63,0

Tab. 7.8: Porovnanie výsledkov tréningu SSL modelu MoCov2 na testoch TEST1 a TEST3

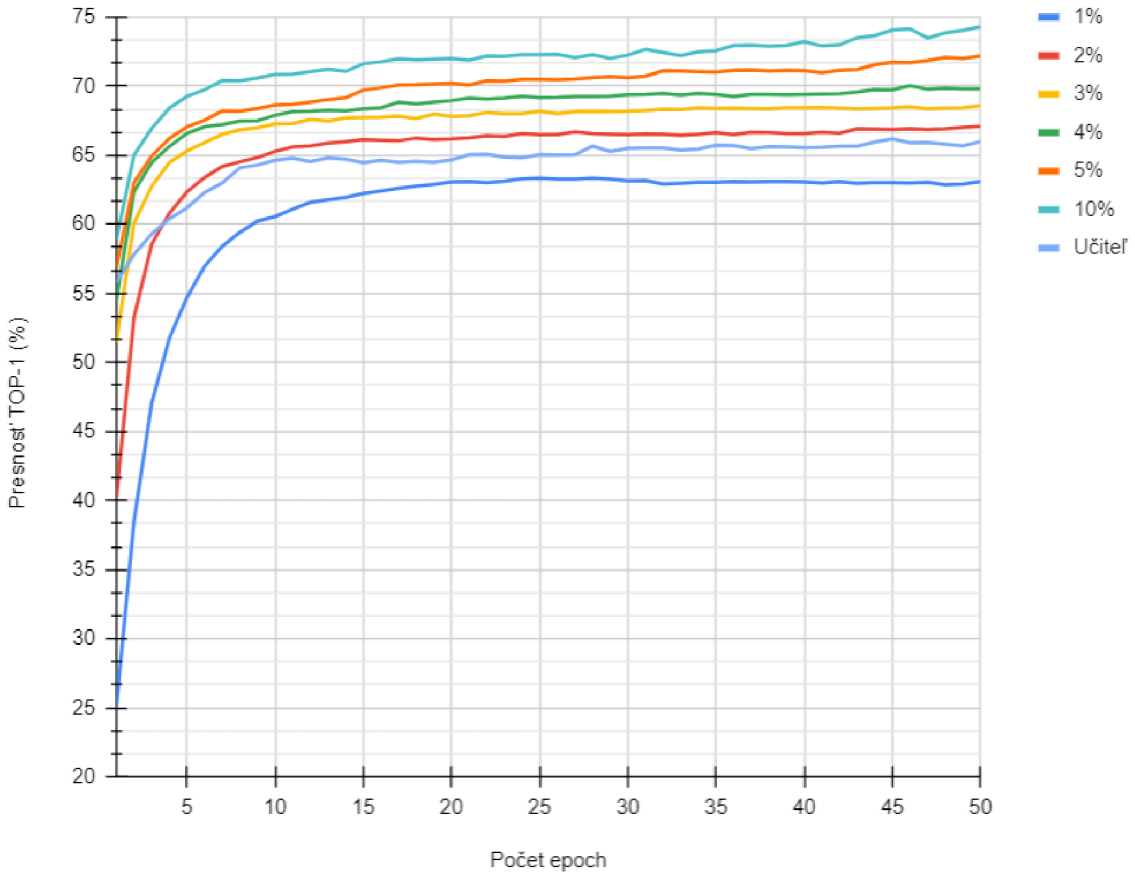
SSL		Vyhodnotenie SSL metódy MoCov2, dotrénovanie 1%/5% dát s označením			
epoch	dávka	Presnosť [%]	AUC	PR plocha	F1 skóre
50	200	64,3 / 71,5	0,884 / 0,896	0,598 / 0,612	0,61 / 0,69
100	200	63,1 / 72,2	0,893 / 0,908	0,599 / 0,613	0,62 / 0,70
300	200	68,4 / 74,5	0,899 / 0,910	0,608 / 0,624	0,64 / 0,73
500	200	70,2 / 74,2	0,902 / 0,911	0,610 / 0,637	0,66 / 0,72
50	50	60,0 / 70,1	0,886 / 0,894	0,587 / 0,609	0,59 / 0,68
50	100	62,4 / 71,9	0,894 / 0,911	0,592 / 0,612	0,62 / 0,69
50	300	67,1 / 74,3	0,895 / 0,914	0,607 / 0,625	0,65 / 0,72
50	500	66,5 / 76,2	0,900 / 0,921	0,604 / 0,642	0,65 / 0,74

Tab. 7.9: Tabuľka porovnaní presností pre dotrénovanie s 1% a 5% dátach (SSL model MoCov2, TEST1)

Presnosť modelov	Množstvo označených dát	
	1%	5%
Tréning s učiteľom	40%	63%
SSL + tréning s učiteľom	63,1%	72,2%

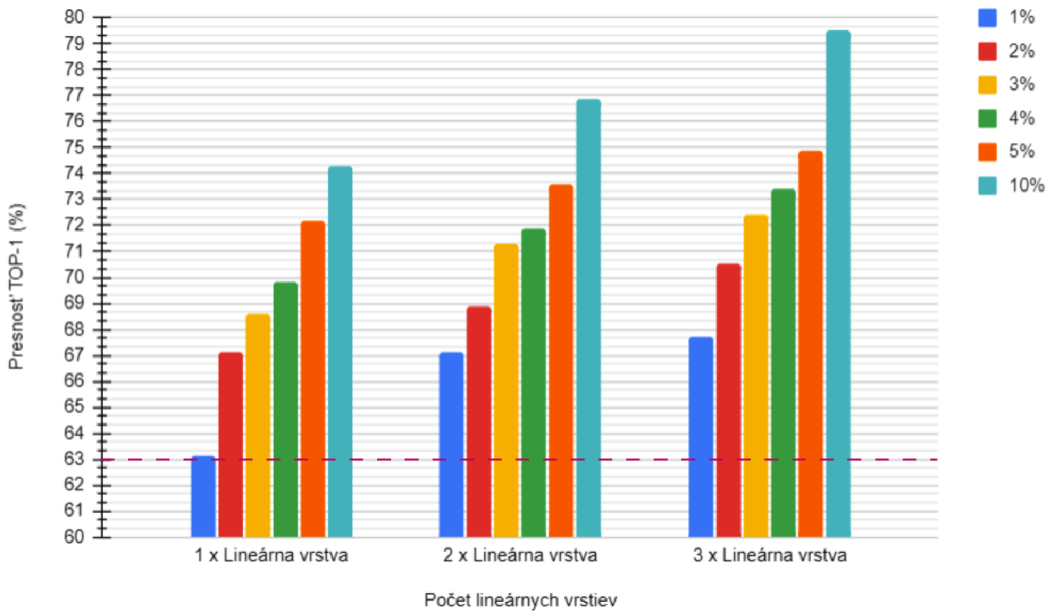
Tab. 7.10: Porovnanie predtrénovaného SSL modelu MoCov2 a nepredtrénovaného modelu pri trénovaní s učiteľom na 1% a 5% trénovacích dát (TEST4)

vrstvy 512 neurónov a výstupnej 128 neurónov. V tabuľke Tab. 7.8 je zaznamenaný TEST1 a TEST3. Z dát je vidieť, že k väčšiemu nárastu dochádza pri zvyšovaní

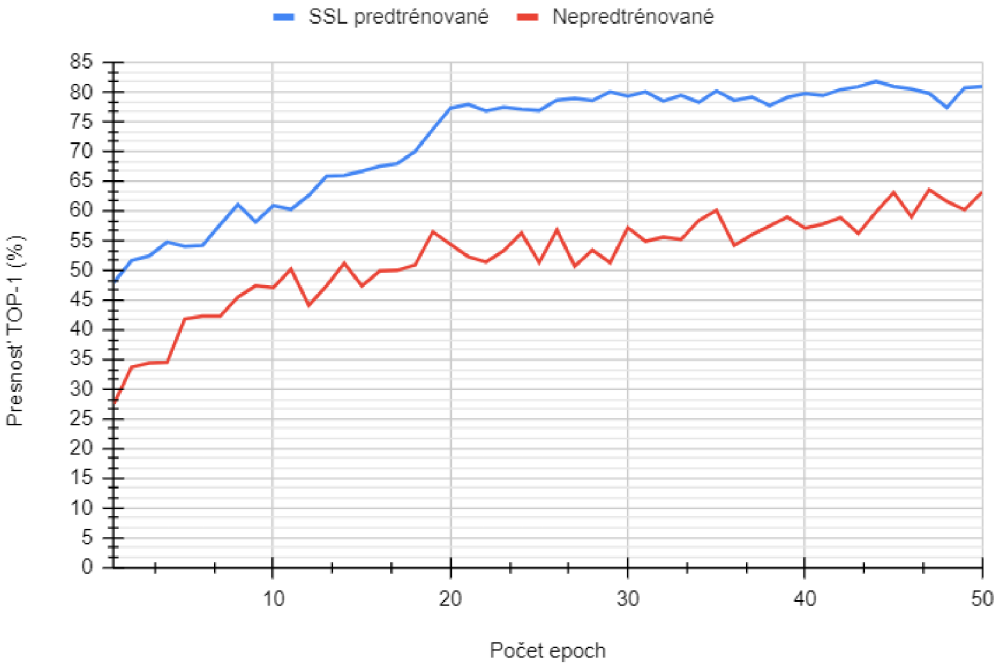


Obr. 7.12: Graf porovnania rôznych veľkostí označených dát na dotrénovanie SSL modelu MoCov2 (TEST1)

veľkosti dávky a samotné maximum presnosti 79,3% je pri dotrénovaní s 10% dát s označením. Model trénovaný s učiteľom dosiahol presnosť 63% a pri dotrénovaní 66%. Takýto model je prekonaný SSL metódou MoCov2 už pri dotrénovaní na 1% dát s označením pri štyroch z ôsmich prípadov trénovania. Pri dotrénovaní na 3 a viac % dátach s označením SSL metóda MoCov2 jasne prevyšuje presnosť modelu trénovaného s učiteľom. Výsledky pre TEST2 sú na obrázku Obr. 7.13. Prerušovaná čiara ružovou farbou značí presnosť modelu trénovaného s učiteľom. Pri použití 1 vrstvy dotrénovanie 2 a viac % prekonáva metódu trénovania s učiteľom. Maximálnu presnosť sa podarilo dosiahnuť pri použití 3 vrstiev a 10% dát a to 79,5%. TEST3 svoje výsledky zobrazuje na vývoji grafu Obr. 7.14. SSL model má väčší náskok v presnosti od prvej až po poslednú 50 epochu. Výsledok TEST4 je zaznamenaný v tabuľke Tab. 7.10, z ktorého je vidieť, ako už pri 1% dát dosahuje SSL model presnosť 63,1% a model bez predtrénovania 40%. Pri 5% sa tento rozdiel zníži, ale stále model s predtrénovaním dosahuje vyššiu presnosť o 9,2% (SSL - 72,2%, nepredtrénovaný - 63%). Posledný experiment zaznamenaný v tabuľke Tab. 7.11



Obr. 7.13: Porovnanie počtu lineárnych vrstiev pri dotrénovaní SSL modelu s rôznou veľkosťou dát (TEST2)



Obr. 7.14: Porovnanie trénovania SSL modelu MoCov2 a prázdneho modelu trénovaného s učiteľom (TEST3)

	Lineárna evaluácia (presnosť) [%]					
Veľkosť dynamického slovníku	1%	2%	3%	4%	5%	10%
8192	64,1	69,2	70,1	70,8	71,5	73,9
4096	61,2	63,4	65,1	65,3	66,2	67,6
2048	60,6	62,4	63,4	65,1	66,2	67,3
1024	58,7	61,2	62,3	63,1	65,3	67,2
512	57,6	60,9	61,8	63,0	65,1	67,0

Tab. 7.11: Porovnanie výsledkov tréningu SSL modelu MoCov2 pri rôznych veľkostiach dynamického slovníka

je trénovanie SSL modelu pri rôznej veľkosti dynamického slovníku. Začal som pri hodnote 512 a následne hodnotu vždy zdvojnásobil. Veľkosť 8192 dosahuje najväčšiu presnosť pri všetkých typoch dotrénovania počas lineárnej evaluácie. Vyplýva z toho, že veľkosť dynamického slovníku má priamy pozitívny vplyv na presnosť daného modelu.

7.5 SSL metóda BYOL:

Poslednou úspešne aplikovanou SSL metódou je BYOL (Bootstrap you own latent), ktorá spolu s SimCLR a MoCov2 patria ku kontrastívnym. Je to aj najnovšia metóda, ktorej publikované výsledky dosahujú najväčšiu presnosť.

7.5.1 Príprava datasetu:

Príprava datasetu spočíva v aplikovaní augmentácii obdobných, ako pri metóde SimCLR a MoCov2. Tieto augmentácie aplikujem pri stiahnutí datasetu STL10 pomocou triedy *ApplyAugmentationsToImages*. Trieda pri svojej inicializácii dostáva parameter augmentácii dva krát a to z toho dôvodu, aby vytvorila 2 augmentované obrázky z pôvodného spoločného originálu. Pri metóde BYOL aplikujem nasledovné augmentácie: náhodné orezanie, náhodné otočenie, náhodná zmena farebného spektra, náhodné aplikovanie šedotónového spektra, rozmazanie gaussovým filtrom a normalizácia. Tieto augmentácie sú spísané vo funkcie *BYOL_data_transforms()*, ktorá očakáva na svojom vstupe veľkosť obrázku a hodnotu premennej aplikovanej na veľkosť sily zmeny pri augmentovaní farebného spektra pri úprave jasu, kontrastu, saturácie a odtieňu, ktorá je rovnaká, ako pri SimCLR. Vyššie spomenuté

augmentácie sú aplikované príkazmi z knižnice *torchvision.transforms* a spojené dokopy príkazom *Compose()* z knižnice *torchvision*. Príklad aplikovania augmentácií na vstupný obrázok z datasetu STL10 je na obrázku Obr. A.4.

7.5.2 Tréning SSL modelu:

Metóda BYOL pri svojom tréningu používa dve CNN s názvami Online a Target (cieľová sieť). Online sieť okrem svojej architektúry obsahuje nad sebou prediktor, ktorého úlohou je predpovedať výstup z cieľovej siete. Tieto prakticky 3 siete musím na začiatok inicializovať. Online aj Target sieť inicializujem bežným spôsobom pomocou funkcie ResNet, ktorej vstupný parameter je názov konvolučnej siete, ktorú chcem používať (Resnet18). Prediktor je tvorený ako projekčná hlava a jeho vstup je veľký hodnote výstupu (počtu neurónov) online sieťe, a vo vnútri obsahuje ešte jednu vrstvu o veľkosti 512 neurónov. Výstupná vrstva má veľkosť 128 neurónov. Test rôznych počtov neurónov je v tabuľke Tab. 7.15. Po vytvorení sietí, inicializujem optimizer, ktorého vstupný parameter je súčet parametrov online siete s prediktorom, veľkosť učiaceho kroku (0.03) a momentu (0.9). Následne dochádza ku kontrole predtrénovaného modulu s rovnakými parametrami. Ak existuje, konvolučné siete sa načítajú z uložených modelov spolu s poslednou epochou, po ktorú sa model trénoval. Ak takýto model neexistuje, pripraví sa tréning od začiatku. Pomocou triedy *BYOLTrainer* vytvorím objekt, ktorého inicializačné parametre sú CNN Online, Target a prediktor. Nasleduje optimizer, dataloader, zariadenie (GPU) a počiatočná epocha. Po inicializácii spúšťam tréning funkciou *BYOLTrainer.train()*.

V úvode tréningu dochádza k tvorbe premenných, do ktorých budem ukladať chybu za epochu a za celý tréning a spúšťam meranie času celého trénovania. Následne inicializujem Target sieť, aby mala obdobné nastavenie váh ako Online sieť a zablokujem potrebu spätného šírenia gradientu pre Target sieť, keďže sa jej upravujú váhy pomocou postupnej optimalizácie váh momentom (krokom), ako pri metóde MoCov2.

```
def Initialize_Target_Net(self):
    for Params_Online, Params_Target in zip(self.Online_Net.parameters(),
                                            self.Target_Net.parameters()):
        Params_Target.data.copy_(Params_Online.data)
        Params_Target.requires_grad = False
```

Tréning spočíva v dvoch vnorených FOR cykloch. Prvý FOR cyklus iteruje cez epochy a meria sa v ňom čas tréningu každej epochy a v druhom vnorenom FOR cykle iterujem cez dataloader. Postupne vyberám pripravené dávky augmentovaných obrázkov do premenných *batch_view_1, batch_view_2* (ktoré obsahujú augmentované dvojice). Tieto dávky následne posielam do zariadenia, v ktorom sú pripravené

aj CNN. Nasleduje výpočet kontrastívnej chyby, v prípade BYOL je to nasledovne. Do Online siete pošlem augmentovaný obrázok *batch_view_1* a jeho výstup priamo do prediktoru. Výstup prediktoru ukladám do premennej *predictions_from_view_1*. Do CNN Target posielam druhý augmentovaný obrázok *batch_view_2* a jeho výstup ukladám do premennej *targets_to_view_1*. Následne oba výstupy (*predictions_from_view_1* a *targets_to_view_1*) posielam do statickej funkcie *Loss()*, ktorá podľa vzorca v kapitole 5.6.5 vypočíta chybu. Takýto postup aplikujem aj v opačnom poradí, kedy do Online siete posielam na vstup *batch_view_2* a do Target siete *batch_view_1*. Na výstupy sietí rovnako aplikujem chybovú funkciu *Loss()* a z oboch chýb spravím priemer, ktorý je výstupnou chybovou hodnotou celej funkcie pre danú dávku obrázkov. Príklad tohto postupu v kóde vyzerá nasledovne:

```
def update(self, batch_view_1, batch_view_2):
    predictions_from_view_1 = self.predictor(self.Online_Net(batch_view_1))
    predictions_from_view_2 = self.predictor(self.Online_Net(batch_view_2))

    with torch.no_grad():
        targets_to_view_2 = self.Target_Net(batch_view_1)
        targets_to_view_1 = self.Target_Net(batch_view_2)

    loss = self.Loss(predictions_from_view_1, targets_to_view_1)
    loss += self.Loss(predictions_from_view_2, targets_to_view_2)
    return loss.mean()

def Loss(prediction, target):
    prediction_norm = F.normalize(prediction, dim=1)
    target_norm = F.normalize(target, dim=1)
    loss = 2 - 2 * (prediction_norm * target_norm).sum(dim=-1)
    return loss
```

Táto hodnota je chyba, ktorá sa spätnou propagáciou posiela naspäť v Online sieti. Toto je dôležitý rozdiel oproti kontrastívnym metódam SimCLR a MoCov2, ktoré pracujú s pozitívnymi aj negatívnymi pármi, zatiaľ čo BYOL iba s pozitívnym párom, čiže s dvomi augmentáciami jedného obrázku. Po úprave váh v Online sieti dochádza zavolaním funkcie *_update_Target_Net_params()* upravením váh Target siete taktiež podľa vzorca v kapitole 5.6.5. Aplikácia tohto vzorcu v kóde je nasledovná:

```
def update_Target_Net_params(self):
    for Params_Online, Params_Target in zip(self.Online_Net.parameters(),
```

```
                                                    self.Target_Net.parameters()):
        Params_Target.data = Params_Target.data * self.moment +
                             Params_Online.data * (1. - self.moment)
```

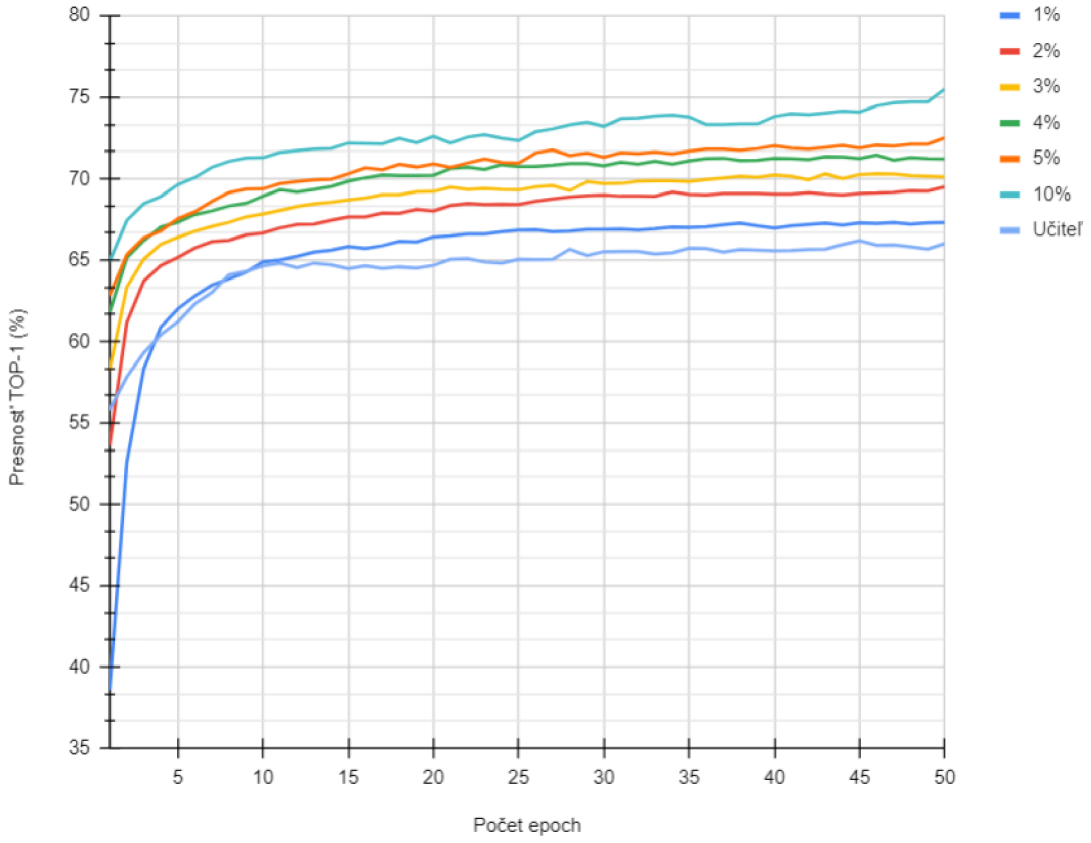
Po úprave váh sa opakuje tento cyklus dokým sa neprečíta celá dávka dát na danú epochu. Nasleduje výpočet chyby pre epochu a výpis informácie o jej veľkosti spolu s časom trénovania do konzoly a textového súboru. Každú N epochu sa vykreslí graf chyby a po dokončení trénovania sa všetky dáta spolu s Online a Target sieťami a optimizerom uložia na prípadné dotrénovanie.

7.5.3 Vyhodnotenie modelu:

SSL metóda - BYOL										
SSL		TEST1 - Lineárna evaluácia (presnosť) [%]							TEST3 - Porovnanie trénovania s učiteľom	
epoch	dávka	1%	2%	3%	4%	5%	10%	Učiteľ	SSL [%]	Učiteľ [%]
50	200	67,4	69,7	70,6	71,3	71,8	73,5	80,0	79,2	63,0
100	200	67,3	69,5	70,1	71,2	72,5	75,5	80,0	80,2	63,0
300	200	69,8	71,2	73,4	74,1	75,1	77,4	80,0	82,1	63,0
500	200	71,1	72,1	74,3	75,8	77,1	79,8	80,0	84,4	63,0
50	50	60,5	62,2	63,2	63,4	64,5	69,2	80,0	75,7	63,0
50	100	64,3	67,8	69,1	70,3	71,2	73,7	80,0	79,6	63,0
50	300	65,6	70,0	71,0	72,2	73,4	76,2	80,0	83,5	63,0
50	500	66,1	66,5	67,0	69,0	74,3	78,2	80,0	85,7	63,0

Tab. 7.12: Porovnanie výsledkov tréningu SSL modelu BYOL na testoch TEST1 a TEST3

Podľa získaných výsledkov TEST1 v Tab. 7.12 sa dá konštatovať, že metóda BYOL dosahuje väčšiu presnosť pri zvyšovaní počtu epoch, ako pri zväčšovaní dávky. Pri predtrénovaní na 500 epochách dosiahne model už pri 1% dát s označením v lineárnej evaluácii 71,1% presnosť. Naopak najväčší rast (+ 12,1%) je zaznamenaný pri predtrénovaní na 50 epochách a 500 veľkosť dávky, medzi 1% až 10% dotrénovaní na dátach s označením. Graf na obrázku Obr. 7.15 potvrdzuje efektívnosť metódy BYOL, ktorej stačí 1% dát pri dotrénovaní na porazenie modelu trénovaného s učiteľom. Rastúci počet epoch zvyšuje aj plochy pod krivkami a tiež skóre F1, zaznamenané v Tab. 7.13 (TEST1). Svoje maximá dosahujú pri trénovaní s 500 epochami a dávkou 200. Metóda BYOL je natoľko efektívna, že už pri použití jednej lineárnej vrstvy model dosahuje presnosť, ktorá poráža model trénovaný s

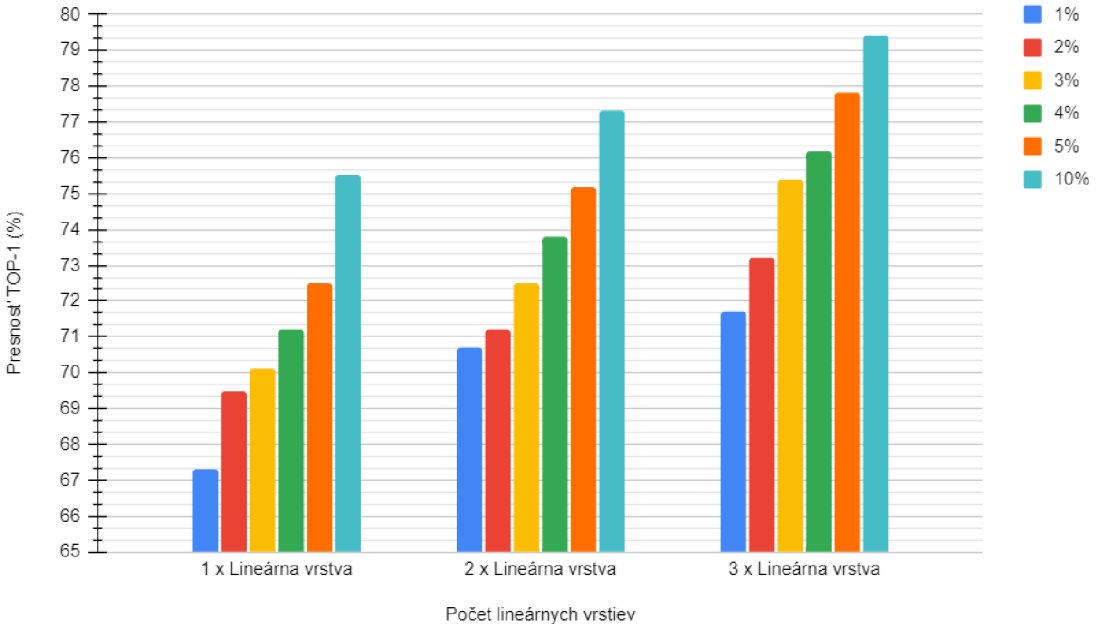


Obr. 7.15: Graf porovnania rôznych veľkostí označených dát na dotrénovanie SSL modelu BYOL (TEST1)

SSL		Vyhodnotenie SSL metódy BYOL, dotrénovanie 1%/5% dát s označením			
epoch	dávka	Presnosť [%]	AUC	PR plocha	F1 skóre
50	200	67,4 / 71,8	0,898 / 0,912	0,571 / 0,602	0,66 / 0,70
100	200	67,3 / 72,5	0,899 / 0,913	0,575 / 0,610	0,66 / 0,71
300	200	69,8 / 75,1	0,902 / 0,922	0,581 / 0,631	0,69 / 0,74
500	200	71,1 / 77,1	0,905 / 0,936	0,592 / 0,641	0,70 / 0,75
50	50	60,5 / 64,5	0,881 / 0,894	0,523 / 0,553	0,59 / 0,63
50	100	64,3 / 71,2	0,893 / 0,904	0,535 / 0,604	0,63 / 0,70
50	300	65,6 / 73,4	0,897 / 0,915	0,552 / 0,614	0,64 / 0,72
50	500	66,1 / 74,3	0,899 / 0,922	0,557 / 0,623	0,65 / 0,74

Tab. 7.13: Tabuľka porovnaní presností pre dotrénovanie s 1% a 5% dátach (SSL model BYOL, TEST1)

učiteľom. Z výsledkov TEST2 vyplýva, že najväčšia maximálna presnosť bola dosiahnutá pri aplikovaní 3 vrstiev pri lineárnej evaluácii a dotrénovaním s 10% dát a to 79,4%. Dáta z TEST3 sú zaznamenané v Tab. 7.12 a vyhodnotené v grafe na Obr.

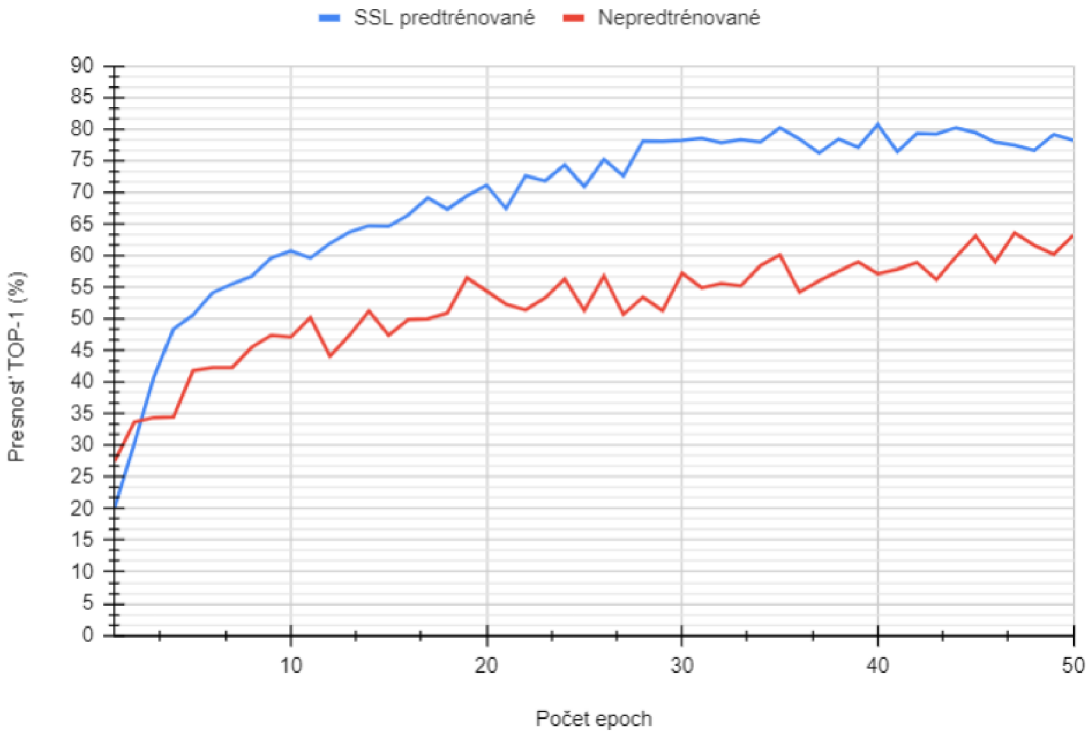


Obr. 7.16: Porovnanie počtu lineárnych vrstiev pri dotrénovaní SSL modelu s rôznou veľkosťou dát (TEST2)

Presnosť modelov	Množstvo označených dát	
	1%	5%
Tréning s učiteľom	40%	63%
SSL + tréning s učiteľom	67,3%	72,5%

Tab. 7.14: Porovnanie predtrénovaného SSL modelu BYOL a nepredtrénovaného modelu pri trénovaní s učiteľom na 1% a 5% trénovacích dát (TEST4)

7.17 (epoch 100, dávka 200). Maximum predtrénovaného modelu bolo dosiahnuté na úrovni presnosti 85,7% (epoch 50, dávka 500) a nepredtrénovaného 63,0%. Výsledok zaznamenaný v tabuľke Tab. 7.14 patrí k TEST4, z ktorého je vidieť ako už pri 1% dát dosahuje SSL model presnosť 67,3% a model bez predtrénovania 40%. Pri 5% sa tento náskok zníži, ale stále model s predtrénovaním dosahuje vyššiu presnosť o 9,5% (SSL - 72,5%, nepredtrénovaný - 63%). Jeden z posledných experimentov je porovnanie testovania modelov, ktoré majú rozdielny počet neurónov v MLPHead alebo rozdielnosť momentu (m) na úpravu váh cieľovej siete. Výsledky sú zaznamenané v tabuľke Tab. 7.15. SSL model bol trénovaný na 100 epochách a 200 dávke. Zvýšením počtu neurónov dochádza k mierne vyššej presnosti modelu. Neuróny zabezpečia kvalitnejšie naučenie vlastností vstupných dát, z ktorých model profituje. Najlepšie výsledky boli zaznamené pri momente (m) s hodnotou 0.99, ktorý som aj



Obr. 7.17: Porovnanie trénovania SSL modelu BYOL a prázdneho modelu trénovaného s učiteľom (TEST3)

	počet neurónov v MLPHead		Lineárna evaluácia (presnosť) [%]					
momentum (m)	skrytá vrstva	výstupná vrstva	1%	2%	3%	4%	5%	10%
0.9	512	128	68,2	69,5	70,1	71,2	72,3	75,2
0.99	512	128	67,4	69,7	70,6	71,3	71,8	75,5
0.999	512	128	65,5	66,1	69,2	70,5	71,3	72,4
1.0	512	128	45,0	48,2	49,5	50,5	51,3	55,2
0.99	1024	526	72,1	74,0	74,5	75,0	76,2	77,3
0.999	1024	526	67,0	69,2	70,4	70,5	71,0	75,2
1.0	1024	526	45,1	46,3	47,2	50,3	51,0	53,6

Tab. 7.15: Porovnanie výsledkov tréningu SSL modelu BYOL pri rôznych veľkostiach momentu a počtu neurónov v MLPHead

používal pri všetkých testoch TEST1, TEST2, TEST3 a TEST4.

Pri metóde BYOL je veľmi dôležité zachovať v projekčnej hlave aj dávkovú normalizáciu (batch normalization - BN), ktorá je zobrazená na obrázku Obr. 5.21 a vysvetlená v kapitole 4.5.1. Bez tejto dávkovej normalizácie predtrénovaný model

metódou BYOL dosiahol pri testovaní presnosť 27,3% až 32,2% pri dotrénovaní na 1% až 10% dátach s označením. Z toho plynie dôležitý poznatok, že normalizovať dáta naprieč dávkami je pri kontrastívnych metódach, ktoré pracujú s dávkami obsahujúcimi veľký počet augmentovaných obrázkov veľmi užitočný a praktický krok, na zvýšenie presnosti pri generalizácii. Dávkovou normalizáciou sa tak model vyhne kolapsu. Príklad kolapsu by bol, ak by výsledný vektor zo siete mal vždy rovnaký výstup ([0, 1, 0, 0,...] a teda sieť sa iba potrebuje naučiť funkciu "q", ktorá sa čo najviac priblíži tomuto vektoru projekcie - "z" a tým dosiahne perfektnú presnosť. Použitím dávkovej normalizácie sa nikdy nemôže stať, že by výstupná projekcia vo vektore "z" bola rovnaká, ako predtým a tým sa zabráni kolapsu pri trénovaní.

Metóda BYOL je podľa výsledkov efektívna a dokáže dosiahnuť najväčšiu presnosť. Už pri dotrénovaní s 1% dát dokáže porážať model trénovaný s učiteľom (ak sa samozrejme SSL model trénuje s väčšou dávkou a viac epochami, ako model trénovaný s učiteľom). Vzájomné porovnanie všetkých metód je v poslednej kapitole mojej diplomovej práce.

7.6 Metóda trénovania s učiteľom:

Aby bolo možné porovnať jednotlivé modely predtrénované rôznymi SSL metódami s modelom, ktorý sa učí bežným spôsobom, teda s učiteľom na dátach s označením, vytvoril som kód na trénovanie takéhoto modelu spolu s mierne upraveným datasetom.

7.6.1 Príprava datasetu:

Dataset STL10 ponúka 3 skupiny obrázkov a to obrázky bez označenia (100 000), obrázky s označením na tréning (5 000) a obrázky s označením na test (8 000). Dokopy je obrázkov s označením 13 000 voči 100 000 obrázkom bez označenia. Tieto obrázky bez označenia používam na SSL tréning. Síce je obrázkov s označením určených na tréning až 5000, rozhodol som sa ich spojiť s obrázkami určenými na test (8 000) a tým vytvoriť jeden dataset o veľkosti 13 000 obrázkov s označením. Následne tento dataset rozdelím na 10 000 obrázkov pre tréning modelu a zvyšných 3 000 na test modelu. Je to z dôvodu lepšieho natrénovania dodaním viacerých obrázkov, presne dva krát toľko, ako by som trénoval iba s pôvodne určenými na tréning. Obrázky sťahujem obdobným príkazom ako pri neoznačených a aplikujem na ne mierne augmentácie. Tie aplikujem z dôvodu zvýšenia presnosti výsledného modelu, vďaka čomu sa mi podarilo zvýšiť presnosť o 8% z pôvodných 55% na 63%. Augmentácie spočívajú iba z horizontálneho otočenia, transformácie do tensoru a normalizácie. Sú aplikované pomocou príkazov z knižnice *torchvision.transforms* a

spojené dokopy príkazom *Compose()* z knižnice *torchvision*. Z takto pripravených dát následne spravím dataloader pre tréning a test a dáta sú pripravené na tréning. Príklad stiahnutia dát, aplikácie augmentácii, rozdelenia na tréningové a testovacie dáta a vytvorenie dataloaderu je v nasledujúcom kóde.

```
data_transforms_supervised = transforms.Compose([
        transforms.RandomResizedCrop(shape),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

train_dataset = datasets.STL10('..datasets..',
        split='train', download=True,
        transform=data_transforms_supervised)

test_dataset = datasets.STL10('..datasets..',
        split='test', download=True,
        transform=data_transforms_supervised)

dataset_together = train_dataset + test_dataset
train_part_dataset, test_part_dataset =
    torch.utils.data.random_split(dataset_together, [10000, 3000])

print("Input shape:", train_part_dataset[0][0].shape)
train_loader = DataLoader(train_part_dataset,
    batch_size=SUPERVISED_BATCH, num_workers=4,
    drop_last=False, shuffle=True)
test_loader = DataLoader(test_part_dataset,
    batch_size=SUPERVISED_BATCH, num_workers=4,
    drop_last=False, shuffle=True)

supervised_datasets_STL10 = {'train': train_part_dataset,
        'test': test_part_dataset}
supervised_dataloaders_STL10 = {'train': train_loader,
        'test': test_loader}

image_datasets = {x: supervised_datasets_STL10[x] for x in ['train', 'test']}
dataloaders =    {x: supervised_dataloaders_STL10[x] for x in ['train', 'test']}
```

7.6.2 Tréning modelu s učiteľom:

Pred tréningom modelu inicializujem novú konvolučnú sieť Resnet18. Musí to byť rovnaká architektúra ako pri SSL modeloch aby ich porovnanie bolo založené na rovnakých parametroch. Následne definujem veľkosť učiaceho kroku na hodnotu 0.005, definujem chybovú funkciu *nn.CrossEntropyLoss()* a optimizer typu SGD. Aby som dosiahol čo najlepší výsledok nastavenia váh, čiže čo najoptimálnejšie nájdenie globálneho minima, definoval som funkciu *lr_scheduler.StepLR()*, ktorá má za cieľ každých 7 epoch znížiť hodnotu učiaceho parametru o 10%. Po definovaní týchto premenných urobím inicializáciu objektu *Supervised* so vstupnými parametrami modelu, optimizeru, dataloadrov, chybovej funkcie a vybraním typu modelu podľa toho, či trénujem model s alebo bez predtrénovania. V tomto prípade je to bez predtrénovania a model bude trénovaný s neupravenými váhami. Následne cez funkciu *train_test_model()* prislúchajúcu objektu spúšťam tréning na nastavený počet epoch (50).

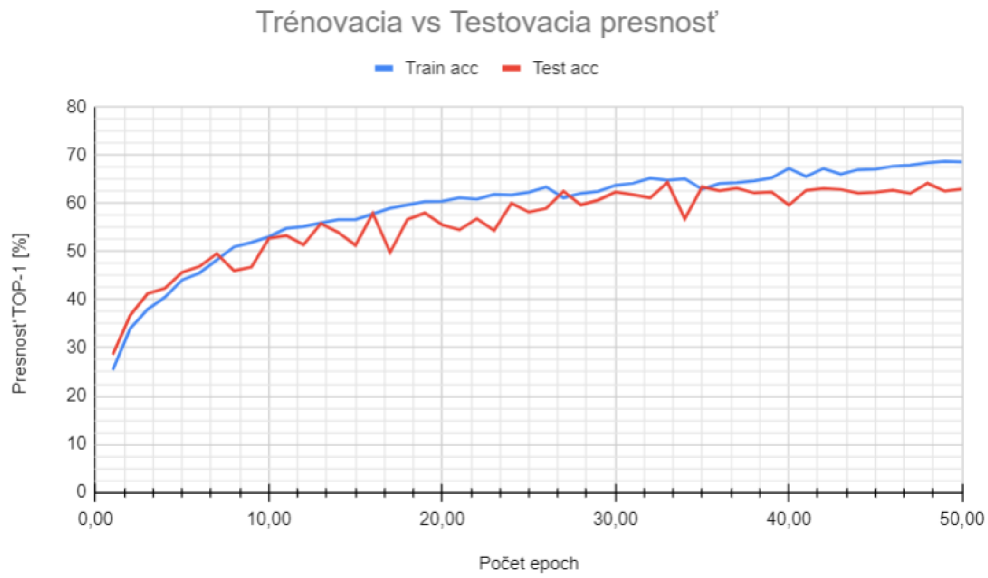
Trénovací cyklus obsahuje 3 vnorené FOR cykly. Prvý FOR cyklus iteruje počet epoch, druhý FOR cyklus iteruje podľa typu fázy, či dochádza k tréningu alebo testu modelu a následne pre jednu z fáz iteruje tretí FOR cyklus cez celý dataloader. Z dataloaderu vyberá dáta a označenia dát, posiela ich na vstup modelu, následne výstup porovnáva so skutočnou pravdou v chybovej funkcii a ak je nastavený tréning modelu, posiela vypočítaný gradient späť na úpravu váh modelu. Chyba aj presnosť sa priebežne sčítava a po konci epochy sa podelí počtom dát v dataloaderi. Hodnoty chyby a presnosti zapisujem do zoznamu a priebežne po konci každej epochy vypisujem. Po skončení tréningu alebo testovania zapíšem namerané hodnoty do textového súboru, konzoly a model uložím na následné porovnávanie s SSL modelmi.

7.6.3 Vyhodnotenie modelu:

Model natrénovaný metódou s učiteľom vyhodnocujem pomocou presnosti, ale aj za aplikovania zložitejších metód ako chybová matica, krivky ROC a PR, plochou pod krivkami a tiež skóre F1.

Na obrázku Obr. 7.18 vykreslujem do grafu presnosť pri tréningu a presnosť pri teste s 50 epochami a 32 veľkosťou dávky.

V prílohe A na obrázkoch Obr. B.2 a Obr. B.3 zobrazujem krivky ROC (Receiver operating characteristic) a PR (precision recall). V ich dolných pravých rohoch sú aj plochy pod krivkami pre jednotlivé triedy z datasetu STL10.



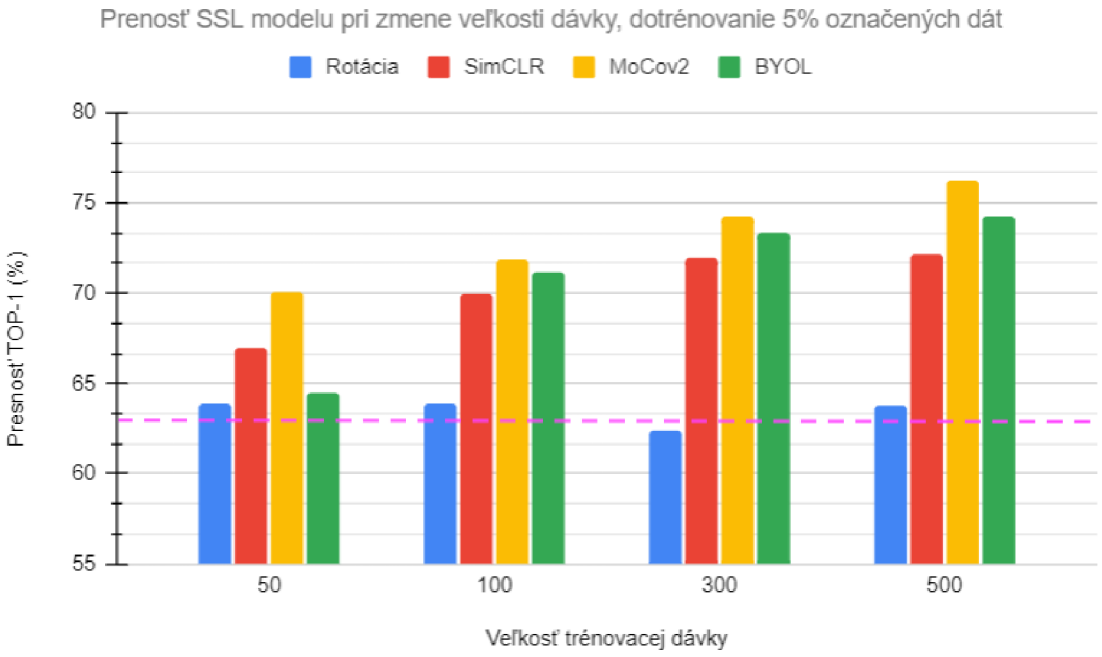
Obr. 7.18: Porovnanie presnosti pri tréningu a pri teste metódy s učiteľom

7.7 Evaluácia modelov na datasete STL10:

V poslednej sekcii vyhodnotenia na datasete STL10 porovnávam presnosť a úspešnosť predtrénovaných modelov SSL metódami. Robím to z dôvodu jasnejšieho vyjadrenia kvality modelov pri rovnakých podmienkach.

Na obrázku Obr. 7.19 zobrazujem graf porovnania presností modelov pri zmene veľkosti dávky a následne dotrénovania na jednej lineárnej vrstve s 5% dát. Jednotlivé modely sú farebne odlíšené. Počas trénovania modelov som použil štyri rôzne veľkosti dávok a to 50, 100, 300 a 500. Počet trénovacích epoch bol konštantný a to vo veľkosti 50. Zo zvyšovania veľkosti dávky najviac ťaží metóda MoCov2, ktorej už stačí aj dávka s veľkosťou 50 a je schopná pri rovnakých podmienkach prekonať ostatné 3 metódy. Nasledujúcim rastom dávky metóda MoCov2 dosahuje vždy svoje maximum v porovnaní s ostatnými metódami. Pri predtrénovaní na najväčšej dávke (500) dosiahla metóda MoCov2 najväčšiu presnosť zo všetkých modelov a to 76,2%. Zvyšné dve kontrastívne metódy (SimCLR a BYOL) taktiež zvyšujú svoju presnosť rastúcou veľkosťou dávky, ale v menšej miere oproti MoCov2. SSL metóda rotácie naopak vôbec nezvyšuje svoju presnosť s rastúcou dávkou. Prerušovaná ružová čiara značí úroveň presnosti modelu trénovaného s učiteľom na datasete STL10, architektúrou Resnet18, veľkosťou dávky 32 a 50 epochami.

Druhé hlavné porovnanie, ktoré sa z nameraných dát dá vykresliť je porovnanie vývoja presnosti modelov pri zmene počtu trénovacích epoch. Modely boli trénované na konštantnej dávke 200 a počet epoch sa vyvíjal 50, 100, 300 a 500. Jednotlivé

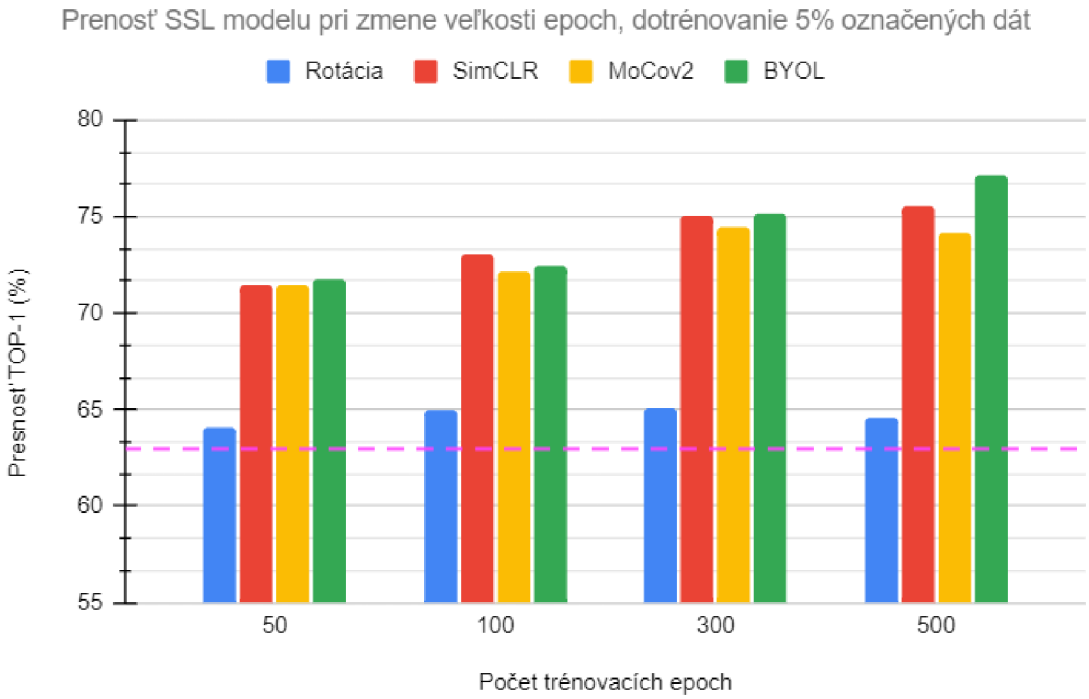


Obr. 7.19: Porovnanie presnosti SSL metód pri dotrénovaní na 5% dát s označením a zvyšovaním veľkosti dávky

metódy na obrázku Obr. 7.20 sú znovu rovnako farebne odlíšené, ako pri predošlom zobrazení. Zvyšovaním počtu epoch presnosť kontrastívnych metód rastie. V tomto prípade metódy SimCLR a BYOL profitujú z rastu epoch viac ako MoCov2 a metóda rotácie. Metóda BYOL prekonáva zvyšné metódy pri počte epoch 500 s presnosťou 77,1%.

Jeden z experimentov (TEST3) bolo trénovanie SSL modelu a nepredtrénovaného modelu spôsobom trénovania s učiteľom. Predtrénované modely pravidelne dokázali udržať od prvých epoch až po koniec tréningu vyššiu presnosť. Vzájomné porovnanie takéhoto tréningu a všetkých metód spolu s tréningom s učiteľom je zobrazené v grafe na obrázku Obr. 7.21.

Posledný experiment v každej metóde (TEST4) pri jednotlivých metódach bolo aj trénovanie predtrénovaných modelov spôsobom tréningu s učiteľom. Tento tréning bol ale urobený iba za použitia 1% alebo 5% trénovacích dát STL10 datasetu. Z porovnania na obrázku Obr. 7.22 vyplýva, že nepredtrénovaný model (modrou farbou) pri 1% označených dát stráca minimálne 18,5% na druhý najmenej presný model Rotácie. Najpresnejší predtrénovaný model pri dodaní iba 1% označených dát je SimCLR. Pri dodaní 5% označených dát už model trénovaný s nepredtrénovanými váhami dobieha SSL modely, ale stále zostáva najpresnejšia metóda SimCLR s 73,1% presnosťou. Všetky modely boli trénované s jednotlivými SSL metódami na



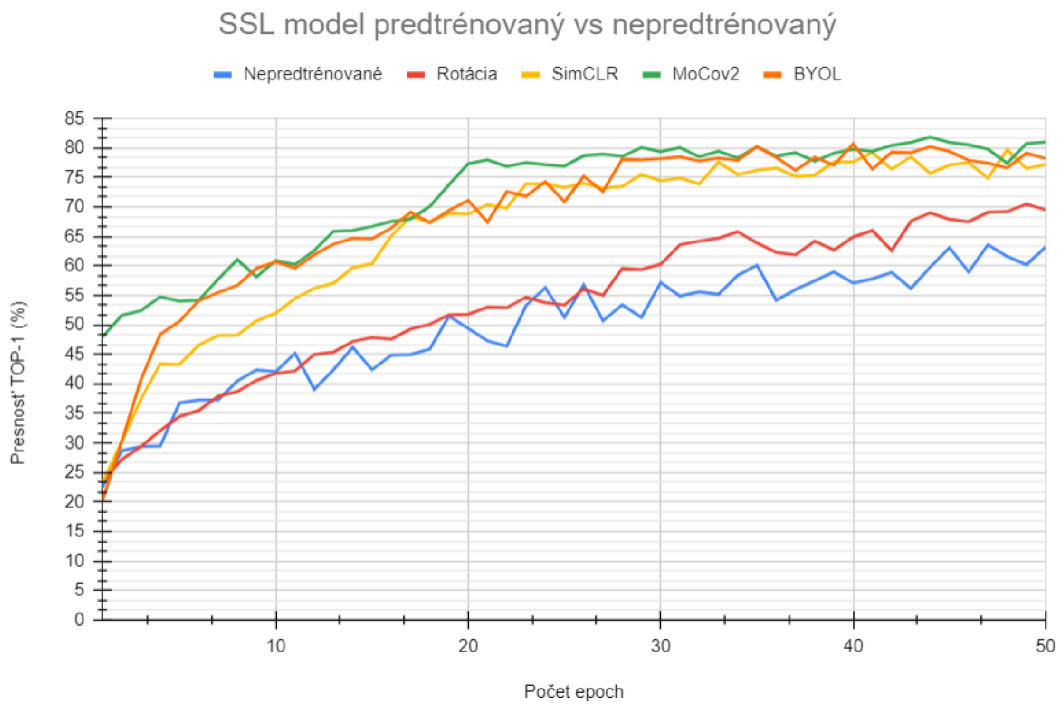
Obr. 7.20: Porovnanie presnosti SSL metód pri dotrénovaní na 5% dát s označením a zvyšovaním počtu epoch

100 epochách s veľkosťou dávky 200 a následne dotrénované pri 1 lineárnej vrstve na 50 epochách a s veľkosťou dávky 500. Tréning modelu s učiteľom sa uskutočnil pri 50 epochách a veľkosti dávky 32.

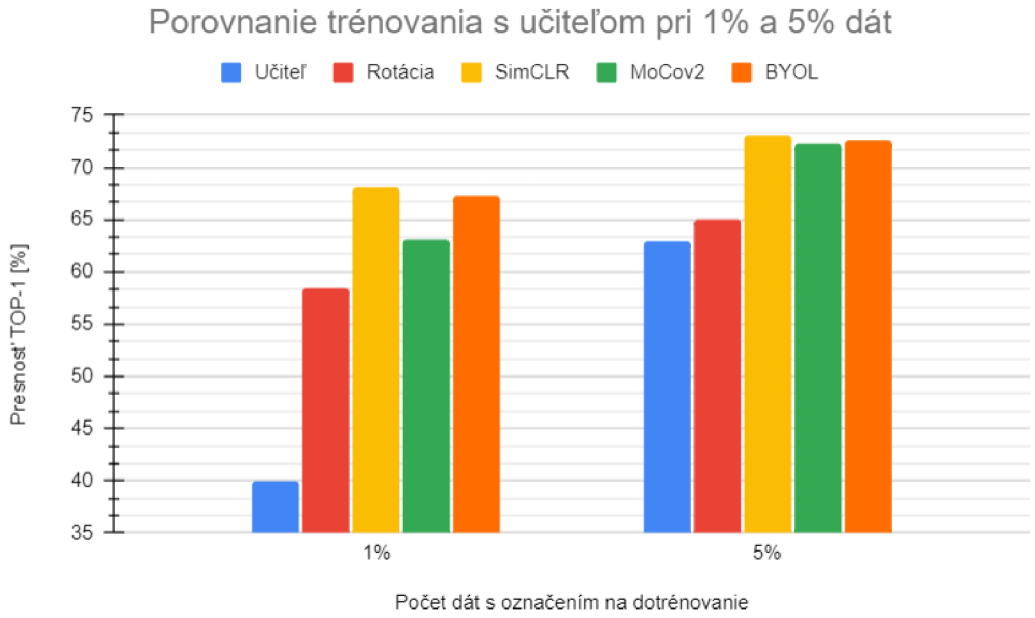
7.8 Evaluácia modelov na rôznych datasetoch:

V predošlej kapitole som sa venoval podrobnému vyhodnoteniu všetkých 4 SSL metód na datasete STL10 a následnému porovnaniu medzi sebou. Aby som poukázal na možné aplikovanie SSL metód aj s inými dátami, čo sa týka veľkosti obrázkov, počtu trénovacích a testovacích dát, v tejto poslednej kapitole vyhodnocujem evaluáciu SSL metód na rôznych datasetoch. Konkrétne ide o skôr spomenuté datasety CIFAR10, STL10, MNIST, Imagenete, Weather a Industry dataset. Podrobne popísané datasety (počet obrázkov a ich veľkosť, je v úvodnej kapitole praktickej časti 6.1).

Presnosť jednotlivých testov datasetov vyhodnocujem v grafoch (Obr. 7.23 - 7.26) vyhodnocujúcim každú metódu zvlášť, kde na vertikálnej osy je zobrazená presnosť metódy a na horizontálnej osy názov datasetov. Porovnané SSL modely boli predtrénované s veľkosťou dávky 500 a 50 epochami, tréning s učiteľom s dávkou 32



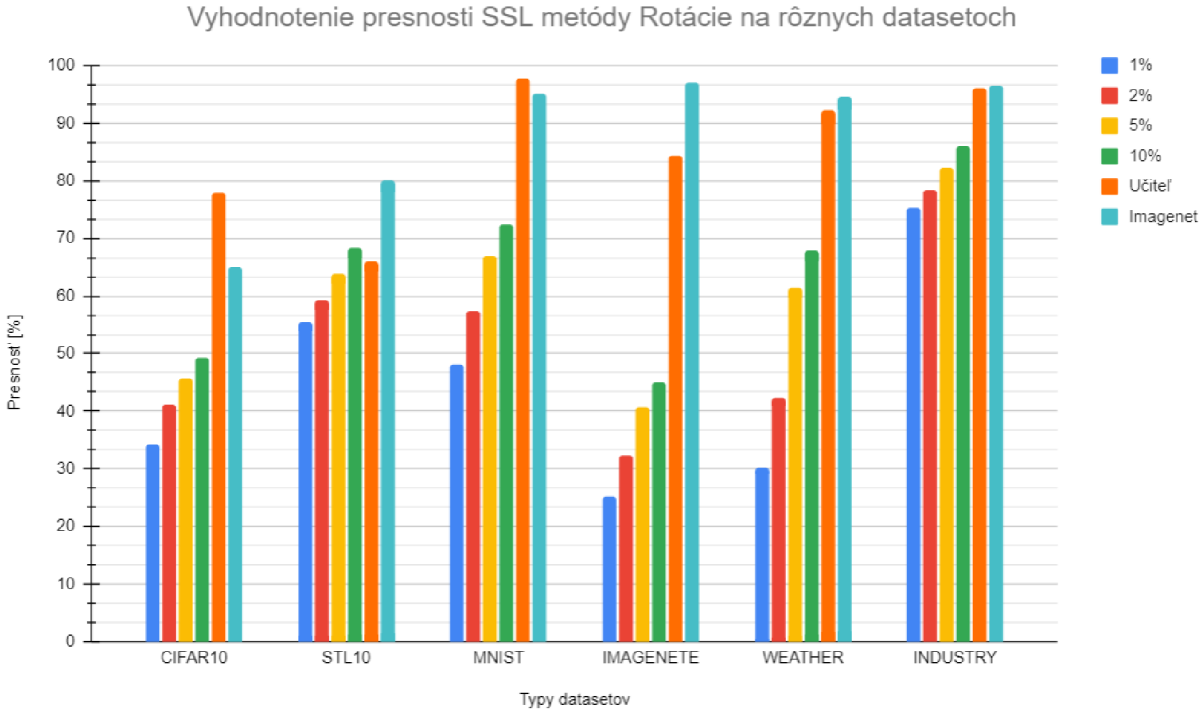
Obr. 7.21: Porovnanie vývoja presnosti pri trénovaní s učiteľom modelov predtrénovaných na SSL metódach a prázdneho modelu (TEST3)



Obr. 7.22: Porovnanie presnosti SSL metód a modelu trénovaného s učiteľom pri dotrénovaní na 1% 5% dát s označením a zvyšovaním počtu epoch (TEST4)

na 50 epochách a lineárna evaluácia s jednou vrstvou pri dávke o veľkosti 500 a 50 epochách. V grafoch je vyhodnotenie predtrénovaných modelov dotrénovaných na 1%, 2%, 5% a 10% dát s označením v porovnaní s modelom trénovaným s učiteľom a tiež s modelom predtrénovanom na veľkom datasete Imagenet.

SSL metóda Rotácie (Obr. 7.23) dosiahla svoje maximá pri Industry datasete a minimá pri Imagenet datasete. Svoju efektivitu preukázala pri datasete STL10, kedy sa jej pri 10% dotrénovaných dátach podarilo prekonať model trénovaný s učiteľom.

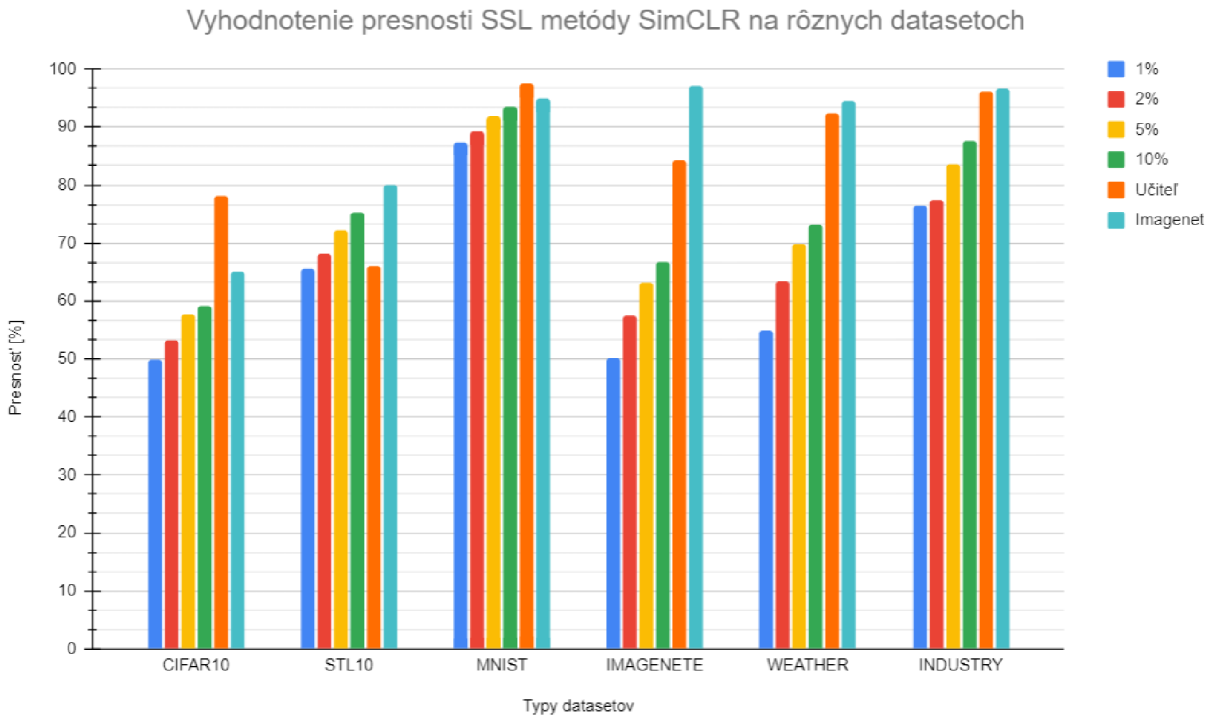


Obr. 7.23: Porovnanie presnosti všetkých datasetov na SSL metóde Rotácie

Metóda SimCLR (Obr. 7.24) dosiahla svoje minimum pri datasete Imagenet a maximum pri datasete MNIST. Pri datasete STL10 sa podarilo prekonať model trénovaný s učiteľom a pri MNIST a Industry datasete sa ku nemu priblížiť.

MoCov2 je metóda, ktorá profituje z počtu obrázkov v datasete. Pri datasete MNIST dosahuje maximum a prekonáva všetky metódy s presnosťou 94,6%. Pri datasete STL10 prekonáva ostatné metódy, ak sa dotrénuje s 10% dát a to s presnosťou 79,3%. Pri datasete Imagenet, kde všetky metódy dosiahli nižšiu presnosť, metóda MoCov2 dosahuje pri 1% dát presnosť 51,3% čo je najviac z SSL metód. Jej vyhodnotenie je na obrázku Obr. 7.25.

Posledná použitá kontrastívna metóda je BYOL (Obr. 7.26), ktorá svoje maximum tiež dosiahla na datasete MNIST a to 93,1%. Pri datasete STL10 sa pri

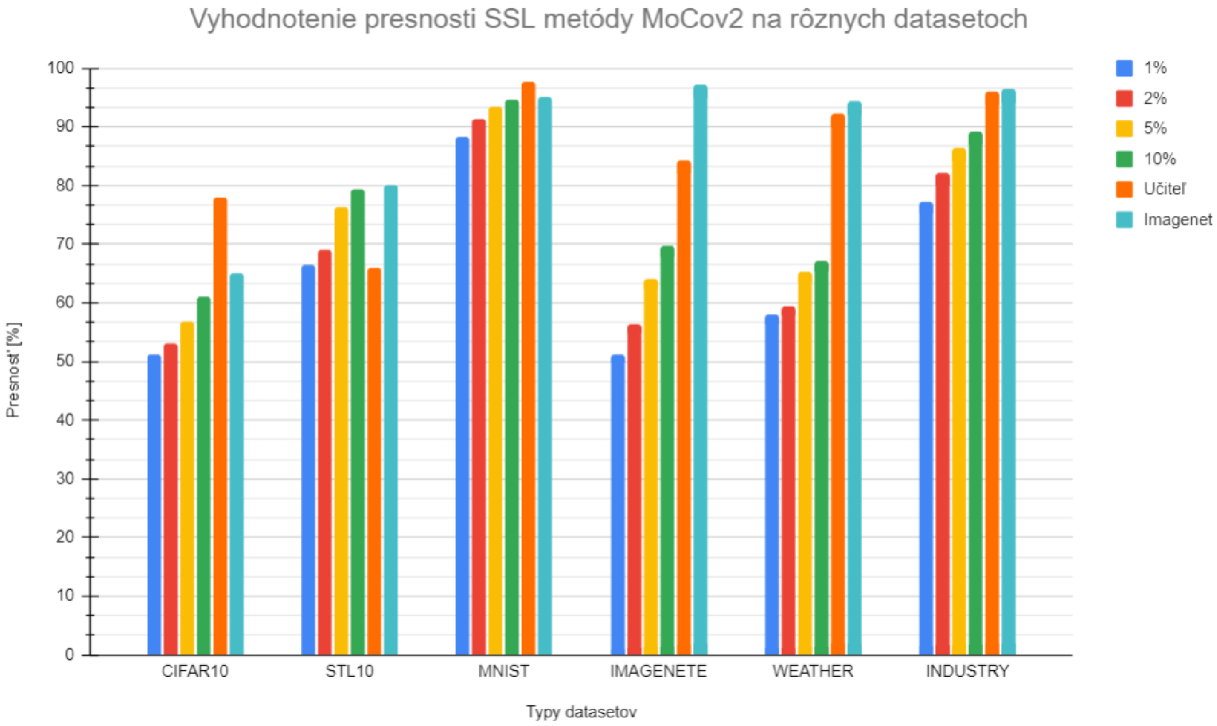


Obr. 7.24: Porovnanie presnosti všetkých datasetov na SSL metóde SimCLR

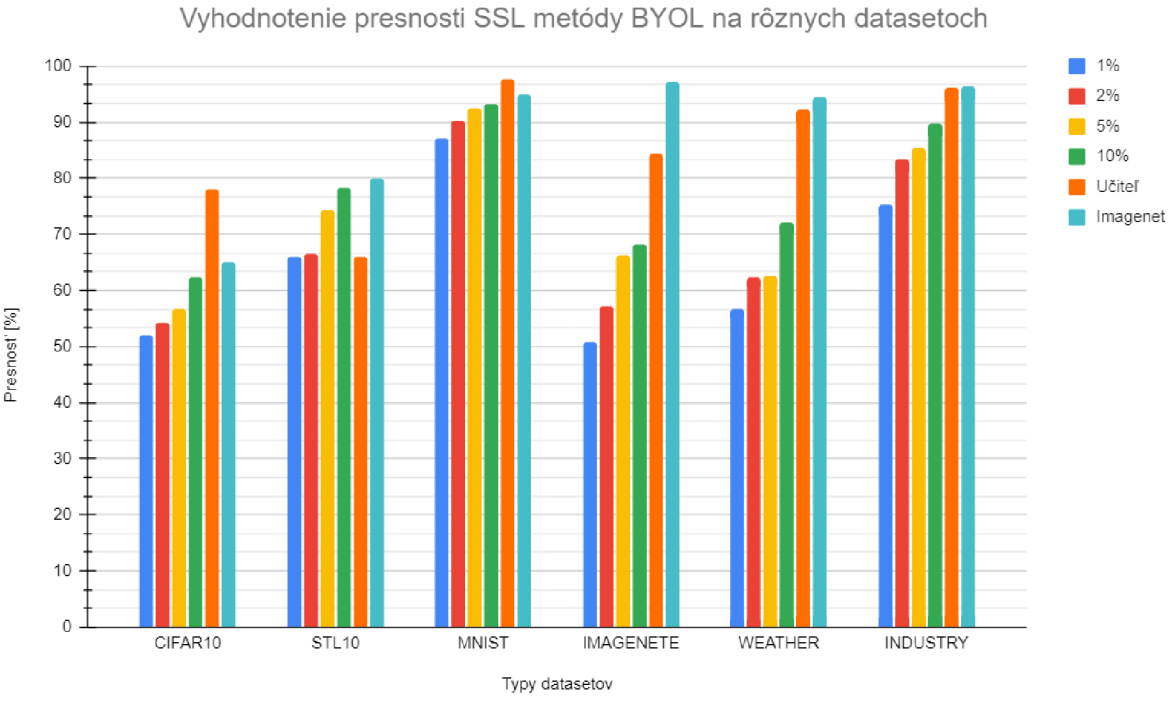
dotrénovaní na 1% dát dokázala dosiahnuť vyššiu presnosť, ako trénovanie s učiteľom.

Na základe zaznamenaných dát z testu SSL metód na rôznych datasetoch môžem konštatovať, že trénovanie datasetu s menším počtom tried výrazne pomáha vyhodnocovanej presnosti. Je to príklad datasetu Industry, na ktorom bola dosiahnutá minimálna presnosť už 75,2% (metóda rotácie pri 1% dotrénovaných dát). Ostatné metódy a dotrénovanie s viac percentami označených dát pomohlo zvýšiť presnosť. Maximálna presnosť bola dosiahnutá pri metóde MoCov2 a to 89,2% pri dotrénovaní s 10% dát. Trénovanie s učiteľom na tomto datasete dosiahlo presnosť 96,1%.

Pri datasetoch Imagenet, CIFAR10 a Weather metódy dosahovali najmenšiu presnosť. Predpokladám, že je to z dôvodu obtiažnosti datasetov. Totižto pri prvých dvoch menovaných sa nachádza 10 tried obrázkov. Je to síce rovnaký počet, ako pri STL10, ale pri CIFAR10 je možnosť trénovať SSL metódy s 50 000 obrázkami a pri Imagenet iba s 10 000. Tento rozdiel oproti STL10 (pri ktorom je 100 000 neoznačených obrázkov na tréning) môže predstavovať aj nižšiu presnosť modelov. Dataset Weather je zložitý aj z hľadiska počasia, kedy sa napríklad CNN môže z obrázku dažďa učiť aj iné vlastnosti, ako napríklad predmety a to ju môžu mýliť. Zároveň je pri datasete Weather možné trénovať iba s 1200 obrázkami.



Obr. 7.25: Porovnanie presnosti všetkých datasetov na SSL metóde MoCov2



Obr. 7.26: Porovnanie presnosti všetkých datasetov na SSL metóde BYOL

Dataset STL10 je hlboko vysvetlený v predošlej kapitole a presnosť metód na tomto datasete je priemerná oproti iným datasetom. Hodnotím ho, ako najvhodnejší dataset vzhľadom na počet trénovacích obrázkov bez označenia (100 000), vďaka ktorým sa metódy naučia dostatok vlastností o obrázkoch. MNIST je dataset, na ktorom modely predtrénované na kontrastívnych metódach dosiahli najväčšiu presnosť. Aplikovanie metódy rotácie zabezpečilo najnižšiu presnosť 48,1% (1% dát) a najvyššiu 72,5% (10% dát). Metóda SimCLR už pri 1% dotrénovacích dát dosiahla presnosť 87,3%. Najvyššia presnosť na datasete MNIST je 94,6% pri metóde MoCov2 s 10% dotrénovaných dát. Model trénovaný s učiteľom dosiahol presnosť 97,6%. Vysoká presnosť metód pri datasete MNIST je z môjho hladiska spôsobená typom obrázkov. V datasete je síce 10 tried, ale obrázky v jednotlivých triedach sú si vždy veľmi podobné. V každej triede sa nachádza 6 000 čísloviek a to je dostatočný počet na naučenia vlastností obrázkov aj bez označenia.

Ako obmedzujúce podmienky by som definoval veľkosť a zložitosť datasetu. Príklad vidíme pri porovnaní počtu tried (dataset Industry voči datasetom STL10, CIFAR10 alebo Weather). Druhý príklad je tiež počet trénovacích obrázkov, kedy napriek rovnakému počtu tried má dataset STL10 dva-krát viac trénovacích obrázkov, ako podobný dataset CIFAR10 a dosiahne vďaka tomu vyššiu presnosť pri všetkých metódach. Treťou obmedzujúcou podmienkou je zložitosť dát, čo môžeme vidieť z presnosti modelov trénovaných na datasete MNIST, kedy modely dosahujú vysokú presnosť a naopak pri datasete Weather alebo Imagenet, kde je presnosť nižšia. Obmedzujúcou podmienkou pri kontrastívnych metódach je aj typ a sila aplikovaných augmentácií. Toto tvrdenie potvrdzuje experiment v metóde SimCLR, kde bez použitia augmentácií náhodného vyrezania a zmeny sfarbenia, model nebol schopný sa naučiť dostatok vlastností o vstupných dátach. Ďalším obmedzujúcim faktorom je výber počtu epoch a veľkosti dávky pri predtrénovaní modelov a tiež definovanie počtu neurónov v MLPHead.

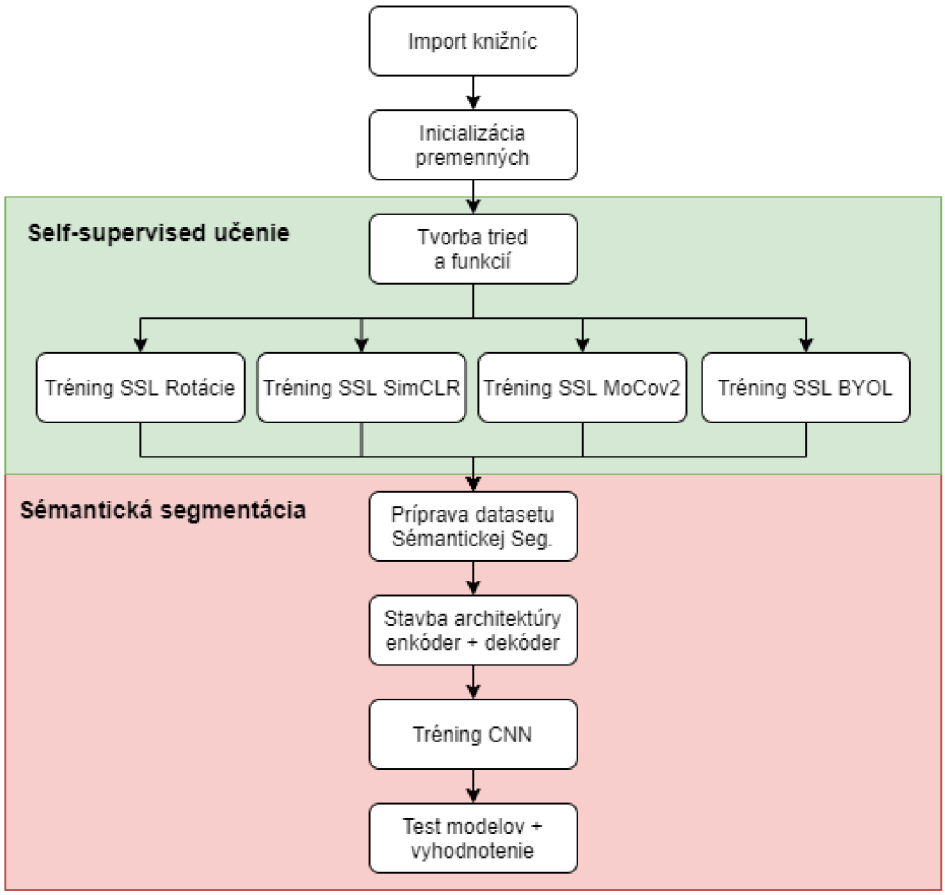
8 Sématická segmentácia

Úloha sémantickej segmentácie je ďalšia z aplikácií počítačového videnia, pri ktorej je cieľom vykresliť triedy priamo do vstupného obrázku s tým, že rovnaká trieda má rovnakú farbu označenia na obrázku. V tejto úlohe som sa rozhodol použiť dataset Cityscapes nakoľko obsahoval už vytvorené triedy k obrázkom a tiež bol verejne dostupný. V nasledujúcich kapitolách popisujem tvorbu kódu, prácu s datasetom, tréning CNN, využitie SSL metód a vyhodnotenie úspešnosti.

8.1 Postup tvorby kódu

Pri sémantickej segmentácii sa rovnako, ako pri iných úlohách počítačového videnia používa CNN. V tomto prípade sa celý model skladá z dvoch CNN a to časti enkóder a dekóder. Prvá časť, enkóder, má za úlohu naučiť sa spočiatku vlastnosti obrazu na nízkej úrovni ako sú čiary, hrany, farby atď. a hlbšie vrstvy CNN sa učia prvky na vyššej úrovni, ako sú tvary alebo celé objekty. Konvolučné vrstvy sú spojené s vrstvami s podvzorkovaním, aby sa vytvoril vektor vlastností s nízkym rozlíšením. Tento vektor vlastností obsahuje informácie z nižšej aj vyššej úrovne o vstupnom obrázku obsiahnutom v jeho rôznych konvolučných kanáloch. Vo väčšine implementácií má tento vektor najnižšie rozlíšenie s najväčším počtom kanálov. Následne je potrebné tento vektor zväčšiť na pôvodný obrázok, aby som splnil úlohu sémantickej segmentácie. Vektor vlastností vstupuje do dekóderu, ktorý sa skladá z nadvzorkovávacích vrstiev a tiež konvolučných vrstiev s cieľom vytvoriť mapy funkcií s vyšším rozlíšením. Zvyšovaním rozlíšenia sa súčasne znižuje počet kanálov v mapách funkcií. Dokopy sa takáto architektúra nazýva enkóder-dekóder a je zobrazená na obrázku Obr. 2.5. Postup celého algoritmu sémantickej segmentácie v mojej diplomovej práci zobrazujem na obrázku Obr. 8.1. V mojej práci som ako enkóder použil sieť Resnet50 vzhľadom na predošlú skúsenosť tréningu na Resnet sieťach a tiež vďaka jej vlastnosti reziduálnych spojení, ktoré dosahujú nižšiu trénovaciu chybu. Jednotlivé modely, ktoré budem trénovať SSL metódou sú taktiež už z predošlej úlohy klasifikácie predpripravené na prácu s Resnet sieťou.

Ako prvý krok aplikujem import knižníc, ktorý je identický k použitým knižniciam v úlohe klasifikácie. Nasleduje Inicializácia premenných, medzi ktoré patrí počet trénovacích epoch pre každý SSL model a úlohu sémantickej segmentácie. Veľkosť dávky pre všetky modely. Názov datasetu a použitej siete. Počet neurónov pre predikčnú hlavu v SSL tréningu. Tvorba súboru na uloženie a zápis všetkých potrebných informácií do textového súboru. Za definovaním potrebných premenných nasleduje rovnaká tvorba tried a funkcií, ako pri úlohe klasifikácie s cieľom vytvoriť architektúry na SSL tréning metódami Rotácie, SimCLR, MoCov2 a BYOL spolu s



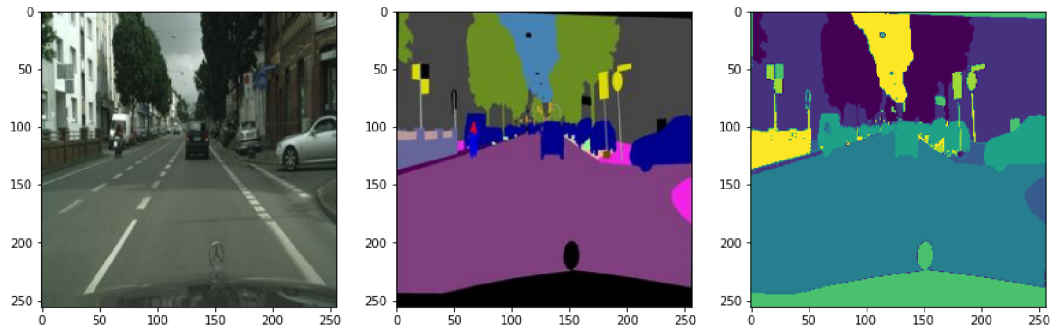
Obr. 8.1: Vývojový diagram sémantickej segmentácie

funkciou na vykreslenie grafov. Po tvorbe architektúry nasleduje trénovanie každej SSL metódy datasetom Cityscapes, z ktorého používam všetkých 2975 obrázkov bez označenia tried na naučenie reprezentácií/vlastností daného predtrénovaného SSL modelu. Tréning metód je podrobne popísaný v predošlých kapitolách. Po úspešnom tréningu metód pokračuje algoritmus k najdôležitejšej časti a to sémantickej segmentácie.

Vzhľadom na to, že SSL tréning niekedy trvá v rádoch hodín, tak algoritmus sémantickej segmentácie je možné spustiť samostatne. Z toho dôvodu niekoľko potrebných knižníc pre túto časť kódu v úvode importujem. Dôležitá je knižnica *torch* a *torchvision*, ktoré používam na prácu s obrázkami a CNN. Taktiež používam *sklearn* a z nej model KMeans, na tvorbu tried. Po importovaní knižníc dochádza k načítaniu obrazových dát do premenných train_dir a test_dir. Počet obrázkov môže byť určený po maximálny počet obsahujúci v datasete a to 2975 na tréning a 500 na testovanie/validáciu. Aby pri porovnaní SSL predtrénovaných modelov bolo možné trénovať s väčším počtom obrázkov, rozhodol som sa použiť všetkých 2975 bez označenia tried na tréning a následne 500 obrázkov aj s označením tried na dotrénovanie

modelov v časti sémantickej segmentácie.

Dataset sa skladá z 8 hlavných tried: zem, človek, vozidlo, budova, objekt, príroda, obloha a zvyšok (ako napríklad poznávacia značka) a dokopy 30 podtried. Ja som sa rozhodol pracovať iba s triedami vzhľadom na cieľ dosiahnutia vyššej presnosti pri sémantickej segmentácii. Všetky obrázky s označením si uložím do poľa, na ktoré aplikujem zhlukovaciu metódu KMeans so zadaným počtom tried. KMeans vytvorí 8 skupín (clusters) a pre každú priradí jednu farbu. Výsledkom tejto operácie je, že všetky autá budú jednotne označené a všetky budovy taktiež, ale inou farbou a to platí pre všetkých 8 tried z obrázku. Príklad vstupného obrázku, obrázku s označením tried a podtried a výstupného obrázku z metódy KMeans je na obrázku Obr. 8.2. Takéto vytvorenie obrázku s označenými triedami priradeného k vstupnému obrázku uskutočňujem pri príprave datasetu a následne dataloadru.



Obr. 8.2: Cityscapes dataset, označenie tried, výsledok KMeans metódy pri označení tried

Tvorba architektúry CNN pre sémantickú segmentáciu pozostáva z enkóderu a dekóderu. Ako enkóder som zvolil CNN Resnet50, ktorú buď používam nepredtrénovanú a ako čistú sieť trénujem od začiatku v architektúre enkóder+dekóder alebo používam predtrénovanú sieť Resnet50 na niektorej zo 4 SSL metód. Spojenie týchto dvoch CNN sa deje pomocou "mostu" (vrstvy spájajúcej enkóder a dekóder), cez ktorý ide vektor vlastností a jeho veľkosť som stanovil na 2048 neurónov. Za mostom dochádza k tvorbe dekóderu a postupnému zväčšovaniu veľkosti vrstiev a skracovaniu dĺžky vektoru. Počet vrstiev je stanovených na 6 a ich veľkosť sa každou úrovňou zmenší o polovicu až v poslednej vrstve je vstupný počet 64 a výstupný počet tried, čiže 8. Pri tvorbe dekódera som sa inšpiroval verejne dostupným kódom od Kevin Chyunlu [84].

Po tvorbe architektúry dochádza k trénovaniu, pri ktorom aplikujem vedomosti z trénovania metód pri úlohe klasifikácie. Na začiatku tréningu inicializujem potrebné premenné, stopujem čas, prechádzam vstupné dáta podľa veľkosti dávky a ako chybovú funkciu používam *CrossEntropyLoss()*. Natrénovaný model po každej

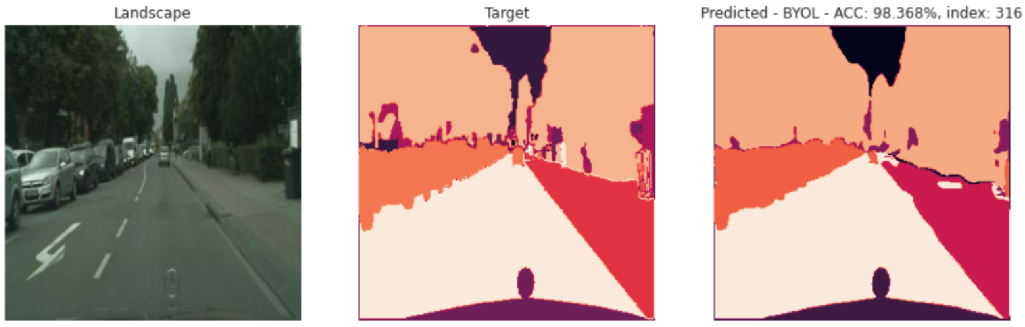
epoche aj testujem a sledujem ako sa vyvíja chyba a presnosť modelu. Informácie o tréningu po každej epoche vypisujem do konzoly a po dotrénovaní dáta o chybe a presnosti vykreslím do grafu a uložím do zoznamu a csv súboru. Takýto proces trénovania uskutočním 5 krát a to pre nepredtrénovanú sieť Resnet50 a následne pre 4 modely s predtrénovanými sieťami z SSL trénovania. Výsledkom tréningu je 5 modelov. Tieto modely ukladám pre neskoršie použitie pri testovaní.

Vyhodnotenie úspešnosti jednotlivých modelov vykonávam vo FOR cykle s cieľom iterovať cez všetky modely a každý z modelov testujem na rovnakom testovacom datasete. Každý model je vždy vstupným parametrom do funkcie *semantic_segmentation()*, v ktorej model na základe vstupného testovacieho obrázku vygeneruje nový obrázok s označením tried (Y_pred) následne porovnávam so skutočnou pravdou, ktorou je obrázok vygenerovaný KMeans metódou. Úspešnosť predikcie je počítaná vo funkcii *compute_all_acc()*, v ktorej počítam špecificitu, senzitivitu, presnosť, chybovú maticu a najdôležitejší prvok vyhodnocovania presnosti sémantickej segmentácie aj IoU. Úspešnosť jednotlivých modelov som sa rozhodol určiť dvomi spôsobmi a to veľkosťou presnosti predikovaných pixelov na základe dát chybovej matice, ktorá sa počíta podľa vzorcu 4.5 a tiež prieniku nad spojením (IoU), ktoré sa počíta podľa vzorcu 4.12.

Metódy navzájom porovnávam podľa počtu obrázkov, ktorých presnosť je viac ako 98% alebo IoU minimálne 0.5 (50%), čo je všeobecne používaná hodnota určujúca kvalitu predikcie pri detekcii alebo sémantickej segmentácii. Výsledok testovania modelov je v tabuľke Tab. 8.1. Hodnoty v stĺpcoch reprezentované "MAX, AVG, MIN" predstavujú maximálnu, priemernú a minimálnu presnosť alebo IoU. Posledné stĺpce v oboch kategóriách značia počet obrázkov väčší ako 98% pri presnosti alebo viac ako 0,5 (50%) pri IoU. SSL modely boli trénované na 50 epochách, s veľkosťou dávky 300 a obrázky som zmenšil na rozmer 50x50 pixelov, aby som mohol trénovať s väčšou dávkou. Trénovanie sémantickej segmentácie bolo uskutočnené pri 50 epochách a veľkosti dávky 8 aby bola veľkosť obrázkov (256x256 pixelov) zachovaná.

	Presnosť				IoU			
Model	MAX [%]	AVG [%]	MIN [%]	>98%	MAX [%]	AVG [%]	MIN [%]	>0,5
Resnet50	98,13	93,93	81,80	9	73,13	46,68	15,11	198
Rotácia	98,20	93,98	81,92	9	73,51	46,28	15,26	206
SimCLR	98,24	94,28	82,05	11	74,34	47,73	16,17	212
MoCov2	98,33	94,37	82,10	12	74,58	48,35	17,44	211
BYOL	98,37	94,42	82,08	12	74,91	49,04	17,60	215

Tab. 8.1: Tabuľka vyhodnotenia presnosti a IoU s maximálnymi, priemernými a minimálnymi hodnotami



Obr. 8.3: Najlepšie predikovaná maska sémantickej segmentácie podľa presnosti pixelov metódou BYOL

Obr. 8.4: Najlepšie predikovaná maska sémantickej segmentácie podľa IoU metódou BYOL

8.2 Vyhodnotenie

Metódy SSL je možné úspešne použiť aj pri úlohe sémantickej segmentácie. Ich výsledok v tomto pokuse, ale neznamenal veľké zvýšenie presnosti. Podľa výsledkov v tabuľke Tab. 8.1 môžem tvrdiť, že predtrénovanie SSL metódou dokáže mierne zvýšiť presnosť. Napríklad počet obrázkov s celkovou presnosťou väčšou, ako 98% bolo pri nepredtrénovanej Resnet50 9 a pri predtrénovanej pomocou metódy BYOL 12. Mierne zvýšenie pri rovnakej metóde nastalo aj pri obrázkoch, ktorých IoU je viac ako 0,5 a to z pôvodných 198 obrázkov na 215. Metóda BYOL bola aj najúspešnejšou pri úlohe sémantickej segmentácie. Na obrázkoch Obr. 8.3 a 8.4 je zobrazený príklad obrázku z datasetu, označeného cieľového obrázku a generovaného pomocou sieťe enkóder+dekóder, v ktorej je enkóder predtrénovaný metódou BYOL. V prvom prípade ide o najlepšie predikovanú masku na základe presnosti (98,368%) a v druhom prípade o najlepšie predikovanú masku na základe IoU (74,907%). Ďalšie príklady najlepšie a najhoršie predikovaných masiek (vyhodnotených podľa presnosti a IoU) všetkými SSL metódami, sú zobrazené v prílohe C.

Záver

Táto diplomová práca sa venuje problematike Self-supervised učenia v aplikácii počítačového videnia. Práca je rozdelená na dve hlavné časti (teoretickú a praktickú).

V prvej kapitole som popísal všeobecné použitie umelej inteligencie v počítačovom videní a vysvetlil základné pojmy rozdelenia dát podľa učenia s učiteľom, bez učiteľa alebo s vlastným učiteľom (self-supervised). Následne som porovnal spôsoby trénovania medzi sebou a podrobne popísal metódy v počítačovom videní ako klasifikácia a detekcia obrazu, sledovanie objektov alebo aplikovanie sémantickej segmentácie. Taktiež som uviedol príklad reziduálnych spojení v konvolučných sieťach, ktorých princíp som použil na demonštráciu self-supervised metód v praktickej časti. Bližšie som popísal spôsoby vyhodnocovania úspešnosti modelov cez presnosť, ROC a PR krivky, F1 skóre a IoU.

V hlavnej sekcii teoretickej často diplomovej práce som podrobne rozobral, ako self-supervised trénovanie funguje, čo je to pretext a downstream úloha, rozdelenie metód na generatívne a kontrastívne. Vysvetlil som ich detaily a rozdiely a z akých metód sa skladá generatívny oblasť - rekonštrukcia obrazu (image colorization, super-resolution, inpaiting alebo cross-channel prediction), common sense reasoning (jigsaw, context prediction, geometric transformation recognition), clustering (deep cluster alebo syntetické obrázky) a z akých metód sa skladá presnejšia, ale zložitejšia kontrastívna oblasť pozostávajúca z metód: SimCLR, MoCov2 alebo BYOL, ktoré sa dokážu svojou kvalitou rovnať metódam trénovania s učiteľom. Ku koncu kapitoly som popísal aj využitie týchto metód v projektoch v praxi napríklad pri predpovedaní hĺbky z 2D obrazu.

V praktickej časti diplomovej práce definujem vhodnú testovaciu úlohu a vyberám 4 self-supervised metódy, ktoré budem aplikovať. Konkrétne ide o metódy Rotácie, SimCLR, MoCov2 a BYOL. Pokračujem v popise použitých datasetov na tréning a test. Hlavným datasetom, ktorý slúži ako ideálny pre testovanie efektívnosti a presnosti self-supervised metód, je dataset STL10, pozostávajúci zo 100 000 neoznačených obrázkov, 5 000 trénovacích a 8 000 testovacích s označením. Každú zo 4 spomenutých metód testujem na spoločných testoch TEST1, TEST2, TEST3 a TEST4 na tomto datasete za použitia rôznych vstupných parametrov, ktorými sú veľkosť trénovacej dávky (50, 100, 300, 500), počet trénovacích epoch (50, 100, 300, 500), rôzna veľkosť a počet lineárnych vrstiev (1, 2, 3) pri dotrénovaní modelov a tiež testovanie trénovania predtrénovaných modelov spôsobom tréningu s učiteľom. Ako vhodnú CNN som vybral Resnet z dôvodu vlastnosti reziduálnych spojení.

Metóda Rotácie je najjednoduchšou aplikovanou SSL metódou z pohľadu tvorby kódu. Jej úlohou je orotovať vstupný obrázok s cieľom predikcie uhlu rotácie SSL modelu. Testovanie tejto metódy s rôznou veľkosťou dávky alebo počtu epoch značilo

iba mierne zlepšenie presnosti výsledného modelu. Pri experimente so zvyšovaním veľkosti dávky dosiahla najväčšiu presnosť 68,3% s dotrénovaním 10% označených dát z datasetu STL10, pri 50 epochách a dávke 200. Najväčší prínos metódy je v stave, ak máme malý počet označených dát (napríklad 1%) a zvyšok neoznačených. Predtrénovanie modelu touto metódou umožní dosiahnuť až o 18% väčšiu presnosť, ako vôbec nepredtrénovaný model (Tab. 7.3). Najväčšia presnosť bola dosiahnutá na datasete Industry (75,2% pri dotrénovaní 1% dát a 86,1% pri dotrénovaní 10% dát).

Kontrastívna metóda SimCLR urobí zo vstupného obrázku 2 augmentované obrázky (pozitívny pár). Cieľom je takýto pozitívny pár priblížiť, čo najviac k sebe a ostatné negatívne páry z dávky odsunúť ďalej. Tento krok sa deje pomocou funkcie chyby NT-Xent. Architektúra CNN sa skladá z Resnet18 a MLPHead. Model predtrénovaný metódou SimCLR a dotrénovaný 1 lineárnou vrstvou prekonáva model trénovaný s učiteľom už pri 1% označených dát na datasete STL10. Najvyššiu presnosť pri dotrénovaní s rôznou veľkosťou dát model dosiahol na datasete MNIST a najmenšiu na datasetoch Imagenete a Weather. Výsledky pokusu v Tab. 7.7 potvrdzujú, že použitie náhodného vyrezania pomôže pri kontrastívnom učení viac, ako náhodná zmena farebného spektra, ale ich kombinácia umožní dosiahnuť skoro dvojnásobnú presnosť oproti ich nepoužitiu.

Metóda MoCov2 je vylepšenou verziou pôvodnej metódy MoCo vďaka použitiu MLPHead (rovnako ako pri SimCLR) a aplikovaniu Gaussovho rozostrenia pri augmentovaní vstupného obrázku. Výstup augmentácie je pár obrázkov (žiadosť a kľúč). Obrázok reprezentovaný kľúčom sa pred použitím metódy ukladá do dynamického slovníku (fronty - FIFO), ktorý bude pri tréningu použitý na hľadanie správneho kľúču pre danú žiadosť. Architektúra siete sa skladá z dvoch CNN typu Resnet pre žiadosti a kľúče. Metóda MoCov2 viac profituje z väčšej dávky alebo v porovnaní s rovnako veľkou dávkou, ako iné metódy, dosahuje mierne vyššiu presnosť (Obr. 7.19). Pri datasete MNIST prekonáva ostatné metódy s presnosťou 94,6% a datasete STL10 prekonáva ostatné metódy, ak sa dotrénuje s 10% dát (79,3%). Pri datasete Imagenet, MoCov2 dosahuje pri 1% dát presnosť 51,3% čo je najviac v porovnaní s inými SSL metódami. Zväčšovanie veľkosti dynamického slovníka pomáha pri dosahovaní vyššej presnosti (Tab. 7.11).

Poslednou použitou SSL metódou je BYOL. Architektúra v použití tejto siete obsahuje 3 CNN a to Online, Target a Prediktor. Cieľom metódy je aplikovať augmentácie na vstupný obrázok, vytvoriť augmentovaný pár, poslať jeden obrázok do Online siete s Prediktorom a predpovedať výstupný vektor vlastností, ktorý bude generovať Target sieť. Metóda BYOL je najpresnejšia a dosahuje najlepšie výsledky v porovnaní s ostatnými metódami. Najviac prosperuje pri raste počtu epoch. Svoje maximum dosiahla s datasetom MNIST (93,1%) a pri trénovaní na datasete STL10

pri použití 1% dotrénovaných dát prekonáva model trénovaný s učiteľom (63,0%) s presnosťou 71,1% pri 500 epochách a 200 dávke. Pri metóde BYOL je dôležité aplikovať dávkovú normalizáciu v MLPHead, lebo bez jej aplikovania sa môže model dostať do kolapsu a zle predikovať výsledný vektor vlastností. Zvýšením počtu neurónov v MLPHead dochádza k mierne vyššej presnosti modelu. Neuróny zabezpečia kvalitnejšie naučenie vlastností vstupných dát, z ktorých model profituje. Najlepšie výsledky boli zaznamené pri momente (m) s hodnotou 0.99 (Tab. 7.15).

Všetky metódy je vhodné aplikovať pri predtrénovaní akéhokoľvek modelu. Môže to iba pomôcť a pri datasete, v ktorom je málo označených dát (napríklad iba 1%), môže byť rozdiel predtrénovania so zvyškom dát bez označenia dôležitý a tento rozdiel je zaznamenaný na obrázku Obr. 7.22 pre všetky SSL metódy na datasete STL10. Predtrénovaný model následne trénovaný spôsobom učenia s učiteľom (TEST3, Obr. 7.21) od začiatku tréningu až po koniec (pre niektoré metódy) dosahuje vyššiu presnosť. Aj toto potvrdzuje dôležitosť predtrénovať model na neoznačených dátach. Ako obmedzujúce podmienky pri SSL metódach sú vlastnosti datasetu, medzi ktoré patrí počet obrázkov na predtrénovanie, rôznorodosť obrázkov s obtiažnosťou a počet tried. Naopak SSL metódy zvyšujú presnosť dotrénovaním na viac lineárnych vrstvách a kontrastívne metódy aj zvyšovaním počtu epoch alebo veľkosti dávky. Obmedzujúcou podmienkou pri kontrastívnych metódach môže byť typ a sila aplikovaných augmentácií, kde bez použitia augmentácií náhodného vyrezania a zmeny sfarbenia sa model nemusí naučiť dostatok potrebných vlastností na následné dotrénovanie. Poslednou obmedzujúcou podmienkou, na ktorú bol uskutočnený experiment je aj výber počtu neurónov do MLPHead.

Praktickú časť diplomovej práce uzatváram aplikáciou úlohy Sémantickej Segmentácie za použitia SSL metód pri predtrénovaní enkóderu. Model trénujem na datasete Cityscapes a ako hlavnú CNN v enkódery používam Resnet50. Predtrénovanie aj v tomto prípade pomáha zvýšiť presnosť vyhodnotenú pomocou celkovej presnosti a IoU v tabuľke Tab 8.1. Model predtrénovaný metódou BYOL dosiahol najväčšiu presnosť (98,37%) a IoU (74,91%) s počtom 12 predikovaných masiek s presnosťou viac ako 98% a 215 s IoU viac ako 0,5. Príklady najlepších a najhorších predikcií (podľa presnosti a IoU) masiek všetkými SSL metódami sú v prílohe C.

Môžem zhodnotiť, že zadanie diplomovej práce bolo úspešne splnené vo všetkých bodoch.

Literatúra

[1] Aidan Wilson. *A Brief Introduction to Supervised Learning* [online]. 2019 – [cit. 8. 11. 2020]. Dostupné z URL: <https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590>.

[2] Sanatan Mishra. *Unsupervised Learning and Data Clustering* [online]. 2017 – [cit. 8. 11. 2020]. Dostupné z URL: <https://towardsdatascience.com/unsupervised-learning-and-data-clustering-eeecb78b422a>.

[3] Prateek Bajaj *Reinforcement learning* [online]. 2020 – [cit. 8. 11. 2020]. Dostupné z URL: <https://www.geeksforgeeks.org/what-is-reinforcement-learning/>.

[4] Anoop Sarkar and Gholamreza Haffar. *Inductive Semi-supervised Learning with Applicability to NLP* [online]. [cit. 10. 11. 2020]. Dostupné z URL: <https://www.cs.sfu.ca/~anoop/papers/pdf/semisup_naacl.pdf>.

[5] Amit Chaudhary. *The Illustrated FixMatch for Semi-Supervised Learning* [online]. 2019 – [cit. 10. 11. 2020]. Dostupné z URL: <https://amitness.com/2020/03/fixmatch-semi-supervised/>.

[6] Alfonso Ibañez. *Semi-Supervised Learning... the great unknown* [online]. 2019 – [cit. 10. 11. 2020]. Dostupné z URL: <https://business.blogthinkbig.com/semi-supervised-learning-the-great-unknown/>.

[7] Dickey Singh. *Self-supervised learning gets us closer to autonomous learning* [online]. 2018 – [cit. 10. 11. 2020]. Dostupné z URL: <https://cutt.ly/mh6yqwA>.

[8] Michal Maj. *Object Detection and Image Classification with YOLO* [online]. 2018 – [cit. 10. 11. 2020]. Dostupné z URL: <https://www.kdnuggets.com/2018/09/object-detection-image-classification-yolo.html>.

[9] Axel Angel. *Towards Distortion-Predictable Embedding of Neural Networks* [online]. 2015 – [cit. 10. 11. 2020]. Dostupné z URL: <https://deepai.org/publication/towards-distortion-predictable-embedding-of-neural-networks>.

[10] Kaiming He, Georgia Gkioxari, Piotr Dollar, Ross Girshick *Mask R-CNN* [online]. 2018 – [cit. 10. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1703.06870.pdf>.

[11] BitRefine Heads. *AI object tracking* [online]. 2020 – [cit. 10. 11. 2020]. Dostupné z URL: <https://heads.bitrefine.group/use-cases/object-recognition/116-object-recognition/297-ai-object-tracking>.

[12] Dhanoop Karunakaran. *Semantic segmentation — Udaity's self-driving car engineer nanodegree* [online]. 2018 – [cit. 10. 11. 2020]. Dostupné z URL: <https://cutt.ly/Kh6ytoz>.

[13] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, Senior Member. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation* [online]. 2016 – [cit. 10. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1511.00561.pdf>.

[14] James Le. *The 5 Computer Vision Techniques That Will Change How You See The World* [online]. 2018 – [cit. 10. 11. 2020]. Dostupné z URL: <https://cutt.ly/bh6yuCJ>.

[15] Gualtiero Piccinini. *The first computational theory of mind and brain* [online]. 2004 – [cit. 15. 3. 2021]. Dostupné z URL: <https://www.umsl.edu/~piccininig/First_Computational_Theory_of_Mind_and_Brain.pdf>.

[16] David Fumo. *A Gentle Introduction To Neural Networks Series — Part 1* [online]. 2017 – [cit. 15. 3. 2021]. Dostupné z URL: <https://towardsdatascience.com/a-gentle-introduction-to-neural-networks-series-part-1-2b90b87795bc>.

[17] Purnasai Gudikandula. *A Beginner Intro to Neural Networks* [online]. 2019 – [cit. 15. 3. 2021]. Dostupné z URL: <https://purnasaigudikandula.medium.com/a-beginner-intro-to-neural-networks-543267bda3c8>.

[18] Dynamic. *Is there any difference between an activation function and a transfer function?* [online]. 2014 – [cit. 15. 11. 2020]. Dostupné z URL: <https://cutt.ly/0h6yolX>.

[19] R Views. *Building A Neural Net from Scratch Using R – Part 1* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://www.r-bloggers.com/2020/07/building-a-neural-net-from-scratch-using-r-part-1/>.

[20] MissingLink.ai. *7 Types of Neural Network Activation Functions: How to Choose?* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>.

[21] ML Glossary. *Loss Functions* [online]. 2017 – [cit. 15. 3. 2021]. Dostupné z URL: <https://cutt.ly/XcFwpvR>.

[22] Sebastian Raschka. *Loss Functions* [online]. 2014 – [cit. 15. 3. 2021]. Dostupné z URL: <http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/>.

[23] Rachel Draelos. *The History of Convolutional Neural Networks* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/>.

[24] Juraj Muráň. *Úvod do konvolučných neurónových sietí* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://umelainteligencia.sk/uvod-do-konvolucnych-neuronovych-sieti/>.

[25] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <https://cutt.ly/MjtML8X>.

[26] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://d2l.ai/d2l-en.pdf>.

[27] Ayeshmantha Perera. *What is Padding in Convolutional Neural Network's(CNN's) padding* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <https://ayeshmanthaperera.medium.com/what-is-padding-in-cnns-71b21fb0dd7>.

[28] Jason Brownlee. *A Gentle Introduction to Padding and Stride for Convolutional Neural Networks* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/>.

[29] Jason Brownlee. *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>.

[30] SAGAR SHARMA. *Activation Functions in Neural Networks* [online]. 2017 – [cit. 15. 11. 2020]. Dostupné z URL: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.

[31] Jason Brownlee. *A Gentle Introduction to Batch Normalization for Deep Neural Networks* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/>.

[32] Sergey Ioffe, Christian Szegedy. *Batch normalization in Neural Networks* [online]. 2015 – [cit. 15. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1502.03167v3.pdf>.

[33] Jason Brownlee. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/>.

[34] Jason Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://cutt.ly/djtMCvd>.

[35] Richmond Alake. *Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://cutt.ly/Ih6yaTv>.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. *Deep Residual Learning for Image Recognition* [online]. 2015 – [cit. 15. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1512.03385.pdf>.

[37] Pablo Ruiz. *Understanding and visualizing ResNets* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8>.

[38] Pablo Ruiz. *ResNets for CIFAR-10* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <https://towardsdatascience.com/resnets-for-cifar-10-e63e900524e0>.

[39] Olaf Ronneberger. *U-Net: Convolutional Networks for Biomedical Image Segmentation* [online]. 2015 – [cit. 6. 4. 2021]. Dostupné z URL: <https://arxiv.org/pdf/1505.04597.pdf>.

[40] Soner Yildirim. *How to Best Evaluate a Classification Model* [online]. 2020 – [cit. 6. 4. 2021]. Dostupné z URL: <https://towardsdatascience.com/how-to-best-evaluate-a-classification-model-2edb12bcc587#:~:text=Classification%20is%20a%20supervised%20learning,to%20create%20a%20robust%20model.>.

[41] Lékařská fakulta Univerzity Karlovy. *ROC křivka* [online]. 2020 – [cit. 6. 4. 2021]. Dostupné z URL: <https://www.wikiskripta.eu/w/ROC_k%C5%99ivka>.

[42] Jeremy Jordan. *Evaluating image segmentation models.* [online]. 2018 – [cit. 17. 4. 2021]. Dostupné z URL: <https://www.jeremyjordan.me/evaluating-image-segmentation-models/>.

[43] Chien Vu. *Train without labeling data using Self-Supervised Learning by Relational Reasoning* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://cutt.ly/Th6yfJt>.

[44] Lilian Weng. *Self-Supervised Representation Learning* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <https://lilianweng.github.io/lil-log/2019/11/10/self-supervised-learning.html>.

[45] Andrei Bursuc, Relja Arandjelović. *Towards Annotation-Efficient Learning Self-Supervised Learning* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://abursuc.github.io/slides/2020_tutorial_cvpr/self_supervised_learning.html#18>.

[46] Shengchao Liu. *Contrastive Learning on Graphs and Some Interpretations* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://chao1224.github.io/material/slides/20200728.pdf>.

[47] Amit Chaudhary. *The Illustrated Self-Supervised Learning* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <https://amitness.com/2020/02/illustrated-self-supervised-learning>.

[48] Richard Zhang, Phillip Isola, Alexei A. Efros. *Colorful Image Colorization* [online]. 2016 – [cit. 15. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1603.08511.pdf>.

[49] Satoshi Iizuka, Edgar Simo-Serra, Hiroshi Ishikawa. *Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification* [online]. 2016 – [cit. 15. 11. 2020]. Dostupné z URL: <http://iizuka.cs.tsukuba.ac.jp/projects/colorization/en/>.

[50] Christian Ledig, Lucas Theis, Ferenc Husza r, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, Wenzhe Shi. *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network* [online]. 2017 – [cit. 15. 11. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1609.04802.pdf>.

[51] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, Alexei A. Efros, University of California, Berkeley. *Context Encoders: Feature Learning by Inpainting* [online]. 2016 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1604.07379.pdf`>.

[52] Richard Zhang, Phillip Isola, Alexei A. Efros. *Split-Brain Autoencoders: Unsupervised Learning by Cross-Channel Prediction* [online]. 2017 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1611.09842.pdf`>.

[53] Mehdi Noroozi, Paolo Favaro. *Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles* [online]. 2017 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1603.09246.pdf`>.

[54] Carl Doersch, Abhinav Gupta, Alexei A. Efros. *Unsupervised Visual Representation Learning by Context Prediction* [online]. 2016 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1505.05192.pdf`>.

[55] Spyros Gidaris, Praveer Singh, Nikos Komodakis. *UNSUPERVISED REPRESENTATION LEARNING BY PRE- DICTING IMAGE ROTATIONS* [online]. 2018 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1803.07728.pdf`>.

[56] Amit Chaudhary. *A Visual Exploration of DeepCluster* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://amitness.com/2020/04/deepcluster/`>.

[57] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. *Deep Clustering for Unsupervised Learning of Visual Features* [online]. 2019 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1807.05520.pdf`>.

[58] Zhongzheng Ren and Yong Jae Lee. *Cross-Domain Self-supervised Multi-task Feature Learning using Synthetic Imagery* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/1711.09082.pdf`>.

[59] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, Dilip Krishnan. *Supervised Contrastive Learning* [online]. 2020 – [cit. 15. 11. 2020]. Dostupné z URL: <`https://arxiv.org/pdf/2004.11362.pdf`>.

[60] Ting Chen, Simon Kornblith, Mohammad Norouzi, Geoffrey Hinton. *A Simple Framework for Contrastive Learning of Visual Representations* [online]. 2020 –

[cit. 10. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/2002.05709.pdf>.

[61] Ganesh Kalyansundaram. *Self-supervised Representation Learning in Computer Vision — Part 2* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://cutt.ly/EvTplx8>.

[62] Thalles Silva. *Exploring SimCLR: A Simple Framework for Contrastive Learning of Visual Representations* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://cutt.ly/oh6yhLS>.

[63] Ting Chen, Geoffrey Hinton. *Advancing Self-Supervised and Semi-Supervised Learning with SimCLR* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html>.

[64] Amit Chaudhary. *The Illustrated PIRL: Pretext-Invariant Representation Learning* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://amitness.com/2020/03/illustrated-pirl/>.

[65] Ishan Misra, Laurens van der Maaten. *Self-Supervised Learning of Pretext-Invariant Representations* [online]. 2019 – [cit. 10. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1912.01991.pdf>.

[66] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, Ross Girshick. *Momentum Contrast for Unsupervised Visual Representation Learning* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1911.05722.pdf>.

[67] Xinlei Chen, Haoqi Fan, Ross Girshick, Kaiming He, Facebook AI Research (FAIR). *Improved Baselines with Momentum Contrastive Learning* [online]. 2020 – [cit. 11. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/2003.04297v1.pdf>.

[68] Aditya Rastogi. *How to Reduce Computational Constraints using Momentum Contrast V2(Moco-v2) in PyTorch* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://www.analyticsvidhya.com/blog/2020/08/moco-v2-in-pytorch/>.

[69] Ann Arbor. *Momentum Contrast for Unsupervised Visual Representation Learning.* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://hammer-wang.github.io/5cents/representation-learning/moco/>.

[70] Nilesh Vijayrania. *Self-Supervised Learning Methods for Computer Vision.* [online]. 2020 – [cit. 10. 12. 2020]. Dostupné z URL: <https://towardsdatascience.com/self-supervised-learning-methods-for-computer-vision-c25ec10a91bd>.

[71] Jean-Bastien Grill, Florian Strub, Florent Altche, DeepMind, Imperial Collage. *Bootstrap Your Own Latent A New Approach to Self-Supervised Learning* [online]. 2020 – [cit. 15. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/2006.07733.pdf>.

[72] Frank Odom. *Easy Self-Supervised Learning with BYOL* [online]. 2020 – [cit. 15. 12. 2020]. Dostupné z URL: <https://medium.com/the-dl/easy-self-supervised-learning-with-byol-53b8ad8185d>.

[73] NeuroHive. *Depth Estimation Using Encoder-Decoder Networks and Self-Supervised Learning* [online]. 2018 – [cit. 15. 12. 2020]. Dostupné z URL: <https://cutt.ly/0h6ykYU>.

[74] M.Ye , E.Johns , A.Handa , L.Zhang , P.Pratt , G.-Z.Yang. *Self-Supervised Siamese Learning on Stereo Image Pairs for Depth Estimation in Robotic Surgery* [online]. 2017 – [cit. 20. 12. 2020]. Dostupné z URL: <https://arxiv.org/pdf/1705.08260.pdf>.

[75] Sermanet, Pierre and Lynch, Corey and Chebotar, Yevgen and Hsu, Jasmine and Jang, Eric and Schaal, Stefan and Levine, Sergey. *Time-Contrastive Networks: Self-Supervised Learning from Video* [online]. 2017 – [cit. 20. 12. 2020]. Dostupné z URL: <https://sermanet.github.io/imitate/>.

[76] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton. *The CIFAR-10 dataset* [online]. 2009 – [cit. 21. 12. 2020]. Dostupné z URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.

[77] Adam Coates, Honglak Lee, Andrew Y. Ng. *STL-10 dataset* [online]. 2011 – [cit. 20. 3. 2021]. Dostupné z URL: <https://cs.stanford.edu/~acoates/stl10/>.

[78] LeCun, Yann. *MNIST Database Examples - Size Normalized* [online]. 2019 – [cit. 21. 3. 2021]. Dostupné z URL: <https://anatomiesofintelligence.github.io/posts/2019-07-25-mnist-dataset-sample>.

[79] Vijay Gupta. *Weather dataset* [online]. 2020 – [cit. 21. 3. 2021]. Dostupné z URL: <https://www.kaggle.com/vijaygiitk/multiclass-weather-dataset>.

[80] Ravirajsinh Dabhi. *casting product image data for quality inspection* [online]. 2020 – [cit. 22. 3. 2021]. Dostupné z URL: <https://www.kaggle.com/ravirajsinh45/real-life-industrial-dataset-of-casting-product>.

[81] DanB. *STL-10 dataset* [online]. 2018 – [cit. 20. 3. 2021]. Dostupné z URL: <https://www.kaggle.com/dansbecker/cityscapes-image-pairs>.

[82] Google. *Welcome To Colaboratory* [online]. 2020 – [cit. 24. 12. 2020]. Dostupné z URL: <https://colab.research.google.com/notebooks/intro.ipynb#recent=true>.

[83] pytorch. *FROM RESEARCH TO PRODUCTION* [online]. 2020 – [cit. 15. 4. 2021]. Dostupné z URL: <https://pytorch.org/>.

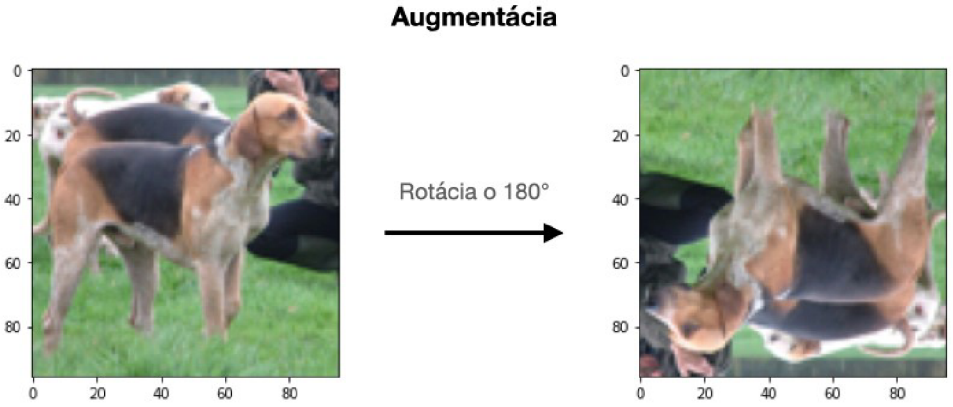
[84] Kevin Chyunlu. *pytorch-unet-resnet-50-encoder* [online]. 2018 – [cit. 15. 4. 2021]. Dostupné z URL: <https://github.com/kevinlu1211/pytorch-unet-resnet-50-encoder/blob/master/u_net_resnet_50_encoder.py>.

Zoznam symbolov a skratiek

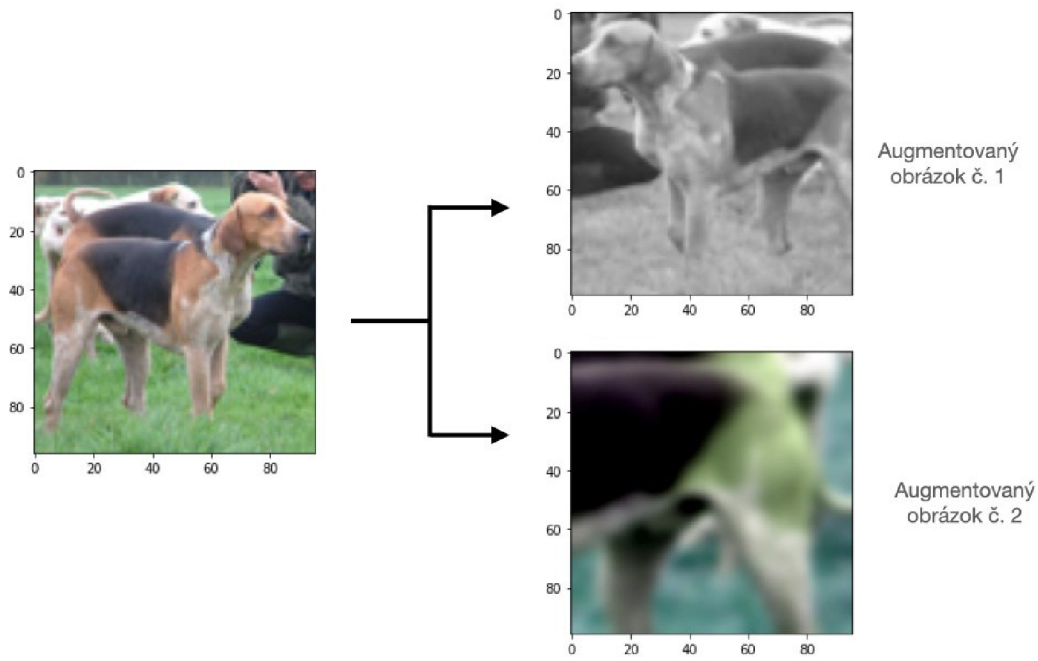
BN	Dávková normalizácia - Batch normalization
BYOL	Bootstrap Your Own Latent
CFN	Bezkontextová sieť – context-free network
CNN	Konvolučné neurónové siete – Convolution neural networks
CPU	Procesor - central processing unit
FCN	Plná konvolučná sieť – Fully Convolutional Networks
GPU	Grafická karta - graphics processing unit
InfoNCE	Information Noise Contrastive Estimation
MLPHead	Multilayer projection head
MoCo	Momentum Contrastive Learning
MNIST	The Modified National Institute of Standards and Technology
NN	Neurónové siete – Neural Networks
NT-Xent	Normalized temperature-scaled cross-entropy loss
PCA	Analýza hlavných komponentov – Principal component analysis
R-CNN	Regionálna konvolučná neurónová sieť – region convolution neural network
R-FCN	Region-Based Fully Convolutional Networks
RPN	Region Proposal Network
SAE	Stacked auto encoders
SimCLR	A Simple Framework for Contrastive Learning of Visual Representations
SSL	Učenie s vlastným učitelom – Self-supervised Learning
SmSL	Učenie s čiastočným učitelom – Semi-supervised Learning
SVM	Metóda podporných vektorov - support vector machine
SSD	Single Shot MultiBox Detector

Zoznam príloh

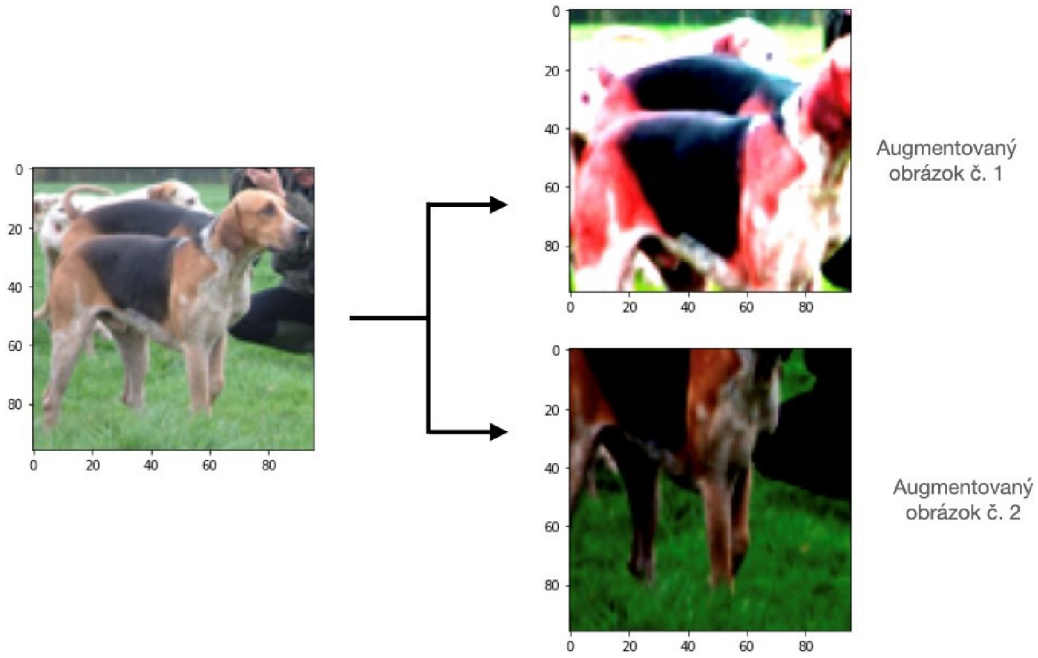
A Aplikovanie augmentácií pri SSL tréningu na vstupný obrázok



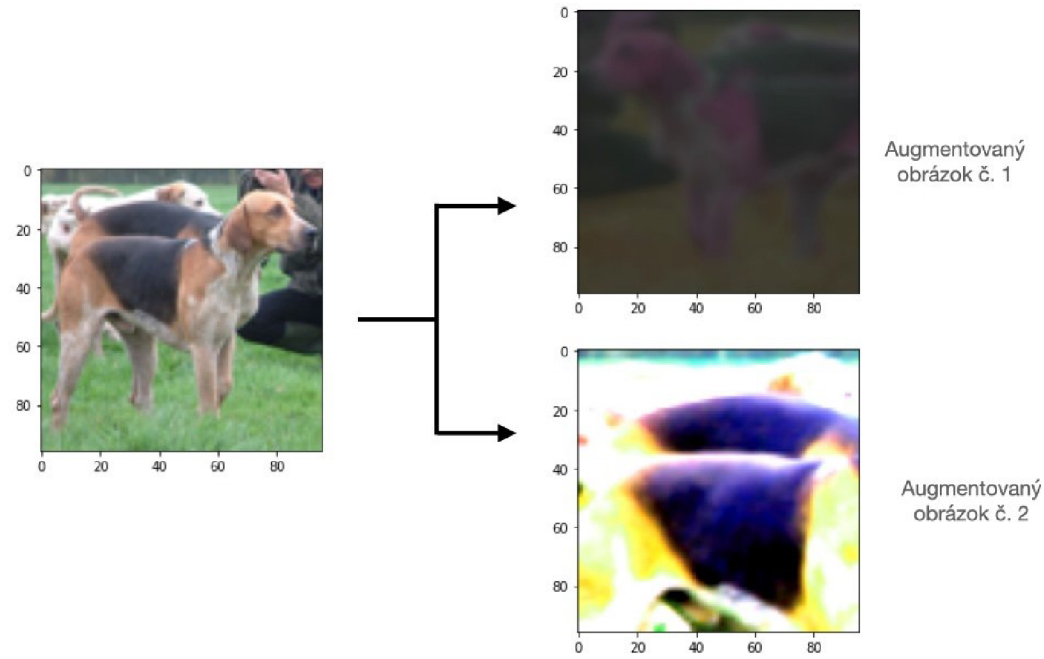
Obr. A.1: Príklad aplikácie augmentácie rotácie na vstupný obrázok



Obr. A.2: Príklad aplikácie augmentácií metódy SimCLR na vstupný obrázok

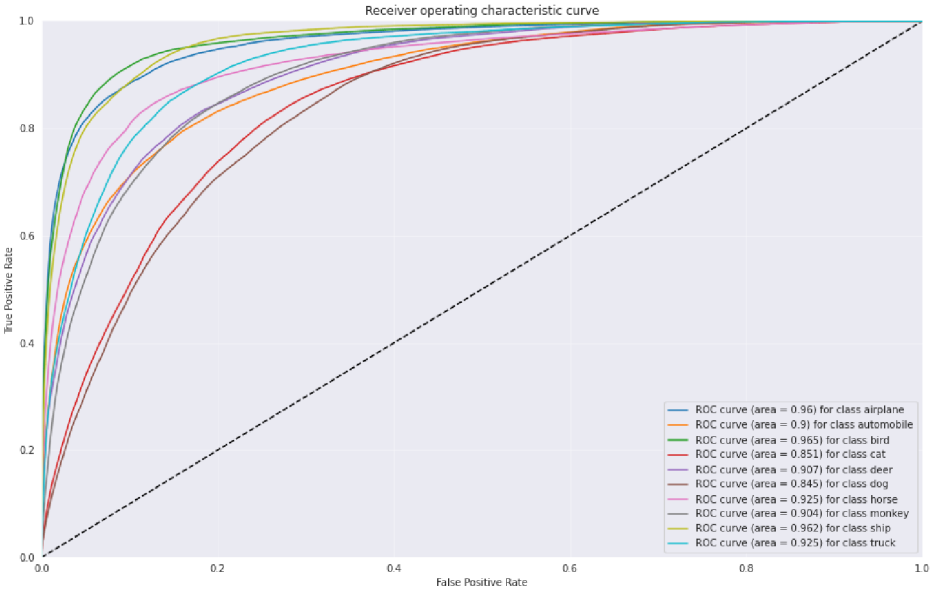


Obr. A.3: Príklad aplikácie augmentácií metódy MoCov2 na vstupný obrázok

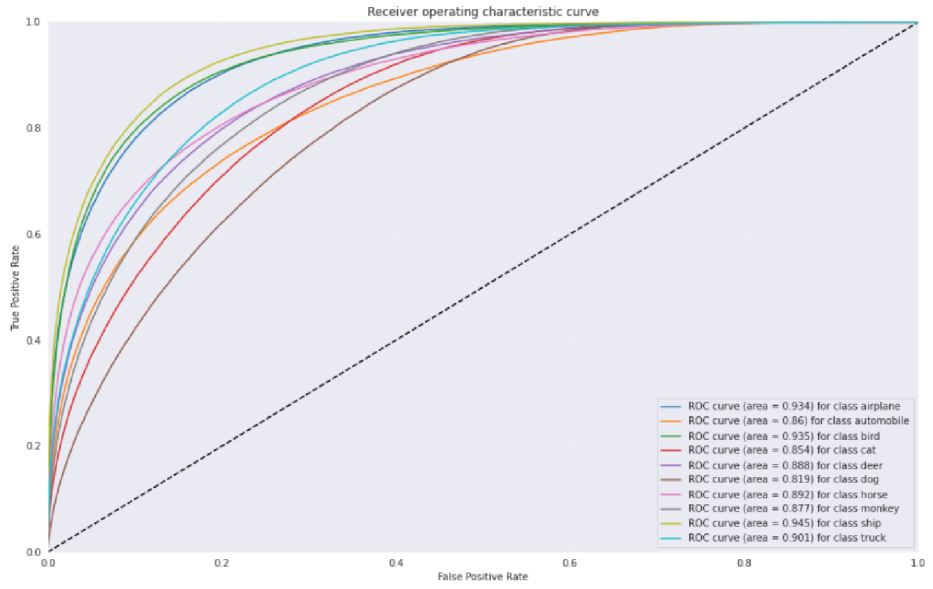


Obr. A.4: Príklad aplikácie augmentácií metódy BYOL na vstupný obrázok

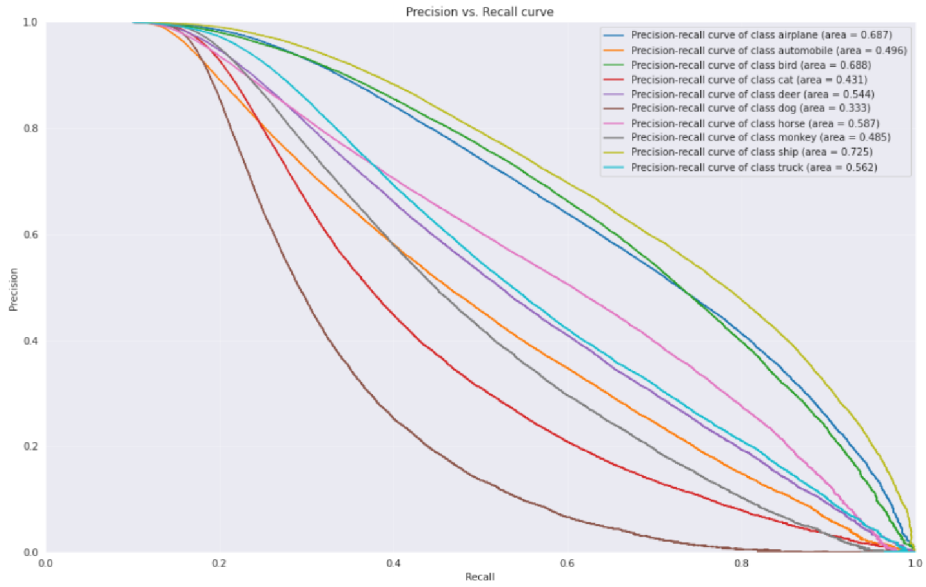
B ROC a PR krivky



Obr. B.1: Príklad ROC krivky pre model Rotácie trénovaný na datasete STL10

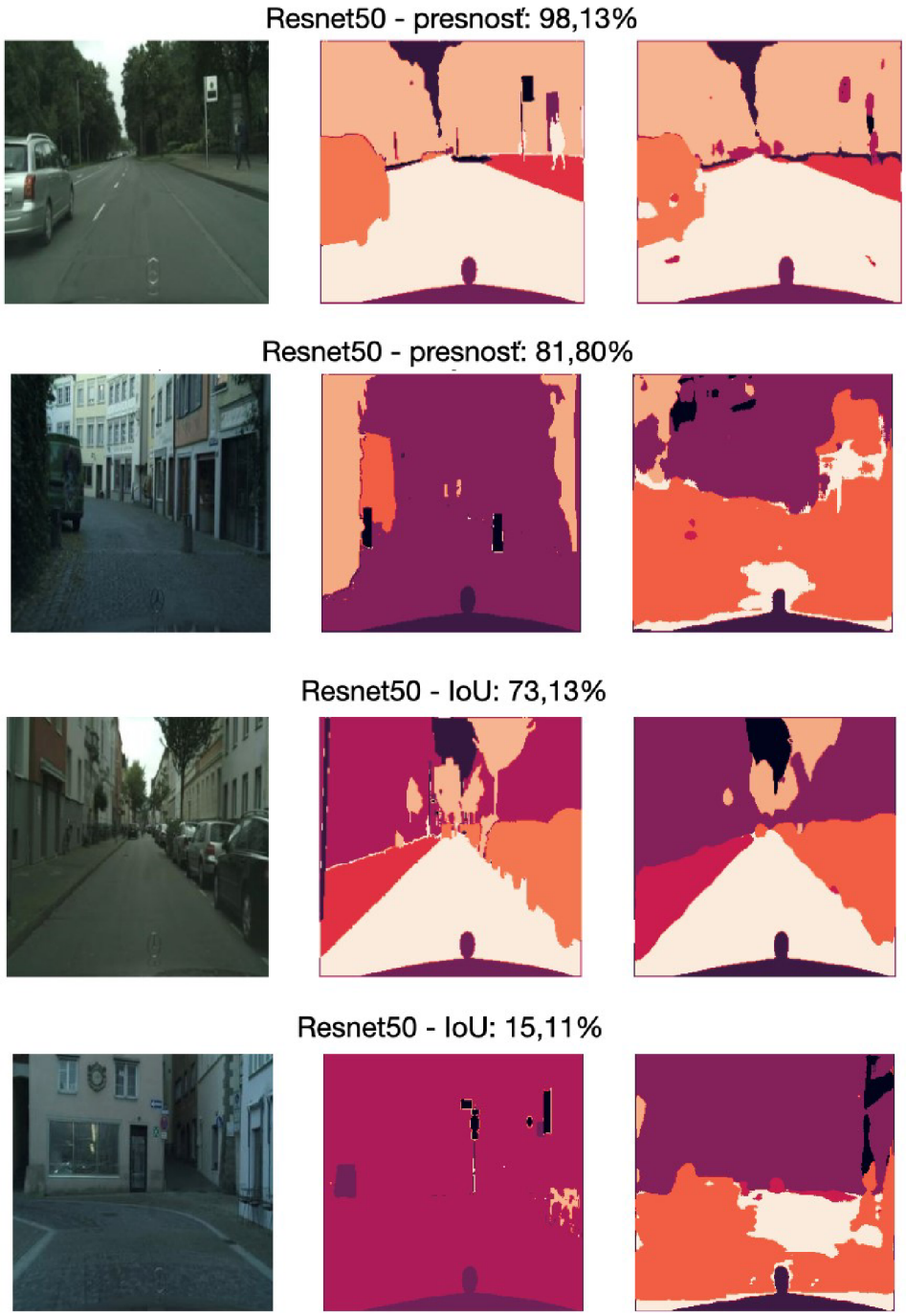


Obr. B.2: ROC krivka s veľkosťou AUC pre jednotlivé triedy datasetu STL10 modelu trénovaného s učiteľom



Obr. B.3: PR krivka s veľkosťou plochy pod krivkou pre jednotlivé triedy datasetu STL10 modelu trénovaného s učiteľom

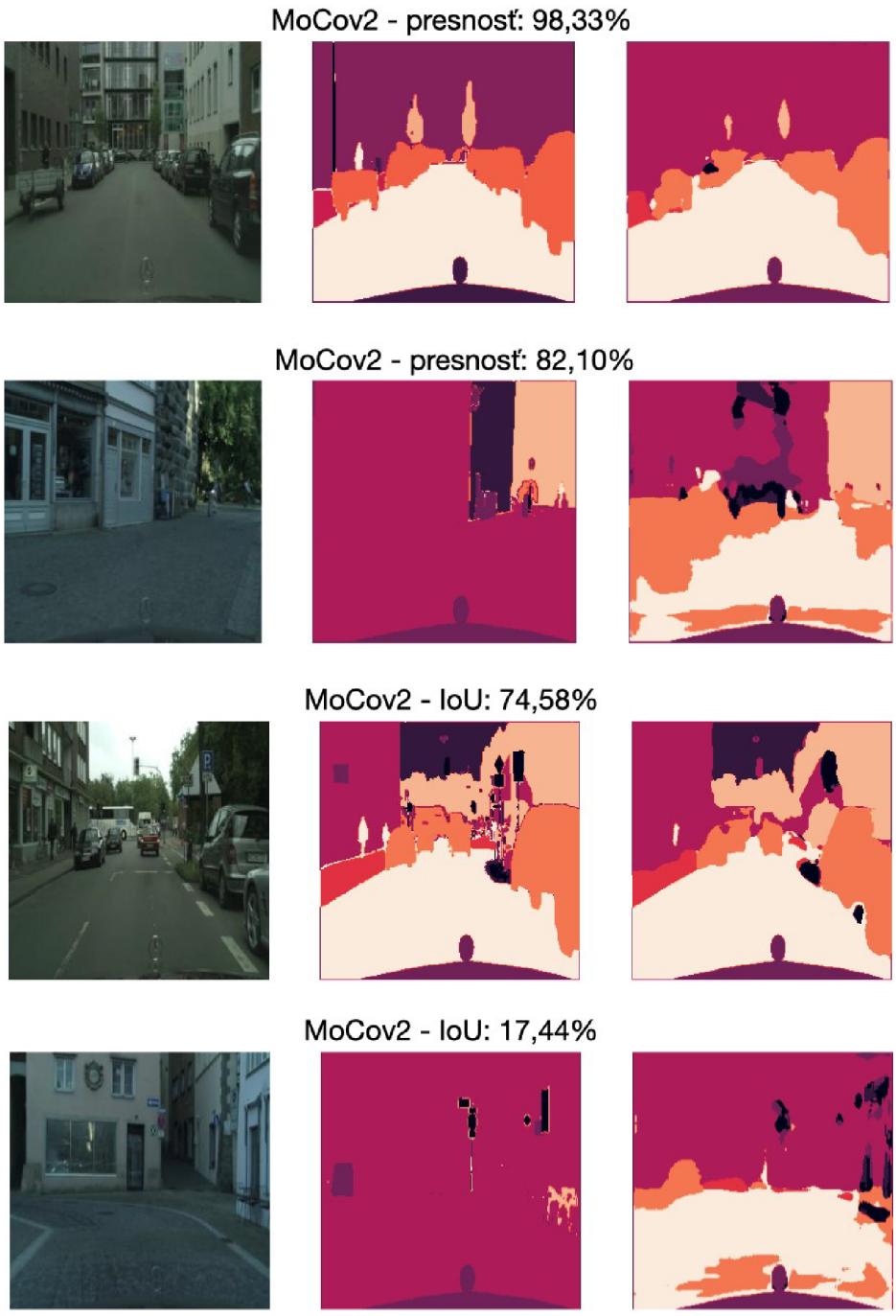
C Výsledky sémantickej segmentácie pri rôznych SSL metódach



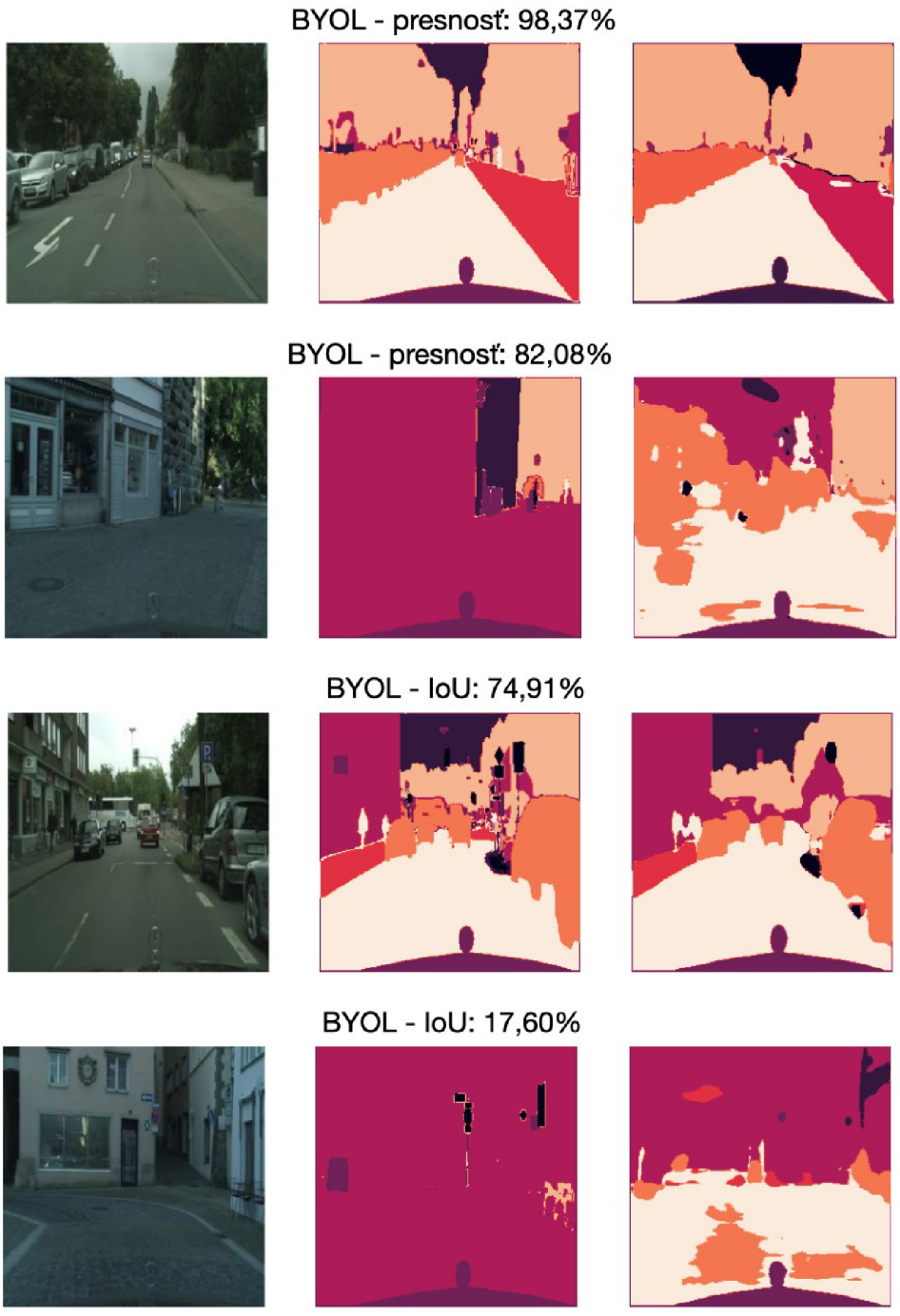
Obr. C.1: Enkóder Resnet50 nepredtrénovaný. Vyhodnotenie najlepšej a najhoršej presnosti a IoU. Na ľavo vstupný obrázok, v strede označenie tried masky, na pravo predikovaná maska.

Obr. C.2: Enkóder predtrénovaný SSL metódou Rotácie. Vyhodnotenie najlepšej a najhoršej presnosti a IoU. Na ľavo vstupný obrázok, v strede označenie tried masky, na pravo predikovaná maska.

Obr. C.3: Enkóder predtrénovaný SSL metódou SimCLR. Vyhodnotenie najlepšej a najhoršej presnosti a IoU. Na ľavo vstupný obrázok, v strede označenie tried masky, na pravo predikovaná maska.



Obr. C.4: Enkóder predtrénovaný SSL metódou MoCov2. Vyhodnotenie najlepšej a najhoršej presnosti a IoU. Na ľavo vstupný obrázok, v strede označenie tried masky, na pravo predikovaná maska.



Obr. C.5: Enkóder predtrénovaný SSL metódou BYOL. Vyhodnotenie najlepšej a najhoršej presnosti a IoU. Na ľavo vstupný obrázok, v strede označenie tried masky, na pravo predikovaná maska.

D Obsah elektronickej prílohy

Na priloženom CD sa nachádza diplomová práca v pdf formáte. CD taktiež obsahuje jednotlivé kódy ku aplikáciám Self-supervised metód napísané v prostredí Google Colab a s koncovkou .ipynb. Na CD sa taktiež nachádzajú použité datasety.

```
/ ..... koreňový adresár priloženého CD
├── Diplomová práca
│   └── Diplomová práca - Timotej Vančo.pdf
├── Codes ..... kódy jednotlivých Self-supervised metód
│   ├── Python.....kódy metód s koncovkou (.py) pre python
│   │   ├── Rotation_STL10
│   │   ├── SimCLR_STL10
│   │   ├── MoCov2_STL10
│   │   ├── BYOL_STL10
│   │   ├── Rotation_all
│   │   ├── SimCLR_all
│   │   ├── MoCov2_all
│   │   ├── BYOL_all
│   │   └── Cityscapes_SSLmethods_SemanticSegmentation
│   └── Jupyter ..... kódy metód s koncovkou (.ipynb) pre jupyter/colab
│       ├── Rotation_STL10
│       ├── SimCLR_STL10
│       ├── MoCov2_STL10
│       ├── BYOL_STL10
│       ├── Rotation_all
│       ├── SimCLR_all
│       ├── MoCov2_all
│       ├── BYOL_all
│       └── Cityscapes_SSLmethods_SemanticSegmentation
└── Datasets.....použité datasety
    ├── CIFAR10
    ├── Cityscapes
    ├── Imagenete
    ├── Industry_circles
    ├── MNIST
    └── Weather
```