ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DOLOVANIE TRIED V ČIASTOČNE ANOTOVANÝCH OBRAZOVÝCH DÁTACH

Dizertačná práca

Žilina, 2021 **Ing. Peter Lukáč**

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DOLOVANIE TRIED V ČIASTOČNE ANOTOVANÝCH OBRAZOVÝCH DÁTACH

Dizertačná práca

28360020213010

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Pracovisko:	Katedra informačných sietí
	Fakulta riadenia a informatiky Žilinskej univerzity v Žiline
Školiteľ:	prof. Ing. Martin Klimo, PhD.
Školiteľ špecialista:	Ing. Peter Tarábek, PhD.

Žilina, 2021 **Ing. Peter Lukáč**

Abstrakt

Lukáč Peter, Ing.: Dolovanie tried v čiastočne anotovaných obrazových dátach

(písomná práca k dizertačnej skúške). Žilinská univerzita v Žiline. Fakulta riadenia a informatiky. Katedra informačných sietí.

Cieľom práce je navrhnúť postupy a metódy na zefektívnenie procesu získavania anotovaných dát pre potreby vývoja metód počítačového videnia založených na hlbokom strojovom učení. Práca predstavuje rôzne postupy pri manuálnom anotovaní obrazových dát a nové postupy pri automatickom rozhodovaní o tom či obraz bude anotovaný alebo nie. Okrem známych metód strojového učenia založených na softmaxe sú navrhnuté nové postupy využívajúce detekčné a opravné kódy s podporou fuzzy logiky.

Kľúčové slová: strojové učenie, DNN, CNN, anotácia, GPGU

Abstract

Lukáč Peter, Ing.: Mining classes in partially annotated image data

(written part of dissertation examination). University of Žilina. Faculty of Management Science and Informatics. Department of Information Networks .

This theses aims to design methods for streamlining the process of obtaining annotated data in training computer vision methods based on deep machine learning. The work presents various procedures for manual annotation of image data and new procedures for an automatic decision whether the image will be annotated or not. New procedures using detection and correction codes with fuzzy logic support are proposed in addition to the known softmax-based machine learning methods.

Key words: machine learning, DNN, CNN, annotation, GPGU

Predhovor

Chcel by som sa poďakovať svojmu školiteľovi, prof. Ing. Martinovi Klimovi, PhD., ako aj Ing. Petrovi Tarábkovi, PhD. za konzultácie, ochotu, pripomienky a cenné rady k mojej práci.

Prehlasujem, že som predloženú písomnú prácu k dizertačnej skúške vypracoval samostatne pod odborným vedením školiteľa, vlastných teoretických poznatkov, praktických skúseností získaných v priebehu štúdia a na základe uvedenej literatúry.

Peter Lukáč

Obsah

Zoznam použitých značiek

ANN	-	Artifical neural network
CNN	-	Convolutionl neural network
BN	-	Batch normalizácia
AE	-	Autoenkóder
BBOX	-	Vyznačenie oblasti v obraze v tvare obdĺžnika
ILSVRC	-	ImageNet Large Scale Visual Recognition Challenge
GPGPU	-	General-purpose computing on graphics processing
DNN	-	Deep neural network
LR	-	Learning rate
AI	-	Artifical inteligence
FC	-	Fullly connected
MSE	-	Mean square error
RMSE	-	Root mean square error
MAE	-	Mean absolute error
MSE	-	Mean square error
SGD	-	Stochlastic gradient descent
GAN	-	Generative adversarial networks
ML	-	Machine learning
AMT	-	Amazon mechanical trunk
RA	-	Random Annotation
HCA	-	High confidence annotation
LCA	-	Low confidence annotation
ReLU	-	Rectified Linear Unit
ROC		Receiver operating characteristic
AUC		Area unde ROC curve

Úvod

Ľudia sa snažili v celej histórii uľahčiť si prácu. V dávnych dobách to bolo pomocou jednoduchých nástrojov, neskôr prišli na rad komplikovanejšie stroje, bez ktorých by dnes určité činnosti nebolo možné robiť. Jedným z odvetví, ktoré dnes patria do tejto skupiny sú informačné technológie (IT), ktoré akcelerovali ďalší pokrok naprieč všetkými odvetviami.

Jedna z dávnych túžob ľudí, ktorá siaha až do starovekého Grécka, je vytvoriť stroj, ktorý myslí. Inšpirovaný biológiou a podporovaný IT vznikol odbor umelá inteligencia (AI), ktorý sa snaží túto túžbu naplniť. Dnes je AI prosperujúca oblasť s mnohými praktickými aplikáciami a aktívnymi výskumnými témami. Vytvárame inteligentný softvér, ktorý má za úlohu automatizovať rutinné práce, pochopiť reč a obraz, robiť diagnostiku v medicíne a podporovať základný vedecký výskum.

V počiatkoch sa oblasť AI snažila riešiť problémy, ktoré sú z intelektuálneho hľadiska pre ľudí zložité, ale relatívne jednoduché pre počítače – problémy, ktoré môžu byť popísané matematickými pravidlami. Naozajstná výzva AI je riešiť úlohy, ktoré sú ľahké pre ľudí, ale je ťažké ich popísať formálne – problémy, ktoré vieme riešiť intuitívne, ako napr. rozpoznávanie hovorených slov, alebo tváre v obraze.

Táto práca sa zaoberá rozpoznávaním obrazu a problémami s nimi súvisiacimi. Hlavným problémom dneška pri práci s obrazom sú dáta. Ich získavanie je relatívne náročná práca, pretože si vyžaduje nielen jednoduché operácie (ako označovanie relevantných oblastí na obrázku a ich následné kategorizovanie), ale aj operácie náročné na sústredenie, dôsledkom čoho sa môžu vnášať chyby, ktoré nepriaznivo pôsobia na výsledné riešenie. Potrebné získanie veľkého množstva takýchto dát nie je lacná, ani jednoduchá záležitosť.

1 Úvod do problematiky

V posledných rokoch odbor počítačového videnia značne napreduje a jeho aplikácie sú čoraz viac používané v bežnom živote a na vedeckú scénu sa dostávajú rôzne problémy, ktoré je potrebné vyriešiť. Z pohľadu tejto práce ktorá sa zaoberá rozpoznávaním obrazu môžeme použiť nasledovné kategorizovanie základných úloh s učiteľom v počítačovom videní pre statické obrázky:

Obrázok 1.1. **typy úloh v počítačovom videní**

Klasifikácia – Ako vstupné dáta existujú obrázky, ktoré majú určené, do akej kategórie (triedy) patria. Úloha spočíva v určení triedy hlavného objektu na vstupnom obrázku.

Klasifikácia + Lokalizácia – Rozšírenie úlohy klasifikácie o úlohu lokalizácie. Okrem samotného obrázku a triedy je potrebné určiť aj oblasť na obrázku, kde sa daný objekt nachádza. Pre úspešné riešenie stačí nájsť ľubovoľný výskyt hlavného objektu.

Detekcia objektov – Klasifikácia a lokalizácia rieši zjednodušenú úlohu - na obrázku sa hľadá len jeden hlavný objekt. Detekcia objektov je úloha, pri ktorej je potrebné označiť všetky objekty, ktoré sa majú rozpoznať a priradiť im príslušnú kategóriu.

Segmentácia – Je to v podstate detekcia objektov, len presnejšia. Namiesto označenia objektov pomocou BBOX-u je objekt označený presne podľa tvaru objektu formou pixelov.

Pre každú z týchto úloh je potrebné, aby boli vytvorené datasety, ktoré zahŕňajú také obrazové dáta, ktoré majú požadovanú kvalitu, čím sa myslí, že reprezentujú doménu úloh a majú čo najmenšiu chybovosť parametrov, dostatočnú diverzitu (nie je dobré, ak existujú

dáta, ktoré sú veľmi podobné. napr. ten istý objekt v rovnakom uhle) a početnosť (početnosť dát v kategóriách by mal byť rovnaký). Ako sa odbor rozpoznávania obrazu vyvíjal, pre vedecké účely boli časom vytvorené rôzne datasety. Jeden z prvých a dodnes používaných datasetov je *MNIST* [1], ktorý reprezentuje ručne písané číslice, v ktorom obrázky tvorí čierne popredie na bielom pozadí. Počtom ho tvorí 60000 dát pre tréning a 10000 pre testovanie. Dnes je tento dataset príliš jednoduchý, ale stále sa v niektorých prípadoch používa na overenie konceptu riešenej úlohy. Jeden z najväčších datasetov pre úlohy rozpoznávania obrazu je *ImageNet* [2]. Za jeho vznikom bola motivácia vytvoriť čo najväčší dataset 80 000 tried (pod pojmom trieda si môžeme predstaviť skupinu dát reprezentujúcu rovnakú množinu ako napr. zviera, auto,…) a na každú 500 až 1000 obrázkov. Predstavuje testovací dataset na ktorom sa môžu porovnávať vedecké riešenia. Súťaž *ILSVRC* (ImageNet Large Scale Visual Recognition Challenge), bola jednou v ktorej tímy z celého sveta súťažili v úlohách rozpoznávania obrazu na tomto datasete. Ako základ pre vznik datasetu si vybrali WorldNet [3] databázu slov čo je hierarchická štruktúra anglického jazyka, kde jednotlivé slová sú poprepájané vzťahom namiesto abecedného poradia. Napr. "mačka" je pod slovom "mačkovité šelmy" a tie sú pod "cicavce". Tento slovník obsahuje viac než 155 000 indexovaných slov. V roku 2009 keď vyšla prvá verzia ImageNet-u obsahovala 3.2 milióna anotovaných obrázkov rozdelených do 5 247 kategórii a zotriedených do 12 podstromov ako cicavce, nábytok,... (dnes je to 14 miliónov obrázkov v 22 000 kategóriách). Počiatok vzniku datasetu bol náročný, keďže najatí študenti, ktorý mali hľadať a pridávať obrázky nestačili na objem práce, ktorú bolo potrebné vykonať. Pri ich tempe by to trvalo 19 rokov než by sa dataset stal dostatočne veľkým. Použitie služby Amazonu *Mechanical Turk,* čo je systém kde veľké množstvo ľudí po celom svete sedí za počítačom a vykonáva jednoduché operácie za malú finančnú odmenu pár centov, pomohlo vytvoriť dataset v takej miere, aby ho bolo možné zverejniť. Pracovalo na ňom cez 50 000 ľudí prehľadávalo viac ako 160 miliónov obrázkových kandidátov, pričom pre zefektívnenie vyhľadávania bol vytvorený software na hľadanie nových obrázkov a tak isto aj na ich samotnú validáciu. Jednotlivé obrázky sa validovali 2-5 krát. Zozbieranie takéhoto množstva dát potrebuje veľké úsilie ľudí, a nemalé technické a finančné prostriedky. Je preto dôležité sa venovať tejto problematike za účelom zjednodušenia a zlacnenia postupov pri získavaní dát. Taktiež kvalita dát je kľúčová pri riešení úloh, kedy chybne anotované dáta - či už z pohľadu nepresného označenia, alebo nepresnej kategórie - nie sú žiadúce, pretože to spôsobuje zhoršenie výsledného riešenia. Hľadanie takýchto chýb a ich následná oprava sa zdá ako jednoduchá úloha, ale prechádzanie 1000-ky obrázkov je z pohľadu človeka monotónna činnosť, a ľahko dokáže urobiť chyby.

Preto si táto práca kladie za jeden z cieľov nájsť také metódy a postupy, ktoré by zefektívnili spôsob získavania a opravovania dát či už automatickým, alebo poloautomatickým spôsobom.

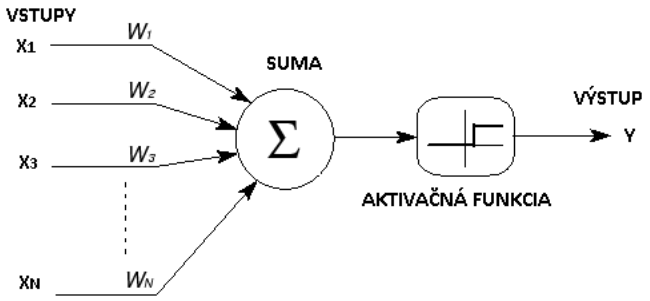
Vznik a rozmach riešení úloh počítačového videnia založených na hlbokom strojovom učení je do veľkej miery umožnený existenciou veľkých datasetov a vývojom hardwaru, keď nástup GPGPU umožnil veľkú paralelizovateľnosť algoritmov a tým sa úlohy s veľkým množstvom dát dokázali prepočítať v rozumnom čase (rádovo dni). Aj keď existuje viacero vhodných datasetov, pre riešenie praktických úloh je ich použitie veľmi obmedzené. Ak chceme v praxi vyvinúť riešenie špecifické pre niektorú doménu, potrebujeme vždy aj vhodný dataset, ktorý väčšinou nie je k dispozícii. Existujúce datasety sú skôr určené pre výskumné účely (v niektorých prípadoch aj licenčne). V praxi nastávajú situácie, kedy je veľmi ťažké získať dáta - či už z dôvodu neexistencie situácie, alebo malého výskytu reálnych situácii a dáta sú v týchto prípadoch málo zastúpené, čo vytvára nerovnomerné rozdelenie vzoriek vzhľadom na kategórie a s tým súvisiace problémy (pretrénovanie málo zastúpených tried). Preto je potrebné dáta rozširovať (napr. generovaním umelých dát), aby sme dosiahli aspoň minimálne množstvo potrebné na realizáciu úlohy. Existuje aj opačný problém, kedy dát je príliš veľké množstvo. V týchto súvislostiach je dobré si položiť otázku, aké množstvo a typ dát sa oplatí generovať, resp. použiť a ako to ovplyvní výsledné riešenie. V konečnom dôsledku pri hľadaní riešenia, ktoré môže prehľadávať veľký počet architektúr a ich hyperparametrov, môže zredukovanie dátovej množiny nezanedbateľne urýchliť čas potrebný na vyriešenie úlohy. V tejto práci sa predpokladá práca s typom úloh klasifikácie objektov.

2 Súčasný stav

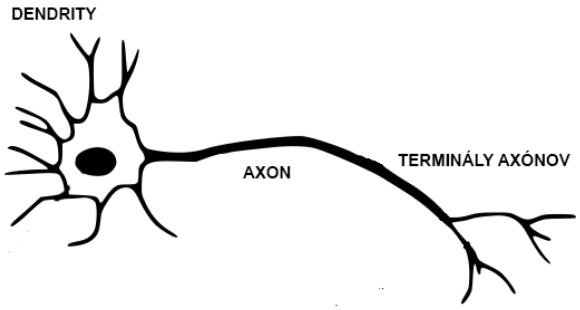
Pre úlohy klasifikácie v obraze sa používa učenie s učiteľom, nakoľko máme pre každý obrázok definovanú výstupnú triedu. Taktiež hovoríme o end-to-end učení kde medzi vstupom a výsledkom je iba neurónová sieť t.j. na vstupe nemusíme dáta zložito predspracovať a taktiež na výstupe dostávame priamo výsledok, ktorý netreba ďalej spracovávať. Nasleduje krátky prehľad komponentov modelov neurónových sietí a ich vlastností.

2.1 Umelé neurónové siete ANN

Vychádzajú z matematického modelu neurónu, ktorý je preformulovaný do matematickej reči z neurofyziologického neurónu. Je to základná stavebná jednotka neurónových sietí. Najznámejším modelom neurónu je tzv. **formálny neurón.** Jeho štruktúra je znázornená na obrázku 2.1



Obrázok 2.1. Formálny neurón



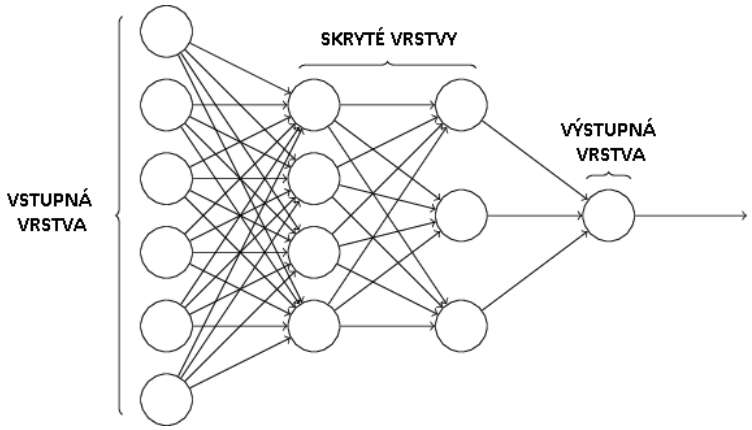
Obrázok 2.2. Biologický neurón

Na vstupe má ľubovoľný konečný počet vstupov $(X_1,X_2,...X_N)$ modelujúcich dendrity. Vstupy sú ohodnotené odpovedajúcimi všeobecne reálnymi synaptickými váhami W_1 ,.., W_N určujúcimi ich priepustnosť. Suma vstupných hodnôt a váh predstavuje vnútorný potenciál neurónu:

$$\varepsilon = \sum_{i=1}^{N} X_i W_i \tag{1}$$

Výstup je potom daný aktivačnou funkciou $f(\varepsilon)$. Aktivačná funkcia býva rôzna. Najčastejšie sa používali sigmoidálna a tangenciálna funkcia. Tieto funkcie majú problémy ako napr. dlhšia konvergencia pri tréningu kvôli saturácií. Dnes sa skôr používajú aktivačné funkcie typu ReLU.

Neurónová sieť sa skladá z formálnych neurónov, ktoré sú vzájomne prepojené tak, že výstup neurónu je vstupom do viacerých neurónov, podobne ako terminály axónov biologického neurónu sú cez synoptické väzby spojené s dendritami iných neurónov. Počet neurónov a ich vzájomné prepojenie v sieti určuje tzv. architektúru (topológiu) neurónovej siete. Z hľadiska využitia rozlišujeme v sieti vstupné, skryté a výstupné neuróny. V čase sa neurónová sieť vyvíja, mení sa stav neurónov, adaptujú sa váhy. V súvislosti so zmenou týchto charakteristík v čase je dobré celkovú dynamiku neurónovej siete rozdeliť do 3 dynamík (organizačná, aktívna a adaptívna) a uvažovať tak tri režimy práce so sieťou [4].



Obrázok 2.3. Umelá neurónová sieť

Organizačná dynamika

Špecifikuje architektúru siete a jej prípadnú zmenu. Zmena topológie sa väčšinou uplatňuje v rámci adaptívneho režimu tak, že sieť je v prípade potreby rozšírená o ďalšie neuróny a príslušné spoje. Avšak organizačná dynamika prevažne predpokladá pevnú architektúru

neurónovej siete, ktorá sa už nemení. Rozlišujeme v zásade dva typy architektúry: cyklická (rekurentná) a acyklická. Neuróny v acyklickej architektúre môžeme vždy disjunktne rozdeliť do tzv. vrstiev, ktoré sú usporiadané (napr. nad sebou) tak, že spoje medzi neurónmi vedú len z nižších vrstiev do vyšších a všeobecne môžu preskočiť jednu, alebo viac vrstiev.

Aktívna dynamika

Špecifikuje počiatočný stav siete a spôsob jeho zmeny v čase pri pevnej topológii a konfigurácii. V aktívnom režime sa na začiatku nastavia stavy vstupných neurónov na vstupe siete a zvyšné neuróny sú v uvedenom počiatočnom stave. Po inicializácií stavu siete prebieha vlastný výpočet. Vo všeobecnosti sa uvažuje spojitý vývoj stavu neurónovej siete v čase a hovorí sa o tzv. *spojitom modeli,* kedy stav siete je spojitou funkciou času, ktorá je obvykle v aktívnej dynamike zadaná diferenciálnou rovnicou. Väčšinou sa však predpokladá diskrétny čas, t.j. na počiatku sa sieť nachádza v čase 0 a stav siete sa mení v čase 1,2,3,....atď. V každom takomto časovom kroku je podľa pravidla aktívnej dynamiky vybraný jeden neurón (tzv. sekvenčný výpočet), alebo viac neurónov (tzv. paralelný výpočet), ktoré aktualizujú svoj stav na základe svojich vstupov. Stav výstupných neurónov, ktorý sa všeobecne mení v čase sa nazýva *výstup neurónovej siete*.

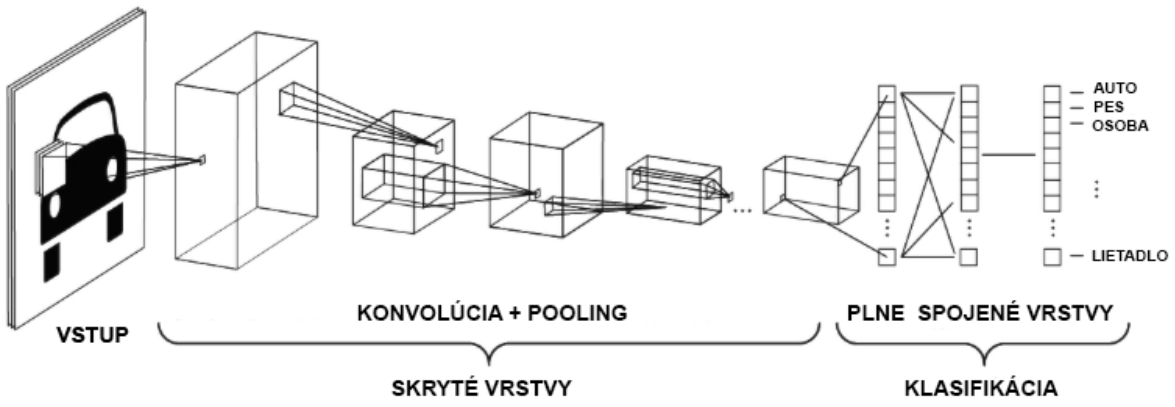
Adaptívna dynamika

Špecifikuje počiatočnú konfiguráciu siete a to, akým spôsobom sa menia váhy v sieti v čase. Všetky možné konfigurácie siete tvoria tzv. váhový priestor neurónovej siete. V adaptívnom režime sa na začiatku nastavia váhy všetkých spojov v sieti na počiatočnú konfiguráciu. Po inicializácii konfigurácie prebieha vlastná adaptácia. Podobne ako v aktívnej dynamike všeobecne sa uvažuje spojitý model, väčšinou sa však predpokladá diskrétny čas adaptácie. Cieľom adaptácie je nájsť takú konfiguráciu siete vo váhovom priestore, ktorá by v aktívnom režime realizovala predpísanú funkciu. Ak sa aktívny režim siete využíva k vlastnému výpočtu funkcie siete pre daný vstup, potom adaptívny režim slúži k učeniu tejto funkcie. Požadovaná funkcia siete je obvykle zadaná tzv. tréningovou množinou dvojíc vstup - výstup. Vstupy tréningových vzorov môžu byť napr. vzorové obrázky písmen, ktorým ako výstupy zodpovedajú jednotlivé písmená (napr. u výstupného neurónu, ktorý reprezentuje písmeno „A“ je požadovaná hodnota stavu rovná „1“ práve keď je na vstupe príklad obrazu písmena „A“ a ostatné výstupné neuróny sú v takomto prípade neaktívne). Tento spôsob popisu požadovaného chovania siete modeluje učiteľa, ktorý pre vzorové vstupy siete informuje

adaptívny mechanizmus o správnom výstupu siete. Preto sa tomuto typu adaptácie hovorí *učenie s učiteľom (supervised learning*). Iným typom adaptácie je tzv. *samo organizácia.* V tomto prípade tréningová množina obsahuje iba vstupy siete. To modeluje situáciu kedy nie je k dispozícii učiteľ a preto sa nazýva *učenie bez učiteľa* (*unsupervised learning*). Neurónová sieť sama organizuje tréningové vzory napr. do zhlukov a učí sa ich súborové vlastnosti. Autoenkóder je ďalším príkladom učenia bez učiteľa.

2.2 Konvolučné neurónové siete

Konvolučné neurónové siete (CNN, obrázok 2.4) sú dopredné siete, pri ktorých informačný tok ide iba jedným smerom - zo vstupu k výstupu. Ich história sa začala neurobilogickými experimentmi vedenými Hubelom a Wieselom už v roku 1959 [5]. Hlavný prínos ich práce spočíva v objave, že neuróny v rozličných štádiách vizuálneho systému silne zodpovedali určitým stimulovým vzorom, zatiaľ čo iné ignorovali. Inými slovami objavili, že neuróny v skorých štádiách primárneho vizuálneho kortexu silne zodpovedali precízne orientovaným vzorom svetla (ako sú napr. čiary), ale ignorovali komplexnejšie vzory vložených podnetov, ktoré vyústili v silné odpovede z neurónov v neskorých štádiách. Tiež objavili, že vizuálny kortex pozostával z jednoduchých buniek, ktoré mali lokálne receptívne polia a komplexných buniek, ktoré boli nemenné k posunutým, alebo skresleným vstupom. Tieto poznatky motivovali architektúry CNN. CNN architektúry majú viacero modifikácií, ale vo všeobecnosti sú tvorené konvolučnými a pooling vrstvami, za ktorými nasledujú plne spojené vrstvy, ktoré tvoria klasifikáciu vstupu.

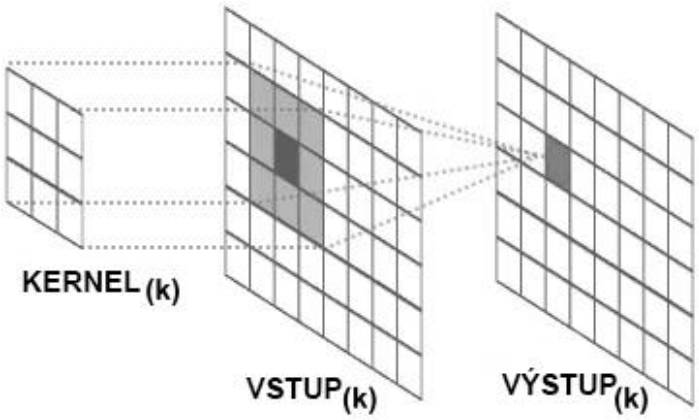


Obrázok 2.4. Konvolučná neurónová sieť

Konvolučná vrstva

Konvolučné vrstvy slúžia ako extraktor vlastností a teda sa dokážu učiť rôzne reprezentácie vlastností ich vstupov. Neuróny v konvolučných vrstvách sú usporiadané do kernelov v príznakových mapách. Každý neurón v príznakovej mape má pole vnímania (receptívne pole), ktoré je spojené so susediacimi neurónmi v predchádzajúcej vrstve cez sadu natrénovaných váh. Všetky neuróny vo vnútri príznakovej mapy majú váhy, ktoré sú nútené byť rovnocenné avšak rozličné príznakové mapy vo vnútri tej istej konvolučnej vrstvy majú rozličné váhy tak, aby rozličné príznaky mohli byť extrahované na každej pozícii. Matematicky sa dá *K*-tý výstup príznakovej mapy Y_k vypočítať ako

$$Y_K = f(W_K * x) \tag{2}$$



Obrázok 2.5. Konvolučný výpočet

kde vstup je označený ako *x* a konvolučný filter vzťahujúci sa ku *K*-tej príznakovej mape je označený ako W_K (obrázok 2.5). Znak * je 2D konvolučný operátor, ktorý je použitý na výpočet skalárneho súčinu filtrovaného modelu na každej pozícii vstupného obrazu a *f(.)* reprezentuje nelineárnu aktivačnú funkciu. Pri konvolučných vrstvách, ktoré majú na vstupe viac príznakových máp je pre každú vstupnú mapu počítaný vlastný kernel, ktorý sa nakoniec sčíta do jedného výstupu.

Nelineárne aktivačné funkcie dovoľujú extrahovanie nelineárnych vlastností. Tradične boli používané sigmoid a hyperbolický tangens, ale v posledných rokoch sa stala populárna ReLU aktivačná funkcia a jej modifikácie. Výhodou konvolučnej vrstvy oproti plnej vrstve kde je každý neurón prepojený so všetkými výstupmi z predchádzajúcej vrstvy je menší počet parametrov, ktoré v konvolúcii tvoria iba kernely. Tým pádom je výpočet takejto vrstvy menej náročný čo spôsobilo, že siete môžu mať viac vrstiev a byť komplexnejšie pri menšom

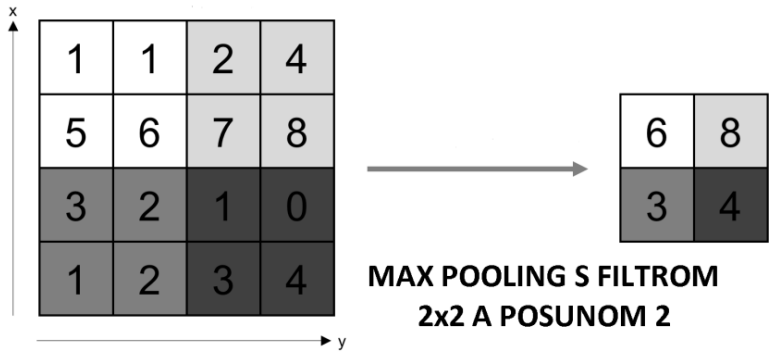
počte parametrov. Ďalšou výhodou je, že vlastnosti, ktoré konvolučná vrstva hľadá v obraze sú pozične invariantné t.j., že časť, ktorá sa vyskytuje na vstupe viac krát napr. tvár človeka sa sieť nemusí učiť na presné miesto ako je to pri plných vrstvách, ale kernel pri prechádzaní obrazu reaguje len na tie časti, ktoré majú požadované vlastnosti.

Pooling vrstva

Účel pooling vrstiev je redukovať rozlíšenie príznakových máp a tak docieliť nižšiu výpočtovú náročnosť (keďže sa zmenší počet vstupov neurónov). Tak ako operácia konvolúcie je aj pooling operácia daná veľkosťou filtra, v ktorom sa vykonáva a veľkosťou posunu daného filtra. Pooling sa aplikuje na každú mapu príznakov samostatne, čo zaručuje, že počet máp v danej vrstve sa nezmení. Existuje viacero pooling vrstiev, z ktorých za najpoužívanejšie môžeme označiť max pooling a spriemerovaný (average) pooling.

Max pooling

Max polling vyberá z príznakovej mapy maximálnu hodnotu. Ako to ilustruje obrázok:



Obrázok 2.6. Max pooling

Spriemerovaný pooling

Spriemerovaný pooling počíta z filtra priemernú hodnotu.

LP pooling

Hoci použitie max poolingu má výborne empirické výsledky, môže sa pretrénovať na tréningových dátach a negarantuje generalizáciu na testovacích dátach. Na druhej strane, spriemerovaný pooling zvažuje všetky elementy v oblasti poolingu a teda oblasti s nízkou aktiváciou môžu znížiť efekt oblastí s vysokou aktiváciou [5,6]. Tieto problémy rieši

biologicky inšpirovaný L_P pooling, ktorý je modelovaný na komplexných bunkách [7,8]. V danej poolingovej oblasti R_j vezme vážený priemer aktivácií a_i nasledovne:

$$s_j = \left(\sum_{i \in R_j} a_i^p \right)^{1/p} \tag{3}$$

V prípade, že *p=1*, táto rovnica korešponduje so spriemerovaným poolingom zatiaľ čo $p = \infty$ sa interpretuje ako maximálny pooling. Pre hodnoty $1 < p < \infty$,L_P pooling môže byť chápaný ako kompromis medzi spriemerovaným a maximálnym poolingom. Navyše vykonaná teoretická analýza [9,10,11] naznačuje, že poskytuje lepšiu generalizáciu s porovnaným max poolingom.

Zmiešaný Pooling (Mixed Pooling)

Inšpirovaný stochastickou povahou poolingovej techniky opísanej v [6] a inými úspešnými stochastickými regularizačnými technikami ako Dropout [12,13] a DropConnect [14], v 2014 uviedli novú poolingovú techniku nazvanú *zmiešaný pooling*, aby viac podporili reguralizačné schopnosti neurónových sietí a určili známe problémy spojené s priemerným a maximálnym poolingom. Taktiež využili stochastický postup na náhodné použitie maximálneho, alebo priemerného poolingu počas tréningu neurónových sietí. Matematicky vyjadrené, výstup y_{kij} zmiešaného poolingu vo vzťahu ku *k*-tej mape príznakov je vypočítaný použitím vzorca:

$$y_{kij} = \lambda . \max_{(p,q) \in R_{ij}} X_{kpq} + (1 - \lambda) \frac{1}{|R_{ij}|} \sum_{(p,q) \in R_{ij}} X_{kpq} \tag{4}$$

kde element na pozícii *(p,q)* vnútri poolingovej oblasti R_{ij} s veľkosťou $|R_{ij}|$ je reprezentovaný X_{kpq} a buď maximálny, alebo spriemerovaný pooling je vybraný pomocou premennej λ, ktorá má náhodnú hodnotu buď 1 alebo 0.

Plne prepojená vrstva

Cieľ plne prepojenej (FC) vrstvy v CNN je zobrať výsledok poslednej konvolučnej, alebo pooling vrstvy v bloku, ktorý sa učí extrahovať vlastnosti (obr.3-4) a použiť ho na

klasifikáciu obrazu triedy. Počet FC vrstiev na konci neurónovej sieti môže byť rôzny. Výstup z FC vrstvy je transformovaný do 1D vektoru, ktorý hovorí s akou pravdepodobnosťou patrí daný vstup kategórii. Preto sa na výstupe používa prevažne softmax funkcia, ktorá výstupný vektor vyjadruje ako pravdepodobnosť, že daný vstup patrí do konkrétnej triedy.

Aktivačné funkcie

Voľba aktivačnej funkcie ovplyvňuje čas potrebný na natrénovanie siete a to má podstatný vplyv na výkon veľkých sietí na veľkých datasetoch [15]. V minulosti boli používane aktivačné funkcie typu sigmoid a tangent, ktorých hlavná nevýhoda bola, že pri nich dochádzalo k ich saturácii. To má za následok spomalenie tréningu, keďže je potrebných veľa iterácii na zmenu saturovaného stavu.

ReLU

Po častiach lineárna funkcia ktorá má tvar $f = max(x,0)$ sa nazýva ReLU. Ponecháva si len pozitívnu časť aktivácie redukovaním negatívnej časti na 0, zatiaľ čo integrovaný max operátor podporuje rýchlejší výpočet. Vedie k rýchlejšej konvergencii a netrpí vanishing gradient problémom. Ako aktivačná funckia bola prvýkrát popísaná v roku 2000 [16] a spopularizovaná Nair a Hintonom v roku 2010 [17]. Nevýhodou je, že rozsah výstupu je v intervale $[0, \infty)$ čo môže viesť k explodujúcim gradientom.

LReLU

Aj keď ReLU vedie k rýchlejšej konvergencii a netrpí vanishing gradient problémom, v ktorom nižšie vrstvy majú gradienty blízke nule (pretože vyššie vrstvy sú takmer saturované), môže byť znevýhodňovaný počas optimalizácie, keďže gradient je nula keď neurón nie je aktívny. Toto môže viesť k prípadom, kde neuróny nie sú nikdy aktivované, keďže populárne optimalizačné algoritmy založené na gradiente zlepšujú len váhy neurónov, ktoré boli predtým aktivované. Nazýva sa to *problém umierajúcich neurónov*. Podobne ako pri vanishing – gradient problému ReLU trpí pomalou konvergenciou, keď trénované siete obsahujú konštantné nulové gradienty. Aby sa toto kompenzovalo, Maas a kolektív [18] uviedli LReLU (leaky rectified linear units), ktoré zohľadňujú malé nenulové gradienty keď jednotka nie je aktívna, ale saturovaná. Matematicky je LReLU vyjadrené:

$$f(x) = \max(x, 0) + \lambda . \min(x, 0) \quad (5)$$

kde λ je predefinovaný parameter z rozsahu *(0,1).*

PReLU

Kým LReLu sa spoliaha na predefinovaný parameter, aby sa zredukovala negatívna časť aktivačného signálu, PReLU (parametric rectified linear unit) [19] bol navrhnutý tak, aby sa parameter aktivačnej funkcie učil počas tréningu. Matematicky je PReLU rovnaký ako LReLU až na to, že λ je nahradená λ_k schopnou sa naučiť rôzne vstupné kanály označené *k.* Matematicky je PreLU vyjadrené:

$$f(x_k) = \max(x_k, 0) + \lambda_k . \min(x_k, 0) \quad (6)$$

ELU

Exponencial Linear Unit aktivácia podobná iným nesaturovaným aktivačným funkciám ako ReLU netrpia *vanishing gradient* a *exploding gradient* problémom. Na rozdiel od ReLU netrpí ani problémom umierajúcich neurónov. Má lepšie výsledky ako ReLU a jemu podobných aktivačných funkcií PReLU, LReLU. ELU aktivačná funkcia je spojitá a diferencovateľná na celom rozsahu.

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases} \quad (7)$$

Na rozdiel od ReLU, ELU má záporné hodnoty čo spôsobuje, že priemer aktivácii je posunutý smerom k 0. Podľa [20] v dôsledku tohto posunu tréning modelu konverguje rýchlejšie než pri ostatných aktivačných funkciách. Napriek pomalšiemu výpočtu ELU funkcie táto rýchlejšia konvergencia dovoľuje trénovať model rýchlejšie než ReLU a jeho varianty.

Softmax

Táto aktivácia sa používa väčšinou na poslednej vrstve pri viacnásobnej klasifikačnej úlohe. Je to druh logistickej regresie ktorá normalizuje vstupné hodnoty do pravdepodobnostného rozdelenia, ktorého suma je 1.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_j^k e^{x_j}} \quad (8)$$

2.3 Trénovanie pomocou učiteľa

V klasifikačných úlohách s učiteľom dáta môžu obsahovať kategóriu zvyčajne nie číselného typu. Napríklad zvieratá majú hodnoty ako mačka, pes, kôň. Farba môže mať hodnotu červená, biela, modrá. Niektoré algoritmy ako *rozhodovací strom* dokážu pracovať s takto zadefinovanými hodnotami ale väčšina algoritmov ML požaduje na výstupe číselnú hodnotu. Preto jednou z prvých úloh pri riešení klasifikácie je konvertovať takto zadané hodnoty do číselnej podoby. Je veľa prístupov ako zakódovať výstup. Dva základné prístupy sú:

2.3.1 Kódovanie tried

Tento prístup konvertuje každú hodnotu do čísla. Ak napr. máme kategórie ako auto, zviera, človek, rastlina potom zakódovanie vytvoríme ako pridadenie postupnej sekvencie čísel 0, 1, 2, 3.

auto	zviera	človek	rastlina
0	1	2	3

Tabuľka 2.1 Kódovanie tried 1

Tento prístup použitia sekvencie čísiel prináša problém kedy vzťah medzi jednotlivými kategóriami zavádza prednosť a algoritmus môže nesprávne interpretovať poradie auto < zviera < človek < rastlina .t.j rastlina je 4x dôležitejšia ako auto. Ak budeme uvažovať iné kategórie napr. zmena hladiny rieky tak v tomto prípade takto zakódovaná kategória má zmysel. Je preto potrebné zvoliť správne kódovanie výstupu dát.

malá	nízka	stredná	vysoká	veľmi vysoká
0	1	2	3	4

Tabuľka 2.2 Kódovanie tried 2

One-hot kódovanie

Rieši problém interpretácie poradia kategórie. V tejto stratégii, je každá kategória konvertovaná do hodnoty 1 alebo 0 (true/false) tak, že hodnota 1 sa nachádza pre jednu z kategórii iba raz.

	auto	zviera	človek	rastlina
auto	1	0	0	0
zviera	0	1	0	0
človek	0	0	1	0
rastlina	0	0	0	1

Tabuľka 2.3 one-hot kódovanie

Pri učení na takto zakódované dáta je použitá účelová funkcia ktorá spočíta chybu a na základe optimalizačného algoritmu sú zmenené váhy neurónovej siete. Vyhodnotenie úspešnosti natrénovanej neurónovej siete sa vypočíta na základe rôznych metrík. Jedny z najzákladnejších sú *Precision* a *Recall*.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{22}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{23}$$

Precision (22) hovorí o tom ako presne model predikuje pozitíva. Koľko z pozitív je naozaj pozitívnych. Je to dobrá metrika na určenie či chybne pozitívnych prípadov je veľa. Napr. ak máme spam filter ktorý identifikuje mail ktorý nie je spam ako spam. Spam filter ktorý má nízku *Precision* môže užívateľovi stratiť dôležitý mail.

Recall (23) počíta koľko pozitív model dokáže zachytiť ktoré su označené ako pozitíva v dátach, t.j koľko dát ktoré majú danú triedu vieme identifikovať. Napr. ak chorý človek ktorý je nakazený infekčnou chorobou je označený za zdravého môže mať veľmi zlé dôsledky pre jeho okolie.

Pri použití rôznych pravidiel na výstupe aktivačných funkcií (ako napr. softmax, sigmoid, tanh,..) na konci neurónovej siete vieme výstup spracovať ďalšími pravidlami. Jeden z najzákladnejších prístupov je dosiahnutie prahu (*threshold*) pod hranicou ktorého môžeme výstup považovať za nedôveryhodný. Existujú ďalšie metódy ako napr. spájanie výstupov viacerých neurónových sietí do jedného tzv. *ensemble.* Ak budeme uvažovať o klasifikačnej úlohe ktorá má *N* kategórii a natrénujeme sieť iba na tieto kategórie, pri vložení iného typu vstupu sieť nemá inú možnosť ako vyhodnotiť vstup tak, že patrí do jednej z kategórii. Je to jeden z problémov neurónových sietí ktoré nevedia povedať že *neviem*. Tento problém sa dá čiastočne odstrániť spomínaným prahom alebo pridaním ďalšej kategórie s názvom *neviem* v ktorom budú všetky ostatné možnosti. Takýto prístup má však ďalšie problémy pretože musíme nájsť dostatočný počet a rôznorodosť dát, ktoré nám túto kategóriu budú zastupovať čo v praxi nie je možné z dôsledku nekonečného množstva možností, ak neriešime špecifickú úlohu kde iné vstupy nemôžu nastať. Každopádne takýto prístup si vyžaduje ďalšiu dodatočnú prácu nad dátami. V ďalších kapitolách bude rozobratý postup ako sa tento problém snažíme riešiť v tejto práci.

2.3.2 Účelové funkcie

Účelová funkcia, alebo po anglicky *loss function* meria ako dobre, alebo zle model neurónovej siete dokáže odhadovať predikciu či daný vstup patrí/nepatrí do konkrétnej kategórie. Inak povedané účelová funkcia je použitá na výpočet gradientov, ktoré sú použité na zmenu váh pri trénovaní neurónovej siete. Tak isto ako pri vrstvách a ich aktiváciách je potrebné, aby účelová funkcia bola diferencovateľná v celom rozsahu. Podľa druhu úlohy môžeme účelovú funkciu rozdeliť na:

Regresná funkcia

Kedy model predikuje spojitú hodnotu ako je napr. cena akcií na burze, veľkosť plochy, objemu. Najviac používané funkcie sú: MSE, RMSE, MAE, Huber.

Binárna klasifikačná funkcia

Používa sa keď chceme predikovať jednu z dvoch kategórii indikovaných ako 0 alebo 1. Výstup binárnej klasifikácie je najčastejšie skóre, ktoré určuje s akou presnosťou je

klasifikácia správna. Na určenie do ktorej triedy patrí predikcia sa používa prah zväčša z hodnotou 0,5. Najčastejšie sa používa binárna krížová entropia:

$$L = -y.\log(p) + (1-y).\log(1-p), \tag{9}$$

kde y je binárny indikátor 0 alebo 1 ak vzor klasifikácia patrí/nepatrí do kategórie. P je predikcia pravdepodobnosti siete, že vzor patrí do triedy.

Viacnásobná klasifikačná funkcia

Je to rozšírenie binárnej klasifikácie kedy chceme klasifikovať viac ako 2 kategórie. Vypočíta sa ako suma chyby pre každú kategóriu

$$L = -\sum_{1}^{M} y_c.\log(p_c) \tag{10}$$

Kde y_c je binárny indikátor (0 alebo 1) ak vzor patrí do kategórie c a p_c je predikcia pravdepodobnosti, že vzor patrí do kategórie c.

Multinásobná klasifikačná úloha

Nejedná sa o funkciu, ale o úlohu kedy chceme, aby na výstupe NN nebol aktívny len jeden vstup, ale viacero. Napríklad pri označovaní kategórie filmu chceme aby na výstupe sme dostali naraz kategórie: akčný, sci-fi, dobrodružný. Pre tento typ úloh sa ako posledná aktivačná funkcia používa sigmoid a chybová funkcia binárna krížová entropia.

2.3.3 Regularizačné mechanizmy

Výskum v regularizácii je motivovaný tendenciou neurónových sietí učiť sa špecifické vlastnosti datasetu, na ktorých boli trénované na úkor učenia sa všeobecných vlastností, ktoré sú aplikovateľné na dáta, ktoré sieť nikdy nevideli, čo je známe ako *pretrénovanie*. Cieľ strojového učenia s učiteľom je aproximovať funkciu, ktorá mapuje vstupy do výstupov na danej dátovej množine. Dôležitým predpokladom je, že modely sú trénované na dátach, ktoré reprezentujú skutočné rozdelenie dát a cieľovej funkcie, ktorá sa má aproximovať. Lenže množstvo tréningových dát obsahuje šum špecifický pre daný výber a rôzne náhodné

odchýlky od skutočného vstupného dátového rozdelenia. Vzhľadom na túto skutočnosť aproximácia funkcie reprezentujúca veľmi presne tréningové dáta je nežiadúca. Algoritmus sa bude učiť šum z tréningových dát a bude veľmi pravdepodobné, že nebude pracovať dobre na dátach, ktoré nikdy nevidel. Regularizácia pomáha učiť skutočnú funkciu a ignorovať šum. Najviac všeobecne známe formy regularizácie sú:

Penalizácia váh

Pridáva ďalší prvok do chybovej funkcie, ktorého veľkosť súvisí s veľkosťou všetkých váh v neurónovej sieti. Parameter λ určuje veľkosť uloženej penalizácie, pretože sa vynásobí váhovou pokutou. Vyššia λ znamená vyššiu penalizáciu a $\lambda = 0$ znamená žiadnu. λ parameter nie je učiaci sa parameter, ale je určený pri ladení hyperparametrov. Cieľom penalizácie váh je v priemere znížiť veľkosť váh pri trénovaní siete, pretože veľké váhy vedú k väčšej chybe. Ak neurónová sieť nemá penalizáciu váh, potom váhy majú tendenciu byť tlačené do väčších hodnôt a to aj v prípade, že výsledok je len nepatrný pokles chyby. Zvažujme prípad kedy výstup siete je rozloženie pravdepodobnosti nad možnými kategóriami dosiahnutý tým, že máme aktivačnú funkciu softmax v poslednej vrstve siete. Výstupné hodnoty sú z intervalu (0,1), kde výstup je 1, ak trieda je správna a 0 pre ostatné. Sieť nebude mať nikdy hodnotu výstupu 1, ale môže pokračovať v nastavení váh do viac extrémnych hodnôt, aby sa výstup priblížil bližšie k 1. Či si je sieť istá na 99% alebo 99.9%, že dáta patria do konkrétnej kategórie, má malý benefit na jej spoľahlivosť a namiesto toho vzniká riziko pretrénovania. Je to preto, lebo veľká váha má za následok, že výstup siete je citlivý na malé zmeny príznakov, pretože zmeny v tomto príznaku sú zosilnené vynásobením veľkou váhou. Teoreticky to môže byť správne, ale pravdepodobnejším výsledkom je, že sieť sa učí šum. Neurónové siete by mali byť relatívne lokálne necitlivé, t. j. vstupy z podobných oblastí by mali produkovať podobné výstupy. Malé zmeny na vstupe by nemali viesť k veľkým zmenám na výstupe. Dve najčastejšie používané formy regulácie sú: L1 a L2. Rozdiel je v tom, ako sa váhy penalizujú v chybovej funkcii. Nech *C* je akoukoľvek chybovou funkciou použitou na trénovanie siete a nech C_i je chyba pre jednu vzorku:

$$L_1: C = \frac{1}{n}\left(\sum_{i=1}^{n} C_i + \lambda \sum_{w \in W} |w|\right) \tag{11}$$

$$L_2: C = \frac{1}{n}\left(\sum_{i=1}^{n} C_i + \frac{\lambda}{2}\sum_{w \in W} w^T w\right) \quad (12)$$

$$L_1: \frac{\partial C}{\partial w} = \lambda.sign(w) - iba\ regularizačný\ komponent\ chybovej\ funkcie \quad (13)$$

$$L_2: \frac{\partial C}{\partial w} = \lambda.w - iba\ regularizačný\ komponent\ chybovej\ funkcie \quad (14)$$

Pri L_2 regularizácii, derivácia reguralizačnej zložky chybovej funkcie vzhľadom na maticu váh w je $\lambda.w$. Príspevok do aktualizácie váh závisí na aktuálnej hodnote váh. Pri L_1 reguralizácií , derivácia regularizačnej zložky chybovej funkcie vzhľadom na maticu váh w je $\lambda.sign(w)$, kde $sign(w)$ je matica rovnakej veľkosti ako w, v ktorej každý prvok je znamienko zodpovedajúceho prvku w. Príspevok z regularizácie L_1 počas aktualizácie váh je úmerný λ, a nezávisí od aktuálnej hodnoty w. To má za následok zmenšenie w k nule, berúce λ ako veľkosť kroku. Naproti tomu L_2 zmenšuje w k nule s veľkosťou kroku $\lambda.w$. Čím bližšie je w k nule, tým menšia je aktualizácia s L_2. Toto neplatí v prípade L_1 reguralizácie. Zatiaľ čo váhy sú zvyčajne malé s reguralizáciou L_2 nemajú tendenciu byť 0. Naproti tomu reguralizácia L_1 má tendenciu presadzovať riedkosť na modeli a vytvára veľa nulových váh. Takže je len niekoľko aktívnych príznakov, ktoré prispievajú k mapovaniu vstupov na výstup.

Skoré zastavenie tréningu

Veľmi intuitívna technika je skoré zastavenie tréningu**.** Regularizačný efekt je zastavenie tréningového procesu predtým, ako sa model začne učiť šum na tréningovom datasete. Keď sa model trénuje je vidieť ako chyba trénovania klesá na tréningovej a testovacej množine. Ako sa zvyšuje počet epoch trénovania, tréningová chyba sa postupne zmenšuje, až sa zastaví na určitej hodnote a tréning má tendenciu stagnovať, alebo pomaly klesať. Avšak keď testovacia chyba dosiahne svoje minimum začne rásť a to je znamenie, že model sa začína pretrénovávať.

Dropout

Je dômyselná technika pre budovanie skupín nezávislých neurónových sietí s rovnakou dátovou a sieťovou architektúrou výpočtovo efektívnym spôsobom. Vždy, keď je v dávkach spracovávaný dataset v priebehu učenia, každý neurón v skrytých vrstvách je vypnutý (výstup je 0) s nejakou pravdepodobnosťou *p* (typicky 50%). To má za následok náhodné generovanie podsiete z hlavnej neurónovej siete. Práve s touto podsieťou sa počíta aktívna a adaptívna dynamika siete. Pri každej novej dávke z datasetu sa generuje nová podsieť, a preto sa na vytvorenie predikcie používa odlišná podmnožina celkových dostupných funkcií. Zakaždým, keď sa model testuje na testovacích dátach sa musí každý neurón zapnúť a vynásobiť váhy pravdepodobnosťou *p*. Násobenie *p* sa ukazuje ako dobrá aproximácia očakávanej hodnoty výstupu neurónu nad všetkými možnými podsieťami a je známa ako *weight scaling inference rule* [21]. Je to mechanizmus pomocou ktorého sa vypočíta väčšina hlasov. Výsledok je ekvivalentný spriemerovaniu výstupu veľmi veľkého počtu neurónových sietí, ktoré boli trénované na rovnakých dátach.

Počas tréningu sa každý skrytý uzol nemôže spoliehať na prítomnosť konkrétneho vstupu. Takže, keď neurón detekuje určitú vlastnosť, sieť sa to musí dokázať naučiť niekoľkými rôznymi spôsobmi. To by mohlo replikovať schopnosť detekcie príznakov. Napríklad ak vrstva detekuje prítomnosť tváre tým, že identifikuje prítomnosť nosa potom predchádzajúca vrstva môže mať viacero funkcií na detekciu nosa, takže funkcia detekcie tváre je robustná pre akýkoľvek z príznakov nosa, ktoré sú vypnuté. Alternatívne, vrstva sa môže naučiť detekovať tvár tým, že identifikuje, či je prítomné jedno, alebo viac očí, úst, alebo nosov. Sieť robí robustnou, keď ktorýkoľvek z nosných, ústnych, alebo očných detektorov je vypnutý v nižšej vrstve. Znamená to, že dropout má za následok zabránenie tomu, aby sa neuróny nadmerne spoliehali na niektorý z jeho vstupov pri detekcii príznakov. Možno mu tiež pripísať produkciu neurónov, ktoré detekujú príznaky, ktoré nie sú dobré len v jednom kontexte, ale sú dobrými vlastnosťami vo viacerých kontextoch.

Významná výhoda tohto prístupu je, že je efektívny, pretože parametre medzi sieťami majú spoločné to, že sú súčasťou rodičovskej siete. Je tak možné zastupovať exponenciálne množstvo neurónových sietí v prijateľnej potrebe pamäti. Výpočtovo dropout zaberie iba dvojnásobné množstvo operácii ako rovnaká sieť bez dropout.

Zväčšenie dátovej množiny

Veľmi efektívnym nástrojom regularizácie je zväčšenie tréningovej množiny. V praxi je však tento spôsob často veľmi časovo a finančne náročný. Dataset sa preto môže rozširovať aj použitím rôznych transformácií na existujúcich tréningových dátach, ktoré napodobňujú varianty pravdepodobne sa vyskytujúce v reálnych dátach a je nepravdepodobné, že zmenia triedu vzoru. To má za následok, že sieť je robustná pre tieto typy transformácií dát. Pre obrazové dáta by to mohlo byť napríklad posunutie všetkých pixelov vľavo, vpravo, hore, alebo dole o niekoľko pixelov, zmena sýtosti farieb a jasu, malá rotácia, zväčšenie a zmenšenie, perspektívne transformácie, pridanie šumu, rozmazanie a pod.

Konfliktný tréning

V mnohých prípadoch začali neurónové siete dosahovať podobné výsledky ako ľudia na testovacích datasetoch. Je preto prirodzené sa pýtať, či tieto modely získali skutočné pochopenie týchto úloh na ľudskej úrovni. Szegedy a kol. [22] zistili, že aj siete, ktoré dosahujú pri úlohách úroveň ľudí majú takmer 100%-nú chybu na príkladoch, ktoré sú zámerne vytvorené pomocou optimalizácie, ktorá hľadá vstup *x'* blízko dátového bodu ***x*** tak, že výstup modelu je veľmi odlišný na hľadanom ***x'***. V mnohých prípadoch ***x'*** je podobné ***x*** tak, že ľudský pozorovateľ nemôže povedať, že existuje rozdiel medzi originálnym a nájdeným príkladom. Goodfellow a kol. [23] ukázali, že jednou z hlavných príčin týchto generovaných príkladov je nadmerná lineárnosť. Neurónové siete sú postavené predovšetkým z lineárnych stavebných blokov. V niektorých experimentoch funkcia, ktorú implementujú, poskytuje vysokú linearitu ako výstup. Tieto lineárne funkcie sa dajú ľahko optimalizovať. Nanešťastie hodnota lineárnej funkcie sa môže veľmi zmeniť ak má veľký počet vstupov. Ak sa zmení každý vstup o ε, potom lineárna funkcia s váhami w sa môže zmeniť až o $\varepsilon\|w\|$, čo môže byť o veľké množstvo, ak w má veľký stupeň rozmeru. Konfliktný tréning zamedzuje tejto vysokej citlivosti na lokálne linearity posilňujúc sieť, aby bola lokálne konštantná v blízkych tréningových dát [21].

2.3.4 Rozšírené inicializačné schémy

Zlá inicializácia parametrov, ktorých množstvo je rádovo v miliónoch váh môže brzdiť proces trénovania kvôli *vanishing exploding* problému [24]. Preto správna inicializácia váh je nevyhnutná na to, aby sa zlepšila konvergencia siete.

Xavier inicializácia

Glorot a Bengio v r. 2010 [25] vyhodnotili, ako sa spätne šírené gradienty a aktivácie šíria v jednotlivých vrstvách. Na základe týchto zistení navrhli normalizačnú inicializačnú schému. Pri tejto inicializácii sú počiatočné váhy odvodené z rovnomerného, alebo gausovského rozdelenia s nulovou strednou hodnotou a rozptylom.

$$\mathrm{Var(W\_Init)} = \frac{2}{n_x + n_y} \tag{15}$$

kde n_x reprezentuje vstup a n_y výstup neurónu. Neskoršie bola schéma označená ako *Xavier inicializácia* a pre jednoduchšiu implementáciu bola zjednodušená Jiaom a kol. [26], na rozdelenie so strednou hodnotou 0 a rozptylom vypočítaným ako:

$$\mathrm{Var(W\_Init)} = \frac{1}{n_x} \tag{16}$$

Bolo ukázané, že vedie k rýchlejšej konvergencii siete. Jeho hlavným obmedzením je, že jeho odvodenie je založené na predpoklade, že aktivácie sú lineárne, a preto sú nevhodné pre ReLU a PReLU aktivácie. Obídenie tohto problému viedlo k odvodeniu inicializácie váh s nulovou strednou hodnotou gausovského rozdelenia a štandardnou odchýlkou $\sqrt{2/n_l}$, kde *n* je počet spojení vchádzajúce do vrstvy a *l* je index vrstvy.

Normalizácia dávky

Ang. batch normalization (BN): pri trénovaní hlbokých sietí vzniká fenomén zvaný *internal covariate shift.* Je to spôsobené zmenami v distribúcii vstupov každej vrstvy z dôvodu zmien parametrov v predchádzajúcej vrstve. Tento jav má vážne dôsledky, medzi ktoré patrí pomalší tréning v dôsledku nižšieho learning rate, keďže je potrebné starostlivo vyberať inicializačné parametre keď sa trénujú CNN so saturovanými nelineárnymi aktiváciami. Pre zamedzenie tohto problému bola navrhnutá technika batch normalizácia [27]. Táto technika zavádza normalizačný krok, ktorý je jednoduchou nelineárnou transformáciou aplikovanou na každú aktiváciu, ktorý upravuje strednú hodnotu a rozptyl vstupnej vrstvy. Umožňuje integráciu SGD, ktorý tiež používa batch v priebehu tréningu. BN počíta strednú hodnotu

a rozptyl po každom batch-i, a nie na celom tréningovom datasete. Pre batch $B = \{x_1;..x_n\}$, s aktiváciou x a rozmerom n, stredná hodnota a rozptyl je vypočítaná nasledovne:

$$\mu_B = \frac{1}{n}\sum_{j=1}^{n} x_j \tag{17}$$

$$\sigma_B^2 = \frac{1}{n}\sum_{j=1}^{n} (x_j - \mu_B)^2 \tag{18}$$

j-tý rozmer je potom počítaný:

$$\hat{x}_j = \frac{x_j - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{19}$$

kde ε je konštanta pre aritmetickú stabilitu. Normalizované hodnoty $\hat{x}$ sú potom zmenené a posunuté pre rozšírenie reprezentácie nasledujúcou formulou:

$$Y_j = \gamma \hat{x}_j + \beta \equiv BN_{\gamma,\beta}(x_j) \tag{20}$$

kde γ a β sú učiace sa parametre. Výsledok druhej transformácie Y je šírený do ďalších vrstiev siete.

2.3.5 Hyperparametre neurónovej siete

Na tréning neurónovej siete okrem dátovej množiny, účelovej funkcie a architektúry majú vplyv rôzne ďalšie parametre, ktoré je potrebné správne nastaviť. Sú to parametre, ktoré nemajú vplyv na samotný výpočet natrénovanej siete v aktívnej dynamike, ale na jej tréning.

Najviac používané sú:

Rýchlosť učenia

DNN sú trénované použitím optimalizzátorov založených na stochlastickom poklese gradientu. Rýchlosť učenia angl. *Learning rate* (LR) je parameter, ktorý kontroluje do akej miery treba zmeniť model v reakcii na odhadovanú chybu pri každej aktualizácii váh modelu.

Zvolenie správnej hodnoty je náročné, pretože príliš malá hodnota môže mať za následok dlhý tréningový proces, zatiaľ čo príliš veľká hodnota môže mať za následok príliš rýchle naučenie neoptimálnej sady váh, alebo nestabilný tréning. Používajú sa rôzne prístupy napr. LR sa určuje dynamicky kedy s počtom narastajúcich epoch klesá hodnota.

Veľkosť dávky

Alebo po anglicky *batch size* je parameter, ktorý určuje koľko dát sa použije v jednom cykle tréningu pri zmene váh. Čím je hodnota väčšia, tým zmena váh je spôsobená väčšou vzorkou a smer gradientu pri tréningu bude viac presný. Naopak pri menšom počte odhad gradientu bude menej presný a bude viac kopírovať „lokálny“ pokles účelovej funkcie. Nevýhoda pri väčšej dávke je, že výpočet spotrebuje viac pamäti a preto pri niektorých veľkých úlohách musí byť počet malý.

Počet epoch

Určuje ako dlho sa sieť má učit adaptovať váhy podľa tréningových dát. Jedna epocha určuje, že všetky tréningové dáta boli použité raz v procese učenia. Z počtom epoch súvisí aj regularizácia skorého zastavenia kedy sa tréning zastaví po n epochách ak sa výsledok nezlepší.

Reguralizačné konštanty

V jednotlivých vrstvách ANN, alebo chybových funkciách sú pridávané konštanty, ktoré zabraňujú aby sa sieť pretrénovala.

Okrem hore uvedených existujú ďalšie hyperparametre ktorých správne nastavenie si vyžaduje expertné znalosti. S počtom menených parametrov a ich hodnôt rastie čas a počet experimentov, ktoré je potrebné vykonať aby sa trénovaný model učil. Aby tento proces nebol užívateľský náročný existujú rôzne prístupy [28], ktoré sa snažia hyperparametre nájsť. Jedným z nich je *Grid search* kde sa prehľadáva celý zadaný priestor hyperparametrov a postupne sa skúšajú rôzne parametre pri trénovaní. V *Random search* prístupe sa hodnoty hyperparametrov určujú náhodne.

2.3.6 Optimalizátory neurónovej siete

Optimalizátory sú algoritmy, ktoré sa používajú na zmenu parametrov neurónovej siete pri tréningu ako sú váhy, learning rate, momentum a iné za účelom zníženia chyby.

Gradient descent

Je to základný optimalizačný algoritmus používaný v lineárnej regresii a klasifikačných algorimtoch. Používa prvé derivácie účelovej funkcie na zmenu váh tak, aby funkcia dosahovala minimum. Jeho nevýhody sú, že môže uviaznuť v lokálnom minime a váhy sa menia až po prepočítaní celého datasetu.

Stochlastic gradient descent

Varianta Gradient Descent algoritmu, ktorá robí update parametrov nie na základe celého datasetu, ale len jeho časti tzv. *mini-batch*, ktorá je vybraná náhodne spomedzi všetkých tréningových dát. Nevýhoda tochto algoritmu je, že parametre ako learning rate, momentum sa musia určovať explicitne. Taktiež môže uviaznuť v lokálnom minime.

AdaGrad

Adaptuje learning rate podľa sumy štvorcov všetkých minulých gradientov. V počiatku je kumulatívny gradient malý, learning rate je vysoký a učenie je rýchlejšie . Metóda je vhodná na riešenie problémov s riedkym gradientom. Ako tréning prebieha akumulovaný gradient sa stáva väčší, learning rate má tendenciu byť blízky 0 čo vedie k neefektívnemu updatu parametrov. Nije je vhodný na nekonvexné problémy.

RMSprop

Upravuje AdaGrad algorimtus zmenou akumulácie gradientu na exponenciálne vážený kĺzavý priemer, t.j. zahodí históriu z dávnej minulosti. Vylepšuje neefektívny učiaci sa proces v neskorších fázach AdaGrad. Je vhodný na optimalizovanie pre nestacionárne a nekonvexné problémy. V neskorej fáze tréningu sa môže proces aktualizácie opakovať okolo lokálneho minima.

Adam

Kombinuje adaptívne metódy a momentovú metódu kde pomocou momentu prvého rádu a odhadu gradientu druhého rádu dynamicky upravuje rýchlosť učenia. Je vhodný pre väčšinu konvexných optimalizačných problémov s veľkými množstvom dát, ktoré sú reprezentované vo vysoko rozmernom priestore. V tejto práci sa používa Adam ako základný algoritmus pre tréning.

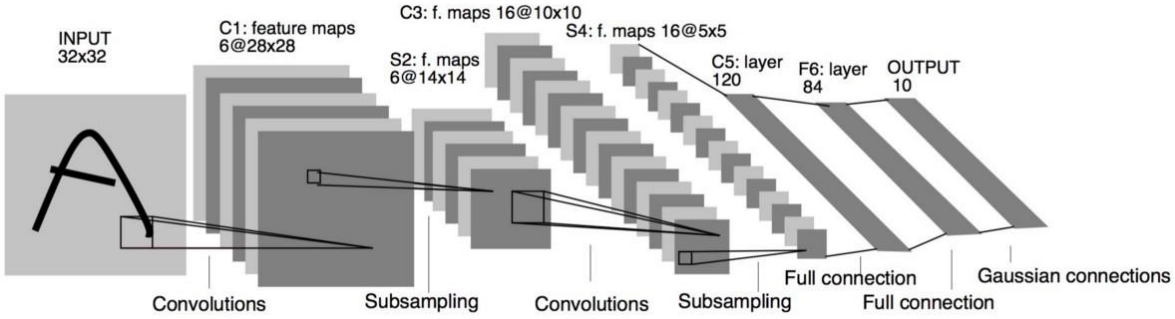
2.4 Študované architektúry neurónových sietí

S vývojom neurónových sietí sa ruka v ruke vyvíjali aj ich architektúry. V nasledujúcej kapitole stručne popíšem najdôležitejšie architektúry, ktoré boli vytvorené pre úlohy rozpoznávania obrazu.

2.4.1 Konvolučné siete

LeNet5

Lecun model [29] bol vytvorený v roku 1998 pre identifikáciu ručne písaných číslic. Tento priekopnícky model z veľkej časti predstavil konvolučnú neurónovú sieť ako ju poznáme dnes. Architektúra bola zásadná v tom, že konvolúcia s učiacimi sa parametrami je efektívny spôsob ako extrahovať podobné príznaky na rôznych pozíciách s málo parametrami čo je výhodné, keďže dôležité obrazové príznaky sú rozmiestnené v celom obraze. V tom čase neexistovali GPU, ktoré by pomohli učiť sieť a procesory boli tiež pomalé. Preto zredukovanie parametrov a zmenšenie výpočtov bolo kľúčové. To je v rozpore s použitím každého pixelu ako samostatného vstupu veľkej viacvrstvovej neurónovej siete. LeNet5 ukázal, že plné vrstvy by sa nemali používať v prvých vrstvách, pretože obrázky sú veľmi priestorovo korelované a použitie jednotlivých pixelov obrazu ako samostatných vstupných funkcií by tieto korelácie efektívne nevyužilo.

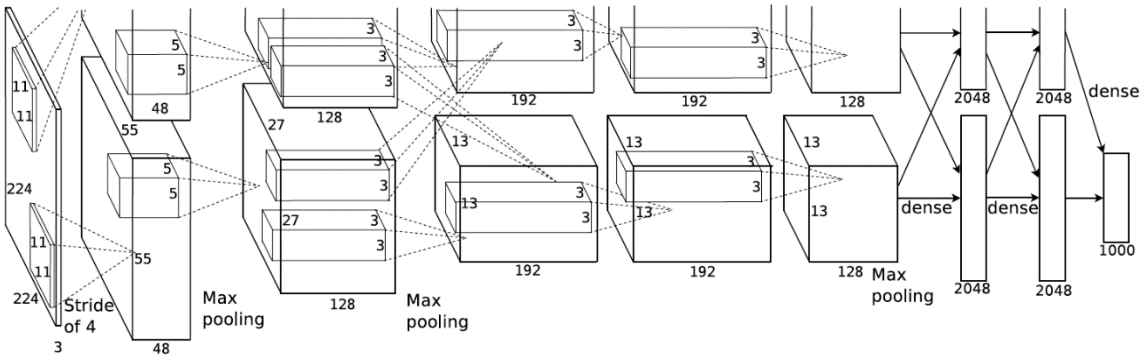


Obrázok 2.7. architektúra siete LeNet5

AlexNet

V r. 2012, Alex Krizhevsky vytvoril AlexNet [30], ktorá bola hlbšia a širšia verzia (mala cca 60 miliónov parametrov) ako LeNet, pomocou ktorej mohol sieť naučiť viac komplexné objekty a ich hierarchiu. Pri učení s veľkou rezervou vyhral súťaž ILSVRC 2012, kde na top-5 test chybe dosiahol 15.3 % oproti druhému najlepšiemu 26.2%. Prínos tejto práce bol nasledovný:

- použitie ReLU aktivačných funkcií
- používanie dropout kvôli zabráneniu pretrénovaniu
- prekrývajúci sa max pooling a nepoužitie spriemerovaného poolingu
- využitie GPU na urýchlenie trénovania



Obrázok 2.8. AlexNet architektúra

VGG

VGG siete [31] pochádzajú z výskumného tímu *Visual Graphics Group at Oxford* (odtiaľ VGG). V architektúre boli použité malé filtre o veľkosti 3x3 vo všetkých konvolučných

vrstvách a tak isto boli kombinované ako sekvencia konvolúcií. Sieť používa viacnásobné konvolučné vrstvy 3x3 na reprezentovanie zložitých funkcií namiesto použitia filtrov 9x9 alebo 11x11 ako v AlexNet. Najväčšia výhoda je, že sekvencia filtrov 3x3 môže emulovať väčšie receptívne polia (napríklad filtrov 5x5 a 7x7) pri ušetrení parametrov. Vrstvy 256x256 a 512x512 s filtrami 3x3 sú použité viac krát za sebou, aby extrahovali viac komplexné príznaky a kombináciu príznakov. Tento princíp je viac efektívny ako mať veľký klasifikátor, ktorý má konvolučné 3 vrstvy o veľkosti 512x512. To samozrejme, predstavuje obrovský počet parametrov a tiež výpočtový výkon. Tréning týchto sietí bol zložitý a musel byť rozdelený na menšie siete s postupne pridávanými vrstvami jednej po druhej (z dôvodu nedostatku spôsobov, ako regulovať model, resp. ako obmedziť veľký prehľadávaný priestor podporovaný veľkým množstvom parametrov). VGG siete sú voľne dostupné a používajú sa v niektorých úlohách ako extraktor vlastností.

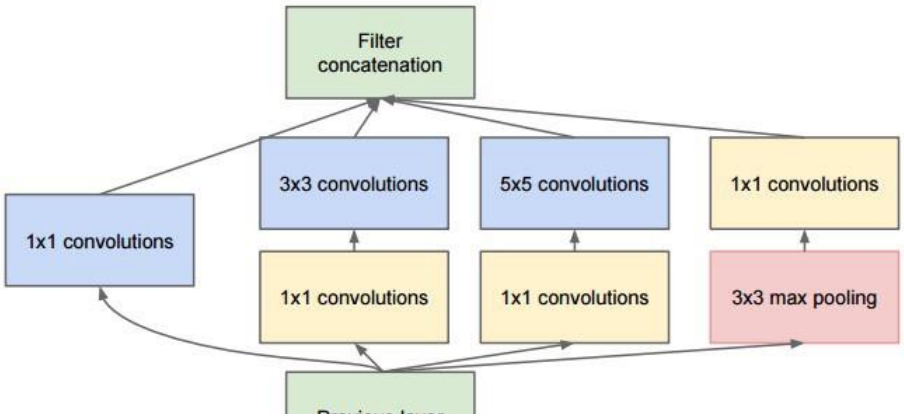
Konfigurácie					
A	**A-LRN**	**B**	**C**	**D**	**E**
11 váhových	11 váhových	13 váhových	16 váhových	16 váhových	19 váhových
vstup (224x224 RGB obrázok)					
conv3-64	conv3-64	conv3-64	conv3-64	conv3-64	conv3-64
max pooling					
conv3-128	conv3-128	conv3-128	conv3-128	conv3-128	conv3-128
max pooling					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
max pooling					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
max pooling					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
max pooling					
FC-4096					
FC-4096					
FC-1000					
softmax					

Tabuľka 2.4. Konfigurácie neurónových sietí VGG

Inception (GoogLeNet)

V roku 2014 výskumníci spoločnosti Google predstavili sieť *Inception* [32], ktorá v r. 2014 zaujala prvé miesto v súťaži ImageNet pre problémy klasifikácie a detekcie. Model sa skladá zo základnej jednotky označovanej ako *"Inception bunka"* (obrázok 2.9), v ktorej sa vykonávajú rôzne konvolúcie v rôznych mierkach a následne sa výsledky spájajú. Na úspory výpočtov sa používajú konvolúcie 1x1, ktoré majú nižší počet príznakových máp ako vstup a tak znižujú

hĺbku vstupného kanála. Pre každú bunku sa sieť učí sadu filtrov 1x1, 3x3 a 5x5, ktoré sa naučia extrahovať príznaky v rôznych mierkach. Tiež sa používa max pooling s rovnakým zarovnaním, aby sa zachovali rozmery tak, že výstup môže byť spojený. Navrhnutá sieť má až o 90% nižší počet učiacich sa parametrov ako to bolo u AlexNet. Google použil jeden z prvých modelov, ktorý predstavil myšlienku, že vrstvy CNN nemusia byť vždy ukladané za sebou postupne. Inception modul ukázal, že kreatívne štruktúrované vrstvy môžu viesť k zvýšenému výkonu a výpočtovej efektívnosti.



Obrázok 8. inception bunka [32]

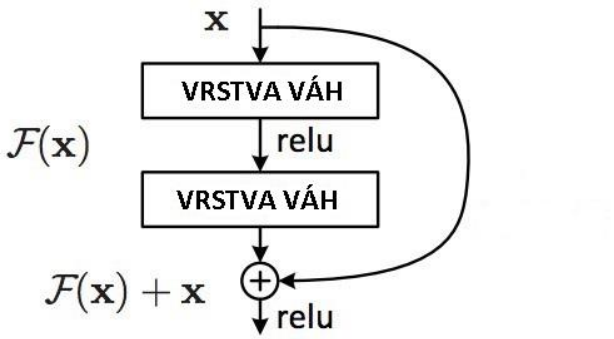
Obrázok 2.9 Inception bunka

ResNet

Je všeobecne prijateľný princíp, že hlbšie siete sú schopné učiť sa zložitejšie funkcie a reprezentácie vstupov, ktoré by mali viesť k lepším výsledkom siete. Lenže pridanie

ďalších vrstiev malo nakoniec negatívny vplyv kvôli *vanishing gradient* problému, keďže hlbšia sieť sa nedokázala naučiť tak dobre ako jej plytký variant. Tieto nedostatky sa pokúsila obísť architektúra ResNet [33], ktorá vychádza z jednoduchej myšlienky prepojiť výstup z dvoch po sebe nasledujúcich konvolučných vrstiev do ďalšej vrstvy (obrázok 2.10).

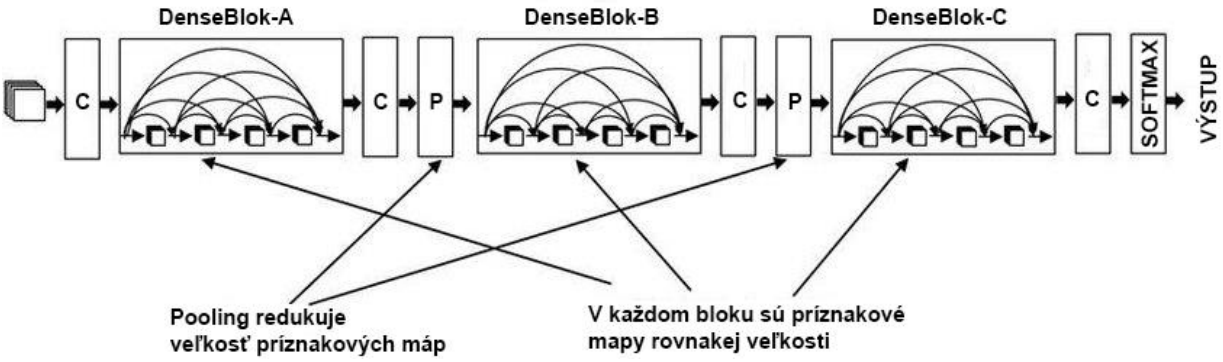
Tento blok sa učí zostatkovú funkciu vzhľadom na predchádzajúcu funkciu bloku. Môže sa to chápať ako krok zdokonaľovania, v ktorom sa učí ako nastaviť vstupnú mapu príznakov pre príznaky vyššej kvality. Je to porovnateľné s „obyčajnou“ sieťou, v ktorej sa očakáva, že každá vrstva sa učí odlišné mapy príznakov. V prípade, že nie je potrebná zmena, môžu sa vrstvy učiť upravovať váhy tak, že sú blízke nule a tak blok reprezentuje rovnakú funkciu.



Obrázok 2.10 Inception bunka

DenseNet

Husto prepojená konvolučná sieť DenseNet [34] je ďalším krokom ako zvýšiť hĺbku CNN. Ako hĺbka siete narastá “cesta” informácie zo vstupnej vrstvy k výstupnej (a pre gradient v opačnom smere) sa stáva príliš dlhá a dochádza k jej zníženiu predtým ako dosiahne opačnú stranu siete. DenseNet tento problém rieši, pretože každá vrstva má priamy prístup ku gradientu z účelovej funkcie a aj iných vrstiev. Niektoré variácie ResNet sietí navyše dokázali, že mnoho vrstiev prispieva len veľmi málo do výsledku a je ich možné zrušiť. V skutočnosti je počet parametrov pre ResNets veľký, pretože každá vrstva sa musí naučiť svoje váhy. Namiesto toho sú vrstvy DenseNets veľmi úzke (napr. 12 filtrov) a pridávajú iba malú sadu nových máp.



Obrázok 2.11. schéma DenseNet siete

Jeden z rozdielov medzi ResNet sieťou je v tom, že DenseNet nesčítava výstupné mapy príznakov z prichádzajúcih vstupných máp, ale ale ich spája do väčšieho výstupného priestoru. Preto je sieť rozdelená do *DenseBlokov* kde rozmer príznakových máp v bloku ostáva konštantný, ale počet filtrov sa medzi blokmi mení. Tieto vrstvy sa medzi blokmi nazývaju *prechodové vrstvy,* kde sa aplikuje pooling pre zmenšenie mapy príznakov, konvolúcia na zmenu filtrov a batch normalizácia. Spájanie príznakových máp zvyšuje dimenziu každej pridanej vrstvy a počet výstupných máp vrstvy definuje tzv. *growth rate,* ktorý je definovaný ako:

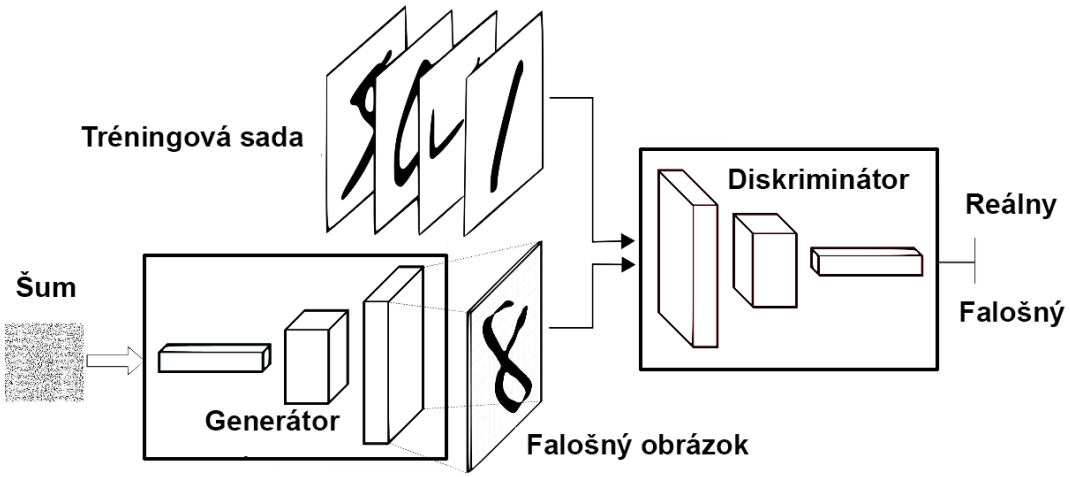
$$k_l = k_0 + k * (l - 1) \tag{21}$$

DenseNet potrebuje menšiu šírku vrstiev, pretože tak ako sú vrstvy prepojené existuje len malá redundancia pri učení príznakov. Všetky vrstvy rovnakého dense bloku zdielajú *kolektívne vedomosti.*

GAN

Generative adversarial networks [35] sú *generatívne modely*: vytvárajú nové dáta tak, že sa podobajú na tréningové dáta. Napr. môžu vytvoriť obrázky, ktoré vyzerajú ako fotky ľudských tvári i keď žiadna z týchto fotografii nepatrí reálnej osobe. GAN dosahujú vysokú úroveň reálnosti párovaním generátora, ktorý sa učí produkovať požadované výstupy s diskriminátorom, ktorý sa učí rozlišovať medzi reálnymi dátami a výstupom generátora. Generátor sa snaží oklamať diskriminátor a diskriminátor sa snaží nebyť oklamaný generátorom. Učenie sieti prebieha na v dvoch fázach. V prvej fáze sa učí diskriminátor na vygenerovaných z generátora a reálnych dátach rozlíšiť tieto množiny. Diskriminátor je v

princípe klasifikátor, ktorý rozlišuje či daný obrázok je reálny, alebo nie. Po čiastočnom natrénovaní diskriminátora sa vypne jeho učenie a začne sa učiť generátor. Generátor vygeneruje falošný obrázok, ktorý vyhodnotí diskriminátor a spätným šírením gradientu sa upravuje generátor tak aby oklamal diskriminátor. Jedným s problémom GAN sietí je, že sú náročné na veľký výpočtový výkon a tréning je nestabilný. Jedným z problémov je tzv. *Helvetica scenário* kedy pre ten istý tréningový obraz *x* je pridelené veľké množstvo vstupného šumu *p(z),* z ktorého generátor skolabuje. Vzhľadom na dobré výsledky sa siete GAN začali uplatňovať v tzv. Deep fake kedy do existujúceho obrázka, alebo videa je aplikovaná iná osoba vydávajúca sa tak za niekoho iného. GAN siete majú veľké množstvo aplikácii a modifikácii ako napr. *text-to-image, image-to-image, super resolution, 3D object generator* a dalšie. V tejto práci sú GAN siete zahrnuté z dôvodu preskúmania učenia klasifikátora ako generátora, ktorý na výstup neudáva triedu, ale rozdelenie.



Obrázok 2.12. Architektúra GAN siete

3 Cieľ práce

Pre riešenie veľkých úloh rozpoznávania obrazu je vytvorenie veľkého datasetu základnou nutnou podmienkou úspešného riešenia. Vytvorenie takéhoto datasetu predstavuje veľkú finančnú aj časovú záťaž. Buď je to z dôvodu náročného získavania dát z dôvodu absencie niektorých prípadov, alebo je potrebné zapojiť veľké množstvo ľudskej sily (dáta sú rozsiahle a je potrebná kontrola z dôvodu prevencie zavedenia chýb). Preto si táto práca dáva za svoj cieľ:

Navrhnúť postupy a metódy na zefektívnenie procesu získavania anotovaných dát pre potreby vývoja metód počítačového videnia založených na hlbokom strojovom učení.

V prvom rade je to potreba minimalizovať množstvo dát anotovaných človekom. Získavanie dát je drahá záležitosť a môže trvať mesiace úsilia vytvoriť požadovaný dataset s potrebnou kvalitou. Čím sú dáta horšej kvality (nepresnosť anotácii, alebo chybné zaradenie kategórii), tým sa trénované modely stávajú menej použiteľné. Je preto dôležité hľadať spôsoby ako zamedziť vnášaniu chýb do datasetov - či už anotovaním človekom prípadne automatizovanými postupmi anotovania, alebo hľadaním takých, ktoré prinášajú veľkú informačnú hodnotu. Tak isto čistenie a zlepšovanie kvality samotných dát sa môže robiť poloautomaticky kedy sieť sama dokáže identifikovať chybne anotované vzorky, ktoré sú v tréningových dátach ak je ich percento malé. Takýto postup sa môže iterovať čo by malo viesť k zlepšeniu kvality dát. V tejto práci za hlavnú výzvu sme si vytýčili inú organizáciu priestoru výstupných dát z neurónovej siete oproti klasickému unárnemu kódovaniu. Predpokladáme, že by to mohlo viesť k lepšiemu modelovaniu neurčitosti klasifikácie a tým identifikovaniu problematických dát či už z pohľadu chýb v datasete, alebo z pohľadu dôležitosti pre anotovanie.

4 Metódy riešenia

Pri riešení klasifikačných úloh podporovanými algoritmami hlbokého strojového učenia s učiteľom sú kritickým faktorom pre dosiahnutie požadovaného výkonu algoritmu dobre anotované dáta. Anotovanie je typicky manuálny proces kde anotátor nasleduje požadované usmernenia akým spôsobom má proces vykonávať t.j. aké metadáta anotovaným dátam priradiť, či je to označenie nejakej oblasti (napr. orámovanie) alebo nastavenie rôznych príznakov. Výsledok takto spracovaných dát je vysoko závislí od zručnosti a skúseností anotátora čo môže mať dopad na konečný výsledok riešenia. Preto vo väčšine prípadov sa dáta validujú iteračnými postupmi tak, aby potvrdenie správnosti anotácie bolo viacnásobné a až potom sa výsledok prehlási za správny. Takéto získavanie dát je pritom neefektívne z pohľadu zdrojov a času. Existujú rôzne prístupy ako začleniť čiastočne anotované alebo neanotované dáta do procesu tréningu [36,~~37,38,39~~]. Úloha vytvoriť dataset pre úlohy strojového učenia nie je nová a preto v nasledujúcih kapitolách ukážem základné prístupy a problémy, pri ktorých riešitelia narazili.

4.1 Prístupy a metódy anotovania

4.1.1 ImageNet

Je to jeden z najväčších datasetov, ktorý vznikal niekoľko rokov a autori k nemu vydali niekoľko publikácií [2,56,57]. Pre vývoj algoritmov pracujúcich s obrazom je potrebné vytvoriť dostatočnú databázu anotovaných obrázkov, ktorá poskytne dáta pre trénovanie a testovanie. Jednou z takýchto iniciatív bolo vytvorenie databázy ImageNet. Tá si za cieľ dala vytvorenie databázy obrázkov, ktorá by indexovala všetky možné objekty na svete. Je založená na štruktúre WorldNet [3], ktorá vychádza z konceptu, že slovná fráza alebo viacero slov je popísaných synonymom, tzv. synset, a uložených do štruktúry pomocou ontológie. Týchto podstatných slov je v databáze okolo 80 000 a ImageNet si kládol za úlohu poskytnúť v priemere 500 až 1000 obrázkov na každé slovo, čo sú desiatky miliónov presne anotovaných obrázkov. Následne bola k tomuto datasetu vytvorená súťaž ILSVRC [56], ktorá dala možnosť vyvíjať a testovať algorimy na rozpoznávanie obrazu. Trvala 7 ročníkov od roku 2010 do 2017, kedy 29 z 38 súťažiacich tímov mali presnosť väčšiu ako 95% [53]. V 2018 bola pridaná úloha pre detekciu 3D objektov. Vytvorenie 3D model objektu je náročnejšia úloha ako anotovať 2D obrazové dáta. Tento typ úlohy mal pomôcť pri úlohách

navigácie v robotike. Dnes už súťaž neprebieha z dôvodu že existuje konsenzus o tom že rozpoznávanie obrazu (klasifikácia a lokalizácia) je v princípe vyriešená téma a rôzne tímy sa presunuli na iné úlohy. Neskôr vznikli ďalšie datasety ako napr. COCO [54], v ktorom prebiehajú súťaže aj dnes. Tie su už zamerané na iné úlohy, ako je napríklad segmentácia obrazu. Keďže dataset sa mal používať na porovnávanie nových metód bolo na začiatku potrebné zadefinovať úlohy, na ktoré sa bude používať. Cieľom ILSVRC bolo riešenie nasledujúcich úloh:

Klasifikácia obrazu – Z pohľadu datasetu, sú dáta anotované podľa prítomnosti najvýraznejšej triedy patriacej do jednej z 1000 kategórií. Každý vzor obsahuje iba jednu pravdivú anotáciu. Pre každý vzor algoritmy produkujú zoznam najpravdepodobnejších 5 tried prítomných na obrázku. Metóda je úspešná ak sa pravdivá anotácia nachádza medzi týmito piatimi predikciami (tzv. TOP 5 metrika) Pri tvorbe tried bola snaha pokryť čo najviac anglických slov určujúcich podstatné mená. Kategórie a obrázky boli zafixované pre poskytnutie štandardizovaného testu keďže sa dataset časom zväčšoval.

Lokalizácia jedného objektu – ide o rozšírenie úlohy klasifikácie. Okrem samotnej predikcie triedy objektu je cieľom aj orámovanie jednej inštancie daného objektu v obrázku. Dáta teda pochádzaju z úlohy *klasifikácia obrazu* a boli rozšírené o orámovania objektov. Algoritmy vytvárajú zoznam objektov v obraze pomocou orámovania, ktoré definuje pozíciu a veľkosť jednej inštancie objektu danej kategórii. Vyhodnotenie prebiehalo na základe správnej predikcie kategórie a zároveň sa porovnávalo aj prekrytie predikovaného orámovania voči anotácii.

Detekcia objektu – Dáta pre túto úlohu obsahujú nové fotky zozbierané z databázy Flickr použitím hľadaných výrazov pre scény. Táto úloha mala za cieľ otestovať algoritmy, ktoré budú vedieť rozpoznávať viaceré objekty v scéne. Obrázky sú anotované pomocou orámovania určujúce pozíciu a veľkosť každej inštancii objektu danej kategórie. Trénovacia množina je obohatená (a) z úlohy *lokalizácia jedného objektu*, ktorý obsahuje anotácie pre všetky inštancie iba jedného objektu danej kategórie a (b) negatívne obrázky, o ktorých vieme že neobsahujú žiadnu kategóriu v trénovacom datasete.

Takto stanovené úlohy pre vytvorenie ImageNet datasetu si vyžiadali vytvorenie nových prístupov k anotovaniu. Aby boli modely schopné vyriešiť úlohy, ktoré si autori stanovili bolo potrebné splniť nasledujúce vlasnotsti datasetu:

Hierarchickosť – štruktúra ImageNet vyhádza z hierarchie WorldNet, čo je vlastne hierarchycký strom kde sú uložené jeho listy podľa ontológie (napr. na vrchu podstromu sú cicavce, pod ním sú domáce zvieratá, pod ním je vetva mačka a tá sa následne delí na rôzne druhy). Takto zadefinovaná štruktúra umožňuje vyberať rôzne dáta tak, aby sa ich triedy neprekrývali v úlohách a dali sa vyberať podľa rôznej úrovne abstrakcie.

Presnosť - požadovaná presnosť dát bola na všetkých úrovniach datasetu. To je náročné docieliť hlavne na nízkych úrovniach. Rozlíšiť mačku a psa nie je problém (vyššia úroveň v hierarchii). Rozlíšiť Siamskú a Barmskú mačku (nižšia úroveň v hierarchii) je aj pre človeka, ktorý nie je odborníkom, problém.

Rôznorodosť – objekty v obrázkoch by mali mať rôzne pozície, uhly pohľadu, pózy, rôzne druhy pozadí a prekážok, ktoré zakrývajú objekt. Pre potreby ohondotenia rôznorodosti autori počítali priemer obrázkov každého synset a merali bezstratovú veľkosť JPEG súboru, ktorý odráža množstvo informácí v obrázku. Na základe toho vyberali potencionálne odlišných kandidátov.

Prie vytváraní datasetu autori postupovali v nasledujúcih krokoch:

Zbieranie obrazových kandidátov – prvým krok bolo zhromaždiť dostatok kandidátov pre každý synset. Priemerná presnosť hľadania obrázkov z internetu, v dobe keď dataset bol tvorený, bola 10% [46]. Keďže cieľ bol ponúknnuť 500-1000 obrázkov na synset bolo treba zozbierať obrovské množstvo kandidátov. Obrázky sa zbierali z internetu pomocou rôznych vyhľadávačov a pre každý sysnet sa hľadala sada synoným. Výsledky vyhľadávania boli v stovkách až tisícoch. Pre zvýšenie počtu nájdení sa používali názvy z rodičovského mena synset. Napríklad ak sa hľadal výraz "chrt" a vo WordNet databáze bol záznam "Štíhli pes šlachtený v Anglicku" tak do hľadaného výrazu sa pridal "pes chrt". Pre nájdenie viacerych a rôznych kandidátov boli hľadané výrazy preložené do ďalších jazykov ako čínština, španielčina, holandština a taliančina.

Čistenie obrazových kandidátov – na to aby bolo možné získať vysoko presné dáta zozbierané v predchádzajúcom kroku je potrebné ich verifikovať človekom. Toto bolo zabezpečené pomocou online platfromy Amazon Mechanical Turk (AMT). Pomocou nej je možné zadať úlohu pre užívateľov, za ktorú dostanú zaplatenú odmenu. Pri každej úlohe anotovania boli ponúknuté sady obrázkov a definícia vzoru zo synset, vrátane odkazu na

Wikipédiu. Potom užívatelia overovali či obrázok pochádza z daného sysnet. Tak tiež bolo užívateľom povedané aby vyberali obrázky, v ktorých je objekt čiastočne zakrytý, prípadne bol na scéne "neporiadok" aby sa zabezpečila rôzna diverzita. Aj keď sa užívatelia snažili robiť robotu najlepšie ako vedeli, bolo potrebné vytvoriť systém kontroly kvality, pretože je treba dbať na dva problémy. Prvým je chybovosť ľudí a v niektorých prípadoch aj nedodržiavanie inštrukcií. Druhým je, že užívatelia nie vždy navzájom súhlasia, hlavne ak sa jedná o sysnset, ktorý sa nachádza hlbšie v strome. Riešenie pre tieto problémy je mať viacero užívateľov pre rovnaký obraz. Obraz je považovaný za správne anotovaný ak dostane väčšinu hlasov. Zistilo sa že pre rôznu obtiažnosť anotácie je potrebný aj rôzny počet hlasov. Napríklad pri "Barmskej mačke" je potrebný väčší konzenzus ako pri obraze "mačka". Preto bol vytvorený jednoduchý algoritmus, ktorý dynamicky určuje počet súhlasných anotácií. Pre každý synset bola vybratá náhodne podmnožina obrázkov a potom najmenej 10 užívateľov malo anotovať tieto obrázky. Z tejto anotácie sa potom vypočíta miera spoľahlivosti podľa zhody užívateľov, ktorá pri dosiahnutí stanoveného prahu považuje anotáciu za spoľahlivú. Pre všetkých ostatných kandidátov daného sysnet, ktorý sa má anotovať, je tento prah použitý pri rozhodovaní o správnosti anotovania. Pri niektorých synset nebolo možné nájsť dostatočnú zhodu, čo ukazuje že nie všetky môžu byť znázornené obrazom. Tento jednoduchý prístup viedol k vysokej spoľahlivosti datasetu pre rôzne podstromy na úrovni 99,7%.

Keďže dataset je konštruovaný pre rôzne úlohy, autori ukázali rôzne prístupy a výhody takéhoto datasetu. Ukázali že aj použitím neparametrických metód dokážu využívať výhody ktoré dataset má.

Neparametrické rozpoznávanie - Torralba a kol.[46] demonštrovali, že na veľkom počte obrázkov pomocou metód nejbližších susedov môžeme dosiahnuť dobrý výsledok napriek vysokej úrovne šumu. Ukázali, že dobre anotovaný dataset vykazuje lepšie výsledky na algoritmoch, špecálne ak sa využívajú hlbšie obrazové vlasnosti daného vzoru. Spustili štyri rôzne experimenty rozpoznávania objektov. Vo všetkých experimentoch testovali obrázky zo 16 kategórií medzi Caltech256 [47] a podstromom cicavcov z. Pre každú kategóriu vznikla negatívna sada, ktorá obsahovala všetky obrázky z ostatných 15 kategórií. Výsledky sú znázornené na obrázku 4.1. jednotlivé experimenty sa pokúsim popísať a priblížiť v čom má dataset výhody k dovtedy používaným a je potrebné na ne brať ohľad pri vytváraní datasetu.

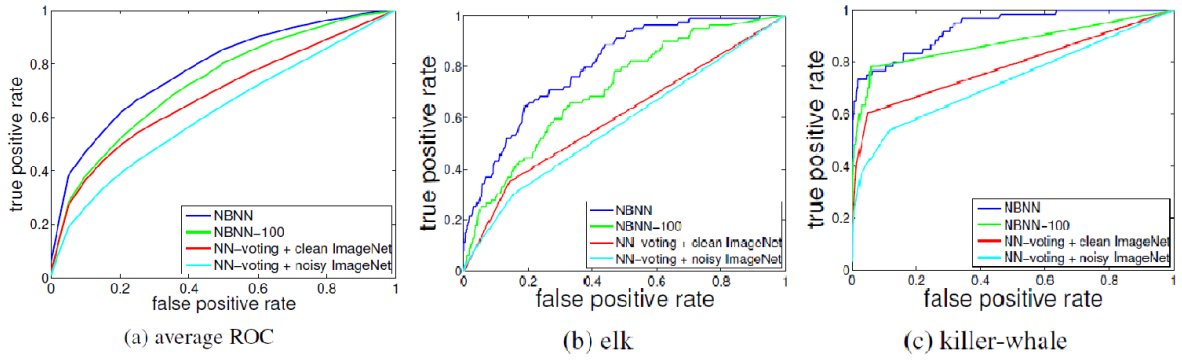
Najbližší Sused hlasovanie + zašumený ImageNet – Replikovali experiment z [48]. Aby imitovali TinyImage [46] dataset (obrázky zozbierané z vyhladávačov bez ľudského čistenia), použili originálnych kandidátov pre každý sysnset a zmenšili ich rozlíšenie na 32x32. Pre každý obrázok vybrali pomocou algoritmu najbližších susedov 100 kandidátov kde vzdialenosť merali ako SSD (sum of square differencies) z podstromu cicavcov. Potom vykonali klasifikáciu na základe väčšinového hlasovania (počet najbližších susedov) vnútri stromu danej kategórie.

Najbližší Sused hlasovanie + čistý ImageNet – Pustili ten istý experiment ako popisovaný vyššie s tým rozdielom že dataset bol očistený ľudmi. Tento experiment ukazuje že mať presnejšie anotované dáta vedie k lepším výsledkov klasifikácie.

NBNN – Naivná bayesovská metóda najbližšieho suseda navrhnutá v [49] ktorá bola vybraná aby podčiarkla užitočnosť mať dataset anotovaný v plnom rozlíšení (objekty v obraze majú minimálny podiel šumu z pozadia). NBNN používa sadu filtrov reprezentujúce obrázky. Ako deskriptor bol použitý SIFT [50]. Pre daný obrázok Q s deskriptormi $\{di\}, i = 1,..,M,$ a pre každý objekt triedy C sa počíta vzdialenosť $D_C = \sum_{i=1}^{M} \left\| d_i - d_i^C \right\|^2$ kde d_i^C je najbližší sused z d_i zo všetkých obrazových deskriptorov v triede C. Potom sa všetky triedy zoradili podľa D_C a určili klasifikačné skóre ako minimálne skóre cieľovej triedy a jej podtried. Výsledky ukazujú že NBNN dáva podstatnejšie lepšie výsledky domonštrujúce výhody využitia viac sofistikovaných vlasností z obrazu vďaka anotáciám v plnom rozlíšení.

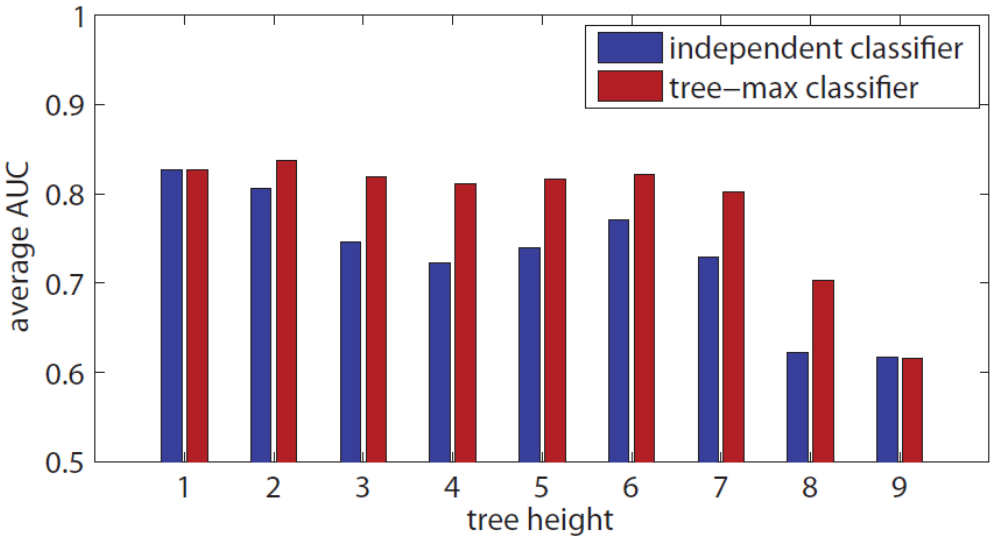
NBNN-100 – Pustený NBNN experiment, ale limit pre počet obrázkov na kategóriu bol 100. Výsledky potvrdzujú zistenia z [46]. Úspešnosť môže byť signifikantne zvýšená rozšírením datasetu. Keď sa pozrieme na výsledky obrázka.4.1 tak tento prístup prekonáva prístup „Najbližší sused hlasovanie“, aj napriek tomu že táto metóda má dáta z celého datasetu. Ukazuje to aj výhody mať dataset vytvorený tak že sa dá využiť celé rozlíšenie obrázka ktoré popisuje obrazové vlastnosti objektu využívané na rozpoznanie kategórie.

Aj keď použité metódy hore uvedených experimentov neboli modely neurónových sietí, demonštrovali že využitie datasetu ktorý je presný (má málo zle anotovaných dát), má plné rozlíšenie obrázka (tzn. že objekt ktorý reprezentuje je zachytený maximálnym počtom pixelov a šum v pozadí je minimálny) dokáže zlepšiť výsledné riešenie. Okrem toho existujú metódy ktoré vedia využívať ďalšiu vlasnosť a tou je štruktúra datasetu.



Obrázok 4.1. Zdroj [2]. ROC krivky pre experimenty neparametrického rozpoznávania. a) priemerný ROC výsledok so 16 objektov medzi Caltech256 a stromom cicavcov. b),c) ROC krivky pre losa a zabíjačskej veľryby.

Klasifikácia pomocou stromovej štruktúry – Autori demonštrovali na jednoduchej úlohe klasifikácie objektov metódu, ktorú nazvali „*tree-max*“ klasifikátor. Pre každý nod stromu je



Obrázok 4.2. Zdroj [2]. Ukazuje AUC (oblasť pod ROC krivkou) využitie klasifikátora ktorý využíva stromovú štruktúru datasetu na rôznych úrovniach stromu, oproti nezávislému klasifikátoru.

vytvorený klasifikátor. Klasifikačné skóre sa nepočíta len z nodu kategórie, ako napr. „pes“, ale aj z jeho potomkov, ako napr. „nemecký vlčiak“ ,“nemecká doga“ . Klasifikačné skóre je

vypočítané ako maximum zo všetkých výsledkov klasifikátorov takéhoto podstromu. Obrázok 4.2 ilustruje výsledky pre podstrom cicavcov. Na trénovanie autori použili klasifikátor založený na AdaBoost [51], kde rozdelili dataset v pomere 90% tréning a 10% testovanie. Vytvorili aj negatívnu sadu obrázkov, kde náhodne vybrali 10 obrázkov z každého synset. Pri tréningu používali pozitívnu sadu z podstromu, ktorý trénovali, a negatívnu sadu, z ktorej vylúčili rodičov a deti podstromu z pozitívnej sady. Na obrázku 4.2 je vidieť rôzne výsledky AUC (oblasť pod ROC krivkou) pre daný synset na rôznych úrovniach stromu porovnaný s nezávislým klasifikátorom ktorý nevyužíva stromovú štruktúru datasetu ImageNet.

Graf ukazujezuje že je jednoduchšie klasifikovať vo väčšej hĺbke stromu ako je napr. "Polárny medveď" oproti abstraktnejším kategóriám ako napr. "cicavec". Toto je pravdepodobne možné kvôli silnejšej vizuálnej väzbe v blízkosti listov uzlu stromu. Takmer na všetkých úrovniach je úspešnosť „*tree-max* klasifikátoru vyšia ako nezávislý klasifikátor. Aj tento výsledok ukazuje, že jednoduchý spôsob využitia hierarchie datasetu môže poskytnúť podstatné zlepšenie pre úlohu klasifikácie obrázkov.

Autori pri vytváraní datasetu dbali na to aby obrázky v datasete pre jednu kategóriu boli v čo najrôznejších variáciách či sú to rôzne uhly alebo situácie. Nasledujúci experiment ukazuje zastúpenie rôznorodosti objektov datasetu.

Automatická lokalizácia objektov – V začiatkoch vytvárania datasetu neboli do obrázkov pridané metadáta pre lokalizačné úlohy, ako je napríklad orámovanie pre úlohy lokalizácie objektov. Tieto dáta umožňujú využitie datasetu pre lokalizačné úlohy, prípadne pre úlohy detekcie vo forme testovacej množiny. Autori prezentovali výsledky lokalizácie na 22 kategóriách z rôznej hĺbky stromu WordNet hierarchie.

Výsledky ukazujú aké rôznorodé vzory v jednotlivých kategóriach sú. Použili neparametrický model popísaný v [52] na naučenie vizuálnej reprezentácie objektov oproti globálnej triede pozadia. Anotovali 100 obrázkov v 22 rônych kategóriách pre podstrom *cicavce* a *vozidlá*. Samotný výsledok lokalizácie je z pohľadu vtedajších a dnešných metód minoritný. Čo stojí za ukážku je obrázok 4.3**.** Autori spustili K-Means zhlukovací algoritmus na detekované orámovanie po tom čo ich konvertovali do odtieni šedej a zmenšili ich na rozlíšenie 32x32 pixelov. Takto vytvorili skupiny podobných vzorov. Všetky priemerné obrázky sú vytvorené

približne zo 40 vzorov. Je ťažko identifikovať objekt v spriemerovanom obrázku z dôvodu rôznorodosti datasetu, ale priemerné obrázky z jedného zhluku ukazujú pózi a a pohľady.

Ako je vidno anotácia na komplexnej úrovni či je to rôznorodosť, čistota datasetu alebo štruktúra, pomáha zlepšiť výsledky algoritmov detekcie a klasifikácie. Ukázalo sa že použitie anotovaných objektov ktoré využívajú plné rozlíšenie obrázka pomáha využívať hlbšie obrazové vlastnoti klasifikovaného objektu. Je preto potrebné pracovať aj s takýmto typom dát ktoré sú využívané pri detekcie objektov aj keď sa táto práca nezameriava priamo na detekciu. V nasledujúcej časti by som chcel ukázať ďalšie prístupy anotovania dát ktoré stavajú na úlohách detekcie.

Obrázok 4.3. Zdroj [2]. V ľavo spriemerované obrázky a detekované orámovanie z kategórie “slon” a “neviditeľné lietadlo” . V Pravo spriemerované obrázky a príklady z troch zhlukov po K-MEANS algoritme. Rôzne pohľady a pózi sa vynárajú v kategórii “slon”. Prvý riadok ukazuje slonov z bočného, z predného pohladu a v profile. Ďalší zhluk lietadiel ukazuje lietadlá na zemi.

4.1.2 Nepotrebujeme Orámovanie

V tejto kapitole priblížim ďalšie prístupy anotácie dát z článku „*We don't need no bounding-boxes: Training object class detectors using only human verification"*[55].

Anotácia orámovania je únavná, časovo náročná a drahá práca. Napríklad anotovanie ILSVRC požadovala priemerne 42 sekúnd na vytvorenie anotácie pri použití AMT. Navyše bol potrebný špecializovaný SW, ktorý zefektívnil proces vytvárania orámovania. Za účelom zredukovania ceny orámovania sa vývoj sústredil na dve stratégie. Prvá, je učenie čiastočne využívajúce učiteľa kde napr. učenie rozpoznávania je iba na trénované na triedach, ktoré sú prítomné na obrázku bez bližšej špecifikácie. Lenže takýto model má iba polovičnú presnosť ako keby sme ho učili z orámovaných vzorov [58,59,60,61,64]. Druhá stratégia je aktívne učenie, kedy pre vytvorenie vzorov trénovania potrebujeme od človeka vytvoriť rám okolo objektu. Táto stratégia produkuje vysokú kvalitu detektorov, ale stále požaduje od ľudí kresliť rám okolo objektu a je tak limitovaná časom anotácie [62,63]. V tomto článku autori ukázali novú schému ako učiť detektor iba tak, že automaticky vytvára návrhy anotácii a ľudská interakcia je potrebná iba na overenie orámovania či je správne alebo nie. Odpoveď na túto otázku zaberie ďaleko menej času než nakreslenie orámovania. Schéma je založená na iteratívnej úprave objektov detekcie. V každej iterácii je použitý kontrolný signál od anotátora dvoma spôsobmi. Prvý spôsob spočíva v menení objektu iba vtedy keď je informácia od anotátora kladná. Tým pádom overený vzor už nemusí byť zaradený do ďalšej iterácie. Druhý spôsob je, že orámovanie, ktoré je označené za chybné, môže byť použité na určenie priestoru kde sa objekt nenachádza a zredukovať tak priestor, ktorý sa má prehľadávať. Oba tieto spôsoby pomáhajú zvýšiť efektivitu hľadania objektov v ostávajúcich obrázkoch a minimalizujú tak ľudské úsilie, pretože eliminujú potrebu kresliť orámovanie. Nasledujú súvisiace riešenia, ktoré pristupovali k tomuto problému inak.

Weakly-supervised object localization (WSOL) - Mnoho predchádzajúcich techník sa pokúšalo učiť detektory objektov s čiastočnou informáciou (weakly supervised). Napr. tréningové obrázky, ktoré obsahujú príklady istých tried objektov, ale nie ich lokáciu. Úloha je lokalizovať tieto objekty v tréningových obrázkoch a učiť detektor lokalizovať inštancie v nových obrázkoch. Práca na WSOL [58,59,64] ukázala značný progres, hlavne z dôvodu použitia konvolučných neurónových sietí , ktoré značne vylepšujú úlohy vizuálneho rozpoznávania. Avšak, učenie detektora bez existencie orámovania je náročné a výkon je stále pomerne nízky v porovnaní s metódami založenými na učení s učiteľom metódami

(typicky okolo polovice mAP toho istého detekčného modelu trénovaného na tých istých obrázkoch ale z manuálnym orámovaním anotácie [58,59,64]).

Často WSOL je konceptualizované ako MUIL (Multiple Instance Learning). Obrázky sú považované ako sady rámov (inštancie). Negatívny obrázok obsahuje len negatívne príklady. Pozitívny obrázok obsahuje prinajmenšom jeden pozitívny príklad, zamiešaný väčšinou s negatívnymi objektmi. Cieľom je nájsť skutočné pozitívne príklady, z ktorých sa klasifikátor bude učiť. Toto sa typicky deje iteratívnym striedaním medzi (A) znovu trénovaním objektového detektoru na dátach s aktuálnou selekciou pozitívnych inštancií a (B) znovu lokalizovaním rámov inštancií v pozitívnych obrázkoch použitím súčasného objektového detektora. Autormi navrhovaná schéma taktiež obsahuje tieto kroky. Avšak navrhli ľudský verifikačný krok, ktorého signál je použitý v kroku (A) aj (B) a fundamentálne ich mení. Výsledná schéma vedie k podstatne lepším objektovým detektorom s malým pridaným anotačného úsilia.

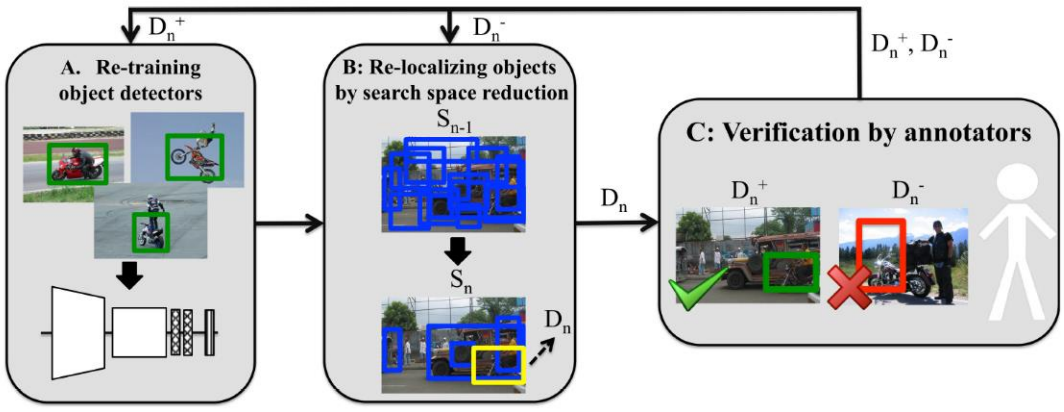
Človek v slučke (human in the loop) – Prístupy spolupráce ľudí a strojov boli úspešne použité v úlohách, ktoré sú aktuálne príliž náročné aby boli riešené len počítačovým videním, ako napr. jemnozrnné rozpoznávanie (fine grained) [65, 66], zhlukovanie s čiastočným dohľadom (semisupervised clustering) [67] a na atribútoch založená klasifikácia obrazu[65]. Tieto práce kombinujú výstupy predtrénovaných počítačových modelov na novom testovacom obrázku, s ľudským úsilím na to aby vyriešili úlohu. V doméne detekcie objekov, Russakovsky a kol. [68] navrhli takú schému, aby plne detekovala všetky objekty v obrázkoch komplexných scén. Avšak je dôležité poznamenať že ich detektory sú predtrénované na orámovaných objektoch z datasetu ILSVRC [56], nakoľko ich cieľom nie je vytvoriť tréningovú schému ktorá znižuje prácu na anotovaní.

Aktívne učenie - schémy aktívneho učenia iteratívne trénujú modely a v iteráciách požadujú ľudskú anotáciu pre sadu dát, ktorá je najviac zaujímavá pre učiaci sa algoritmus. Predchádzajúce práce o aktívnom učení boli zameraná na klasifikáciu obrazu [67,68], freeform region labelling [69,70]. Niekoľko prác navrhlo aktívne učiace schémy špeciálne pre trénovanie detektorov [63]. Vijayanarasimhan a Grauman [62] navrhli prístup, pri ktorom trénigové obrázky nepochádzajú z preddefinovaného datasetu, ale sú získané z WWW. Potom sú anotátori požiadaný aby vytvorili orámovanie okolo cieľových objektov (pre približne 1/3 tréningových vzorov [62]). Yao a kol.[63] navrhujú ručne opravovať orámovania detekované vo videu. Zatiaľ čo oba prístupy [62,63] produkujú vysokú kvalitu

detektorov, dosahujú len malé úspechy v šetrení anotačného času lebo kreslenie alebo oprava orámovania objektov je časovo náročná úloha. Naproti tomu navrhovaná schéma autorov článku [55] požaduje aby anotátori len overovali orámovanie, nie aby ich kreslili. Toto vedie k podstatnejším úsporám anotačného času.

Iné spôsoby redukcie anotačného úsilia – Niektorý autori sa pokúšali učiť detekciu objektu z videí, kde priestorová a časová súdržnosť snímok uľahčuje objektovú lokalizáciu [71]. Alternatívou je transferové učenie (transfer learning), kde učenie modelu pre novú triedu je podporované príbuznými triedami [72]. Hoffman a kol. [73] navrhli algoritmus, ktorý transformuje obrázkový klasifikátor na obrázkový detektor bez orámovaných dát použitím doménovej adaptácie. Ďalšie typy dát, ako text z webových stránok alebo novín [74] a sledovanie pohybu očí[75], boli tiež použité ako slabý anotačný signál pre trénovanie detektorov objektov.

Metóda "nepotrebujeme orámovanie" – Na začiatku je daný dataset obrázkov anotovaný triedami na úrovni celého obrázka. Cieľom je získať objekty anotované orámovaním a natrénovať dobré detektory za minimalizovania ľudského anotačného úsilia. Autori navrhli schému kde anotátori potrebujú skontrolovať orámovanie ktoré je automaticky vytvorené učiacim sa detektorom. Ich schéma iteratívne strieda medzi (A) znovu natrénovaním objektových detektorov, (B) znovu lokalizovaním objektov v tréningových obrazoch a (C) žiadaním anotátorov o kontrolu (obrázok 4.4). Dôležité je že používame výsledok overenia aby nám pomohol znovu trénovať a znovu lokalizovať objekty.



Obrázok 4.4. Zdroj [55]. Metóda navrhovaná autormi iteruje medzi (A) znovu naučením detektorov, (B) znovu lokalizovaním objektov a (C) žiadaním anotáorov o kontrolu. Výsledný signál z (C) je použitý v oboch krokoch (A) aj (B)

Nech I_n je sada obrázkov pre ktoré ešte nemáme pozitívne overné orámovanie na iterácií *n*. Nech S_n je zodpovedajúca sada možných objektových lokácií. . Na začiatku I_0 je kompletná tréningová množina a S_0 je množina navrhovaných objektov extrahovaných z obrázkov (bola použitá metóda EdgeBoxes[76]. Je to prístup ako generovať orámovanie objektov priamo z hrán obrázka). Na iterácii *n* máme sadu automaticky detekovaných orámovaní D_n ktoré sú dané anotátorom aby boli skontrolované. Detekcie ktoré sú posúdené ako správne D_n^+ sú použité pre znovunatrénovanie detektora (A) v ďalšej iterácii. Výsledok kontrolovania je tiež použitý na redukovanie priestoru S_{n+1} ktorý sa prehľadávania pre znovulokalizáciu (B).

Overenie anotátormi- v tejto fáze sú anotátori požiadaný aby overili automaticky generované detekcie D_n na iterácií *n*. Z tohto dôvodu sa preskúmali dve stratégie: jednoduchá *Áno-Nie* overenie a komplexnejšie v ktorom anotátori sú požiadaný aby kategorizovali typ chyby.

Áno-Nie overovanie - v tejto úlohe je anotátorom ukázaná detekcia l_d a označenie triedy. Sú inštruovaný aby odpovedali *Áno* ak detekcia správne lokalizuje objekt danej triedy a *Nie* v opačnom prípade. Toto delí sadu detekcií D_n na D_n^+ a D_n^-. Autori definovali *"správnu lokalizáciu"* založenú na štandardnej PASCAL IoU kritériu (Intersection over union) [77]. Nech l_d je detekovaný objekt orámovania a l_{gt} nech je aktuálne orámovanie objektu ktoré nie je ukázané anotátorovi). Nech $IoU(l_a, l_b) = |l_a \cap l_b|/|l_a \cup l_b|$ kde $|.|$ označuje oblasť. Ak $IoU(l_a, l_b) \geq 0{,}5$, detekované orámovanie by malo byť považované za správne a anotátor by mal odpovedať *Áno*. Intuitívne *Áno-Nie* overovanie je relatívne jednoduchá úloha ktorá by mala viesť k zrýchleniu anotácie objektov.

Áno, Čiastočne, Obsahujúci, Zmiešaný, Chýbajúci overovanie - v tejto úlohe požiadali anotátorov označiť detekciu objektu l_d ako *Áno* (správna), *Čiastočná* , *Obsahujúca*, *Zmiešaná* alebo *Chýbajúca*. *Áno* je definované ako $IoU(l_{gt}, l_d) \geq 0{,}5$. Pre nesprávne detekcie sú anotátori požiadaní diagnostikovať chybu buď ako *Čiastkovú* ak obsahuje časť cieleného objektu a žiadne pozadie; *Obsahujúcu* ak obsahuje celý objekt a nejaké pozadie; *Zmiešanú* ak obsahuje časť objektu a nejaké pozadie; *Chýbajúcu* ak objekt kompletne chýbal. Tento overovací krok delí D_n na D_n^+ a D_n^- a D_n^{ypcmm-}. Intuitívne určenie typy chyby je náročnejšie vedúce k dlhším anotačným časom ale tiež prináša viac informácii ktoré môžme použiť v ďalších krokoch.

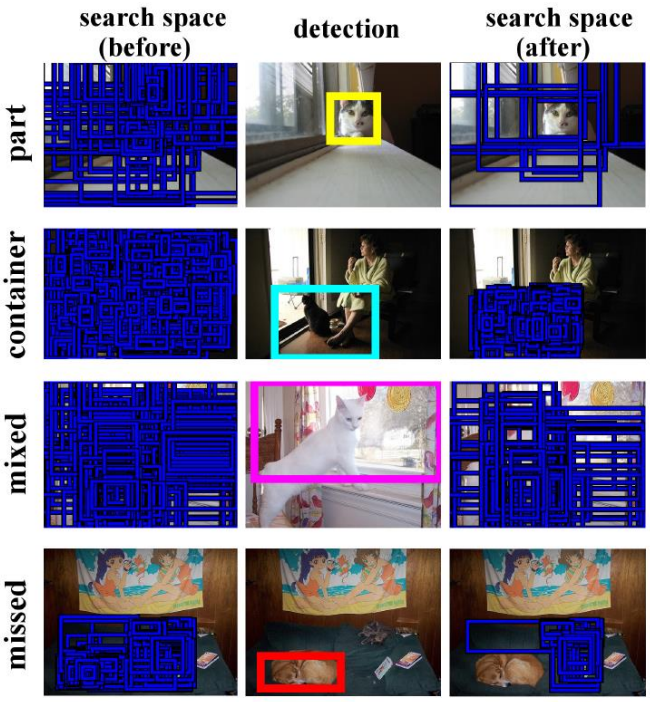
Znovutrénovanie objektových detektorov - v tomto kroku znovu trénujeme detektor. Po overovacom kroku vieme že D_n^+ obsahuje dobre lokalizované objekty zatiaľ čo D_n^- a D_n^{ypcmm-} neobsahujú. Teda sa trénuje len s použitím orámovania $D_1^+ \cup \ldots \cup D_n^+$ ktoré boli pozitívne označené v predchádzajúcej iterácii. Pre získanie tréningových vzorov pozadia sa vybrali návrhy ktoré majú IoU v rozsahu [0-0,5) s pozitívne overenými orámovaniami.

Znovulokalizácia objektov pri redukovaní prehľadávacieho priestoru - v tomto kroku sa znovu lokalizujú objekty v tréningovćyh obrázkoch. Pre každý obrázok sa aplikuje aktuálny objektový detektor na vyhodnotenie navrhovaných objektov v ňom, a vyberú sa návrhy s najvyším skóre ako nová detekcia pre tento obrázok. Dôležité je že nehodnotíme všetky návrhy S_0, ale namiesto toho používame výsledok overovania ktorým redukujeme prehľadávanie priestoru odstránením návrhov. Pozitívne overené detekcie D_n^+ sú správne podľa definície a teda ich obrázky nepotrebujú byť v nasledových iteráciách, v relokalizačnom kroku a ani v overovacom kroku. Pre negatívne overené detekcie autori redukovali prehľadávaný priestor podľa verifikačnej stratégií ako je popísané nižšie.

***Áno-Nie* overovanie –** V prípade že anotátor hodnotí detekciu ako zlú (D_n^-) môže sa môže návrh vyradiť z prehľadávacieho priestoru. To vedie k auktualizovaniu prehladávacieho priestoru S_{n+1} kde jeden návrh bol odstránený z každého obrázku s nesprávnou detekciou. Avšak dá sa ešte lepšie využiť výsledok overenia. Keďže nesprávne detekcie majú $IoU < 0.5$ so správnym orámovaním, dajú sa odstrániť všetky návrhy s $IoU \geq 0.5$. Táto agresívnejšia forma odstraňovania môže viesť k odstráneniu niektorých správnych návrhov podľa IoU kritéria, avšak neodstráni to najlepší návrh. Dôležité je že táto stratégia eliminuje oblasti prehľadávania ktoré sú vysoko hodnotené lokácie v ktorých je nepravdepodobné že obsahujú objekt.

***Áno, Čiastočne, Obsahujúci, Zmiešaný, Chýbajúce* overovanie -** V prípade kedy anotátori kategorizujú nesprávne detekcie ako *Čiastočne, Obsahuje, Zmmiešané, Chýbajúce,* môžeme použiť tento typ chyby aby sa dosiahla väčšiu redukcia prehľadavaného priestoru. Závisiac na type chyby eliminujeme rozličné návrhy (obrázok 4.5): *Čiastočná* – eliminuje všetky návrhy ktoré neobsahujú detekciu; *Obsahujúca*: elimunjue všetky návrhy ktoré nie sú vo vnútri detekcie; *Zmiešaná*: eliminuje všetky návrhy ktoré nie sú vo vnútri detekcie alebo ju neobsahujú alebo majú s ňou nulový IoU, alebo s ním majú s $IoU \geq 0.5$; *Chýbajúca*: eliminujú všetky návrhy ktoré majú nenulovú IoU s detekciou. Aby sa presne určilo čo je

„*vo vnútri*“ a „*obsahujúci*“ autori uviedli „*Prienik cez A*“ (intersection-over-A) meranie: $IoA(l_a, l_b) = |l_a \cap l_b|/|l_a|$. Všimnite si že $IoA(l_{gt}, l_d) = 1$, ak detekcia l_d obsahuje skutočný objekt l_{gt}, tak $IoA(l_d, l_{gt}) = 1$, ak l_d pokrýva časť l_{gt}. V praxi ak l_d je ohodnotené anotátorom ako *Čiastočne*, eleminujú sa všetky návrhy l_s s $IoA(l_d, l_s) \leq 0.9$. Podobne ak l_d je považované za *Obsahujúci*, eliminujeme všetky návrhy l_s s $IoA(l_s, l_d) \leq 0.9$. Takto zmenšenie vyhľadávacieho priestoru výrazne uľahčuje opätovnú lokalizáciu.



Obrázok 4.5. Zdroj [55]. Vizualizácia redukcie prehľadávaného priestoru podľa YPXMM overovania na triedach mačky part (Čiastočne), container (Obsahujúci), mixed (Zmiešaný), and missed (Chýbajú). Na poslednom riadku redukciu vyhľadávacieho priestoru riadi proces relokalizácie smerom k malej mačke na vpravo a ďalej od psa vľavo.

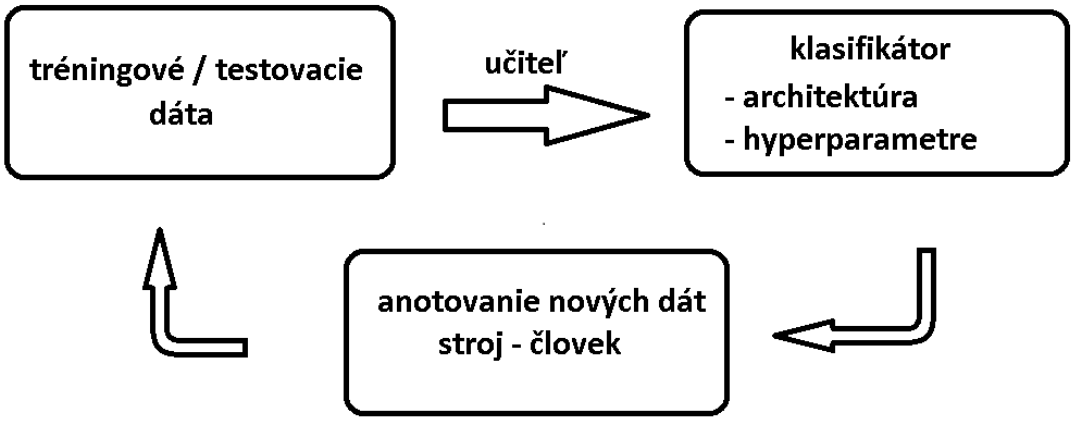
Inicializácia podľa MIL (multiple instance learning) – Autori vykonali MIL pre slabo naučenú lokalizáciu objektov [58,76] , aby získali počiatočnú sadu detekcií D_0. Začínali s trénovacími obrázkami I_0 a sadou navrhovaných objektov S_0 extrahovných použitím EdgeBoxes[76]. Nasledujúc [58] extrahovali vlastnosti z posledných vrstiev CNN nad ktorými trénovali SVM. Iterovali medzi (A) znovutrénovaním detektora objektu a (B) znovulokalizovaním objektov v tréningových obrázkoch. Prestali keď za sebou idúce

lokalizačné kroky poskytovali tie isté detekcie čo sa typicky dialo v rámci 10 iterácií. Tieto detekcie sa stali D_0. V úplne prvej iterácií trénovali klasifikátor na všetkých obrázkoch ktoré použili ako pozitívne tréningové príklady [60]. Aplikovali dve zlepšenia na štandardnú MIL schému. Po prvé, vo vysokom dimenzionálnom priestore vlastností SVM klasifikátor môže relatívne ľahko separovať akékoľvek pozitívne príklady od negatívnych inštancií, čo znamená že najpozitívnejšie inštancie sú ďaleko od rozhodnutia nadroviny. Teda tie isté pozitívne tréningové príklady použité pre znovutrénovanie (A) sú často znovulokalizované v (B) vedúc k predčasnému *locked-in* správaniu. Aby tomu prdišli Cinbis a kol.[78] uviedli viac žložkový (viac žložkový) MIL: podobné ako krížová validácia kde dataset je rozdelený na 10 podčastí a relokalizácia na každej z týchto podčastí je vykonávaná použitím detektorov trénovaných na spojení ostatných podčastí. Po druhé ako [60,61,72] kombinovali skóre detektora objektov s všeobecným meraním "objektovosti" ktoré meria nakoľko je pravdepodobné, že návrh tesne ohraničuje objekt nejakej triedy (napr. vták, auto, ovca) ako protiklad k pozadiu (obloha, voda, tráva). V tejto práci použili meranie "objektovosti" [76].

4.2 Metóda dolovania tried v čiastočne anotovaných dátach

Pri riešení klasifikačných úloh okrem správnej zvolenej architektúry a algoritmu je veľmi dôležitou častou získanie potrebného množstva dát ale aj ich kvality (t.j mať správne nastavené atribúty podľa ktorých chceme klasifikovať.) Existujú situácie kedy zastúpenie niektorých typov dát je veľmi málo lebo sa reálne nevyskytujú a je potrebné úrehladať obrovské množstvo kandidátov, prípadne počiatočné dáta pri riešení úlohy nie sú dostatočné. Získanie nových dát môžeme realizovať buď anotovaním pomocou človeka alebo vytvoriť metódu ktorá pomáha človeku tieto dáta anotovať a to buď automatickým alebo poloautomatickým spôsobom. Pri poloautomatickom spôsobe predpokladáme že anotátor dostane vopred vyplnené atribúty dát neurónovou sieťou a ten výsledky potvrdí prípadne upraví. V taktomto prípadne je efektivita anotovania vyšia. Ostatné dáta ktoré nie sú v problémové a výstup pokladáme za dostatočne dôveryhodný nie je potrebný zásah človeka a anotovanie sa stáva rýchlejšie. Vtedy môžeme hovoriť o automatickom anotovaní. Otázka je kedy si môže byť algoritmus natoľko istý aby nemusel požadovať potvrdenie od človeka? Existujú rôzne prístupy pomocou ktorých by sa dala takáto podmienka vyhodnotiť. My sme sa rozhodli použiť binárne kódovanie (viď kapitola 4.6) ktoré nám na výstupe vytvára nové rozdelenie priestoru a na základe tejto informácie sa pokúšame vyhodnotiť či danný výstup je dôveryhodný alebo nie. Pri počiatočnom anotovaní by mal mať algoritmus prísne kritéria aby

nedochádzalo k zanášaniu chýb do tréningovej množiny. U takto nastavených kritériách je možné že dáta z ktorých sa bude dolovať budú akceptované len v malej miere a pre zníženie interakcie s človekom je lepšie pustiť algoritmus niekoľko krát, ako naraz nechať anotátora opravovať veľké množstvo dát. Takýto iteratívny prístup je preto základom celého dolovania dát. Ako vidno na obrázku 4.6 schéma je jednoduchá. V Prvom kroku vychádzame z počiatočných dát kedy natrénujeme CNN. Takto natrénovanú sieť potom môžeme použiť na neanotovaných alebo na čiastočne anotovaných dátach ktorým môže chýbať atribút, prípadne kedy chceme overiť správnosť dát z predchádzajúceho anotovania. Výstup potom podľa dôveryhodnosti ponúkneme anotátorovi alebo ho rovno zaradíme do trénovacej množiny.



Obrázok 4.6 Schéma iteračného postupu

Ako je asi zrejmé celý proces takéhoto iteratívneho prístupu závisí na tom ako dobre dokážeme sieť natrénovať. Pri tréningu nemusíme trénovať len jednu sieť, ale môžme meniť rôzne hyperparametre ktoré maju vpliv na celkový tréning a tým pádom aj celkový výsledok. Hľadanie takejto siete si vyžaduje spustenie väčšieho počtu experimentov. Preto sme vytvorili jednoduchý prístup (viď kapitola 4.3) ako s pomedzi množstva experimentov vybrať v počiatočných fázach iba jeden a ten dotrénovať do konca a tak zlepšiť výsledok iterácie dolovania dát.

4.3 Jednoduchý prístup zlepšenia riešenia DNN

Pri vývoji nových metód, prípadne riešení novej úlohy sa nevyhneme pomerne veľkému množstvu experimentov, ktoré sú časovo veľmi náročné. Aktuálna prax je manuálne skúšať

rôzne variácie architektúr a ich hyperparametrov. Toto je veľmi prácna úloha. Navyše je to extrémne náročné na výpočtové zdroje, najmä pri technikách, ako je grid alebo náhodné prehľadávanie [40], ktoré sú použité na nájdenie najlepších hyperparametrov. Trénovanie siete veľkého rozsahu použitím GPU môže trvať dni alebo týždne.

Jedným z cieľov práce je navrhnutie iteračného poloautomatického postupu na anotovanie dát. Súčasťou tohto riešenia je aj potreba automatizovať hľadanie dobrého riešenia, ktoré bude v danej iterácii použité na anotovanie dát. V práci [41] sme navrhli jednoduchý prístup pre zlepšenie nájdeného riešenia. Myšlienka tohto prístupu je založená na hypotéze, že signifikantné zlepšenie v presnosti predikcie môže byť získané trénovaním každej konfigurácii viac krát (s rôznou inicializačnou schémou váh a všetko ostatné je ponechané rovnaké). Na zefektívnenie tohto postupu bola testovaná druhá hypotéza, ktorá hovorí o tom, že siete, ktoré dávajú lepší výsledok v skoršej fáze trénovania, patria medzi najlepšie na konci tréningu. Stratégia bola vybrať najlepšiu sieť v skorej fáze trénovania a ukončiť trénovanie zvyšných sietí, aby sa ušetrili zdroje.

4.4 Anotovanie dát na základe miery istoty predikcie

V predchádzajúcih kapitolách boli spomínané metódy, ktoré pomáhaju zredukovať cenu anotovania čo najviac tak, že anotátor robí čo najmenej zložitej práce (ako kreslenie orámovania objektu alebo výber triedy objektu z veľkého množstva možností) a tak zvýši svoju efektivitu. V našom prístupe sme sa zamerali na vytvorenie *anotovaného* datasetu špeciálne pre úlohy klasifikácie (na jednom celom obrázku je len jeden objekt). Testovali sme dve anotačné stratégie, ktoré sme nazvali *vysoká miera istoty anotácie* HCA (anglicky high confidence annotation) a *nízka miera istoty anotácie* LCA (anglicky low confidence annotation). V oboch stratégiách bol dataset vytvorený iteratívnym spôsobom tak, že sa postupne trénoval klasifikátor a následne sa použil jeho výstup na anotovanie nových ešte nevidených dát. Stratégia, ktorá vedie ku datasetu s vyššou klasifikačnou presnosťou pri danom množstve práce vynaloženej pri jeho tvorbe, je považovaná za lepšiu. Obe stratégie sa spoliehajú na manuálnu anotáciu, čo je aj kľúčová časť pri vytváraní spoľahlivého datasetu. Náš prístup k anotovaniu kandidátov je podobný ako u spomínaných metód (vybratie kandidátov a následné anotovanie) s tým, že využíva hodnotu predikcie, ktorá riadi výber anotovaných dát a zjednodušuje anotáciu navrhovaním správnej triedy. Zatiaľ čo HCA poskytuje presnejší výsledok triedy, LCA by mala identifikovať vzory s vysokou

informačnou hodnotou pre dataset. Tieto stratégie je možné rozšíriť použitím ďalších prístupov, ktoré riešia problémy s nedostatkom údajov a pretrénovaním, ako sú napr umelé dáta, ktoré sme v experimentoch použili. Tieto stratégie sme vyhodnocovali na dvoch datasetoch: Fashion MNIST a CIFAR-10. Vyhodnocovanie bolo založené na troch kritériách: počet všetkých anotácií, počet komplexných anotácií, a počítačový čas ktorý bolo treba na použitie danej stratégie. Obe stratégie sú porovnávané so základnou metódou založenou na náhodnom anotovaní vzorov (vzor, ktorý sa pridá do trénovacej množiny je vybraný náhodne).

Metóda

Empiricky sme vyhodnotili obe stratégie HCA a LCA pri vytváraní datasetu. Anotačný proces je iteratívny. V každej iterácii je klasifikátor naučený na aktuálnych dátach pustený na množinu neanotovaných kandidátov. Pre každého kandidáta je získaná miera istoty predikcie anotácie (maximálna hodnota zo softmax) ako aj trieda (argmax v softmax). Takto predpovedaná hodnota dôvery je použitá pre výber nových dát z anotácií. V HCA stratégii sú dáta vybrané začínajúc vzorom s najväčšou mierou istoty. Tieto môžeme považovať ako najjednoduhšie prípady pre klasifikátor. Preto navrhovaná trieda od klasifikátora je častejšie správna než pri LCA stratégii, kedy sú vyberané vzory s malou mierou istoty. Nakoniec sú vybrané vzory manuálne anotované a použité na trénovanie nového klasifikátora. Tento proces sa iteratívne opakuje. Naše porovnávanie s náhodným anotovaním (RA) je vykonané tak, že sa náhodne vyberú dáta, na ktorých sa natrénuje klasifikátor, a výsledky sa porovnajú s navrhovanými stratégiami.

Za účelom vyhodnotenia sme definovali dva typy anotácií: *jednoduchú* a *komplexnú.* Keď sa klasifikátor použije na dáta, ktoré nevidel, jeho výstup predpovedá mieru istoty a triedu pre každý vzor. V priebehu anotácie je operátorovi predložený vzor a zároveň je mu tiež poskytnutý kandidát na triedu. Ak je kandidát na triedu správny, anotátorovi stačí odpovedať, že daný kandidát je správny. Toto nazývame *jednoduchá anotácia.* Vo veľa klasifikačných úloh potvrdenie že kandidát triedy je správny je výrazne jednoduchšia úloha ako vybrať správnu triedu. Ak je kandidát nesprávny, alebo ho nie je možné poskytnúť, operátor musí vybrať správnu triedu. Toto voláme *komplexná anotácia*, nakoľko ide o náročnejšiu úloha, hlavne v prípade úloh s veľkým počtom tried. Všetky anotácie v *náhodnej* stratégii (RA), validačné dáta použité pre zastavovanie tréningu a počiatočné tréningové dáta, z ktorých začínaju HCA a LCA stratégie, sú považované za *komplexné anotácie.* Počet všetkých

anotácií je našim hlavným kritériom na meranie práce potrebnej na vytvorenie datasetu. Obsahuje obe, jednoduché aj komplexné, anotácie. Počet komplexných anotácií je dobrý indikátor pre úlohy, kde vybratie správnej triedy pre vzor je výrazne viac namáhavá práca ako skontrolovať, či kandidát na triedu je správny, alebo nie. Počítačový čas potrebný na anotovanie je pomocný indikátor na meranie ako je anotačná stratégia výpočtovo zložitá. Je počítaný ako počet všetkých tréningových dát použítých vo všetkých epochách a iteráciách.

4.5 Kódovanie výstupov neurónovej siete

Skôr než vysvetlíme použitie detekčných a opravných kódov pre rozhodovanie o anotácii dát, poukážeme na hlavné rozdiely medzi najčastejšie používaným spôsobom kódovania výstupov neurónovej siete, ktorým je one-hot kódovanie a všeobecným binárnym kódom na ktorom sú založené binárne detekčné a opravné kódy. Termín one-hot kód ponecháme v angličtine, pretože v slovenskej literatúre nemá výstižný preklad a aj slovenské učebnice ho takto používajú. Znamená to kód, ktorého kódové slová majú jeden bit jednotkový a ostatné bity nulové. To znamená, že na zakódovanie dát do *n* tried potrebujeme *n* bitov. Pretože Hammingova vzdialenosť medzi kódovými slovami one-hot kódu sú dva bity, kód nemá dobré opravné schopnosti. Avšak tým, že bity v kódovom slove sú viazané takýmto silným pravidlom (nazveme to holistickou vlastnosťou kódových slov), má veľmi dobré vlastnosti pre trénovanie neurónovej siete. Použitím metódy softmax na spracovanie výstupov neurónovej siete sa ľahko určí jednotkový bit, ktorý sa priradí výstupu po určitej transformácii s najvyššou hodnotou a ostatným bitom sa priradia nuly. Ak navyše transformované výstupy normujeme na tvar rozdelenia pravdepodobnosti, trénovanie konvolučných sietí spätným šírením gradientu sa ukázalo veľmi úspešným. Kódové slová one-hot kódu vytvárajú ortogonálnu bázu.

Môžeme však použiť aj nejednotkové ortogonálne bázy na tvorbu kódových slov. Takým príkladom je použitie Hadamardových kódových slov [79]. Hadamardova matica vytvára kódové slová pomocou rekurentného predpisu

$$\mathbf{H}_{i+1} = \begin{pmatrix} \mathbf{H}_i & \mathbf{H}_i \\ \mathbf{H}_i & -\mathbf{H}_i \end{pmatrix}, \mathbf{H}_0 = (1), \tag{22}$$

kde na konci sú hodnoty “-1” nahradené hodnotami “0”. Okrem prvého riadku, v ktorom sú všetky prvky nulové, ostatné riadky obsahujú polovicu bitov jednotkových a polovicu bitov

nulových. Užitočnou vlastnosťou Hadamardovej matice je aj ekvidištantné rozdelenie kódových slov v kódovom priestore s Hamingovou vzdialenosťou $2n - 1$ bitov medzi kódovými slovami, kde $2n$ je počet výstupov neurónovej siete. Poznamenajme, že počet tried musí byť menší alebo rovný počtu výstupov siete. Ako upozorňuje [80], aj stĺpce (okrem prvého) Hamingovej matice majú rovnaký počet núl a jednotiek, nakoľko transpozíciou Hamingovej matice dostávame tú istú maticu. Túto vlastnosť sa snažili zachovať aj v prípade, že počet tried je menší ako počet riadkov v matici. Z konštrukcie matice (22) vyplýva, že počet riadkov je mocnina dvoch a po vynechaní nulového riadku, dostávame všetky možné kódové slová.

Všeobecné binárne kódovanie umožňuje vyššiu kapacitu kódu než one-hot alebo Hadamardove kódovanie. V uvedených dvoch kódoch sú kódové slová ortogonálne a teda vytvárajú bázu kódového priestoru, binárne kódovanie vytvára kódové slová aj lineárnou kombináciou bázických slov. Literatúra uvádza, že one-hot kódy sú menej odolné voči adverziálnym útokom [81], zatiaľ čo binárne kódy (napr. multi-way kódy) sú odolné viac [85]. Priradenie kódových slov jednotlivým kategóriám však nie je také priamočiare ako u one-hot kódu. Dobre preštudovaným prípadom je Error Correcting Output Code (ECOC) predstaveným v [82]. Rozpoznanie každého bitu výstupného kódu (ktorý je určený maticou kódu) je trénované zvlášť na binárnom rozdelení vstupných dát alebo príznakov. Cieľom je dosiahnuť nezávislosť bitov. Návrh matice kódu však zatiaľ ostáva nedoriešeným problémom [86,87,88,89]. ECOC kódové slová sú potom množinou nezávislých bitov, v ktorej každý bit indikuje prítomnosť odpovedajúceho príznaku v dátach. V tejto práci sa zameriavame na viactriedne vzory (multi-class), kde trieda nie je priradená jednotlivým bitom, ale kódovému slovu ako celku. Pripustenie závislosti medzi bitmi umožňuje využitie vzájomnej závislosti medzi bitmi podobne ako v softmaxe.

Hlavnou výhodou použitia binárnych kódov je vynikajúca úroveň poznania, ktorú dosiahla teória kódovania. Používame lineárne blokové kódy pre ich schopnosť oddelenia informácie o triedach od zabezpečovacej časti kódového slova. Pre porovnanie vlastností one-hot kódu a binárnym kódom sme pre databázu CIFAR-10 s desiatimi triedami zvoli desať bitové kódové slova, aby sme zachovali dimenziu kódového priestoru. Akceptovanie prijatého kódového slova ako správneho je založené na výpočte syndrómu. Chybu detekujeme v prípade, že syndróm nie je nulový [84], alebo je nulový, ale kódové slovo nie je priradené žiadnej triede (napr. v našom prípade desiatich tried ide o šesť slov zo šestnástich).

Ak toto pravidlo určí kódové slovo ako chybné, považujeme ho za nedôveryhodné a pri anotácii sa k nemu správame podľa stratégií popísaných v predchádzajúcich kapitolách. V tejto práci analyzujeme použitie (10, 4) blokových systematických cyklických kódov ako príkladu všeobecných binárnych kódov. Metodika hľadania suboptimálneho binárneho kódu je načrtnutá v neskoršej podkapitole, ale jej overenie sme nestihli a ponechávame ho na ďalší výskum. Tabuľky 4.1. predstavuje generujúce polynómy študovaných 10 bitových cyklických kódov. Všetky polynómy sú nutne ireducibilné. Prvých päť polynómov je aj primitívnych, čo sa v tomto prípade ukazuje ako nepodstatné.

CRC1: 100,0011	CRC2: 101,1011	CRC3: 110,0001	CRC4: 110,0111	CRC5: 111,0011
CRC6: 100,1001	CRC7: 101,0111	CRC8: 110,1101	CRC9: 111,0101	

Tabuľka.4.1. Binárny tvar generujúcich polynómov použitých cyklických kódov CRC.

4.5.1 Rozhodovacie metódy.

Ako priradiť dáta odpovedajúcim triedam? Pokiaľ samotná neurónová sieť implementuje nelineárnu transformáciu $\mathcal{R}^m \rightarrow \mathcal{R}^n, m > n$, úlohou klasifikácie je priradiť vstupnej vzorke triedu z konečnej množiny, v našom prípade predstavenú kódovým slovom. Základnou otázkou je, ktoré ktoré slovo sa najviac podobá získanému výstupu neurónovej siete. Hoci kódové slovo je binárne, môžeme ho považovať za bod v Euklidovom vektorovom priestore. Potom môžeme nájsť k vektoru výstupu najbližší vektor spomedzi kódových slov, v závislosti na zvolenej metrike. Takýto spôsob rozhodovania založenom na vzdialenosti je postačujúci, pokiaľ rozhodnutie o triede je posledným krokom a nepotrebujeme ho pre ďalšie úkony. Na priradenie kódového slova výstupu sa môžeme pozerať ako na logický výrok a teda môžeme naň aplikovať pravidlá logiky a skúmať ďalšie vlastnosti výstupu neurónovej siete. Na binarizované výstupy sa môžeme pozerať aj ako na kódové slová a použitím skalárov nad Galoisovým poľom vytvoriť konečno-rozmerný vektorový (kódový) priestor. Pomocou teórie kódovania potom môžeme študovať ďalšie vlastnosti výstupov. Avšak táto bohatá sada logických a kódovacích nástrojov je k dispozícii až po binarizácii výstupov neurónovej siete. Takýmto preklopením z Euklidovho vektorového priestoru do kódového však strácame informáciu o podobnosti výstupu a kódového slova. Aby sme ju zachovali a spojili výhody obidvoch priestorov, pozeráme sa na výsledok algebraickej operácie ako výrok, a určujeme jeho mieru pravdivosti použitím fuzzy logiky. Tento prístup nám napríklad umožní priradiť pravdivostnú hodnotu syndrómom a ukážeme, že najpravdivejšie kódové

slovo má najpravdivejší syndróm. Ako bude uvedené neskôr, zvolili sme Zadehovu (štandardnú) fuzzy logiku a $t \in [0,1]$ je mierou pravdivosti.

Preto prvým krokom je normalizácia individuálnej hodnoty výstupu neurónovej siete. Nech odozvou neurónovej siete na daný vstup je $\boldsymbol{y} = (y_0, \dots, y_{n-1}) \in \mathcal{R}^n$, a $\tilde{\boldsymbol{y}} = (\tilde{y}_0, \dots, \tilde{y}_{n-1}) \in [0,1]^n$ je normalizovaný výstup, kde *n* je počet výstupov / bitov (*n*=10 v našom prípade). Používame tri druhy normalizácie pre *i = 0,…,n*−1:

$$\text{sigmoid } \tilde{y}_i = \sigma(y_i) = \frac{1}{1 + e^{-y_i}} \tag{23}$$

$$\text{lineárna } \tilde{y}_i = \frac{y_i - y_{min}}{y_{max} - y_{min}}, y_{min} = \min\{y_0, \dots, y_{n-1}\}, y_{max} = \max\{y_0, \dots, y_{n-1}\} \tag{24}$$

$$\text{vektorová } \tilde{\boldsymbol{y}} = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_2} \tag{25}$$

Po normalizácii priraďujeme vektor výstupu kódovému slovu. Najjednoduchším spôsobom, ktorý je rozšírením jednorozmerného prípadu, je rozhodnúť o každom bite nezávisle

$$c_i = \begin{cases} 0, \tilde{y}_i < 0.5 \\ 1, \tilde{y}_i \geq 0.5 \end{cases}, i = 0,1, \dots, n-1. \tag{26}$$

Túto metódu nazývame "prahová bitová metóda". Z nášho pohľadu budeme výstupnú hodnotu považovať za pravdivosť výroku, že daný výstup určuje bitovú zvolenú hodnotu. Podrobnejšie to vyjadruje vzťah (30).

Zložitejšie prístupy, ktoré nazveme holistickými, berú výstupný vektor $\tilde{\boldsymbol{y}}$ ako celok. Najpopulárnejší je softmax prístup založený na one-hot kódovaní, v ktorom celistvosť kódového slova je opretá o štruktúru v ktorej všetky bity sú rovné nule s výnimkou jedného, ktorý je jednotkový. Po exponenciálnej transformácii

$$\tilde{y}_i \leftarrow \frac{e^{\tilde{y}_i}}{\sum_{j=0}^{n-1} e^{\tilde{y}_j}} . \tag{27}$$

normujeme hodnoty na rozdelenie pravdepodobnosti. Najvyššej hodnote priradíme jednotkový bit, ostatným priradíme nulové bity. V [83] je predstavená metóda, ktorá kombinuje ECOC so softmaxom. Metóda nepoužíva detekčné kódovanie. Implementovali sme túto metódu s nahradením ECOCu cyklickým kódom, aby sme ohodnotili prínos použitia Zadehovej fuzzy logiky pri rozhodovaní.

Na vysvetlenie ako používame binárne kódové slová, začneme s dvojhodnotovou logikou. V tomto Booleovom prípade je výstup $\tilde{\boldsymbol{y}}$ mapovaný na kódové slovo $\tilde{\boldsymbol{c}} = (\tilde{c}_0, \dots, \tilde{c}_{n-1}) \in \{0,1\}^n$ a každá trieda N je priradená k reprezentujúcemu kódovému $\boldsymbol{c^N} = (c_0^N, \dots, c_{n-1}^N)$, $N \in \{0,1, \dots, n-1\}$. Vstupná vzorka bude klasifikovaná ako vzor triedy N, vtedy a len vtedy ak $\tilde{\boldsymbol{c}} = \boldsymbol{c^N}$, t.j. $\bigwedge_{i=0}^{n-1}(\tilde{c}_i = c_i^N)$. V Booleovej logike sa kódové slovo rovná kódovému slovu triedy N vtedy a len vtedy, ak pravdivostné hodnoty zhody všetkých bitov porovnávaných kódových slov sa rovnajú jednej

$$\mathrm{S^N} = \bigwedge_{\mathrm{i=0}}^{\mathrm{n-1}}\left(\tilde{\mathrm{c}}_\mathrm{i} = \mathrm{c}_\mathrm{i}^\mathrm{N}\right) \Rightarrow \mathrm{t}(\mathrm{S^N}) = \prod_{\mathrm{i=0}}^{\mathrm{n-1}} \mathrm{t}\left(\tilde{\mathrm{c}}_\mathrm{i} = \mathrm{c}_\mathrm{i}^\mathrm{N}\right) = 1, \mathrm{N} \in \{0,1, \dots, 2^\mathrm{n} - 1\} \tag{28}$$

kde $S^N = (\tilde{\boldsymbol{c}} = \boldsymbol{c}^N)$, $\tilde{\boldsymbol{c}}, \boldsymbol{c}^N \in \{0,1\}^n$, $t(true) = 1, t(false) = 0$.

Aby sme zachovali tento holistický prístup k rozhodnutiu o celom kódovom slove, nahradili sme Booleovu logiku Zadehovou (štandardnou) fuzzy logikou a $t \in [0,1]$ je pravdivostná hodnota výroku. pravdivostná hodnota v Zadehovej fuzzy logike konjunkcie, disjunkcie a negácie výrokov A, B je

$$\begin{aligned} \mathrm{t}(\mathrm{A} \wedge \mathrm{B}) &= min\left(\mathrm{t}(\mathrm{A}), \mathrm{t}(\mathrm{B})\right), \\ \mathrm{t}(\mathrm{A} \vee \mathrm{B}) &= max\left(\mathrm{t}(\mathrm{A}), \mathrm{t}(\mathrm{B})\right), \\ \mathrm{t}(\overline{\mathrm{A}}) &= 1 - \mathrm{t}(\mathrm{A}). \end{aligned} \tag{29}$$

Hodnotu i-teho výstupu $\tilde{y}_i \in [0,1]$ budeme interpretovať ako pravdivostnú hodnotu. Aby sme určili správnosť bitu nezávisle na jeho hodnote, použijeme ako mieru zhody mieru pravdivosti [93] výroku $\tilde{y}_i = a, a \in \{0,1\}$. Aby sme sa vyhli nerozhodnuteľným výrokom (ak

$\tilde{y}_i = 1/2$), pripočítame v tomto prípade k miere pravdivosti zanedbateľnú hodnotu $\varepsilon > 0$ a teda $t(\tilde{y}_i = a) \neq 1/2$

$$\mathrm{t}(\tilde{\mathrm{y}}_{\mathbf{i}} = \mathrm{a}) = \begin{cases} \tilde{\mathrm{y}}_{\mathbf{i}}, & \mathrm{a} = 1, \tilde{\mathrm{y}}_{\mathbf{i}} \in [0, ½) \cup (½, 1] \\ 1 - \tilde{\mathrm{y}}_{\mathbf{i}}, & \mathrm{a} = 0, \tilde{\mathrm{y}}_{\mathbf{i}} \in [0, ½) \cup (½, 1] \\ ½ + \varepsilon, & \mathrm{a} = 1, \tilde{\mathrm{y}}_{\mathbf{i}} = ½, 0 < \varepsilon \ll 1 \\ ½ - \varepsilon, & \mathrm{a} = 0, \tilde{\mathrm{y}}_{\mathbf{i}} = ½, 0 < \varepsilon \ll 1 \end{cases}. \tag{30}$$

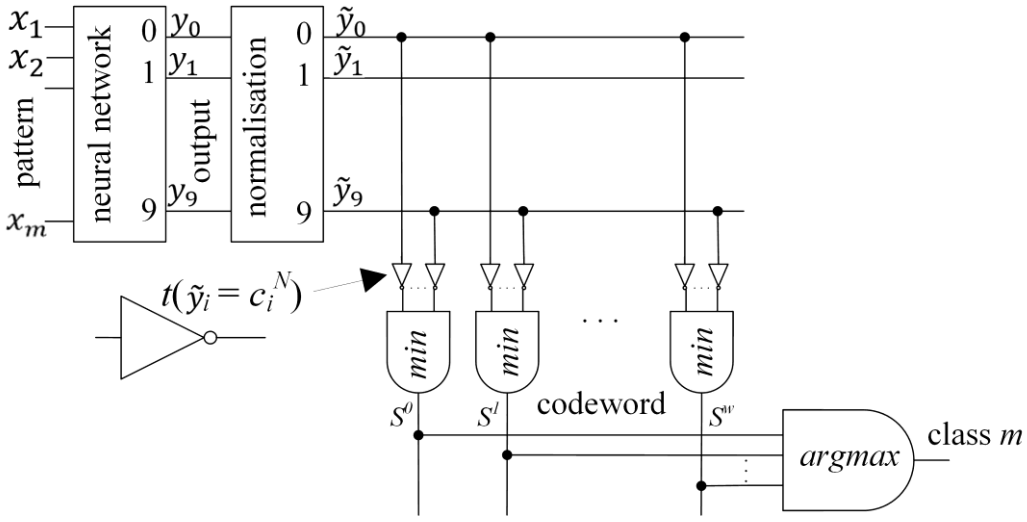
Pri implementácii nemusíme túto ochranu používať, pretože pravdepodobnosť nadobudnutia hodnoty $\tilde{y}_i = ½$ na spojitom jednotkovom intervale je nulová. Potom pri použití Zadehovej fuzzy logiky

$$t(S^N) = t(\tilde{\boldsymbol{c}} = \boldsymbol{c}^N) = \min_{i=0,\dots,n-1} t(\tilde{c}_i = c_i^N),\ N \in \{0,1,\dots,2^n - 1\}. \tag{31}$$

Notácia (31) je platná aj v prípade použitia Booleovej logiky, pretože

$$\min_{\mathrm{i}=0,\dots,\mathrm{n}-1} \mathrm{t}(\tilde{\mathrm{c}}_\mathrm{i} = \mathrm{c}_\mathrm{i}^\mathrm{N}) = \prod_{\mathrm{i}=0}^{\mathrm{n}-1} \mathrm{t}(\tilde{\mathrm{c}}_\mathrm{i} = \mathrm{c}_\mathrm{i}^\mathrm{N}),\ \text{for}\ \mathrm{t}(\tilde{\mathrm{c}}_\mathrm{i} = \mathrm{c}_\mathrm{i}^\mathrm{N}) \in \{0,1\}. \tag{32}$$

To je aj dôvod, prečo na obrázku. 4.7 je použitá funkcia minima miesto násobenia aj v prípade Booleovej logiky.



Obr.ázok 4.7. Zdroj [97]. Navrhnutá klasifikačná schéma založená na Zadehovej fuzzy logike.

4.5.2 Použitie detekčných kódovania na určenie potreby anotácie.

Potreba spracovania analógových výstupov binárnymi kódmi nie je nová [94]. Náš prístup je založený na fuzzy lineárnom kódovaní [90,91]. Vo všeobecnosti fuzzy kódové slovo je podmnožina kódového priestoru s priradenou mierou príslušnosti. Na priradenie vstupu triede, vypočítame (31) a funkcia (33)

$$m = \underset{\boldsymbol{c}^N \in C}{\operatorname{argmax}}\{t(\tilde{\boldsymbol{y}} = \boldsymbol{c}^N)\} \tag{33}$$

určí index kódového reprezentanta triedy s najvyššou pravdivostnou hodnotou. Výstup $\tilde{\boldsymbol{y}}$ a následne vzor ***x,*** budú priradené kódovému slovu $\boldsymbol{c}^m$. V tejto práci používame 10 kódových slov na trénovanie a všetkých 1024 kódových slov na testovanie, aby sme porovnali vlastnosti one-hot a binárneho kódovania. Ak kódové slovo $\mathbf{c}^m$ nepatrí do množiny kódových slov priradených triedam, interpretujeme to ako detekciu chyby pri vyhodnocovaní a vstupná vzorka nie je klasifikovaná. Podľa stratégie anotácie je potom vzorka zahodená alebo manuálne anotovaná. V tejto práci používame na detekciu chyby v binárnych kódoch nad Galoisovým poľom $GF(2)^n$ hodnotu syndrómu [84]. Zadehova fuzzy logika má užitočnú vlastnosť vyjadrenú Vetou 1, ktorú využijeme pri testovaní vstupných dát nad celou kódovou množinou.

Veta 1. Nech $\{\boldsymbol{c}^j = (c_0^j, \dots, c_{n-1}^j) \in \{0,1\}^n\}, j \in \{0,1, \dots, 2^n - 1\}$ je binárny kódový priestor, $\tilde{\boldsymbol{y}} = (\tilde{y}_0, \dots, \tilde{y}_{n-1}) \in [0,1]^n$ je normalizovaný výstup neurónovej siete, $t(\tilde{y}_i = c_i^j)$ je definovaná podľa (30) a o víťaznom kódovom slove $\tilde{\boldsymbol{c}}$ je rozhodnuté podľa (33). Potom

a) pravdivostná hodnota výroku že $\tilde{\boldsymbol{c}}$ je víťazným slovom je viac než jedna polovica,
b) je len jedno kódové slovo ktorého pravdivostná hodnota podľa je viac než jedna polovica,
c) víťazné kódové slovo dostaneme tiež zaokrúhlením výstupu $\tilde{\boldsymbol{y}}$ t.j. $\tilde{\boldsymbol{c}} = (\approx \tilde{y}_0, \dots, \approx \tilde{y}_{n-1})$, kde $\approx \tilde{y}_i = \lfloor \tilde{y}_i + ½ \rfloor$.

Dôkaz.

Predpokladajme, že $\mathbf{c}^{\mathrm{j}} = \underset{\boldsymbol{c} \in \{0,1\}^n}{\operatorname{argmax}}\big(t(\tilde{\boldsymbol{y}} = \boldsymbol{c})\big)$ a $\mathrm{t}\big(\tilde{\boldsymbol{y}} = \boldsymbol{c}^j\big) < ½$. Potom v kódovom slove $\boldsymbol{c}^j$ existujú bity $i \in I \subseteq \{0, \dots, n-1\}$ s $t\big(\tilde{y}_i = c_i^j\big) < ½$. Po výmene týchto bitov ich negáciou

dostávame kódové slovo $\boldsymbol{c}^k$ with $t(\tilde{y}_i = c_i^k) > ½, i = \{0, \dots, n-1\}$. Preto, $t(\tilde{\boldsymbol{y}} = \boldsymbol{c}^k) > ½$, čo je v rozpore s predpokladom.

Predpokladajme, že

$$\mathbf{c}^{\mathrm{j}} = \underset{\mathbf{c}\in\{0,1\}^{\mathrm{n}}}{\operatorname{argmax}}\big(\mathrm{t}(\tilde{\mathbf{y}} = \mathbf{c})\big),\ \mathbf{c}^{\mathrm{k}} = \underset{\mathbf{c}\in\{0,1\}^{\mathrm{n}}}{\operatorname{argmax}}\big(\mathrm{t}(\tilde{\mathbf{y}} = \mathbf{c})\big)\ \mathrm{j} \neq \mathrm{k}\ \mathrm{a}\ \mathrm{t}\big(\tilde{\mathbf{y}} = \mathbf{c}^{\mathrm{j}}\big) > ½,\ \mathrm{t}\big(\tilde{\mathbf{y}} = \mathbf{c}^{\mathrm{k}}\big) > ½.$$

Pretože $\mathbf{c}^{\mathrm{j}} \neq \mathbf{c}^{\mathrm{k}}$, potom existuje bit $i \in \{0,1, \dots, n-1\}$ tak, že $c_i^j = \overline{c_i^k}$.

Hodnoty $t(\tilde{y}_i = c_i^j) > ½, = t(\tilde{y}_i = \overline{c_i^k}) > ½$ implikujú $\mathrm{t}(\tilde{\mathrm{y}}_{\mathrm{i}} = \mathrm{c}_{\mathrm{i}}^{\mathrm{k}}) < ½$, $\mathrm{t}(\tilde{\mathbf{y}} = \mathbf{c}^{\mathrm{k}}) < ½$, čo je v rozpore s predpokladom.

Pre zaokrúhlený výstup platí $t(\tilde{\boldsymbol{y}} = (\approx \tilde{\boldsymbol{y}})) > ½$. Podľa b) $\approx \tilde{\mathbf{y}}$ je jediné víťazné kódové slovo.

Myšlienka pravdivostných hodnôt syndrómov je založená na určení algebraických výrazov v ich určitom (crisp) výsledku a ich pravdivosti. Binárne operácie modulo 2 sú určované nasledovne:

$$a \odot b,\ \ a, b \in \{0,1\} \qquad a \oplus b,\ \ a, b \in \{0,1\}$$

$\odot$	0	1
0	0	0
1	0	1

$\oplus$	0	1
0	0	1
1	1	0

(34)

Podľa (29) pre $a, b \in [0,1]$ platí

$$t(a \odot b = 1) = t\big((a = 1) \wedge (b = 1)\big),$$

$$t(a \odot b = 0) = 1 - t(a \odot b = 1),$$

(35)

$$t(a \oplus b = 1) = t\big((a = 0) \wedge (b = 1) \vee (a = 1) \wedge (b = 0)\big),$$

$$t(a \oplus b = 0) = 1 - t(a \oplus b = 1),$$

Použijeme (35) aj pre $t(\tilde{y}_j \odot h_{ij}^T)$, $\tilde{y}_j \in [0,1]$, $h_{ij}^T \in \{0,1\}$, kde h_{ij}^T je crisp hodnota v kontrólnej matici *H* [84]. Ak normalizovaný výstup siete $\tilde{\boldsymbol{y}}$ je pravdivostnou hodnotou kódového slova $\boldsymbol{c}$ t.j., $\tilde{\boldsymbol{y}} = t(\tilde{\boldsymbol{c}} = \boldsymbol{c})$ alebo $\tilde{y}_j = t(\tilde{c}_i = c_i), i = (0, \dots, n-1)$, potom odhadom syndrómu je $\tilde{\boldsymbol{s}}$, kde $\tilde{\boldsymbol{s}} = \tilde{\boldsymbol{c}}\boldsymbol{H}^T$, $\boldsymbol{H} = (\boldsymbol{P}|\boldsymbol{I})$, a $\boldsymbol{I}$ je jednotková matica. Pre testovaný cyklický kód CRC7

$$\boldsymbol{P}^T = \begin{pmatrix} 0\,1\,0\,1\,1\,0 \\ 0\,0\,1\,0\,1\,1 \\ 1\,0\,1\,1\,1\,0 \\ 0\,1\,0\,1\,1\,1 \end{pmatrix}.$$

Použitím (30) a (35), pravdivostná hodnota nulového syndrómu (nie je detekovaná chyba) je:

$$\begin{gathered} \mathrm{t}(\tilde{\mathrm{s}}_\mathrm{i} = 0) = \mathrm{t}\left(\left(\sum_{\oplus \mathrm{j}} \tilde{\mathrm{y}}_\mathrm{j} \odot \mathrm{h}_{\mathrm{ij}}^{\mathrm{T}}\right) = 0\right), \\ \mathrm{t}(\tilde{\mathrm{s}}_\mathrm{i} = 0) = \underset{\mathrm{j}=0,\dots,\mathrm{n}-1}{max} \left\{ min\left\{\mathrm{t}\left(\tilde{\mathrm{y}}_\mathrm{j} = \mathrm{c}_\mathrm{j}\right), \mathrm{h}_{\mathrm{ij}}^{\mathrm{T}}\right\}\right\} \end{gathered} \tag{36}$$

Poznamenajme, že v nasledujúcej vete platí $t(\tilde{y}_i \odot a) \neq 1/2$ a (n,k) lineárny blokový kód znamená kódové slová s *n* bitmi a *k* informačnými bitmi o triede.

Veta 2. Nech $\left\{\boldsymbol{c}^j = \left(c_0^j, \dots, c_{n-1}^j\right) \in \{0,1\}^n\right\}, j \in \{0,1, \dots, 2^n - 1\}$ je binárny kódový priestor, $\tilde{\boldsymbol{y}} = (\tilde{y}_0, \dots, \tilde{y}_{n-1}) \in [0,1]^n$je normalizovaný výstup neurónovej siete, $t\left(\tilde{y}_i = c_i^j\right)$ je definovaná podľa (30), víťazné kódové slovo $\tilde{\boldsymbol{c}}$ je vybraté podľa

$$\tilde{\boldsymbol{c}} = \underset{\tilde{\boldsymbol{c}} \in \{0,1\}^n}{argmax}\{t(\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{c}})\}, \tag{37}$$

pravdivostné hodnoty syndrómu $\boldsymbol{r} = (r_0, \dots, r_{n-k-1}) \in [0,1]^{n-k}$sú počítané $\boldsymbol{r} = \tilde{\boldsymbol{y}}\boldsymbol{H}^T$, kde $\boldsymbol{H}^T$je kontrolná matica daného detekčného lineárneho (n,k) blokového kódu, a víťazný syndróm $\tilde{\boldsymbol{s}}$ je určený

$$\tilde{\boldsymbol{s}} = \underset{\tilde{\boldsymbol{s}} \in \{0,1\}^{n-k}}{argmax}\{t(\boldsymbol{r} = \tilde{\boldsymbol{s}})\}. \tag{38}$$

Potom $\tilde{\boldsymbol{c}}\boldsymbol{H}^T = \mathbf{0} \Leftrightarrow \tilde{\boldsymbol{s}} = \mathbf{0}$, t.j. výrok „víťazné kódové slovo je správne“ je zhodný s výrokom „víťazný syndróm je nulový“.

Dôkaz.

Podľa Vety 1, pre každý výstup $\tilde{\boldsymbol{y}}$ existuje jediné víťazné slovo $\tilde{\boldsymbol{c}}$ s $t(\tilde{\boldsymbol{y}} = \tilde{\boldsymbol{c}}) > ½$, t.j., $t(\tilde{y}_i = \tilde{c}_i) > ½$, $i = 0, \dots, n-1$ a pre každé slovo $\boldsymbol{r} = \tilde{\boldsymbol{y}}\boldsymbol{H}^T$ existuje jediný víťazný syndróm $\tilde{\boldsymbol{s}}$ s $t(\boldsymbol{r} = \tilde{\boldsymbol{s}}) > ½$ i.e., $t(r_i = \tilde{s}_i) > ½$, $i = 0, \dots, n-k-1$. Víťazné kódové slovo $\tilde{\boldsymbol{c}}$ je správne práve vtedy keď $\tilde{\boldsymbol{c}}\boldsymbol{H}^T = \mathbf{0}$, víťazný syndróm je nulový práve vtedy keď $t(\boldsymbol{r} = \mathbf{0}) > ½$. Podľa (14) platí $t(\boldsymbol{r} = \mathbf{0}) = t(\tilde{\boldsymbol{y}}\boldsymbol{H}^T = \mathbf{0})$. Kontrolná matica obsahuje jednotkovú podmaticu, takže každý stĺpec kontrolnej matice obsahuje aspoň jednu jednotku. Potom

$$t(\tilde{y}_j = c_j) > ½ \; j = 0, \dots, n-1 \Longrightarrow t(r_i = 0) = \max_{j=0,\dots,n-1} \left\{ min\{t(\tilde{y}_j = c_j), h_{ij}^T\} \right\} > ½ \quad (39)$$

$$i = 0, \dots, n-k-1.$$

Pravdivostná hodnota nulového syndrómu je $t(\boldsymbol{r} = \mathbf{0}) = \min_{i=0,\dots,n-k-1} t(r_i = 0)$. Ak $t(r_i = 0) > ½$, potom $t(r_i = 1) < ½$, $i = 0, \dots, n-k-1$ a zmenou ľubovoľného bity syndrómu pravdivostná hodnota syndrómu sa zmenší. □

Doplnok do jednotky je pravdivostnou hodnotou detekcie chyby $t(\boldsymbol{r} \neq \mathbf{0}) = 1 - t(\boldsymbol{r} = \mathbf{0})$. Veta 2 sa dá jednoducho zovšeobecniť pre korekčné kódy: ak má víťazné syndróm $\tilde{\mathbf{s}} = \tilde{\mathbf{c}}\boldsymbol{H}^T$, potom $\tilde{\mathbf{s}}$ je víťazný syndróm

$$\tilde{\boldsymbol{s}} = \underset{\tilde{\boldsymbol{s}} \in \{0,1\}^{n-k}}{argmax}\{t(\boldsymbol{r} = \tilde{\boldsymbol{s}})\}, \text{ kde } \mathbf{r} = \tilde{\mathbf{y}}\mathbf{H}^{\mathrm{T}} \quad (40)$$

alebo jednoduchšie: “víťazné kódové slovo má víťazný syndróm”. Táto veta zdôvodňuje aj intuitívne použitie opravných kódov v schéme ECOC.

V tejto práci porovnávame rozhodovanie na základe Zadehovej fuzzy logiky s rozhodovaním pomocou softmaxu s použitím one-hot kódu. Pre jednoduché porovnanie sme použili rovnaký

počet výstupov. Pri používaní Zadehovej fuzzy logiky sme určovali najpravdivejšiu triedu. Pre rozhodnutie o anotácii sa pri softmaxe používa prah dôveryhodnosti Δ (confidence). Preto sme ho použili aj pri rozhodovaní pomocou Zadehovej fuzzy logiky, kedy vo vzťahu (33) môžeme tiež použiť prahovú hodnotu a víťazné kódové slovo bude použité v rozhodnutí o anotácii len ak $t(\boldsymbol{r}=\boldsymbol{0}) \geq \Delta, \Delta \in [0,1]$.

Skúšali sme aj súčinovú t-normu a pravdepodobnostnú t-konormu [92]. Zdá sa, že nie sú vhodné pre metódu spätného šírenia gradientu. Naopak, Zadehova fuzzy logika umožňuje priamu implementáciu gradientu. Môžeme to vyjadriť tak, že gradient tvorí najväčšia chyba najlepšieho kódového slova.

4.5.3 Použitie opravných kódov na určenie potreby anotácie.

V oblasti opravných kódov je k dispozícii bohatá literatúra. Aby sme zachovali jednotnú terminológiu, použijeme ako referenčný zdroj knihu [45]. Nech skaláre nad Galoisovým poľom GF(q), kde q je prvočíslo a n-tice $\boldsymbol{r}=(r_1, r_2, \dots, r_n) \in \{0,1,\dots,q-1\}^n$, tvoria vektorový priestor V. Lineárny blokový (n,k) kód $\mathcal{C}$ je k-rozmerný podpriestor n-rozmerného vektorového priestoru $\mathcal{C} \subset V$. Ak $\boldsymbol{B}=(\boldsymbol{b}_1, \boldsymbol{b}_2, \dots, \boldsymbol{b}_k)^{\boldsymbol{T}}$ je bázou $\mathcal{C}$, každé kódové slovo $\boldsymbol{c} \in \mathcal{C}$ je lineárnou kombináciou bázických kódových slov $\boldsymbol{c}=\boldsymbol{aB}, \boldsymbol{a} \in \{0,1,\dots,q-1\}^k$. V systematickom lineárnom blokovom (n,k) kóde má každé kódové slovo $\boldsymbol{c} \in \mathcal{C}$ k informačných symbolov a n-k kontrolných symbolov. Informačnou časťou bázy je jednotková matica $\boldsymbol{I}$, a bázická matica má tvar $\boldsymbol{B}=[\boldsymbol{I} \vdots \boldsymbol{P}]$ a nazývame ju generujúcou maticou $\boldsymbol{G}$. Matica $\boldsymbol{H}=[-\boldsymbol{P}^{\boldsymbol{T}} \vdots \boldsymbol{I}]$ je kontrolná matica. Pretože platí $\boldsymbol{GH}^T=\boldsymbol{0}$, pre každé kódové slovo $\boldsymbol{c} \in C$ platí $\boldsymbol{cH}^T=\boldsymbol{0}$. Ak je slovo $\boldsymbol{r} \notin \mathcal{C}$ potom $\boldsymbol{rH}^T \neq \boldsymbol{0}$. Toto poskytuje jednoduchý test, či slovo je kódovým slovom. Výsledok násobenia slova kontrolnou maticou sa nazýva syndróm $\boldsymbol{s}$. Pre všetky q^n n-tice z vektorového priestoru V existuje q^{n-k} syndrómov a q^k n-tíc má rovnaký syndróm. *Coset* je sada q^k n-tíc s rovnakým syndrómom. Oprava chyby v blokovom systematickom kóde využíva skutočnosť, že slovo $\mathbf{r}$ a chyba $\boldsymbol{e}=\boldsymbol{r}-\boldsymbol{c}$ majú rovnaký syndróm, pretože

$$\boldsymbol{s_r}=\boldsymbol{rH}^{\boldsymbol{T}}=(\boldsymbol{c}+\boldsymbol{e})\boldsymbol{H}^{\boldsymbol{T}}=\boldsymbol{cH}^{\boldsymbol{T}}+\boldsymbol{eH}^{\boldsymbol{T}}=\boldsymbol{eH}^{\boldsymbol{T}}=\boldsymbol{s_e}. \qquad 38(41)$$

Všetky chyby s rovnakým syndrómom budeme opravovať rovnako. To znamená, že spomedzi q^kchýb cosetu vyberieme jednu ako vedúcu chybu cosetu a všetky prijaté slová s

rovnakým syndrómom budeme opravovať tak akoby sa vyskytla vedúca chyba. Ak $\boldsymbol{e_s}$ je vedúca chyba cosetu pre syndróm **s**, opravené kódové slovo bude $\tilde{\boldsymbol{c}} = \boldsymbol{r} - \boldsymbol{e_s}$. To samozrejme znamená, že správne opravujeme len jednu chybu (vedúcu) a ostatných $q^k - 1$ chýb opravujeme nesprávne. Preto pri výbere vedúcej chyby musíme byť uvážlivý. Základný prístup k výberu vedúcej chyby spočíva v tom, že by sa mala vyskytovať častejšie ako je výskyt ostatných chýb v cosete. História opravných kódov siaha do časov, kedy v komunikačných systémoch prevládali chyby na malom počte bitov (metalické vedenia, magnetické disky). Tento fenomén vzniká aj v prípade ak sú chyby jednotlivých bitov nezávislé a počet chybných bitov má geometrické rozdelenie pravdepodobnosti. Preto za vedúce chyby brali chyby s najmenším počtom jednotkových bitov v cosete (jednotkový bit v chybovom slove určuje chybný bit). Tento spôsob sa aj nazýva štandardná dekódovacia tabuľka. Experimenty s neurónovými sieťami vedú k tomu, že chyby na väčšom počte bitov môžu byť viac pravdepodobné ako chyby na malom počte bitov. Preto pri určovaní vedúcej chyby cosetu vychádzame zo štatistík experimentov a kandidátom na vedúcu chybu cosetu je najčastejšia chyba spomedzi chýb s rovnakým syndrómom. Pokiaľ je však jej výskyt menší ako súčet ostatných chýb cosetu, chyby s týmto syndrómom nebudeme opravovať ale len detekovať. Ich oprava by totiž dávala predpoklad k tomu, že opravou podľa vedúcej chyby ešte zvýšime počet chybne vyhodnotených slov. Formálnejšie vyjadrené:

nech najpravdepodobnejšia chyba v cosete s indexom syndrómu s je

$$\bar{p}(s) = \max_{0 \le j \le 2^k - 1} p_j(s), \text{ and } \bar{\imath}(\boldsymbol{s}) = arg \max_{0 \le j \le 2^k - 1} p_j(s), s = 0, \dots, 2^{n-k} - 1. \quad (42)$$

Potom je pravdepodobnosť valídnej korekcie π a chybnej korekcie π_e je

$$\pi = \sum_{s=1}^{2^{n-k}} \bar{p}(s), \pi_e = \sum_{s=1}^{2^{n-k}} \sum_{i \neq \bar{\imath}(s)} p_i(s). \quad (43)$$

Neskôr ukážeme, ako konštruovať kód s preddefinovanými chybami ako vedúcimi chybami cosetov.

Doplnkovým kódom ku kódu $\mathcal{C}$ je kód $\bar{\mathcal{C}} \subset V$ taký, že platí $\mathcal{C} \cup \bar{\mathcal{C}} = V \; a \; \mathcal{C} \cap \bar{\mathcal{C}} = 0$. Z praktických dôvodov vynecháme prípad $\mathcal{C} = V, \bar{\mathcal{C}} = 0$. Ak $\mathcal{C}$ je lineárny blokový kód (n, k), doplnkový kód je $(n, n-k)$ lineárny blokový kód. Použijeme nasledujúce vlastnosti doplnkového kódu: $\boldsymbol{c} \in \mathcal{C} \Rightarrow \boldsymbol{c} \notin \bar{\mathcal{C}}$, $\boldsymbol{c} \notin \mathcal{C} \Rightarrow \boldsymbol{c} \in \bar{\mathcal{C}}$. Ak v krátkosti spomenieme kód $\mathcal{C}$, máme na mysli systematický lineárny blokový kód $\mathcal{C}(n, k)$. Predpokladáme, že použité

kódové slová sú iba ostrou podmnožinou celého kódu $\mathcal{K} \subset \mathcal{C}$ and $\mathbf{0} \notin \mathcal{K}$, v tejto práci uvažujeme s binárnym kódom $\mathcal{C}(10,4)$ kde $|\mathcal{C}(10,4)| = 16$ a | K | = 10 tj. 10 tried.

Metodika návrhu kódu je predmetom nasledujúcej podkapitoly. Cieľom tejto podkapitoly je dosiahnuť maximálnu pravdepodobnosť platnej opravy pre daný opravný kód. Ako príklad boli testované všetky (10,4) binárne cyklické kódy (CRC) a bol vybratý kód s najsľubnejším výsledkom. Tabuľka 4.2 definuje všetky generujúce polynómy pre (10,4) CRC.

Kód	Generujúci polynóm	π	*m*	Kód	Generujúci polynóm	π	*m*	Kód	Generujúci polynóm	π	*m*
CRC1	$1 + x + x^6$	0,842	99	CRC4	$1 + x + x^2 + x^5 + x^6$	0.847	163	CRC7	$1 + x + x^2 + x^4 + x^6$	0.846	171
CRC2	$1 + x + x^3 + + x^6$	0,844	155	CRC5	$1 + x + x^4 + x^5 + x^6$	0.843	160	CRC8	$1 + x^2 + x^3 + x^5 + x^6$	0.846	145
CRC3	$1 + x^5 + x^6$	0,843	111	CRC6	$1 + x^3 + x^6$	0.838	35	CRC9	$1 + x^2 + x^4 + x^5 + x^6$	0.853	180

Tabuľka 4.2. Generujúce polynómy cyklického (10,4) kódu.

Nech je kód $\mathcal{C} = \{\boldsymbol{c_0}, \dots, \boldsymbol{c_{2^k-1}}\}$ daný kontrolnou maticou $\boldsymbol{H}$. $\mathcal{C}$ je teda kód,ktorého kódové slová sú všetky slová (*n*, *k*) kódu. Túto kapacitu kódu nevyužívame ak je menej tried. Triedy tréningovej sady sú označené $\mathcal{T} = (\boldsymbol{t}_0, \dots, \boldsymbol{t}_{N-1})$, $N \leq 2^k - 1$. Ak zdôrazňujeme, že ide o kódové slovo takéhoto zredukovaného kódu, budeme ho označovať $\boldsymbol{c}_0 \in \mathcal{K} \subseteq \mathcal{C}$. V našom prípade $|\mathcal{K}| = 10, |\mathcal{C}| = 16$. Výstupy z neurónovej siete sú vektory $\boldsymbol{y} \in R^n$, z ktorých po zaokrúhlení (viď predchádzajúca podkapitola) na *n*-tice $\boldsymbol{r} \in \{0,1\}^n$ dostávame najpravdivejšie kódové slová. Tieto sú rozdelené na cosety podľa syndrómov $\boldsymbol{s} = \boldsymbol{r}\boldsymbol{H}^T$. Každý coset má jedinečné slovo – vedúcu chybu cosetu. Rozšírená sada kódových slov je

$$\mathcal{C}^+ = \{\boldsymbol{c}^+\} = \{\boldsymbol{c}_0 + \boldsymbol{e}(\boldsymbol{s}), \boldsymbol{c}_0 \in \mathcal{K}, \boldsymbol{s} \in S\} \tag{44}$$

a rozšírená sekvencia tréningových tried je $\mathcal{T}^+ = (\boldsymbol{t}_0^+, \dots, \boldsymbol{t}_{N-1}^+)$, $\boldsymbol{t}_i^+ \in \mathcal{C}^+, i = 0, \dots, N-1$. Jedineční vedúci coset-ov dajú dekódovaciu tabuľku $\mathcal{D} = \{\boldsymbol{e}(\boldsymbol{s}), \boldsymbol{s} \in S\}$. Ak je prijatá n-tica $\boldsymbol{r} \in \{0,1\}^{\boldsymbol{n}}$ ktorá generuje syndróm $\boldsymbol{s} = \boldsymbol{r}\boldsymbol{H}^T$, koriguje sa na kódové slovo $\tilde{\boldsymbol{c}} = \boldsymbol{r} - \boldsymbol{e}(\boldsymbol{s})$. Vedúci cosetov sa formujú počas tréningu. Aby tréning zostal flexibilný, ak má viac chýb v cosete rovnakú maximálnu pravdepodobnosť, všetky z nich sú označené ako vodcovia coset $\boldsymbol{e}^+(\boldsymbol{s})$, aj keď $p(\boldsymbol{e}^+(\boldsymbol{s})) = 0$. Sú tiež zahrnuté v rozšírenej dekódovacej tabuľke $\mathcal{D}^+ = \{\boldsymbol{e}^+(\boldsymbol{s}), \boldsymbol{s} \in S\}$.

Nasledujúci tréningový postup maximalizuje pravdepodobnosť správnej opravy π:

1. Vyberte jeden z opravných kódov (v našom prípade z CRC1-CRC9). Nastavte počiatočnú postupnosť označenia tréningu $\bar{\mathcal{T}} = \mathcal{T}$. Vytvorte rozšírenú dekódovaciu tabuľku $\mathcal{D}^+$ (2^n chýb je rozdelených do 2^{n-k} cosetov podľa syndrómov $\boldsymbol{s} = \boldsymbol{e}\boldsymbol{H}^T$). V tomto počiatočnom kroku sú všetky chyby označené ako vedúce coset-ov a sú zahrnuté v rozšírenej dekódovacej tabuľke, pretože zatiaľ nemáme žiadne štatistiky. Vytvorte pravidlo pre výber jednej vedúcej chyby u všetkých cosetov (t.j. minimálna Hammingova veľkosť chyby alebo náhodný výber).
2. Pre každú vzorku *i=0,…,N-1* (s označením $\boldsymbol{t}_i^+$) tréningovej sady vyhodnoťe euklidovskú vzdialenosť $d_E \in R$ prijatého výstupného vektora $\boldsymbol{y} \in \boldsymbol{R}^n$ k vygenerovanému kódovému slovu $\boldsymbol{c}_0 \in \mathcal{K}$ s pridanými vedúcimi chybami cosetu $\boldsymbol{e}^+(\boldsymbol{s}) \in \mathcal{D}^+$. Vezmite za prijatú chybu lídra coset-u s najmenšou vzdialenosťou:

$$\boldsymbol{e}_{opt}(\boldsymbol{r}) = \underset{\boldsymbol{s}}{argmin}\, d_E(\boldsymbol{v}, \boldsymbol{c}_0 + \boldsymbol{e}^+(\boldsymbol{s})), \text{ for } \boldsymbol{r} = \boldsymbol{c}_0 + \boldsymbol{e}^+(\boldsymbol{s}) \tag{45}$$

   a nahraďte $\boldsymbol{t}_i^+ \in \bar{\mathcal{T}}$ opraviteľným kódovým slovom $\boldsymbol{t}_i^+ = \boldsymbol{c}_0 + \boldsymbol{e}_{opt}(\boldsymbol{r})$.
3. Trénujte neurónovú sieť, aby ste minimalizovali pravdepodobnosť $P(\boldsymbol{r} \notin \mathcal{C}^+)$. Vypočítajte štatistiku chýb $p(\boldsymbol{e})$, $\boldsymbol{e} \in \{0,1\}^n$. Generujte novú rozšírenú dekódovaciu tabuľku hlavných chýb cosetu

$$\mathcal{D}^+ = \left\{\boldsymbol{e}_{opt}(\boldsymbol{s}) = \underset{\boldsymbol{r}:\, \boldsymbol{r}\boldsymbol{H}^T = \boldsymbol{s}}{argmax}\, p\left(\boldsymbol{e}^+(\boldsymbol{s})\right)\right\} \tag{46}$$

   Ak existuje viac chýb s rovnakou pravdepodobnosťou (aj keď je maximum rovné nule), pridajte všetky do dekódovacej tabuľky. Filtrujte rozšírenú dekódovaciu tabuľku $\mathcal{D}^+$ podľa pravidla z kroku 1 do dekódovacej tabuľky D, ktorá má jedinečné vedúce chyby cosetov.
4. Skontrolujte správnosť na validačnej sade podľa dekódovacej tabuľky D. Ak úspešnosť klesá (v porovnaní s predchádzajúcou epochou), pokračujte v kroku 2, inak vezmite ako výsledok dekódovaciu tabuľku $\mathcal{D}$ a zastavte.

4.5.4 Adaptácia dekódovacej tabuľky opravného kódu.

Základnou metódou trénovania je gradientová metóda minimalizácie účelovej funkcie (54). Modifikáciou základnej metódy je použitie kontrastných strát v účelovej funkcii [42] alebo tiež myšlienky tripletov [43]. Okrem pozitívnych cieľov (kódových slov tried) sú aj negatívne prekážky (kódové slová ktorým sa treba vyhnúť). Pozitívne L-tice sú množiny kódových slov, ktoré má neurónová sieť rozpoznať v každej triede

$$P_i = \{p(i,j) = (p_1(i,j), \dots, p_L(i,j)), j = 1, \dots, |P_i|\} \subset \{0,1\}^L, i = 0,1, \dots, n-1 \qquad (47)$$

kde P_j je množina pozitívnych kódových slov pre triedu i. Na začiatku používame len 10 unárnych pozitívnych slov alebo 10 slov podľa tabuľky 4.3.

trieda	0	1	2	3	4
kódové slová	1000001101	0010101001	0100110101	0110011100	1011000011
trieda	5	6	7	8	9
kódové slová	0001100111	1001101010	1100111000	1111110110	0101010010

Tabuľka 4.3. 10-bit kódové slová priradené k CIFAR-10 triedam

Takýto výber sa pokúša minimalizovať počet binárnych slov s minimálnou Hammingovou vzdialenosťou rovnou 4 a maximalizovať Hammingovu vzdialenosť medzi najčastejšie zamieňanými triedami. Na rozdelenie celého výstupného priestoru medzi kódové slová, sme použili metódu K-FOLD (k=10) [44], ktorá pripisuje rozpoznané kódové slovo vzorkám tréningovej sady, keď sa vyskytujú vo validačnej sade. V opakovanej modifikácii metódy K-FOLD aktualizujeme rozpoznané kódové slová (používa sa k = 5). Poznamenajme, že je možné nechať niektoré podpriestory prázdne, t.j. niektoré kódové slová nie sú priradené k žiadnej triede, ale celý výstupný priestor je rozdelený.

Negatívne L-tice sú vzormi, ktorým sa musí algoritmus učenia vyhýbať pri hľadaní pozitívneho kódového slova.

$$N_i = \{\boldsymbol{n}(i,j) = (n_1(i,j), \dots, n_L(i,j))\ j = 1, \dots, B\} \subset \{0,1\}^L\ i = 1, \dots, N \qquad (48)$$

Všetky falošne pozitívne chyby počas tréningu môžeme považovať za negatívne chyby. Aby sme kontrolovali ich množstvo, ukladáme ich do konečnej pamäte pre každý vzor. Pokiaľ sa rovnaká chyba vyskytla viackrát v rámci posledných B epoch tréningu (kde B je tiež dĺžka vyrovnávacej pamäte), uložia sa tam všetky. Ak je vzorka rozpoznaná správne, zostáva pozícia vyrovnávacej pamäte prázdna.

Náročnejšia verzia na počítačové zdroje je, keď pozitívne kódové slovo patriace do pozitívnej triedy vytvorí množinu všetkých negatívov patriacich do doplnku:

$$\boldsymbol{C} \in P_{\boldsymbol{i}} \Rightarrow N_i = C - P_{\boldsymbol{i}}. \tag{49}$$

Kde C je kódový priestor *n* bitových kódových slov $\boldsymbol{c}_i$, $i = 1,2, \dots, 2^n$, kde $i \neq j \Rightarrow \boldsymbol{c}_i \neq \boldsymbol{c}_j$ a $\bigcup_{i=1}^{2^n} \boldsymbol{c}_i = C$.

Prepojenie každého tréningového vzoru s vhodným pozitívnym kódovým slovom je nasledovné. Nech je odpoveďou neurónovej siete na danú vzorku jej výstup $\boldsymbol{x} = (x_1, \dots, x_n)$, ktorý patrí do podpriestoru $X_{\boldsymbol{c}}$ generovaného kódovým slovom $\boldsymbol{c} = (c_1, \dots, c_n)$, ak

$$x_i \begin{cases} < 0.5, c_i = 0 \\ \geq 0.5, c_i = 1 \end{cases} \tag{50}$$

Hlavná myšlienka rozdelenia priestoru kódu spočíva v priradení každého kódového slova k príslušnéj triede $\lambda(\boldsymbol{c}) \in \Lambda$ (vrátane „prázdnej triedy“) $\Lambda = \{0, l_1, \dots, l_L\}$. Množinu kódových slov priradených k rovnakej triede nazývame „*pozitívne triedy*“ kódových slov $P_i = \{\boldsymbol{c} \triangleq \boldsymbol{p}_i : \lambda(\boldsymbol{c}) = l_i\}$, $i = 1, \dots, L$ a ak sa vyskytne vzor z príslušnej triedy predpovedáme slovo z tejto množiny. Počas tréningu hľadáme každý výstup *x*, ktorý je generovaný triedou *i*, jej najbližším kladným kódovým slovom $\boldsymbol{\pi}(\boldsymbol{x})$:

$$\boldsymbol{\pi}(\boldsymbol{x}) = \underset{\boldsymbol{p}_i \in P_i}{\operatorname{argmin}} \{l_2^2(\boldsymbol{x}, \boldsymbol{p}_i)\} \tag{51}$$

Toto pozitívne kódové slovo generuje pozitívnu časť účelovej funkcie

$$L_p(\boldsymbol{x}) = \frac{1}{2} l_2^2\big(\boldsymbol{x}, \boldsymbol{\pi}(\boldsymbol{x})\big) \tag{52}$$

Podľa princípu kontrastívnej účelovej funkcie [42] používame „*pružinu určenú iba na odpudzovanie*“ alebo negatívne kódové slová v našej terminológii. Na rozdiel od kódových slov označených ako “*pozitívne triedy*“ nie sú spojené s triedami, ale so vzormi. Nech je výstup generovaný vzormi *s* z triedy l_i je $\boldsymbol{x} = v(\boldsymbol{s})$ a výstup je priradený kódovému slovu $\boldsymbol{c} \in C$. Rozhodnutie nie je správne, ak $\boldsymbol{c} \notin P_i$, a preto by sme radi takýmto rozhodnutiam vyhli.

Preto zhromažďujeme tieto nesprávne kódové slová a použijeme ich ako „*negatívne vzorové*“ príklady. Každý tréningový vzor má svoje vlastné „*negatívne vzory*“ kódových slov $N_i =$

$\{\boldsymbol{n}_i, i = 1, \dots, N\}$ a každá vzorka má pre ne svoju vlastnú pamäť (B je veľkosť pamäte). Pridaním rovnakého negatívneho kódového slova do vyrovnávacej pamäte niekoľkokrát zosilníte odpudenie od negatívneho kódového slova. Aby sa zabránilo situáciám, keď kladné kódové slovo a záporné kódové slovo poskytujú opačné gradienty, príspevok do účelovej funkcie udáva iba bity v negatívnych kódových slovách, ktoré sa líšia od zodpovedajúcich bitov v pozitívnom kódovom slove (funkcia XOR $\oplus$):

$$L_n(\boldsymbol{x}) = \frac{1}{2}\sum_{j=1}^{B}\sum_{i}^{n}\Big(\big(\pi_i \oplus n_i(j)\big)\, max\{0, 0.5 - l_1\big(x_i, n_i(j)\big)\}\Big)^2 \tag{53}$$

Štandardná hranica 0,5 je použitá ako kompromis medzi hodnotami 0 a 1, podľa hraníc podpriestorov. V zápornej časti $L_n(\boldsymbol{x})$ úcelovej funcie použijeme parameter α na vyváženie vplyvu pozitívnych a negatívnych kódových slov. Globálny príspevok výstupu vzoru $\boldsymbol{x}$ k účelovej funkcii je:

$$L(\boldsymbol{x}) = L_p(\boldsymbol{x}) + \alpha L_n(\boldsymbol{x}) \tag{54}$$

Nech vzoru $\boldsymbol{x}$ je priradené kódové slovo $\boldsymbol{c} = (c_0, \dots, c_9)$ a výstupy neurónovej siete sú Boolean $c_i \in \{0,1\},\ i = 0, \dots, 9$. Funkcia $\delta(\boldsymbol{c}) = \prod_{i=0}^{9}\delta(c_i)$ identifikuje kódové slovo $\boldsymbol{c}$, kde

$$\delta(c_i) = \begin{cases} c_i, & c_i = 1 \\ 1 - c_i, & c_i = 0 \end{cases}, i = 0, \dots, 9. \tag{55}$$

Teraz máme 1024 funkcií kódových slov (a ich výstupov) $\delta(\boldsymbol{c}_k), k = 0, \dots, 1023$, ktoré identifikujú 1024 10 bitových kódových slov.

4.5.5 Návrh suboptimálneho opravného kódu.

V predchádzajúcej kapitole bola popísaná metóda použitia opravného kódu, ktorý bol vopred zvolený. V našom prípade to bol (10, 4) lineárny blokový cyklický kód. V tejto kapitole načrtneme novú metódu ako hľadať suboptimálny lineárny blokový kód, ktorý bude deliť celý kódový priestor na podpriestory jednotlivých tried, pričom viacero podpriestorov môže byť priradené tej istej triede. Predosielame, že táto metóda je len ideovým námetom pre ďalší výskum a nestihli sme ju experimentálne overiť.

Systém rozpoznávania vzorov rozdelíme na dva základné podsystémy: extrakciu príznakov a klasifikáciu. Transformáciu z množiny originálov do množiny príznakov $\boldsymbol{x} \in \mathcal{R}^m \rightarrow \boldsymbol{y} \in \mathcal{R}^n$, $m > n$ nazývame vnorením do príznakového priestoru. Klasifikácia je transformácia z množiny príznakov do množiny tried $\boldsymbol{y} \in \mathcal{R}^n \rightarrow \mathbf{c} \in \{0,1\}^n$. V obvyklom prístupe (one-hot kódovanie) je trieda reprezentovaná binárnym kódovým slovom, ktoré má toľko bitov, koľko je príznakov (výstupov neurónovej siete). Vo všeobecnosti tomu tak nemusí byť, kvôli zreteľnému porovnaniu sme však zachovali tento počet bitov kódových slov. Primárnou úlohou vnorenia je extrahovať z dát dobre separované príznaky, čo znamená, že vektor príznakov vzoru jednej triedy bude vzdialený od vektora príznakov odlišnej triedy (maximalizácia medzi-triedneho rozptylu). Kvôli jednoduchosti klasifikátora sa používa aj druhá požiadavka: aby vektory príznakov vzorov rovnakej triedy boli k sebe blízko (minimalizácia vnútro-triedneho rozptylu). Spojenie týchto dvoch kritérií vyjadruje Fisherovo kritérium, ktorého hodnota je priamo úmerná vnútro-triednemu rozptylu a nepriamo úmerná medzi-triednemu rozptylu. Fisherovo kritérium bolo úspešne používané už v lineárnej diskriminačnej analýze (LDA). V neurónových sieťach vyjadruje túto myšlienku trénovanie na základe protirečivých strát (contrastive loss)Ak je trieda vyjadrená kódovým slovom, potom minimalizácia vnútro-triedneho rozptylu vedie k trénovaniu neurónovej siete s učiteľom tak, aby sa dekódované výstupy zhodovali s kódovými slovami vopred priradenými jednotlivým triedam. Môžu to byť kódové slová z one-hot kódu (najčastejší prípad), alebo to sú slová zvoleného binárneho kódu (viď kapitolu o detekčných kódoch). V prípade one-hot kódovania je medzitriedna Hammingova vzdialenosť rovná dvom bitom, v prípade všeobecného binárneho kódovania môže byť medzitriedna Hammingova vzdialenosť väčšia. Teória kódovania umožňuje delenie binárneho vektorového priestoru na podpriestory flexibilnejšie a nie je nutné, aby všetky vzory rovnakej triedy patrili do rovnakého podpriestoru. To umožňuje neklásť dôraz na veľkosť medzitriednej Hammingovej vzdialenosti, ale primárne sa sústrediť na to, aby vzory rôznych tried nemali identické príznaky. Aj naše skúsenosti s používaním cyklických detekčných kódov ukazujú, že najčastejšími chybami sú neodhaliteľné chyby, kedy vstup patriaci do jednej triedy vytvorí n-ticu, ktorá patrí inej triede. Zriedkavejšie vytvorí n-ticu, ktorá nepatrí žiadnej triede a teda je detegovateľnou chybou. To nás motivovalo k myšlienke, že hlavnou úlohou neurónovej siete je oddeliť od seba príznaky rôznych tried bez potreby sústredenia príznakov rovnakej triedy do jedného kompaktného podpriestoru. Samotné priradenie príznakov (kódových slov $\boldsymbol{c}$)

jednotlivým triedam t (klasifikáciu $\boldsymbol{c} \in \{0,1\}^n \rightarrow t \in \{0,1,\dots,N-1\}$) je možné ponechať procesu dekódovania. Minimalizáciu a maximalizáciu rozptylov potom modifikujeme:

a) ak neurónová sieť priradí dvom vzorom rovnakej triedy rovnaké kódové slovo, potom upravíme váhy neurónovej siete tak, aby sa zmenšila ich maximálnu odchýlka od kódového slova,
b) ak neurónová sieť priradí dvom vzorom rôznych tried kódové slová s menšou Hammingovou vzdialenosťou než je stanovená hodnota D, potom upravíme váhy neurónovej siete tak, aby sa zväčšila ich vzájomná vzdialenosť,
c) ak nenastane žiadny z vyššie uvedených situácií, váhy neurónovej siete nemeníme.

Voľbou D volíme minimálnu vzdialenosť pre vzory rozdielnych tried, $1 < D \leq n$. Horná hranica prichádza do úvahy len v špecifických, prakticky nezaujímavých prípadoch.

Trénovanie je inšpirované [1] a prebieha nasledujúco:

- náhodne sa vyberú dva vstupy $\boldsymbol{x}^i, \boldsymbol{x}^j \in \mathcal{R}^m$,
- určia sa odpovedajúce výstupy neurónovej siete $\boldsymbol{y}^i, \boldsymbol{y}^j \in \mathcal{R}^n$, $m \gg n$,
- určia sa odpovedajúce normované výstupy neurónovej siete $\tilde{\boldsymbol{y}}^i, \tilde{\boldsymbol{y}}^j \in [0,1]^n$
- zaokrúhlením sa určia najpravdivejšie kódové slová $\tilde{\boldsymbol{c}}^i, \tilde{\boldsymbol{c}}^j \in \{0,1\}^n$, $\tilde{\boldsymbol{c}}^i =\approx \tilde{\boldsymbol{y}}^i$, $\tilde{\boldsymbol{c}}^j = \approx \tilde{\boldsymbol{y}}^j$,
- určia sa chyby zaokrúhlenia na jednotlivých bitoch. Ďalej ich považujeme za pravdivostné hodnoty výroku, že normovaný výstup má danú hodnotu bitu. Pre odstránenie nejednoznačnosti vylučujeme z (30) hodnotu ½ odchýlením o zanedbateľnú hodnotu.
- hodnota účelovej funkcie závisí na váhovej funkcii vzdialenosti $w(x)\begin{cases} > 0, x \in (0, D) \\ = 0, \quad x \geq D' \end{cases}$ ktorá zaručí dosah účelovej funkcie len do vzdialenosti $x = \sqrt{\sum_{k=0}^{n-1}\left(\tilde{y}_k^i - \tilde{y}_k^j\right)^2} < D$.
- ak sú vstupy z rovnakej triedy, hodnota účelovej funkcie je súčinom maximálnej zaokrúhľovacej chyby a váhovej funkcie $L_S\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) = t_{max}^{ij} w_S(x)$, kde

  $$t_{max}^{ij} = \max_{k=0,\dots,n-1}\left\{t\left(\tilde{y}_k = c_k^i\right), t\left(\tilde{y}_k = c_k^j\right)\right\}, \text{ kde } w_S(x) = \frac{\max(\mathrm{D,x}) * max(0, D-x)}{D*(D-x)}.$$

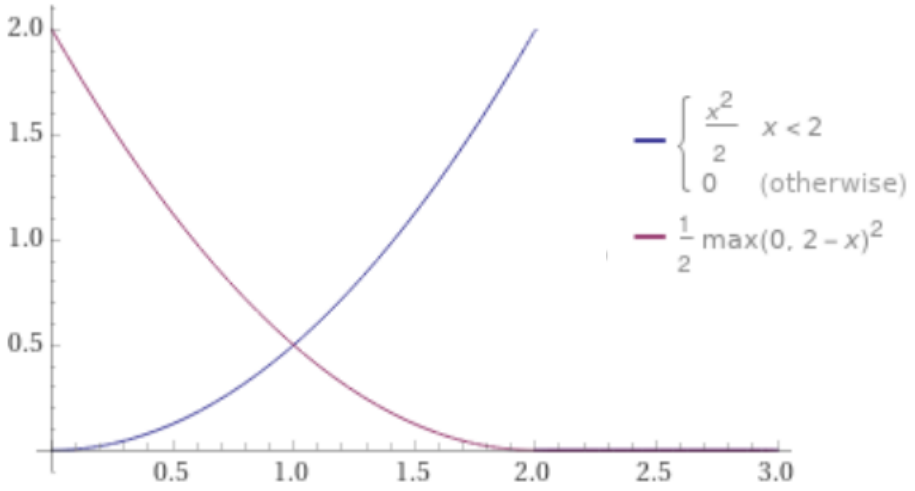
  Jej úlohou je meniť parametre siete tak, aby sa normované výstupy približovali najbližším kódovým slovám,

- ak vstupy nie sú z rovnakej triedy, hodnota účelovej funkcie je daná len váhovou funkciou vzdialenosti

  $$L_D\left(\boldsymbol{x}^i, \boldsymbol{x}^j\right) = w_D(x), \text{ kde } w_D(x) = ½\, max(0, D-x)^2.$$

Jej úlohou je meniť parametre siete tak, aby sa normované výstupy od seba vzďaľovali, ak sú si bližšie než *D*.

Tvar váhových funkcií pre *D*=2 je na obrázku 4.8.



Obrázok 4.8. Tvar váhových funkcií pre *D*=2

- ak je vzdialenosť normovaných výstupov väčšia než *D*, priradenie vzorov kódovým slovám nechávame bezo zmeny, t.j. ich príspevok do účelovej funkcie je nulový. Na rozdiel od [1] sa nesnažíme priťahovať k sebe rovnaké vzory, ktoré sú od seba ďalej než *D*. To umožňuje vytváranie viacerých zhlukov vstupov rovnakej triedy.
- strednú hodnotu odhadujeme z dávky dvojíc vstupov,
- spätným šírením gradientu účelovej funkcie sa upravujú váhy neurónovej siete.

Výsledkom trénovania je štatistika na validačnej množine, v ktorej pre každé kódové slovo celého kódového priestoru $\tilde{\boldsymbol{c}}^i \in [0,1]^n$sa určí v koľkých prípadoch $n^t(\tilde{\boldsymbol{c}}^i)$ bolo priradené danej triede $t \in \{0,1,\dots,N-1\}$. Kódové slovo $\tilde{\boldsymbol{c}}^i$považujeme za pozitívne, ak jedna trieda T je dominantná, t.j. pre triedu $T = \underset{t\in\{0,1,\dots,N-1\}}{\operatorname{argmax}} n^t(\tilde{\boldsymbol{c}}^i)$ platí $n^T(\tilde{\boldsymbol{c}}^i) > \sum_{t\neq T} n^t(\tilde{\boldsymbol{c}}^i)$. Pre pozitívne kódové slovo je odhad pravdepodobnosti, že kódové slovo určuje danú triedu vyšší než jedna polovica. V prípade, že štatistika kódového slova neobsahuje žiadny záznam t.j. $\sum_t n^t(\tilde{\boldsymbol{c}}^i) = 0$, kódové slovo považujeme za nepoužité. Našou snahou pri návrhu kódu je, aby v kóde boli také kódové slová, pre ktoré rozdiel medzi správne a nesprávne klasifikovanými vstupmi bol maximálny:

$$\sum_{\tilde{c}\in C}(n^T(\tilde{\boldsymbol{c}}) - \sum_{t\neq T} n^t(\tilde{\boldsymbol{c}})\ ) \rightarrow \max . \qquad (56)$$

Pre lineárny blokový kód platí $\tilde{\boldsymbol{c}}\boldsymbol{H}^T = \boldsymbol{s}\begin{cases} = \boldsymbol{0}, \tilde{\boldsymbol{c}} \in C \\ \neq \boldsymbol{0}, \tilde{\boldsymbol{c}} \notin C \end{cases}$. Pre *n*-tice, ktoré budú patriť do kódu (s nulovým syndrómom) použijeme ako kritérium vhodnosti kritérium (56).

Pre každý nenulový syndróm $\boldsymbol{s} \neq \boldsymbol{0}$ môžeme určiť najčastejšiu (vedúcu) chybu $\tilde{\boldsymbol{e}}_L$: $\tilde{\boldsymbol{e}}_L\boldsymbol{H}^T = \boldsymbol{s}$. Vedúcu chybu budeme opravovať $\bar{\bar{\tilde{\boldsymbol{c}}}} = \tilde{\boldsymbol{c}} \oplus \tilde{\boldsymbol{e}}_L$, ak sa vyskytuje častejšie ako ostatné chyby s rovnakým syndrómom:

$$\frac{\left(n(\tilde{\boldsymbol{e}}_L)\right)}{\sum_{\tilde{\boldsymbol{e}}:\, \tilde{\boldsymbol{e}}\boldsymbol{H}^T = \boldsymbol{s}} n(\tilde{\boldsymbol{e}})} > \frac{1}{2}. \qquad (57)$$

Ak táto nerovnosť nie je splnená, chybu budeme detegovať, ale ju nebudeme opravovať.

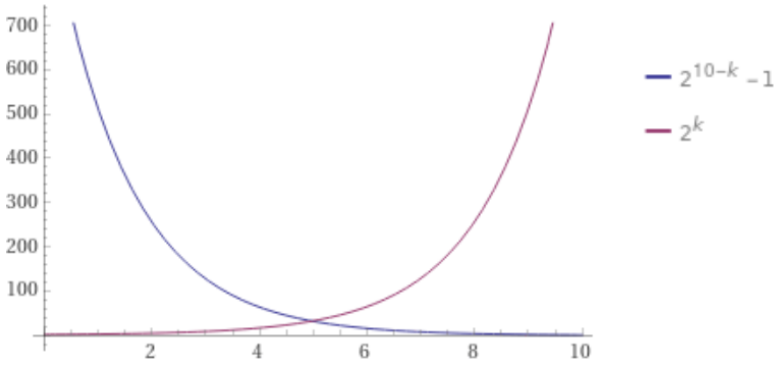
Uvedeným spôsobom vieme hodnotiť jednotlivé *n*-tice ako by prispeli k opravnej a detekčnej schopnosti blokového kódu. Pre nasledujúci algoritmus je určujúce hodnotenie príspevku bázických vektorov k celkovej úspešnosti bázy, ktorá generuje kód. Príspevok sa skladá z dvoch častí:

1. z príspevku bázického kódového slova vo vzťahu (56),
2. z príspevku bázického kódového slova k vlastnostiam syndrómov, t.j. k vedúcim chybám.

Pretože odstránenie jedného bázického kódového slova vedie ku zániku aj iných kódových slov, jednoduchšie je hodnotiť výhodnosť celého kódu, ktorý je určený novou bázou. Výhodnosť kódu určíme ako súčet správne určených a opravených kódových slov:

$$\sum_{\tilde{c}\in C}(n^T(\tilde{\boldsymbol{c}}) - \sum_{t\neq T} n^t(\tilde{\boldsymbol{c}})\ ) + \sum_{\boldsymbol{s}\neq\boldsymbol{0}} \left(n\left(\tilde{\boldsymbol{e}}_L(\boldsymbol{s})\right) - \sum_{\tilde{\boldsymbol{e}}(\boldsymbol{s})\neq\tilde{\boldsymbol{e}}_L(\boldsymbol{s})} n\left(\tilde{\boldsymbol{e}}(\boldsymbol{s})\right)\right) \rightarrow \max. \qquad (58)$$

Všeobecne môžeme povedať, že v (*n*,*k*) blokovom kóde je *k*-bázických vektorov, tj. binárny kód má 2^k kódových slov a 2^{n-k}-*1* nenulových syndrómov (vedúcich chýb). Túto všeobecnú závislosť vyjadruje obrázok 4.9. pre *n*=10.



Obrázok 4.9. Počet kódových slov a vedúcich chýb binárneho blokového (*n*,*k*) kódu

Hľadanie vhodného kódu pozostáva z dvoch krokov:

1. vybratie *n* lineárne nezávislých *n*-tíc najlepších v zmysle (56) a ich ortonormalizácia,
2. postupné redukovanie dimenzie kódu vylučovaním bázických vektorov tak, aby ostávajúce bázické vektory generujúce kód *C* boli najlepšie v zmysle kritéria (58).

Táto „pažravá“ metóda negarantuje nájdenie najlepšieho kódu a bolo by ju možné nahradiť metódou vetiev a hraníc. Pre získanie prvých skúseností ju však považujeme za dostatočnú.

Nasleduje podrobnejší popis algoritmu.

1) Nájdenie východiskového kódu
   a) zoradenie *n*-tíc zostupne podľa kritéria (56),
   b) ortonormalizácia *n*-tíc v uvedenom poradí, pokiaľ nie je dosiahnutých *n* jednotkových vektorov. Pozícia jednotkových vektorov v ortonormalizovanom zozname so zachovaným pôvodným zoradením (nebázické *n*-tice sú nulové), určuje pozíciu bázických vektorov vo východiskovom zozname. Výsledkom je *n* bázických vektorov v poradí podľa kritéria (56). Jednotkové bity nebázických vektorov v zozname pred ortonormalizáciou určujú ich súradnice vzhľadom na vypočítanú bázu.
2) Redukcia dimenzie informačnej časti kódu
   a) predpokladáme existenciu lineárneho blokového (*n*,*k*) kódu. Na začiatku (*n*,*n*) kódu ako výsledku kroku 1),
   b) pre všetkých k bázických kódových slov určíme hodnotu podľa vzťahu (57) a odstránime bázické slovo s najmenšou hodnotou, $k-1 \rightarrow k$,
   c) pokračujeme krokom a), pokiaľ počet bázických vektorov *k* neklesne pod vopred stanovenú hodnotu k_{min}. Minimálna kapacita kódu je daná počtom tried *M*: $2^{k_{min}} \geq M$.

Hlavnou nevýhodou tohto základného algoritmu je, že väčšina náhodne zvolených vstupov bude v príznakovom priestore vzdialená od seba viac než *D* a nebude sa podieľať na trénovaní neurónovej siete. Preto navrhujeme pred každým cyklom trénovania vytvoriť

zoznam vzoriek trénovacej množiny $\mathfrak{X} = \{(\boldsymbol{x}, \tilde{\boldsymbol{c}}): \boldsymbol{x} \rightarrow \tilde{\boldsymbol{c}}\}$ tak, že každej vzorke $\boldsymbol{x}$ trénovacej množiny je priradené najpravdivejšie kódové slovo $\boldsymbol{x} \rightarrow \tilde{\boldsymbol{c}}$ a zoznam $\mathfrak{C} = \{(\tilde{\boldsymbol{c}},\ \mathfrak{X}_{\tilde{c}}): |\mathfrak{X}_{\tilde{c}}| > 0\}$, v ktorom každému kódovému slovu (ktorému je priradená aspoň jedna vzorka) sú priradené vzorky $\boldsymbol{x}$ tak, že $\boldsymbol{x} \rightarrow \tilde{\boldsymbol{c}}$ t.j. $\mathfrak{X}_{\tilde{c}} = \left\{ \bigcup_{x \rightarrow \tilde{c}} \boldsymbol{x} \right\}$. V každom trénovacom cykle sú tieto zoznamy nemenné a v rámci neho postupujeme nasledovne:

1. náhodne vyberieme vzorku $\boldsymbol{x^i}$ z trénovacej množiny a podľa zoznamu $\mathfrak{X}$ vyberieme najpravdivejšie kódové slovo $\tilde{\boldsymbol{c}}^i$,
2. ku kódovému slovu $\tilde{\boldsymbol{c}}^i$ nájdeme kódové slová $\tilde{\boldsymbol{c}}^j$, ktorých vzdialenosť od slova $\tilde{\boldsymbol{c}}^i$ nie je väčšia než *D*. Vytvoríme podzoznam vzoriek $\mathfrak{X}_{\tilde{c}^i}(D)$ tak že im odpovedajúce kódové slová splňujú predchádzajúcu podmienku

$$\mathfrak{X}_{\tilde{c}^i}(D) = \bigcup_{d(\tilde{c}^i, \tilde{c}^j) \leq D} \mathfrak{X}_{\tilde{c}^j},$$

3. zo zoznamu $\mathfrak{X}_{\tilde{c}^i}(D)$ náhodne vyberieme vzorku $\boldsymbol{x^j}$ a s dvojicou vzoriek $\boldsymbol{x^i}, \boldsymbol{x^j}$ pokračujeme v základnom algoritme.

V priebehu tréningového cyklu zoznamy $\mathfrak{X}_{\tilde{c}}(D)$, $\mathfrak{C}$ nemeníme. Prepočítavame ich na začiatku nového cyklu s novo nastavenými váhovými koeficientami neurónovej siete. Dĺžka tréningového cyklu je hyperparameter, ktorého vhodnú veľkosť treba experimentálne nastaviť podobne, ako sa určuje veľkosť dávky.

5 Vykonané experimenty a dosiahnuté výsledky

5.1 Použité architektúry neurónovýc sietí a parametrov

Pre vyhodnotenie experimentov sme použili nasledujúce architektúry neurónových sietí.

CNN-1	layer	conv	pool	conv	pool	conv	local	fc		
	feature maps	16	16	16	16	16	16	10		
	kernel size	3	2	3	2	3	3	1		
	kernel stride	1	2	1	2	1	1	1		
	input:output size	26:24	24:12	12:10	10:5	5:3	3:1	1:1		
CNN-2	layer	conv	conv	pool	conv	conv	conv	conv	local	fc
	feature maps	16	16	16	16	16	16	16	16	10
	kernel size	3	3	2	3	3	3	3	3	1
	kernel stride	1	1	2	1	1	1	1	1	1
	input:output size	26:24	24:22	22:11	11:9	9:7	7:5	5:3	3:1	1:1
CNN-3	layer	conv	conv	pool	conv	conv	conv	conv	local	fc
	feature maps	16	16	16	32	32	32	64	64	10
	kernel size	3	3	2	3	3	3	3	3	1
	kernel stride	1	1	2	1	1	1	1	1	1
	input:output size	26:24	24:22	22:11	11:9	9:7	7:5	5:3	3:1	1:1

Tabuľka 5.1 Architektúry sietí CNN-1,CNN-2,CNN-3

Pre architektúru Resnet20v2 sme použili archítektúru uvádzanú v [95].

			CNN-4	CNN-5
Vrstva	**Veľkosť kernelu**	**Posun Kernelu**	**Počet Výstupných máp**	**Počet Výstupných máp**
Conv + BN	3	1	32	64
Conv + BN	3	1	32	64
Conv + BN	3	1	32	64
Conv + BN	3	1	32	64
Conv + BN	3	1	32	128
MaxP	2	1	32	128
Dropout 0.2				
Conv + BN	3	1	64	128
Conv + BN	3	1	64	256
MaxP	2	2	64	256
Dropout (0.2)				
Conv + BN	3	1	128	256
Conv + BN	3	1	128	256
MaxP	2	2	128	256
Dense	1	1	10	10
Normalizácia	Softmax / Sigmoid / Linear			

Tabuľka 5.2. CNN-4 a CNN-5 architektúry. Conv- konvolúcia, BN – batch normalizácia, MaxP – max pooling, Dense – plná vrstva

5.2 Použité dáta

V tejto práci boli pre potrebu experimentov použité 3 datasety. MNIST [37], Fashion MNIST [38] a CIFAR-10 [39].

MNIST

Dataset použitý na experimenty v ktorých sme ukázali ako sa dá zlepšiť trénovanie CNN pri opakovanom tréningu. Tento dataset je považovaný za jeden z najľahších a tvorí ho 60 000 tréningových vzorov a 10 000 testovacích vzorov o veľkosti 28x28 pixlov pri 10 triedach ručne písaných číslic od 0 po 9. Reprezentuje ručne písane číslice. Dnes sa používa len na overenie správnosti nových algoritmov a vo vedeckých kruhch je považovaný za príliš jednoduchý.

Fashion MNIST

Tento dataset tvorí 60 000 tréningových vzorov a 10 000 testovacích vzorov o veľkosti 28x28 pixelov vo farbách odtieni sivej. Tvorí ho tak isto 10 tried repreznetujúce oblečenie ako sú tričko, nohavice, sveter, šaty, kabát, sandále, košeľa, tenisky, taška, dámske lodičky. Je to náhrada za MNIST dataset ktorý bol na niektoré úlohy príliž jednoduchý. Tento dataset bol využítý pri skúšaní prototypoch algoritmov na GAN sieťach a neskoršie aj klasifikačných úlohách na ladenie vlastnej účelovej funkcie.

CIFAR-10

Hlavný dataset tejto práce použitý vo väčšine experimentov. Podčasť z 80 miliónov malých obrázkov a obsahuje 50 000 tréningových a 10 000 testovacích dát. Veľkosť každého obrázka je 32x32 pixlov v 24bit farbe. Každá trieda obsahuje 5000 obrázkov na trénovanie reprezentujúce: lietadlo, auto, vtáka, mačku, jeleňa, psa, žabu, koňa, loď a nákladné auto.

5.3 Výsledky jednoduchého prístupu zlepšenia riešenia DNN

Na experimenty bol použitý MNIST dataset [37]. Boli testované 3 architektúry (tabuľka 5.1) rôznej veľkosti. Tieto architektúry boli inšpirované klasickými konvolučnými neurónovými sieťami a používali konvolučné filtre 3x3. Každá architektúra mala vstup o veľkosti 26x26. Celkovo bolo natrénovaných 486 sietí. Pre každú architektúru boli použité 3 rôzne LR parametre *(0.01,0.001,0.0001),* momentum (*0.8,0.9,0.999*) a reguralizácia *(0.001, 0.0001, 0.00001*). Pre testovanie hypotéz bol každý experiment pustený 6x s rôznou inicializáciou váh, čo bolo dokopy 81 sád (3 architektúry s 27 rôznymi kombináciami hyperparametrov). Inicializácia váh bola robená pomocou gausového rozdelenia s nulovou strednou hodnotou a smerodajnou odchylkou $\sqrt{2/n}$ (tzv. Xavier inicializácia viď kapitola 2.2.7).

Z vyhodnotenia boli vyňaté experimenty, ktoré mali numerické chyby, a kde počet úspešných opakovaní toho istého experimentu bol menší ako 4. Výsledok pre každú architektúru je znázornený v Tabuľke 5.3. Je vidno, že rozdiel medzi najhoršou sieťou (min) a najlepšou sieťou (max) je veľmi veľký (v relatívnom zlepšení 21.1%, 28.11%, 26.95% pre jednotlivé siete). Ak porovnáme najlepšie výsledky s priemernými sieťami, relatívne zlepšenie je stále významné (10.96%, 17.95%, 17.56%). Z toho vyplýva, že má význam opakovane trénovať tú istú architektúru dokonca aj vtedy, ak sú všetky hyperparametre rovnaké (líšia sa len v inej inicializácii váh).

architektúra	min [%]	max[%]	priemer [%]	max-min relatívne zlepšenie [%]	max-priemer relatívne zlepšenie [%]	početnosť
CNN-1	97.46±0.23	98.01±0.11	97.77±0.08	21.1±8.18	10.96±3.77	10
CNN-2	97.24±0.3	98.02±0.39	97.6±0.3	28.11±11.13	17.95±9.32	12
CNN-3	97.52±0.35	98.24±0.36	98.9±0.28	26.95±13.03	17.56±11.69	12

Tabuľka 5.3. Výsledky pre každú architektúru (spriemerované cez všetky sady) min = siete s najhoršími výsledkami, max = siete s najlepšími výsledkami, priemer = priemer výsledkov sietí, max-min relatívne zlepšenie = relatívne zlepšenie, ak nájdeme najlepšiu sieť namiesto najhoršej, max-priemer relatívne zlepšenie = ak nájdeme najlepšiu sieť namiesto priemernej siete. Početnosť = počet sád experimentov

Aj keď výsledky potvrdili prvú hypotézu, tento postup by zvýšil už tak veľké množstvo experimentov. Pre zefektívnenie procesu bola testovaná druhá hypotéza. Skúmalo sa, či siete,

ktoré dávali najlepšie výsledky v skorom štádiu tréningu, patrili medzi najlepšie aj na konci tréningu. Hypotéza bola testovaná na jednotlivých sieťach, kedy sa vybranie najlepšej siete simulovalo v epochách 2, 4, 16, 64 a 128. V každej z týchto epoch boli vybraté najlepšie siete pre každú sadu experimentov, a následne porovnané oproti najlepšej sieti na konci tréningu a tiež oproti priemeru sieti. Výsledok je znázornený v tabuľke 5.4. Bolo použité kritérium *chybné relatívne zlepšenie,* ktoré meria ako veľmi stratíme výkon siete, ak by sme vybrali najhoršie, priemerné alebo predpovedané (najlepšia sieť v konkrétnej epoche) riešenie namiesto najlepšej siete.

LR	**epocha**	**priemerná sieť [%]**	**predpovedaná sieť [%]**	**najlepšia (max) sieť [%]**	**max - min relatívne zlepšenie [%]**	**max - priemer relatívne zlepšenie [%]**	**max - predpovedané [%]**
0.001	2	97.85	98.01	98.17	24.75	15.32	7.16
	4		97.98				8.2
	16		98.06				5.2
	64		98.07				4.6
	128		98.09				4.04
0.0001	2	97.65	97.62	98.02	26.19	16.03	15.67
	4		97.63				14.9
	16		97.75				10.77
	64		97.89				5.93
	128		97.89				6.01

Tabuľka 5.4. Výsledky experimentov (spriemerované cez všetky sady) posudzujú, ako dobre bude dávať predpovedaná sieť výsledky na konci tréningu v porovnaní s najlepšou a priemernou sieťou. Priemerná sieť je počítaná ako priemer zo všetkých natrénovaných sietí v každej sade. Všetky výsledky sú počítané na základe výsledkov dosiahnutých na konci tréningu. Predpovedaná sieť je vybratá ako najlepšia sieť v zodpovedajúcej epoche.

Môžeme sa pozerať na toto kritérium ako na meranie straty potenciálneho výkonu siete (menšie číslo je lepšie). Ako je vidno, predpovedaná sieť je schopná poraziť priemernú sieť s veľkým odstupom. Pre LR = 0.001 môžeme získať veľké zlepšenie založené na skorej predpovedi v epoche = 2. To znamená, že ak ukončíme všetky siete okrem najlepšej v druhej epoche, tak stratíme iba 7.16% z relatívneho zlepšenia oproti najlepšej sieti. Ak by sme trénovali len jednu sieť, môžeme očakávať stratu 15.32% priemernej sieti. Priemerná epocha, v ktorej je tréning zastavený je okolo 1507 + 1500 = 3007 (ak tréning na validačnej sade nie

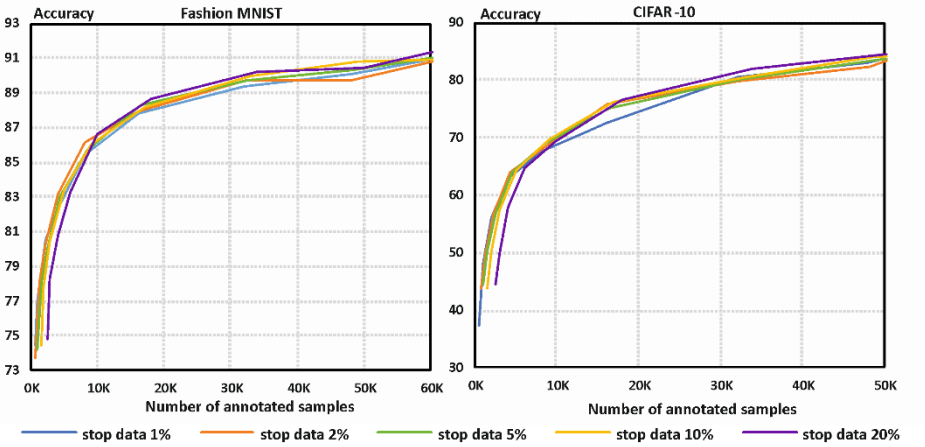
je zlepšený po 1500 epochách, potom je zastavený) čo poskytuje veľkú úsporu zdrojov. Pre LR = 0.0001 bolo dosiahnuté výrazné zlepšenie oproti priemernej sieti použitím stopovacieho pravidla v 64 epoche.

V tejto jednoduchej metóde, ktorá je založená na opakujúcom sa tréningu s rôznymi inicializačnými váhami, bolo ukázané, že veľké zlepšenie môže byť dosiahnuté opakujúcim sa tréningom vedúcim k 10.96%, 17.95% a 17.56% relatívnemu zlepšeniu najlepšej sieti porovnané s výkonom priemernej sieti na rôznych architektúrach.

5.4 Výsledky dolovania v čiastočne anotovaných dátach

Vykonali sme tri sady experimentov. Prvý experiment vyhodnocuje výkon RA stratégie a efektu veľkosti validačnej množiny používanej pri skorom zastavení tréningu. Druhý experiment ukazuje efekt rozdielneho počtu anotovaných dát na iteráciu čo je parameter z LCA a HCA stratégie. Posledný experiment porovnáva efektivitu všetkých stratégií na základe popísaných kritérií.

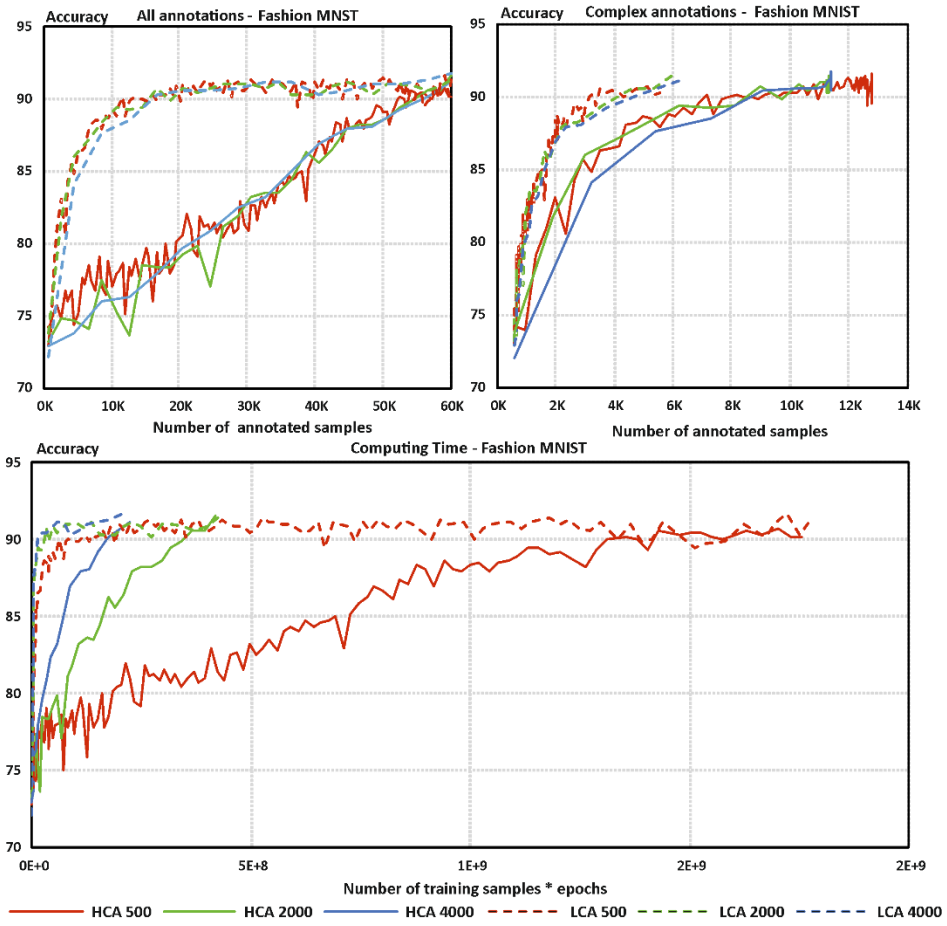
Obrázok 5.1 ukazuje výsledky pre RA stratégiu s použitím rôznej veľkosti stopovacej validačnej množiny: 1%, 2%, 5%, 10% a 20% testovacej sady.



Obrázok 5. 1. Vplyv rôznej veľkosti validačnej množiny na presnosť v RA stratégii.

V prípade základnej stratégie RA sme trénnovali na náhodnej podmnožine tréningových dát a v neskorších experimentoch sme porovnali výsledky oproti navrhovaným stratégiám. Vybrali sme nasledujúce podmnožiny tréningových dát: 500, 1000, 2000, 4000, 8000, 16000,

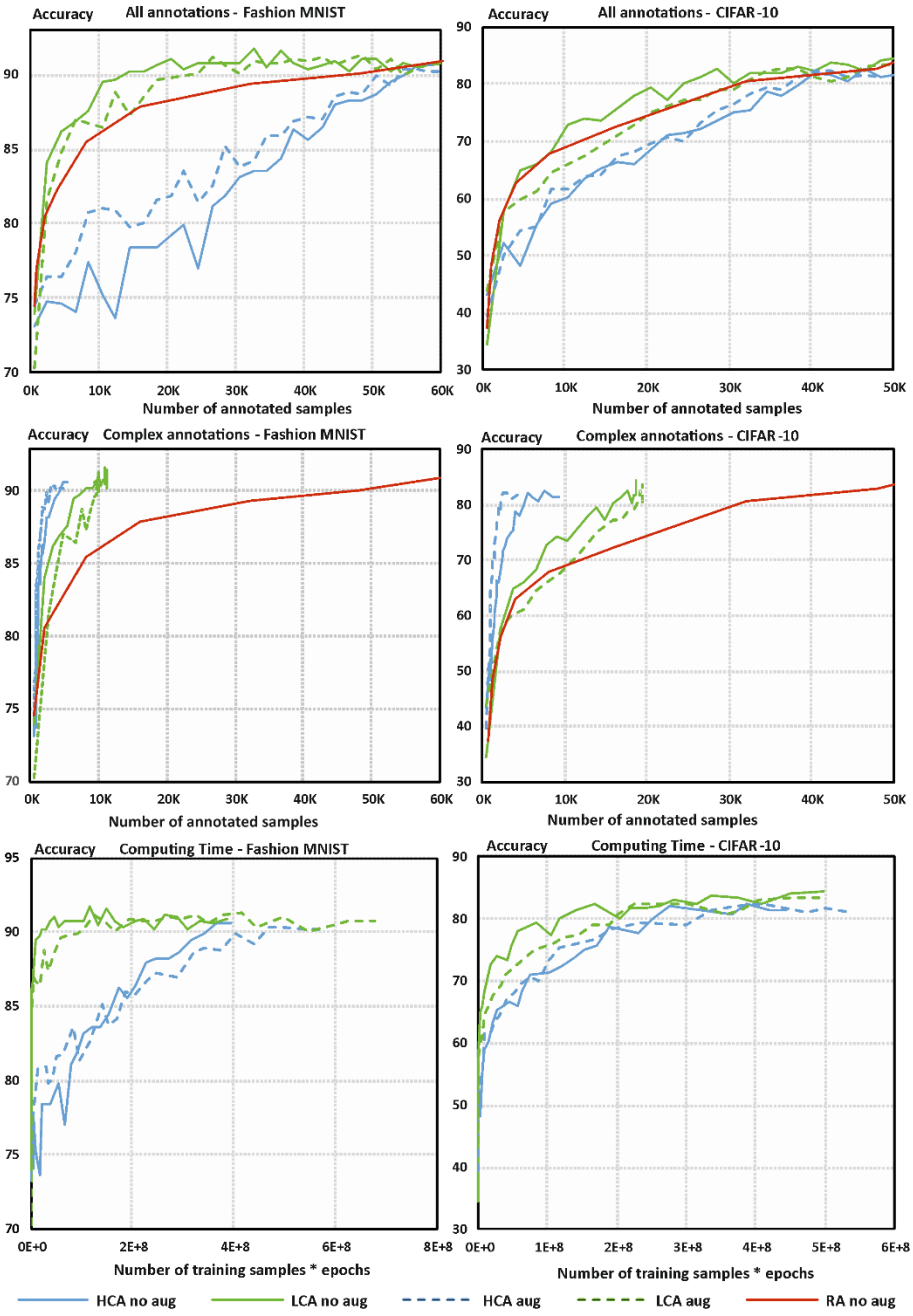
32000, 48000 a 50000/60000 (CIFAR-10/Fashion-MNIST). Rovnako ako v ostatných experimentoch, presnosť bola zmeraná použitím celého testovacieho datasetu. Väčšia sada pre skoršie zastavenie vedie k lepším výsledkom pretože presnejšie odráža realitu. Ale rozdieľ presnosti je malý a trend krivky je rovnaký vo všetkých prípadoch. Ak cheme použiť stopovacíe dáta, je ich potrebná anotovať ako prvé. Tieto anotácie sú považované za komplexné, pretože je ich potrebné získať skôr ako sa nejaký klasifikátor natrénuje a teda je potrebné určiť triedu ručne. Z výsledkov vyplýva, že 1% je dostatočné a toto množstvo sme použili v ďalších experimentoch.



Obrázok 5.2. Vyhodnotenie rôzneho množstva anotovaných dát na iteráciu z pohľadu jednotlivých metrík pre dataset Fashion-MNIST.

Ďalej sme potrebovali nastaviť parametre počet štartovacích dát a počet anotovaných dát na iteráciu pre LCA a HCA stratégie. Zistili sme, že 1% tréningových dát je dostatočné množstvo, z ktorého môžeme začať trénovať. Viac dát by zvýšilo množstvo komplexných anotácií. Vyhodnocovali sme tiež rôzny počet anotovaných dát na iteráciu pre Fashion-MNIST dataset. Na obrázku 5.2 sú znázornené výsledky pre 500, 2000 a 4000 anotácií.

Všetky nastavenia fungovali podobne dobre. Nastavili sme 2000 anotácií na iteráciu pretože 500 požaduje príliž veľa výpočtového času a 4000 dosahuje trochu slabšie výsledky než 2000.



Obrázok 5.3. Vyhodnotenie navrhovaných stratégií ako aj vplyvu využitia umelých dát (no aug – bez umelých dát, aug – s umelými dátami). Ľavý stĺpec zobrazuje výsledky pre Fashion-for a pravý stĺpec pre CIFAR-10. Jednotlivé riadky prezentujú presnosť na základe iného vyhodnocovacieho kritéria (všetky anotácie, komplexné anotácie, výpočtový čas).

Hlavný experiment má za cieľ vyhodnotiť efektívnosť navrhovaných stratégií. (obrázok 5.3). Použili sme RA ako základnú stratégiu ako všeobecne používaný prístup na vytvorenie malých a srtedne malých datasetov .Stratégia ktorá dosiahne vyššiu presnosť v rámci rovnakého počtu práce, je považovaná za lepšiu. Je potrebné poznamenať že na konci všetky dáta z Fashion-MNIST/CIFAR-10 datasetu sú anotované a vŠetky stratégie by mali získať rovnaký výsledok.

Môžeme vidieť podobné trendy pre oba Fashion-MNIST a CIFAR-10 datasety. LCA stratégia je najlepšia pre kritérium, ktoré zahŕňa všetky anotované dáta. Preto je viac vhodná v prípadoch kedy jednoduchá a komplexná anotácia je podobne náročná v zmysle ľudskej práce. Na druhej strane HCA stratégia je lepšia ak komplexné anotácie sú ťažšie na vykonanie a teda náročnosť jednoduchých anotácií môžme zanedbať. To je často prípad v úlohách s vysokým počtom tried. Prirodzene, RA stratégia je najvhodnejšia z pohľadu výpočtovej náročnosti. RA stratégia nepotrebuje trénovať žiadny klasifikátor. Pre tento dôvod je výpočtový čas nulový. LCA stratégia je výpočtovo efektívnejšia než HCA, hlavne v prípade Fashion-MNIST

Pri generovaní umelých dát sme použili štandartné obrazové transormácie (horizontálne otočenie, posun, rotáciu a zväčšenie) pre testovanie dopadu umelých dát v tréningu. Natrénovali sme dva klasifikátory: s a bez umelých dát. Klasifikátor s umelými dátami bol použitý pre výber vzorov pre anotáciu. Klasifikátor bez umelých dát bol použitý pre porovnanie výsledkov presnosti. Dôvodom je, že chceme merať kvalitu vytvoreného datasetu. Umelé dáta nám len pomáhajú zlepšiť výber anotovaných dát, ale nie sú súčasťou výsledného datasetu. Toto nám dovolilo spravodlivo posúdiť kvalitu datasetu, ako aj porovnať výsledky cez všetky experimenty. Prínos umelých dát vidíme pri HCA stratégii v zmysle komplexných anotácii. Umele dáta pomohli natrénovať klasifikátor s lepšou presnosťou. To je pre HCA stratégiu kľúčové hlavne v kontexte komplexnej anotačnej práce. Klasifikátor produkuje presnejších kandidátov na triedu a tým znižuje množstvo komplexných anotácií. Zdá sa, že umelé dáta majú opačný efekt na LCA stratégiu, kedy sú vyberané vzorky s nižšou predikciou. Možné vysvetlenie je, že umelé dáta kompenzujú nedostatok niektorých transformácií vo vzoroch, vďaka čomu sú vzorky s nízkou spoľahlivosťou menej reprezentatívne. Pre definitívny záver je však potrebné vykonať ďalšie experimenty.

5.5 Experimenty využívajúce detekčné CRC kódy pre anotáciu

V naseledujúcom texte sú predstavené výsledky klasifikátorov na rôznych architektúrach pre vyhodnocovacie datasety ktoré sú v rámci aj mimo trénovaciu doménu. Znamená to že obrázky na ktorých sa model trénuje považujeme za dáta „*z rozdelenia*“ a obrázky ktoré pochádzajú a prezentujú inú doménu považujeme za dáta „*mimo rozdelenia*“. Pre základné vyhodnotenie sme používali meranie presnosťi (59) a spoľahlivosťi (60).

	Akceptovaný (A)		**Zamietnutý (R)**	**Σ**
	Pozitívny (P)	**Negatívny (N)**		
True (T)	TP	FN	FR	T
False (F)	FP	TN	TR	F
Σ	P	N	R	Q

Tabuľka 5.5 Konfúzna matica

Vychádzali sme z dvoch základných stavov: „*akceptovaný*“ a „*odmietnutý*“ vzor. Keď sa vykoná rozhodnutie o neznámom vzore, najprv sa akceptuje alebo zamietne. Keď je vzor akceptovaný, určí sa trieda kam má patriť. Vzor je vždy pozitívny (true class label) alebo negatívny (false class label). Zamietnutý znamená že klasifikátor neurčí jeho triedu. Pre tieto dôvody máme 6 možných stavov zobrazených v tabuľke 5.5. Pre dáta mimo rozdelenia sú iba dva stavy: chybne pozitívny (false positive) a správne zamietnutý (true rejected). Pre tieto dáta používame meranie (63)

$$\text{Presnosť } p_{acc} = \frac{TP+TN}{Q} \tag{59}$$

$$\text{Spoľahlivosť } p_{conf} = \frac{TP+TN}{P+N} \tag{60}$$

$$\text{Dôvera } p_{conf} = \frac{TP}{P} \tag{51}$$

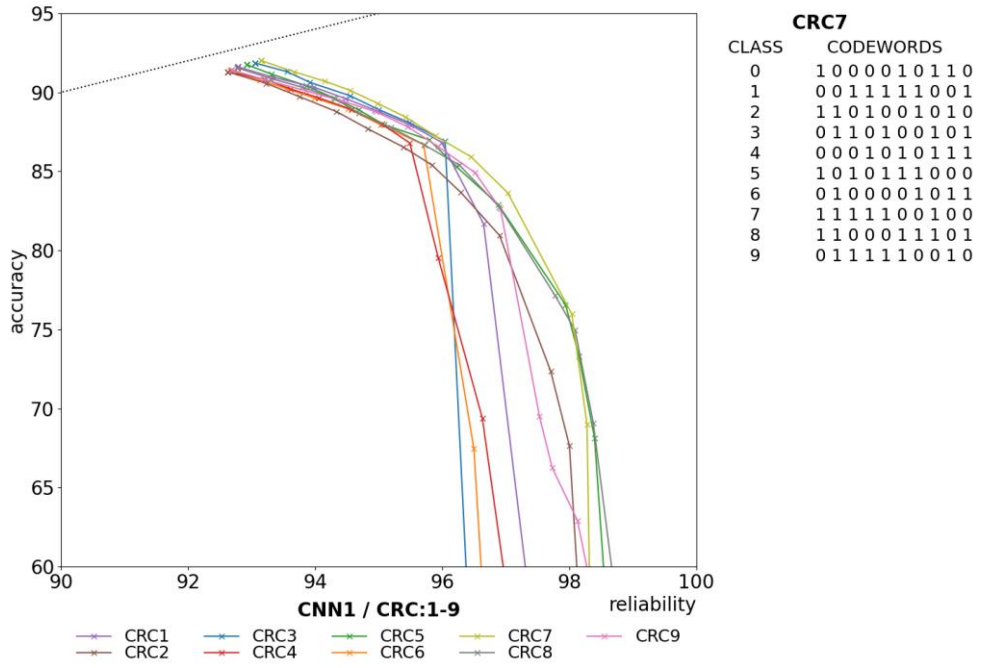
$$\text{Akceptácia } p_A = \frac{P+N}{Q} \tag{62}$$

$$\text{Zamietnutie } p_R = \frac{R}{Q} \tag{63}$$

$$p_{acc} = p_{conf} p_A = p_{conf}(1 - p_R) \tag{64}$$

$$\frac{TP+TN}{Q} = \frac{TP+TN}{P+N} * \frac{P+N}{Q} \tag{65}$$

S ako prvými sme začali experimentovať s lineárnymi blokovými kódmi CRC podľa tabuľky 4.1. Výsledky pre CNN-4 architektúru s lineárnou normalizáciou výstupov (24) a Zadehovou rozhodovacou funkciou sú na obrázku 5.4. Presnosť je počítaná podľa (59). Pre meranie spoľahlivosti (60) sme použili detekčné schopnosti binárnych kódov tak, že sme použili celý kódový priestor využívajúc 1024 kódových slov namiesto 10 tried. Ak výstupný kód nie je zhodný s desiatimi kódovými slovami priradené triedam, vzor je zamietnutý ako nedôveryhodný. Jeden bod na obrázku 5.5 znázorňuje *presnosť* - *spoľahlivosť* pre jednu prahovú hodnotu aplikovanú v celom rozhodovacom procese. Rôzne body môžu byť dosiahnuté s rôznymi prahovými hodnotami. Ak žiadny vzor nie je zamietnutý, potom presnosť je rovná spoľahlivosti. Tomu korešponduje bodkovaná čierna čiara v ľavo hore. CRC kódy môžu detekčnou schopnosťou zamietnuť nedôveryhodný vzor a preto môžu mať spoľahlivosť vyššiu aj pri nulovej prahovej hodnote. Výsledky spoľahlivosti klasifikátora môžu byť zlepšené nastavením vyššieho prahu, výmenou za nižšiu presnosť. Experimenty sme pustili na každom CRC kóde (tabuľka [4.1]) a vytvorili graf presnosti a spoľahlivosti. Je možné vidieť, že každý kód dáva inú presnosť a to ponúka ďalšie možnosti ako hľadať najvhodnejší kód. Čo nebolo cieľom hlbšieho študovania v tejto práci. Pre naše účeli sme vybrali kód CRC7 ktorý mal najlepšiu presnosť.



CRC7

CLASS	CODEWORDS
0	1 0 0 0 0 1 0 1 1 0
1	0 0 1 1 1 1 1 0 0 1
2	1 1 0 1 0 0 1 0 1 0
3	0 1 1 0 1 0 0 1 0 1
4	0 0 0 1 0 1 0 1 1 1
5	1 0 1 0 1 1 1 0 0 0
6	0 1 0 0 0 0 1 0 1 1
7	1 1 1 1 1 0 0 1 0 0
8	1 1 0 0 0 1 1 1 0 1
9	0 1 1 1 1 1 0 0 1 0

Obrázok 5.4 Charakteristika presnosti a spoľahlivosti pre rôzne CRC (v ľavo). CRC7 kódové slovo (v pravo)

V ďalšej sekcií popisujem náš hlavný experiment. Vyhodnotili sme rôzne architektúry, výstupné kódovanie a rozhodovanie klasifikátora na základe presnosti a spoľahlivosti. Všetky experimenty sme robili na CIFAR-10 datasete. Všeobecne v metodológii kllasifikačnej úlohy sa priraďuje každému výstupu neurónovej sieti jedna trieda počas trénovania a testovania. Pri použití detekčných kódov môžeme rozšíriť rozhodovanie na celý kódový priestor. Použili sme 10 bitový výstupný kód pre všetky prípady zaznamenané v tabuľke 5.6 okrem Hadamard kódovania, ktorý požaduje minimálne 15 bitov na pokrytie 10 tried a zaistenie tak vlastností opísaných v [96]. To znamená že celý kódový priestor je 1024 možností pre CRC7 a 10 pre one-hot kódovanie. Presnosť (59) sa používa na meranie výkonu vtedy keď všetky kódové slová (výstupné kódy patriace k triedam) patria triedam. Pri použití celého kódovacieho priestoru, môžeme dostať výstupné kódy, ktoré nepatria žiadnej triede. V tomto prípade je použitá detekcia chyby a vzor je odmietnutý ako nedôveryhhodný. Pre meranie tejto charkteristiky sme zaviedli pojem FCA (full code accuracy), ktorý sa používame keď chceme hovoriť o presnosti na plnom kódovom priestore.

Hoci ako primárny cieľ je určiť vlastnosti binárneho kódovania, metóda rozhodovania značne ovplivňuje správnosť klasifikácie a schopnosť použiť plný kódový priestor. V tabuľke 5.6 porovnávame presnosť podľa bitového rozhodnutia (26) a holistického rozhodnutia slova reprezentovaného euklidovskou vzdialenosťou aplikovanej na Hadamard kóde, softmax použitý na on-hot kóde ako aj navrhnuté Zadeh rozhodnutie (aplikované na one-hot a CRC kóde). Najskôr sme testovali všetky prístupy s CNN-4 architektúrou. Pre každú kombináciu sme robili 10 opakovaní tréningu a použili výsledok s najlepšou presnosťou. Potom sme vybrali 4 najlepšie prístupy podľa presnosti a testovali ich na CNN-5 a ResNet20v2 architektúre. Vykonali sme urobili 5 opakovaní tréningu pre každú architektúru a použili najlepší výsledok podľa presnosti.

Náš hlavný záujem je určiť možnosť použitia binárneho kódovania (repreznetovaného CRC7) s chybovou detekčnou schopnosťou. Najskôr sme testovali prístup prahovej bitovej hodnote (26) nakoľko to je priamy prístup rozhodovaniu. Pre tento prístup bola ako účelová funkcia použitá MSE a bitový prah počas testovania. Ďalej sme testovali Zadeh rozhodovanie na CRC7 a one-hot kódovanií. Použitie Zadeh rozhodovania s one-hot kódovaním je ako použitie binárneho kódovania, kde kódové slová majú one-hot vlasnosti. Pre porovnanie používame one-hot kódovanie so softmax normalizáciou ako základnú metódu. Inšpirovaný [83] sme tiež testovali kódovanie CRC7 so softmax. To umožňuje kombinovať výhody

binárnych kódov a softmax rozhodovania ale pri neschopnosti detekcie chýb. Napokon sme otestovali prístup [96] kde euklidovské rozhodnutie je použité na Hadamard kódovaní. Výsledky naznačujú prevahu rozhodovania Zadeh, softmax, Euklid nad rozhodnutiami na individuálnych bitoch. Môžme vidieť že CRC7 so Zadehovým rozhodnutím dosahuje podobnú presnosť ako one-hot kódovanie so softmax. V one-hot kódovaní so softmax, kódová štruktúra a rozhodovacie pravidlo sa navzájom podporujú. Vieme že len jeden bit je rovný "1" a výstupné hodnoty sú normované ako pravdepodobnosť.

Rozhodnutie a trénovanie založené na euklidovskej vzdialenosti zlepšuje v priemere výstupové hodnoty a taktiež vidíme priemerné výsledky na Hadamardových kódových slovách, napriek použitiu kódových slov dlhých 15 bitov. Podľa Zadehovej fuzzy logiky rozhodnutie má tendendiu znižovať maximálnu výstupnú chybu danú triednemu kódovému slovu počas trénovania. Počas tohto testu je výstup priradený kódovému slovu z minimom maximálnych bitových chýb. Jendoducho povedané zatial čo one-hot, softmax ťahá bitovú hodnotu k "1" navrhované Zadeh rozhodnutie ťahá najhoršiu chybu na "0". Tabuľka 5.6 ukazuje na oboch stratégiách podobné hodnoty a môžeme usúdiť, že to čo dáva softmax one-hot enkódovaniu, dáva Zadehova fuzzy logika binárnemu kódovaniu.

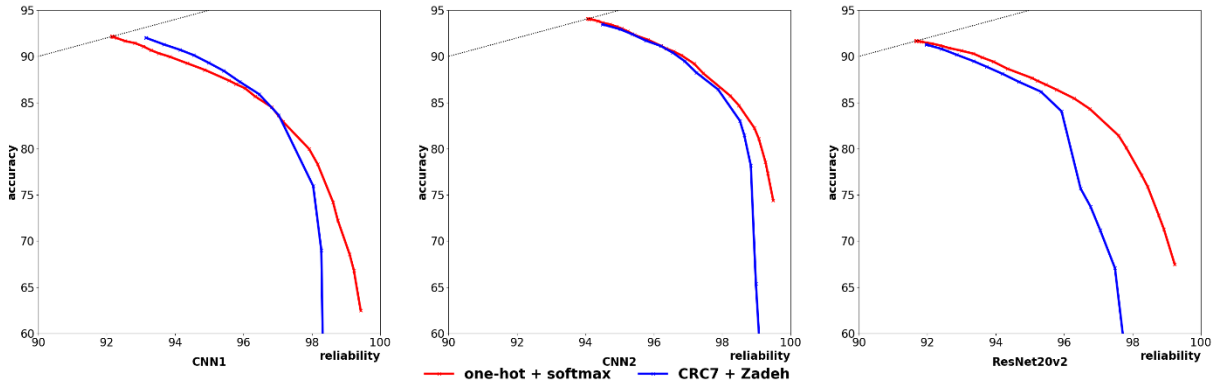
Architektúra	Výstupný Kód	Rozhodnutie	Presnosť	FCA	Nedetekova-teľná Chyba	Spoľahli-vosť
CNN1	CRC7	Zadeh	92.44	92.01	6.77	93.15
CNN1	one-hot	Zadeh	92.01	91.56	7.29	92.63
CNN1	one-hot	softmax	92.15	92.15	7.85	92.15
CNN1	CRC7	softmax	91.85	91.85	8.15	91.85
CNN1	CRC7	bit threshold	89.5609	89.5609	6.87	92.894
CNN1	Hadamard	Euclid	89.92	88.6	8.2	91.53
CNN2	CRC7	Zadeh	93.97	93.47	5.44	94.5
CNN2	one-hot	Zadeh	93.4	92.66	5.76	94.14
CNN2	one-hot	softmax	94.07	94.07	5.93	94.07
CNN2	CRC7	softmax	93.88	93.88	6.12	93.88
ResNet20v2	CRC7	Zadeh	91.56	91.27	7.97	91.97
ResNet20v2	one-hot	Zadeh	84.16	83.28	14.5	85.17
ResNet20v2	one-hot	softmax	91.67	91.67	8.33	91.67
ResNet20v2	CRC7	softmax	91.34	91.34	8.66	91.34

Tabuľka 5.6 Výsledky rôzneho kódovania a rozhodovacích stratégii

V Prípade Zadeh rozhodnutia môžeme ponúknuť viac kódových slov. To ale znižuje FCA presnosť. Rozdiel medzi presnosťou na 10 bitoch naznačuje, koľko vzoriek bolo priradených ku kódovým slovám mimo tried. Ak sú použité iba kódové slová tried, neexistuje rozdiel

medzi presnosťou a FCA. To je prípad one hot kódovania so softmax kde môže byť použité len kódové slovo triedy. Teda ak klasifikácia ponúkne jedno z kódových slov, musí byť akceptované. Nesprávne klasifikované kódové slovo spôsobuje nedetekovateľnú chybu. Tabuľka 5.6 naznačuje niektoré benefity chybnej detekcie. Vylúčenie nedôveryhodných vzoriek zvyšuje klasifikačnú spoľahlivosť.

Dobre známy spôsob ako zvýšiť spoľahlivosť je zaiesť úroveň rozhodovania. Všetky roshodovacie pravidlá sú založené na hodnotení kvantitatívnej hodnoty premenej. Obvyklý spôsob na zlepšenie spoľahlivosti je rozdeliť interval premenej prahom Δ , odmietnuť všetky vzorky pod prahom a obísť rozhodovaciu procedúru. Toto pravidlo môže byť tiež aplikované na softmax koncept v ktorom premenná je v rozmedzí $[1n\diagup,1]$ kde n je počet tried. Teda prahové nastavenie pod $1/n$ nemá žiadny význam a nedáva žiadne zlepšenie. V softmax rozhodnutí je vzorka akceptovaná pre klasifikáciu ak hodnota výstupu je vyšia ako prah. Môžeme aplikovať tento istý prístup v Zadehovom rozhodnutí a rozhodnúť ak $max\{S0,\ldots,Sw\}\geq\Delta$. Theorém 1 hovorí že prah $\Delta<0.5$ neprináša žiadne zlepšenie v plnom kódovom rozhodovaní. Na obrázku 5.5 ukazujeme že graf presnosť-spoľahlivosť s použitím rozličných prahových hodnôt na navrhnuté CRC7 zo Zadehovým rozhodnutím a základným one-hot softmax kódovaním.



Obrázok 5.5 Charakteristika presnosti a spoľahlivosti podľa one-hot + softmax a CRC7 + Zadeh rozhodovaní pri použití prahovej hodnoty

Obrázok 5.5 zobrazuje hodnoty z tabuľky 5.6 ktoré sú na ľavej strane, kde nie je aplikovaný žiadny threshold. Ako threshold narastá body sa posúvajú na pravo. Vlasnosť detekcie chybových kódov posúva nulové hodnoty prahu mimo líniu $y=x$. Môžeme vidieť že obe metódy poskytujú podobné výsledky.

Detekcia chyby mimo distribúcie.

Detekcia chýb je prístup ktorým sa určuje či kódové slovo sa zhoduje s vlasnosťami ktoré definujú konkrétny podpriestor kódového priestoru [84]. Ako trénujeme sieť na trnsformovanie tréningových vzorov do kódových slov detekujúcich chybu predpokladáme že umiestnime tieto vzory do podpriestoru udržujúceho schopnosť chybovej detekcie. Majme hypotézu že ak vzorka je kvalitatívne odlišná od tréningových vzorov, výstupové kódové slovo bude mať iné vlastnosti, napr. bude ležať mimo kódového podriestoru generovaného tréningovou sadou. Hoci hlavný cieľ je ukázať že binárne kódovanie môže byť použité na natrénovanie dosiahnutia vysokej presnosti, detekcia mimo distribúcie "*outlier"* je postranný efekt.

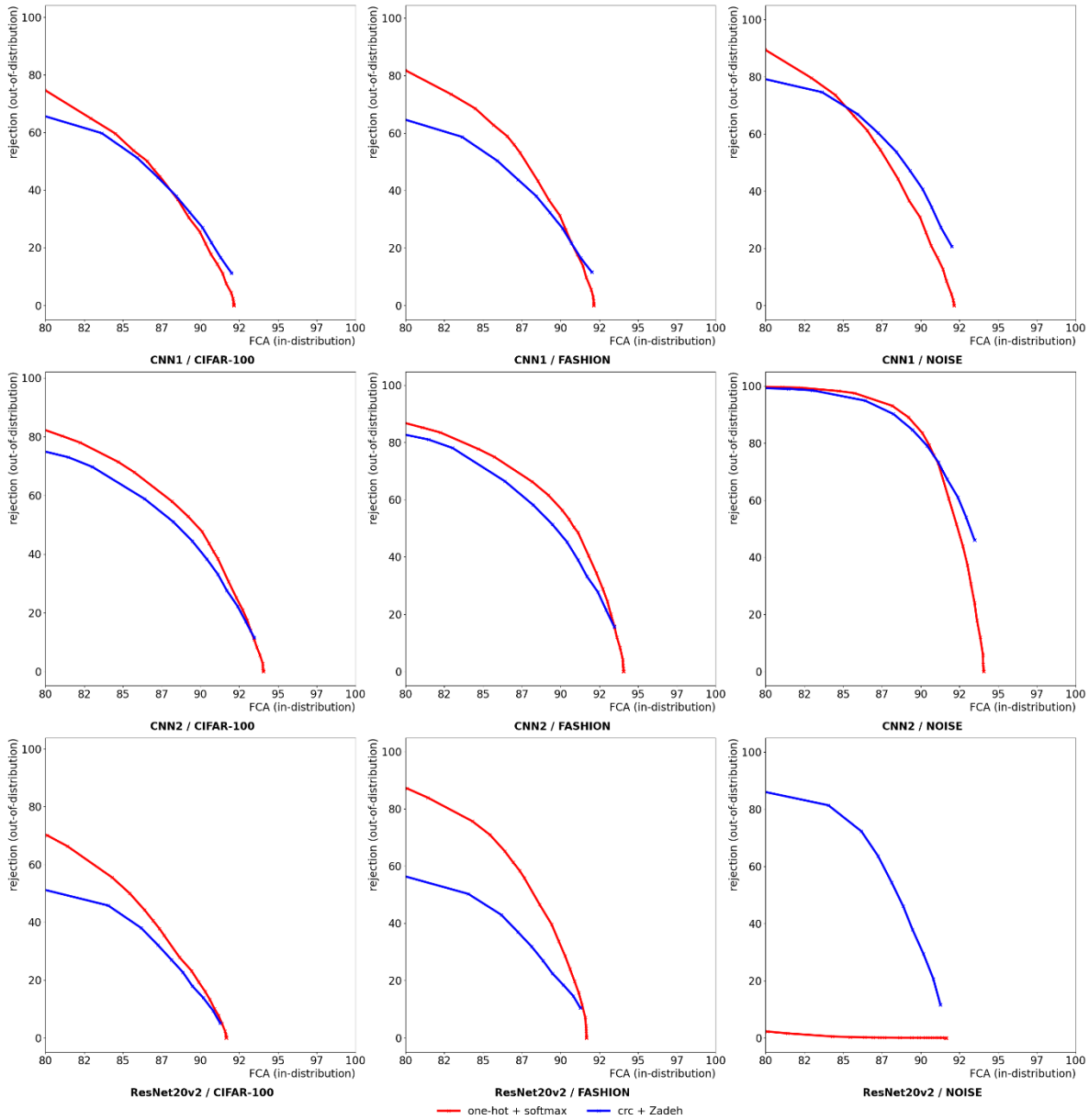
V našom prípade tréningová sada pre "*v rozdelení*" bol CIFAR-10 dataset. Ako "*mimo rozdelenia*" dáta sme si vybrali datasety s rôznymi zložitosťami od najjednoduchších (MNIST, Fashion MNIST), cez podobné (CIFAR-100) až po najzložitejšie (biely šum). Prirodzená (a veľmi nežiadúca) vlasnosť rozpoznávacich systémov ktorá zobrazuje všetky vstupné vzory do jednej z výstupových tried je ich bezbrannosť voči "*outlier"*.

U nás je to prípad softmax rozhodovania. Na druhej strane kódy ktoré detekujú chybu majú implicitnú schopnosť filtrovať kódové slová mimo podpriestoru kódových slov tried. Tento prístup vytvára odmietnuté kódové slová a zlepšuje spoľahlivosť. Tabuľka 5.7 ukazuje presnosť zachytávať "*outlier"* z rozličných mimodistribučných datasetov. Môžme vidieť že "*outlier"* sú prirodzene detekované sieťami trénovanými priamo z chybovej detekcie, dokonca bez aplikovania prahu.

Architektúra	Výstupný kód	Rozhodnutie	MNIST	Fashion MNIST	CIFAR-100	Šum
CNN1	CRC7	Zadeh	10.29	11.59	11.16	20.6
CNN1	one-hot	softmax	0	0	0	0
CNN2	CRC7	Zadeh	12.37	15.55	11.56	46.05
CNN2	one-hot	softmax	0	0	0	0
ResNet20v2	CRC7	Zadeh	10.77	10.49	5.09	11.61
ResNet20v2	one-hot	softmax	0	0	0	0

Table 5.7 Zamietnutie na datasete mimo distribúciu bez prahovej hodnoty

Podobne ako spľahlivosť, presnosť mimo distribúcie dát môže byť zlepšená aplikovaním prahovej hodnoty počas rozhodovania. Obrázok 5.6 ukazuje že odmietnutie mimo distribúcie narastá s prahovou hodnotou. Pre CNN-4/CNN-5 architektúry s CIFAR-100, Fashion MNIST datasete môžeme vidieť podobný vzor v charakteristike presnost-spoľahlivosť.



Obrázok 5.6 Výkon klasifikátorov na rôznych architektúrach a mimo distribučných datasetoch po aplikovaní prahovej hodnoty

Pre vysoké FCA úrovne v rámci distribúcie, presnosť v obch metódach je veľmi podobný. Pre nižsie presnosti v rámci distribúcie, one-hot kódovanie so softmaxom poskytuje lepší výsledok mimo distribúcie. CRC7 so Zadeh rozhodnutím má lepšie detekčné vlastnosti na

datasete bieleho šumu hlavne pre ResNet20v2 architektúru. Naproti tomu one-ho kódovanie so softmaxom ukazuje lepší výkon v zvyšku Resnet20v2 experimentoch.

5.1 Experimenty využívajúce opravné CRC kódy pre anotáciu

Počas doktorandského štúdia sme pri skúmaní novej organizácie výstupného priestoru vykonali doposiaľ približne 15 000 experimentov. Ukázali nielen slepé vetvy architektúr (GAN), ale aj vhodnejšie či menej vhodné hyperparametre neurónovej siete či účelové funkcie.

Referenčným experimentom je najčastejší spôsob trénovania a vyhodnocovania neznámeho vzoru, ktorý označujeme „softmax“ (P0: one-hot coding + softmax) v tabuľke 5.8. Tento prístup spočíva v použití one-hot kódovania a normovania výstupov neurónovej siete do tvaru rozdelenia pravdepodobnosti. V takto upravených výstupoch je maximálna hodnota použitá nielen na určenie triedy vzoru, ale aj jej vzdialenosť od hodnoty 1 je bratá ako chyba výstupu ktorá je minimalizovaná počas trénovania. Softmax tak využíva vnútornú informáciu o štruktúre one-hot kódu nielen na vyhodnocovanie výstupu, ale aj počas trénovania siete. Pri všeobecnom delení priestoru výstupov takéto obmedzenie kódom neexistuje, preto sme urobili dva experimenty (tabuľka 5.8), z ktorých v prvom informácia o kóde nie je využívaná počas trénovania (P1: one-hot coding + vyhodnotenie threshold 0,5) a v druhom ani počas vyhodnocovania (P2: one-hot coding + one hot vyhodnotenie). V obidvoch experimentoch boli výstupy z poslednej vrstvy normované do jednotkového intervalu nelineárnou funkciou sigmoid (23) a v účelovej funkcii bola použitá stredná kvadratická chyba (MSE).

V experimente P1 boli bity určované zaokrúhlením výstupov na prahovej úrovni 0,5. Na rovnosť medzi výstupným kódom a kódom triedy bola potrebná zhoda vo všetkých desiatich bitoch. Hoci spôsob rozhodovania je zhodný so spôsobom pri použití Zadehovej fuzzy logiky, rozdielny je spôsob trénovania siete. V tomto prípade bola pri trénovaní použitá v účelovej funkcii druhá mocnina Euklidovej vzdialenosti medzi výstupom a kódovým slovom (ako vektorom v reálnom priestore). Tento postup odpovedá všeobecnému deleniu výstupného priestoru podľa určujúcich kódových slov. Z úspešnosti na testovacej sade (Tabuľka 5.8) vidíme, že nevyužitie vnútornej informácie o štruktúre kódu vedie k zhoršeniu približne o 2%. Z tohto pohľadu je potrebné posudzovať aj úspešnosti ďalších delení

výstupného priestoru, ktoré na delenie priestoru používajú prahovú hodnotu 0,5 a nevyužívajú vnútornú informáciu o štruktúre kódu.

					úspešnosť [%]	
	experiment	**aktivácia**	**vyhodnotenie**	**účelová funkcia**	**priemer**	**max**
P0	one-hot coding + softmax	softmax	one hot (max)	Categorical crossentropy	85,88	86,07
P1	one-hot coding + vyhodnotenie threshold 0,5	sigmoid	threshold 0.5	MSE	83,75	83,82
P2	one-hot coding + one hot vyhodnotenie	sigmoid	one hot (max)	MSE	87,96	88,13
P3	one-hot coding + threshold 0,5 + negatíva	žiadna	threshold 0.5	pozitiva+negativa (buffer 500)	83,78	84,29
P4	one-hot coding + min-max (10 bitov)	sigmoid	max-min	MSE	88,34	88,47
P5	CRC4 + threshold 0,5 (10 bitov)	žiadna	threshold 0.5	MSE	85,04	85,85
P6	CRC4 + threshold 0,5 + negatíva, (10 bitov)	žiadna	threshold 0.5	pozitiva + negativa (buffer 500)	85,08	85,6
P7	CRC4 + min-max (10 tried)	sigmoid	max-min	MSE	88,41	88,71
P8	CRC4 + threshold 0,5 + negatíva, (opravný kód)	žiadna	threshold 0.5	pozitiva + negativa (buffer 500)	85,85	86,23

Tabuľka. 5.8 Experimenty využívajúce oprávné kódy a one-hot prístup a ich výsledky

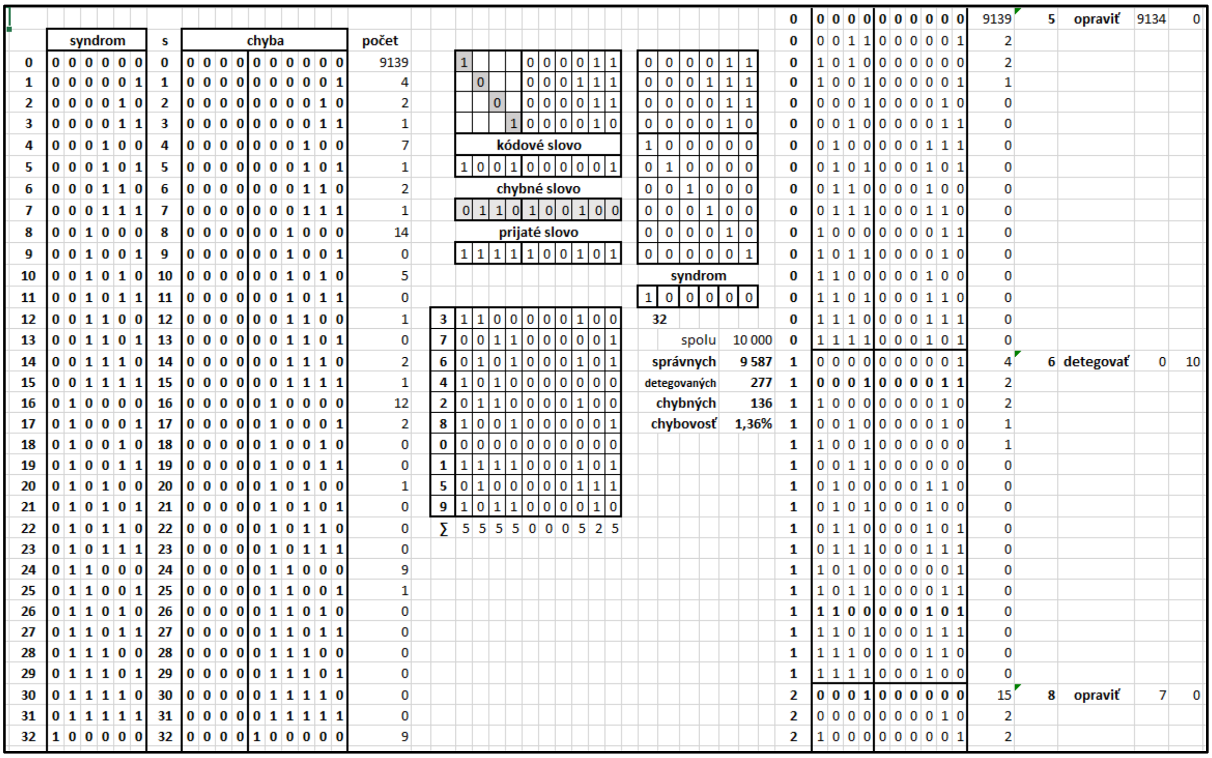
V experimente P2 bol výstup dekódovaný tak, že najväčšiemu výstupu bola priradená hodnota bitu 1, ostatným výstupom hodnota 0. V účelovej funkcii bol použitý výpočet chyby ako druhá mocnina Euklidovej vzdialenosti medzi celým výstupom a celým kódovým slovom. Výsledky poukazujú na to, že informácia o štruktúre kódu je rozhodujúca vo fáze vyhodnocovania. Napriek tomu, že táto informácia nebola použitá počas trénovania, presnosť na testovacej množine je dokonca lepšia ako v referenčnom experimente P0.

Experiment P3 vychádza z experimentu P1. Oproti nemu navyše využíva princíp kontrastných strát, ktorý bol popísaný v metodike práce. Tento spôsob využíva v účelovej funkcii nielen atraktory, t.j. body ku ktorým má výstup neurónovej siete pre vstupný vzor konvergovať (nazývame ich pozitívnymi vzormi), ale aj body, ktorým sa má výstup pri trénovaní vyhýbať (my ich nazývame negatívami). Oproti pôvodnému návrhu, v ktorom sa použilo jedno pozitívum a jedno negatívum, tento experiment využíva viacero negatív, ktoré sú uložené vo vyrovnávacej pamäti s kapacitou pre 500 negatív. Ako negatívne vzory používame kódové slová v ktorých výstup po tréningu skončil chybne (false positive). Našou

snahou bolo, aby od týchto chybných kódových slov bola neurónová sieť počas tréningu odpudzovaná. Chybné kódové slová sa pre daný vstupný vzor počas učenia ukladajú do vyrovnávacej pamäte s nádejou, že neurónová sieť sa bude týmto chybám vyhýbať a adaptívne sa učiť na vlastných chybách. Aby bol vplyv jedného pozitívneho atraktora a viacerých negatívnych vzorov vyvážený, požívame váhovací koeficient C (v tomto experimente C=0,1). V porovnaní s experimentom P1, ktoré využíva rovnaký spôsob vyhodnocovania, vidíme len nepatrné zlepšenie. Je možné, že ďalším ladení hyperparametrov by sa tento postup vylepšil.

Experiment P4 využíva one-hot kódové slová ako reprezentantov tried, v čom je rovnaký ako predchádzajúce experimenty. Pre dekódovanie výstupu neurónovej siete používa Zadehovu fuzzy logiku. Tento nový prístup k dekódovaniu výstupov neurónovej siete, ktorý sme použili pri použití detekčných CRC kódov pri anotácii, sa ukazuje ako nádejný aj pri použití opravných CRC kódov. Pomáha pri trénovaní siete a rovnako ako softmax alebo max v experimentoch P0 a P2, pomáha aj pri vyhodnocovaní výstupu u neznámeho vzoru. Zadehova fuzzy logika (*min* – konjunkcia, *max* – disjunkcia) spôsobuje, že gradientom sa spätne šíri oprava od výstupu s najväčšou chybou u kódového slova s najmenšou chybou, inými slovami opravujeme najväčšiu chybu u najlepšieho slova. Tým si vysvetľujeme (podobne ako u *softmax*-u) vysokú rýchlosť učenia.

Pokiaľ v experimentoch P1–P4 išlo o štúdium rôznych prístupov pri použití one-hot kódovania, ktorý spolu s výstupom softmax považujeme za referenčný, v experimentoch P5, P6, P7 sme aplikovali tieto prístupy na všeobecné binárne kódové slová. Tieto experimenty boli prípravou na použitie detekčných a opravných mechanizmov kódu CRC. Pri voľbe cyklického kódu sme vychádzali z toho, že one-hot kód je 10 bitový (zachováme rovnakú dimenziu priestoru) a na kódovanie 10 tried sú postačujúce 4 informačné bity. Nasledujúci obrázok ukazuje vytvárajúce polynómy (10,4) cyklických kódov. Rozhodli sme sa pre ireducibilné polynómy, ktoré sú zároveň primitívnymi a po predbežných testoch sme vybrali kód CRC4 (neskoršie experimenty s detekčnými CRC kódmi ukázali ako vhodnejší pre detekciu kód CRC7). Zo 16 kódových slov sme vybrali 10 kódových slov tak, aby počet dvojíc medzi ktorými je Hammingova vzdialenosť 3 bity bol minimálny. Tieto kódové slová sme používali ako štartovacie priradenie triedam trénovacej množiny CIFAR-10. Aj počas iného rozdelenia výstupného priestoru v ďalších iteráciách trénovania, neurónová sieť si zachovávala kódové slová CRC4 ako vedúce kódové slová príslušných tried.



Obrázok. 5.7. Opravná schopnosť CRC4 kódu na trénovacej podmnožine 10 tisíc vzorov

Experiment P5 je modifikáciou experimentu P1, v ktorom slová one-hot kódu boli nahradené slovami CRC kódu. Mierne zlepšenie si vysvetľujeme tým, že pokiaľ u one-hot kódu je Hammingova vzdialenosť medzi všetkými kódovými slovami 2 bity, u zvoleného CRC4 kódu je minimálne 3 bity a väčšinovo 4 bity.

V experimente P7 sú podmienky zhodné s experimentom P4. Rozdielne sú len kódové slová, ktoré sú reprezentantmi jednotlivých tried. Pokiaľ v P4 to boli one-hot kódové slová, v P7 sú to všeobecné binárne kódové slová reprezentované CRC kódom. Vidíme, že použitie Zadehovej fuzzy logiky má priaznivý vplyv na presnosť rozpoznania nezávisle na použitom kóde. Avšak vzhľadom na to, že sme jej použitie našli až v ostatnom období, jej výsledky pre väčší počet všeobecných kódových slov ako je tried bude preukázané až pri obhajobe dizertačnej práce, spolu s uvedenou metodikou návrhu suboptimálneho kódu. Bola vyskúšaná aj pravdepodobnostná fuzzy logika (súčin $x.y$ pre konjunkciu, pravdepodobnostný súčet $x+y-x.y$ pre disjunkciu), avšak pri nej sa gradient rozprestiera medzi viacero výstupov, takže učenie nie je také účinné ako pri Zadehovej fuzzy logike, kde sa gradient sústreďuje na najhorší výstup. Na druhej strane nevýhodou funkcie minima pri konjunkcii je, že najhorší výstup určuje hodnotenie rovnakého bitu u všetkých kódových slov. Pokiaľ u niekoľkých kódových slov má bit rovnakú požadovanú hodnotu, výsledok konjunkcie bude u nich

rovnaký a medzi nimi nebudeme vedieť rozhodnúť. Počas učenia to nevadí, pretože sa budú rovnako zlepšovať všetky tieto kódové slová, avšak pri vyhodnocovaní to spôsobuje nerozhodnuteľnosť. Riešenie môže byť v použití sekundárneho rozhodovacieho kritéria (napr. Manhattan vzdialenosť), alebo použitia Zadehovaj fuzzy logiky počas trénovania a pravdepodobnostnej fuzzy logiky počas vyhodnocovania. Na určenie vhodnejšieho prístupu budú potrebné ďalšie experimenty.

Experiment P8 je nosným experimentom pre dosiahnutie cieľa dizertačnej práce, ktorým je zmenšenie množstva anotovaných dát, resp. zmenšenie úsilia na ich anotáciu. Jeho zámerom je preskúmať možnosť použitia bezpečnostných kódov na tento cieľ. Detekčné a opravné kódy môžu určiť, či daný vzor je informačne významný alebo nie. Pokiaľ testovaný vzor sa stotožní s niektorým existujúcim reprezentantom triedy, ide buď o správne rozpoznanie, alebo neodhaliteľnú chybu. Medzi týmito prípadmi nevieme rozhodnúť ak skúšaný vzor má neznámu triedu. Výsledky na testovacej sade ukážu aká je pravdepodobnosť takejto neodhaliteľnej chyby. Očakávame úspešnosť nad 80%, čo nás oprávňuje k tomu, že vzory, ktoré sú priradené ku kódovému slovu triedy budeme považovať za vzor, ktorý nie je potrebné anotovať. Na druhej strane vzory, ktoré sú priradené kódovým slovám ktoré nereprezentujú žiadnu triedu budeme považovať za informačne dôležité a z nich vyberieme typické vzory (okolo nového kódového slova), ktoré je potrebné označiť triedou. Pokiaľ použijeme opravný kód aj na samotné rozpoznávanie, vzory ktoré sieť priradí kódovým slovám ktoré kód opravuje (priraďuje triedam), tieto vzory rovnako nie je potrebné anotovať.

Rozdeľovanie priestoru prebiehalo v iteráciách, kedy po každej iterácii boli triede priradené nové kódové slová. V prvej iterácii boli triedam priradené kódové slová kódu CRC4 a sieť bola trénovaná metódou k-fold (k=10). Touto metódou sa na konci 1. iterácie ukázalo ako vyhodnocuje sieť trénovaná na 45 000 vzoroch zvyšných 5 tisíc vzorov. Tento výsledok bol použitý na zostavenie frekvenčnej tabuľky chýb od najčastejších chýb až po najzriedkavejšie. Najčastejšie boli vzory správne priraďované kódovým slovám CRC4 kódu. Druhý najčastejší prípad boli „false positive“ chyby, kedy vzory boli síce priradené slovu CRC4 kódu ale nesprávnemu. Pri reorganizácii výstupného priestoru sme postupovali od najčastejších chýb a týmto slovám sme priradili kódové slovo ktoré bolo iné ako nesprávne zvolené slovo, ale malo k nemu najmenšiu Hammingovu vzdialenosť, pokiaľ takéto slovo už nebolo obsadené vzormi s väčšou chybovosťou. Takto sme postupovali až po obsadenie všetkých 1 024 kódových slov.

V ďalších iteráciách boli kódové slová do tried opätovne prerozdeľované. Počty kódových slov v jednotlivých triedach dáva tabuľka 5.9.

	rozdelenie po iterácii									
trieda	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	114	12	8	11	5	13	8	4	4	4
1	83	3	2	3	2	5	2	3	3	3
2	116	7	10	5	10	18	13	9	6	8
3	181	10	13	10	10	12	13	9	8	11
4	119	8	6	6	9	8	4	5	4	4
5	94	5	8	9	6	10	8	11	5	8
6	96	4	1	2	4	9	3	2	2	6
7	66	3	2	2	4	8	8	4	5	2
8	76	5	2	5	5	11	6	3	2	3
9	79	6	4	5	8	13	8	4	3	5

Tabuľka. 5.9. Počet kódových slov priradených do jednotlivých tried

Vidíme, že v ďalších iteráciách nemajú vzory tendenciu obsadzovať veľkú časť priestoru a zhlukujú sa okolo menšieho počtu (v závere 54) kódových slov. Počet kódových slov na jednu triedu kolíše po poslednej iterácii od 2 do 11 kódových slov (s priemerom 5,4).

Vývoj úspešnosti po jednotlivých iteráciách uvádza tabuľka. 5.10. Tieto výsledky môžeme porovnať

iterácia	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
úspešnosť	85,5	85,73	85,89	85,84	85,07	85,94	86,03	86,1	86,18	86,23

Tabuľka. 5.10. Vývoj úspešnosti po iteráciách prerozdeľovania kódových slov vo výstupnom priestore.

s výsledkom experimentu P6, ktorý prebiehal za podobných podmienok. Komparácia hovorí o tom, že využívanie väčšej časti výstupného priestoru vedie k malému zvýšeniu úspešnosti, ktoré je aj porovnateľné so zvýšením úspešnosti medzi prvou a desiatou iteráciou.

Do prezentácie dizertačnej práce bude vykonaný experiment P8 modifikovaný experimentom P7, v ktorom bola použitá Zadehova fuzzy logika.

6 Záver

Metódy strojového učenia v posledných rokoch ukázali že sa dokážu vyrovnať s takými úlohami, ktoré sú pre človeka intuitívne a jednoduché (napr. rozpoznávanie tváre) ale sú veľmi ťažko interpretovateľné v algoritmickom jazyku. Rozpoznávanie obrazu je jednou z domén, kde strojové učenie, konkrétne hlboké neurónové siete, dosahuje dobré výsledky, ktoré vedia prekonať už aj človeka. V tejto práci som sa zaoberal rozpoznávaním obrazu na úrovni tvorby trénovacieho dátového súboru tak, aby úsilie človeka bolo pri jeho vytváraní čo najmenšie. Cieľom bolo navrhnúť postupy a metódy na zefektívnenie procesu získavania anotovaných dát pre potreby vývoja metód počítačového videnia založených na hlbokom strojovom učení. Ako už bolo spomínané, tvorba nového datasetu je únavná, monotóna práca a citlivá na ľudské chyby. Je ťažké anotovať ten istý typ objektu stovky krát za sebou v jednotkách sekúnd určením vzoru a potom si všimnúť iný podobný objekt ktorý nepatrí do množiny. Na to aby sme dokázali eliminovať chyby ktoré vytvárame pri tvorbe datasetu je nutné kontrolovať dáta opakovane. Až po určitom počte opakovaní je možné prehlásiť anotáciu za dostatočne dôveryhodnú. Na takýto prístup je potrebné veľké množstvo ľudí (pri ImageNet s AMT to boli desiatky tisíc), ktorí musia byť zaplatení, inštruovaní a musia sa pre nich vytoriť softvérové nástroje. Preto sa výskum v tejto oblasti zameral na zefektívnenie procesu anotovania. Niektoré prístupy sa snažili zredukovať množstvo ľudskej práce na minimum a zjednodušiť samotnú úlohu. Stále však prístup učenia s učiteľom ostáva pri klasifikácii rovnaký a to, že na výstupe sa objaví trieda objektu ktorú určí model. Anotátor potom musí potvrdiť alebo vyvrátiť tento výstup. Neurónová sieť nemá priamu podporu ako povedať že si nie je istá výsledkom, alebo že nevie. Neznámu triedu môžeme určiť len na základe miery ohodnotenia výsledku. Existujú prístupy, v ktorých môžeme vytvoriť novú triedu pre objekty neznámej neznámej triedy, ale na tento prístup potrebujeme dáta ktoré chceme považovať za neznáme, čo zvyšuje nároky na vytváranie dátového súboru. Bolo by však dobré, keby model dokázal priamo určil výstup neurónovej siete o ktorom vieme povedať že vstup patrí do iného rozdelenia resp. triedy. Takouto úvahou sme boli vedení aj v tejto práci. Snažili sme sa pre výstup neurónovej siete neučiť jeden bod v priestore (ako je to pri one-hot kódovaní), ale naučiť model aby na výstupe dával rozdelenie do viacerých bodov ktoré môžeme potom ďalej vyhodnocovať. Za základ sme si zobrali binárne kódovanie pomocou ktorého kódové slová určujú výstupné body priradené jednotlivým triedam. Tento prístup nám ponúka použiť teóriu kódovania na vyhodnocovanie výstupov. Naše zámer bol

naučiť neurónovú sieť viacero binárnych kódových slov ktoré reprezentujú vstupy danej triedy, namiesto toho aby sme učili len jeden výstup ako je to pri one-hot kóde. Takto naučená sieť nám dáva možnosť, že pri neznámom alebo ťažko rozponateľnom vzore dostaneme na výstupe modelu iné kódové slovo ako je jedno hlavné priradené triede, alebo dokonca slovo ktoré sme nepriradili žiadnej triede. V tejto práci sme sa snažili naučiť CNN túto vlastnosť, a potom ju využiť pri anotovaní novej dátovej sady, alebo pri kontrole už existujúcej. Kedže trénovanie modelu nie je záležitosť jedného tréningu ale je potrebný systematický postup od vytvárania datasetu po ladenie hyperparametrov až po niekoľkonásobné natrénovanie konečného modelu, snažili sme sa zefektívniť tento postup jednoduchým prístupom popísaným v kapitole 4.3. Ukázalo sa, že trénovanie viacerych inštancií toho istého modelu a ich zastavenie v skorom štádiu trénovania a pokračovania trénovania iba u najlepšieho modelu, zvyšuje rýchlosť nájdenia lepšieho riešenia pri ušetrení trénovacieho času. Taktiež sme sa zamerali na samotnú anotáciu a snažili sme sa priniesť pohľad na to, ako je časovo aj výpočtovo náročné robiť anotáciu ktorá je len ponúkaná na kontrolu, alebo ktorú treba namáhavým spôsobom vykonať. Tieto príspevky priamo neponúkaju riešenie nášho cieľa ale ukazujú ako sa dajú niektoré procesy zefektívniť a potvrdzujú že anotácia človekom je drahá úloha. Za hlavný príspevok tejto práce považujeme učenie neurónovej siete na klasifikáciu obrazov pomocou binárnych kódov. Bola to náročná úloha, pretože one-hot kódovanie so softmax rozhodovacou funkciou je dlhoročný spôsob používania hlbokých neurónových sietí na klasifikačné úlohy, s ktorým sú bohaté skúsenosti pri optimalizácii sieťových architektúr aj hyperparametrov, čo umožňuje dobré výsledky ktoré sme ťažko dosahovali. Prvý náš prístup ako naučiť CNN rozdelenie binárnych kódov bolo učiť sieť ako GAN, kde generátor zastupoval klasifikátor so vstupnými obrázkami namiesto šumu a výstup bol náš binárny vektor. Aj keď sa javia GAN siete ako vhodné z pohľadu učenia rozdelenia, my sme sme tento názor opustili z dôvodu ťažkého a náročného tréningu. Tréning bol nestabilný a zdĺhavý a navyše neponúkal také výsledky ako spomínaný one-hot + softmax. Ďalším spôsobom a zároveň najdlhšie skúmaným ako naučiť CNN binárne kódovanie bol tzv. prahový prístup. V tompto prístupe sme sa pokúšali učiť CNN pomocou rôznych regresných funkcií ako je MSE, MSA, Čebyšev,..., tak, aby sme na výstupe dostali požadovaný kód. Tento prístup fungoval len čiastočne, pretože výsledná sieť dokázala na natrénovanej množine rozpoznávať dáta, ale jej úspešnosť stále zaostávala za softmax výsledkami. Pri týchto experimentoch sme sa pokúšali meniť aj učiaci sa binárny kód v iteráciách aby čiastočne natrénovaná sieť určila rozdelenie a potom sme sa snažili toto rozdelenie pomocou binárnych kódov sieť doučiť. Bohužiaľ, aj pri naprogramovaní desiatok

algorimov a vykonaní desiatok tisíc experimentov ktoré mali rozdeľovať priestor, alebo hľadať lepšie kódy, sa nám nepodarilo natrénovať algoritmus ktorý by bol porovnateľný so softmaxom alebo mal iné významnejšie vlastnosti ako je napr. odhaľovanie anomálnych dát. Preto sme museli upustiť od tohto prístupu aj keď sme mu venovali najviac času a hľadať iné riešenie. Na one-hot softmax sa môžeme pozerať ako na špecíalny prípad binárneho kódu ktorý je učený len jedným bitom a jeden výstup je do neho pri učení tlačený. Tým pádom gradient daného vzoru pri učení ovplyvňuje len jeden bit a nie všetky ako je to napr. pri MSE. Preto náš ďalší postup bol v hľadaní spôsobu ako trénovať binárny kód pomocou jedného bitu. To nám ponúkla Zadehova fuzzy logika ktorá nám pomáha tlačiť najväčšiu chybu kódového slova smerom k nule. Tento prístup sa osvedčil a v niektorých prípadoch ukázal aj mierne zlepšenie. Jeho hlavným prínosom teda je, že umožnil natrénovať CNN podľa rozdelenia binárneho kódu rovnako dobre softmax u one-hot kódovania. Hoci výsledky dokumentujú len použitie CRC kódov, metodická časť obsahuje aj návrh suboptimálneho opravného kódu ktorý minimalizuje pravdepodobnosť chyby nesprávnej opravy. Predpokladáme, že výsledky experimentov budú prezentované v rámci obhajoby dizertačnej práce. Samotné anotovanie a ukážka anotácie takéhoto datasetu sa už nevošla do tejto práce, pretože rozsah vysvetlenie učenia binárnych kódov a dosiahnutých výsledkov zabralo väčšinu práce. Naše navrhované riešenie nie je úplne hotové pre priamu implementáciu anotácie, ale ponúka príspevok ako sa inak pozerať na dáta počas anotovania nového datasetu alebo hľadania anomálií v ňom. Ukázali sme, že metóda ktorá dokáže odhaľovať dáta dôležité pre anotáciu má potenciál znížiť množstvo práce, prípadne iniciovať hľadanie nových dát ktoré sa javia z pohľadu binárneho kódu ako odlišné.

7 Literatúra

[1] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*: Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE, 86(11):2278-2324

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *IEEE Computer Vision and Pattern Recognition (CVPR), 2009.*

[3] C. Fellbaum.WordNet: An Electronic Lexical Database. BradfordBooks, 1998.

[4] Šíma, Jiří a Roman Neruda. *Teoretické otázky neuronových sítí*. Vyd. 1. Praha: Matfyzpress, 1996.

[5] Hubel, D. H., & Wiesel, T. N. (1959). *Receptive fields of single neurones in the cat's striate cortex. Journal of Physiology*, 148(1), 574–591.

[6] Zeiler, M. D., & Fergus, R. (2013). *Stochastic pooling for regularization of deep convolutional neural networks*. arXiv 1301.3557.

[7] Sainath,T.N.,Kingsbury,B.,Mohamed,A.,Dahl,G.E.,Saon,G.,Soltau,H,.Ramabhadran, B. (2013). *Improvements to deep convolutional neural networks for LVCSR*. In2013 IEEE Workshop

[8] Simoncelli,E.P.,&Heeger,D.J.(1998*). A model of neuronal responses in visual area MT.* Vision Research, 38(5), 743–761.

[9] Hyvärinen,A.,&Köster,U.(2007). *Complex cell pooling and the statistics of natural images.* Network: Computation in Neural Systems, 18(2), 81–100.

[10] Boureau, Y., Ponce, J., & LeCun, Y. (2010). *A theoretical analysis of feature pooling in visual recognition.* In Proceedings of the 27th International Conference on Machine Learning (pp. 111–118). N.p.: International Machine Learning Society.

[11] Bruna, J., Szlam, A., & LeCun, Y. (2013). *Signal recovery from pooling representations.* arXiv 1311.4025.

[12] Gulcehre, C., Cho, K., Pascanu, R., & Bengio, Y. (2014). *Learned-norm pooling for deep feedforward and recurrent neural networks.* In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (pp. 530–546). New York: Springer-Verlag.

[13] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors*. arXiv 1207.0580.

[14] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A simple way to prevent neural networks from overfitting.* Journal of Machine Learning Research, 15(1), 1929–1958.

[15] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks.* In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.),Advances in neural information processing systems, 25 (pp. 1097–1105). Red Hook, NY: Curran.

[16] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung (2000). *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. Nature. 405. pp. 947–951.*

[17] Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted Boltzmann machines.* In Proceedings of the 27th International Conference on Machine Learning (pp. 807–814). N.p.: International Machine Learning Society

[18] Maas, A.L.,Hannun,A.Y.,&Ng,A.Y.(2013). *Rectifier nonlinearities improve neural network acoustic models.* In Proceedings of the 30 th International Conference Machine Learning (pp. 1–8). N.p.: International Machine Learning Society.

[19] He,K.,Zhang,X.,Ren,S.,&Sun,J.(2015). *Deep residual learning for image recognition.* arXiv 1512.03385

[20] Clevert, Djork-Arné & Unterthiner, Thomas & Hochreiter, Sepp. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).

[21] Deep Learning; Goodfellow, Bengio, Courville; ch 7, http://www.deeplearningbook.org/contents/regularization.html

[22] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014b). *Intriguing properties of neural networks.*

[23] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). *Explaining and harnessing adversarial examples.* CoRR, abs/1412.6572.

[24] Bengio, Y., Simard, P., & Frasconi, P. (1994). *Learning long-term dependencies with gradient descent is difficult.* IEEE Transactions on Neural Networks, 5(2), 157–166.

[25] Glorot, X., & Bengio, Y. (2010). *Understanding the difficulty of training deep feedforward neural networks*. In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (pp. 249–256).

[26] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ...Darrell, T. (2014). Caffe*: Convolutional architecture for fast feature embedding*. Proceedings of the 22nd ACM International Conference on Multimedia (pp. 675–678). New York: ACM.

[27] Ioffe,S.,&Szegedy,C.(2015). Batchnormalization: *Accelerating deep network training by reducing internal covariate shift*. In Proceedings of the 32nd International Conference Machine Learning (pp. 448–456). N.p.: International Machine Learning Society.

[28] Petro Liashchynskyi and Pavlo Liashchynskyi (2019). *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS ,* 1912.06059

[29] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*: Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE, 86(11):2278-2324

[30] Alex Krizhevsky and Sutskever, Ilya and Hinton, Geoffrey E (2012): *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems.

[31] Karen Simonyan, Andrew Zisserman (2014). *Very Deep Convolutional Networks for Large-Scale Image Recognition.* arXiv: 1409.1556

[32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich (2014): *Going Deeper with Convolutions*. CoRR abs/1409.4842

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun: *Deep Residual Learning for Image Recognition*, arXiv:1512.03385

[34] G. Huang, Z. Liu and L. van der Maaten, *Densely Connected Convolutional Networks*, 2018.

[35] Ian J. Goodfellow and Jean Pouget-Abadie and Mehdi Mirza and Bing Xu and David Warde-Farley and Sherjil Ozair and Aaron Courville and Yoshua Bengio (2014): *Generative Adversarial Networks,* arXiv: 1406.2661

[36] Xiao, Tianjun & Xu, Yichong & Yang, Kuiyuan & Zhang, Jiaxing & Peng, Yuxin & Zhang, Zheng. (2015). *The application of two-level attention models in deep convolutional neural network for fine-grained image classification*. 842-850. 10.1109/CVPR.2015.7298685.

[37] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*: Gradient-Based Learning Applied to Document Recognition*, Proceedings of the IEEE, 86(11):2278-2324

[38] Han Xiao, Kashif Rasul, Roland Vollgraf: *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.* CoRR abs/1708.07747 (2017)

[39] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[40] J. Bergstra and Y. Bengio, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research, pp. 281–305, 2012.

[41] Peter Lukáč Peter Tarábek, *Simple approach to improve DNN solution,* Mathematics in Science and Technologies 2018: Proceedings of MIST conference 2018

[42] Hadsell, Raia & Chopra, Sumit & Lecun, Yann. (2006). *Dimensionality Reduction by Learning an Invariant Mapping*. 1735 - 1742. 10.1109/CVPR.2006.100.

[43] Schroff, Florian & Kalenichenko, Dmitry & Philbin, James. (2015). FaceNet: A unified embedding for face recognition and clustering. 815-823. 10.1109/CVPR.2015.7298682.

[44] MITCHELL - Machine Learning, Tom Mitchell, McGraw Hill, 1997

[45] Massey, James. (2007). Algebraic Codes for Data Transmission (R. E. Blahut; 2003) [book review]. Information Theory, IEEE Transactions on. 53. 441-442. 10.1109/TIT.2006.887061.

[46] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. PAMI, 30(11):1958–1970, November 2008.

[47] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, Caltech, 2007.

[48] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. PAMI,30(11):1958–1970, November 2008.

[49] O. Boiman, E. Shechtman, and M. Irani. In defense of nearestneighbor based image classification. In CVPR08, pages 1–8, 2008.

[50] D. Lowe. *Distinctive image features from scale-invariant keypoints.* IJCV, 60(2):91–110, November 2004.

[51] B. Collins, J. Deng, K. Li, and L. Fei-Fei. *Towards scalable dataset construction: An active learning approach*. In ECCV08, pages I: 86–98, 2008.

[52] L.-J. Li, G. Wang, and L. Fei-Fei. *OPTIMOL: automatic Online Picture collecTion via Incremental MOdel Learning.* In CVPR07, pages 1–8, 2007.

[53] Gershgorn, Dave (10 September 2017). Gershgorn, Dave (10 September 2017). *"The Quartz guide to artificial intelligence: What is it, why is it important, and should we be afraid?"*. Quartz. Retrieved 3 February 2018. Quartz. Retrieved 3 February 2018

[54] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays,Pietro Perona, Deva Ramanan, Piotr Doll´ar, and C LawrenceZitnick. *Microsoft coco: Common objects in context.* InEuropean conference on computer vision, pages 740–755.Springer, 2014

[55] D. P. Papadopoulos, J. R. R. Uijlings, F. Keller and V. Ferrari, *"We Don't Need No Bounding-Boxes: Training Object Class Detectors Using Only Human Verification,"* 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 854-863, doi: 10.1109/CVPR.2016.99.

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015.

[57] J. Deng, O. Russakovsky, J. Krause, M. Bernstein, A. Berg, L. Fei-Fei. Scalable multi-label annotation. *ACM conference on human factors in computing (CHI)*, 2014.

[58] H. Bilen, M. Pedersoli, and T. Tuytelaars. Weakly supervised object detection with posterior regularization. In BMVC, 2014.

[59]

[60] R. Cinbis, J. Verbeek, and C. Schmid. *Weakly supervised object localization with multi-fold multiple instance learning*. arXiv:1503.00949, 2015.

[61] T. Deselaers, B. Alexe, and V. Ferrari. Localizing objects while learning their appearance. In ECCV, 2010.

[62] S. Vijayanarasimhan and K. Grauman. *Large-scale live active learning: Training object detectors with crawled data and crowds*. IJCV, 108(1-2):97–114, 2014.

[63] A. Yao, J. Gall, C. Leistner, and L. Van Gool. *Interactive object detection*. In CVPR, 2012.

[64] C. Wang, W. Ren, J. Zhang, K. Huang, and S. Maybank. *Large-scale weakly supervised object localization via latent category learning.* IEEE Transactions on Image Processing, 24(4):1371–1385, 2015.

[65] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. *Visual recognition with humans in the loop.* In ECCV, 2010.

[66] J. Deng, J. Krause, and L. Fei-Fei. *Fine-grained crowdsourcing for fine-grained recognition*. In CVPR, 2013.

[67] S. Lad and D. Parikh. *Interactively guiding semi-supervised clustering via attribute-based explanations.* In ECCV, 2014.

[68] O. Russakovsky, L.-J. Li, and L. Fei-Fei. *Best of both worlds: human-machine collaboration for object annotation.* In CVPR, 2015.

[67] A. J. Joshi, F. Porikli, and N. Papanikolopoulos. *Multi-classactive learning for image classification*. In CVPR, 2009.

[68] A. Kapoor, K. Grauman, R. Urtasun, and T. Darrell. *Active learning with gaussian processes for object categorization*. In ICCV, 2007.

[69] B. Siddiquie and A. Gupta. Beyond active noun tagging: Modeling contextual interactions for multi-class active learning. In CVPR, 2010.

[70] S. Vijayanarasimhan and K. Grauman. Multi-level active prediction of useful image annotations for recognition. In NIPS, 2008.

[71] C. Leistner, M. Godec, S. Schulter, A. Saffari, and H. Bischof. Improving classifiers with weakly-related videos. In CVPR, 2011.

[72] M. Guillaumin and V. Ferrari. Large-scale knowledge transfer for object localization in imagenet. In CVPR, 2012.

[73] J. Hoffman, S. Guadarrama, E. Tzeng, R. Hu, and J. Donahue. LSDA: Large scale detection through adaptation. In NIPS, 2014. 3

[74] A. Gupta and L. Davis. Beyond nouns: Exploiting prepositions and comparators for learning visual classifiers. In ECCV, 2008.

[75] D. P. Papadopoulos, A. D. F. Clarke, F. Keller, and V. Ferrari. Training object class detectors from eye tracking data. In ECCV, 2014.

[76] P. Dollar and C. Zitnick. Edge boxes: Locating object proposals from edges. In ECCV, 2014.

[77] M. Everingham, L. Van Gool, C. K. I.Williams, J.Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. IJCV, 2010.

[78] R. Cinbis, J. Verbeek, and C. Schmid. Multi-fold mil training for weakly supervised object localization. In CVPR, 2014.

[79] A. Grigoryan and M. Grigoryan, “Hadamard Transform,” in Brief Notes in Advanced DSP, 2009.

[80] S. Yang, P. Luo, C. C. Loy, K. W. Shum, and X. Tang, “Deep representation learning with target coding,” in Proceedings of the National Conference on Artificial Intelligence, 2015.

[81] N. Akhtar and A. Mian, “Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey,” IEEE Access, vol. 6. Institute of Electrical and Electronics Engineers Inc., pp. 14410–14430, 16-Feb-2018

[82] T. G. Dietterich and G. Bakiri, “Solving Multiclass Learning Problems via Error-Correcting Output Codes,” J. Artif. Intell. Res., 1995.

[83] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera, “Beyond one-hot encoding: Lower dimensional target embedding,” Image Vis. Comput., 2018.

[84] R. E. Blahut, Algebraic Codes for Data Transmission. 2003.

[85] D. Kim, S. A. Bargal, J. Zhang, and S. Sclaroff, “Multi-way Encoding for Robustness,” in Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020, 2020, pp. 1352–1360.

[86] J. Qin et al., “Zero-shot action recognition with error-correcting output codes,” in Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, 2017.

[87] D. H. Wolpert and T. G. Dietterich, “Error-Correcting Output Codes: A General Method for Improving Multiclass Inductive Learning Programs,” in The Mathematics of Generalization, 2018.

[88] M. Lachaize, S. Le Hégarat-Mascle, E. Aldea, A. Maitrot, and R. Reynaud, "Evidential framework for Error Correcting Output Code classification," Eng. Appl. Artif. Intell., 2018.

[89]. K. Sen Li, H. R. Wang, and K. H. Liu, "A novel Error-Correcting Output Codes algorithm based on genetic programming," Swarm Evol. Comput., 2019.

[90] P. A. von Kaenel, "Fuzzy codes and distance properties," Fuzzy Sets Syst., vol. 8, no. 2, pp. 199–204, 1982.

[91] S. A. Tsafack, S. Ndjeya, L. Strüngmann, and C. Lele, "Fuzzy Linear Codes," Fuzzy Inf. Eng., vol. 10, no. 4, pp. 418–434, 2018.

[92] E. P. Klement, R. Mesiar, and E. Pap, "Triangular norms. basic notions and properties," in Logical, Algebraic, Analytic and Probabilistic Aspects of Triangular Norms, 2005.

[93] S. Lian, Principles of imprecise-information processing: A new theoretical and technological system. 2016.

[94] L. D. Rudolph, C. R. P. Hartmann, T. Y. Hwang, and N. Q. Duc, "Algebraic Analog Decoding of Linear Binary Codes," IEEE Trans. Inf. Theory, vol. 25, no. 4, pp. 430–440, 1979.

[95] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2016.

[96] S. Yang, P. Luo, C. C. Loy, K. W. Shum, and X. Tang, "Deep representation learning with target coding," in Proceedings of the National Conference on Artificial Intelligence, 2015.

[97] M. Klimo, P. Lukáč, P. Tarábek, Deep neural networks classification via binary error-detecting output codes In: Applied Sciences. - ISSN 2076-3417 . Roč. 11, č. 8 (2021), https://www.mdpi.com/2076-3417/11/8/3563

Zoznam publikácií

[1] Peter Lukáč Peter Tarábek, Simple approach to improve DNN solution, Mathematics in Science and Technologies 2018: Proceedings of MIST conference 2018

[2] P. Lukáč and P. Tarábek, "Improving DNN Solution using Repeated Training," *2019 International Conference on Information and Digital Technologies (IDT)*, 2019, pp. 311-315, doi: 10.1109/DT.2019.8813418.

[3] P. Lukáč and P. Tarábek, *Confidence Prediction Driven Iterative Data Annotation* **V tlači** https://www.springer.com/gp/book/9789811621017

[4] M. Klimo, P. Lukáč, P. Tarábek, Deep neural networks classification via binary error-detecting output codes In: Applied Sciences. - ISSN 2076-3417 . Roč. 11, č. 8 (2021), https://www.mdpi.com/2076-3417/11/8/3563

Prílohy

Príloha 1 - Publikácie v elektronickej forme s názvom *publikácie.zip*