VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERPRETACE KONVOLUČNÍCH NEURONOVÝCH SÍTÍ

EXPLAINABLE CONVOLUTIONAL NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DANIEL KAMENICKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL HRADIŠ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Kamenický Daniel**
Program: Informační technologie
Název: **Interpretace konvolučních neuronových sítí**
Explainable Convolutional Neural Networks
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s konvolučními neuronovými sítěmi a jejich trénováním.
2. Vytvořte si přehled o současných metodách umožňujících analyzovat, co se neuronové sítě učí, podle jakých vlastností vstupů se rozhodují a podobně.
3. Vyberte sadu úloh, datových sad a vhodných metod pro analýzu natrénovaných neuronových sítí.
4. Implementujte navržené metody a proveďte experimenty.
5. Porovnejte dosažené výsledky a diskutujte možnosti budoucího vývoje.
6. Vytvořte stručné video prezentující vaši práci, její cíle a výsledky.

Literatura:

- Podle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hradiš Michal, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem práce bylo porovnání několika metod pro vizualizaci příznaků jednotlivé třídy na vstupní pixelové vrstvě CNN. Každá metoda využívá jiný algoritmus, založený na gradientech, pro výpočet výsledných hodnot. Pomocí implementace jednotlivých metod jsou získány výsledné hodnoty metod, které jsou pomocí rovnice koncentrace energie porovnány. Výsledné hodnoty jsou uvedeny v tabulkách a grafech z kterých lze vyčíst úspěšnost výsledku práce. Z práce lze vyčíst rozdíl mezi metodami a porovnání jejich výsledků. Díky tomu lze získat přehled o metodách vizualizace založených na gradientech.

Abstract

The aim of this work was to compare several methods for visualizing the features of each class on the input pixel layer of the CNN. Each method uses a different algorithm, based on gradients, to compute the resulting values. Using the implementation of each method, the resultant values of the methods are obtained by using the equation of energy concentration. The resultant values are presented in tables and graphs from which the success rate of the result of the work can be read. The difference between the methods and comparison of their results can be read from the work. This makes it possible to get an overview of gradient based visualization methods.

Klíčová slova

CNN, NN, MLP, gradient, preceptron, sigmoid, bias, práh, váhy, hook

Keywords

CNN, NN, MLP, gradient, preceptron, sigmoid, bias, threshold, weights, hook

Citace

KAMENICKÝ, Daniel. *Interpretace konvolučních neuronových sítí.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Hradiš, Ph.D.

Interpretace konvolučních neuronových sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Hradiše, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Daniel Kamenický
18. května 2021

Poděkování

Tímto bych rád poděkoval panu Ing. Michalu Hradiši, Ph.D. za vedení mé bakalářské práce a také za informace a rady, které mi poskytl v průběhu řešení.

Obsah

Kapitola 1

Úvod

Rozpoznávání objektů, čísel nebo tváří je pro člověka v celku jednoduchá záležitost. Rozpoznat že je na papíře napsané číslo 8 dokáže už i čtyřleté dítě. Díky několika desítkám miliónů neuronů, které máme v mozku, dokážeme jednoduše určit co vidíme. V hlavě nosíme superpočítač, který se vyvíjel několik miliónů let k tomu, aby dokázal pochopit vizualizaci světa.

Jednoduchá rozhodnutí pro člověka se stávají extrémně náročná pro počítač. Abychom dokázali popsat algoritmem, jak se má počítač rozhodovat, museli bychom extrahovat různé malé sektory, podle kterých by se počítač dokázal rozhodnout. Číslovka osm by mohla být rozdělena na dvě malé kružnice nad sebou. Ovšem to je moc všeobecné, a i zde bychom mohli narazit na různá úskalí, i když se jedná pouze o jednu číslovku. Proto se využívá moderních technologií, kde existují veliké datové sady o několika tisících až miliónech obrázků, které jsou využívány k trénování konvolučních neuronových sítí. Místo abychom počítači ukázali, podle kterých faktorů se má rozhodovat, necháme počítač, aby se sám naučil, podle čeho se bude rozhodovat.

Tím se dostáváme k hlavní myšlence této práce. Jelikož necháme počítač, aby se sám naučil, vzniká nám z celkového algoritmu black box. Dokážeme říct co bylo na vstupu a co jsme dostali za výsledek, ale podle čeho se počítač rozhodl že se daná věc na obrázku nachází? Rozhoduje se správně, nebo byl výsledek náhodný? Tyto otázky jsou zodpovězeny v metodách vizualizace, které dokáží na vstupní pixelovou vrstvu zobrazit konkrétní místa obrázku, které byly aktivovány konvoluční neuronovou sítí. Hlavní otázkou je, která metoda je nejvhodnější pro vizualizaci, a při tom nedošlo k vizualizaci, která nemá s danou třídou nic společného. Tímto tématem se zabývá tato práce.

Cílem je výzkum, který otestuje metody `Backpropagation` [5], `Deconvolution` [8], `Guided Backpropagation` [7], `Grad-CAM` [6], `Guided Grad-CAM` [6], rozepsané v kapitole 3, zaměřující se na vizualizaci konvolučních neuronových sítí (CNN). Výzkum ukáže, podle čeho se daná CNN rozhoduje a jak moc její rozhodnutí bylo ovlivněno konkrétní segmentací třídy. Jednotlivé metody jsou testovány na jednotné datové sadě za stejných podmínek. V samém závěru dojde k porovnání všech metod a určení nejlepší metody pro vizualizaci.

Kapitola 2

Vizualizace CNN

2.1 Vizualizace

Neuronové sítě s hlubokým učením jsou neprůhledné, což znamená, že i když mohou vytvářet užitečné předpovědi, není jasné, jak a proč byla daná předpověď provedena. Často jsou algoritmy hlubokého učení [1] spojovány s termínem "black box", jelikož uživatel nevidí, co se uvnitř algoritmu děje. Pro důvěryhodnost algoritmů je potřeba vědět jak fungují. Existují různé metody vizualizací CNN, díky kterým je možné odhalit, co se v algoritmech hlubokého učení děje a podle čeho algoritmy vyhodnotí svoji předpověď.

Tato práce se zaměřuje na metody vizualizace zaměřené na gradienty, díky kterým lze prokázat, podle čeho se daná CNN rozhodla. K tomu, aby metody byly dobře srozumitelné, je potřeba znát podklad, jak se neuronová síť učí a jak se daný gradient zpětně vypočítá. Proto jsou nejdříve vysvětleny pojmy týkající se struktury neuronové sítě a jejího učení a až poté jsou vysvětleny jednotlivé metody.

2.2 Neuronová Síť

Umělá neuronová síť [3] (zkráceně neuronová síť neboli NN) se skládá z několika částí, kterými jsou vstupní vrstva neuronů, několik skrytých vrstev neuronů a poslední vrstva výstupních neuronů. U typické architektury NN, má každé propojení neuronu přidruženou váhu. Výstup h_i, ve skryté vrstvě, neuronu i je,

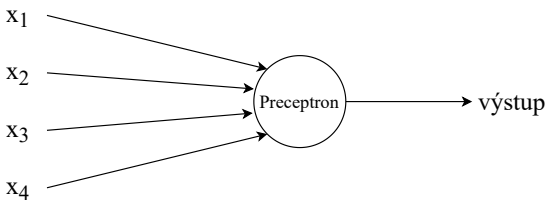
$$h_i = \sigma(\sum_{j=1}^{N} W_{ij}x_j + b), \tag{2.1}$$

kde N je počet vstupních neuronů, W_{ij} jsou váhy, x_j hodnoty ze vstupního neuronu, σ je aktivační funkce a b je bias.

Preceptron

Umělý neuron, který se nazývá `preceptron` [4], je základní jednotka NN. V dnešních moderních sítích se využívají již jiné neurony, které se nazývají sigmoidy. Pro definici sigmoidy je zapotřebí znát definici preceptronu, jelikož z něj vychází.

Funkce preceptronu je jednoduchá. Preceptron bere několik vstupů $x_1, x_2, \ldots$, a produkuje jeden binární výstup:



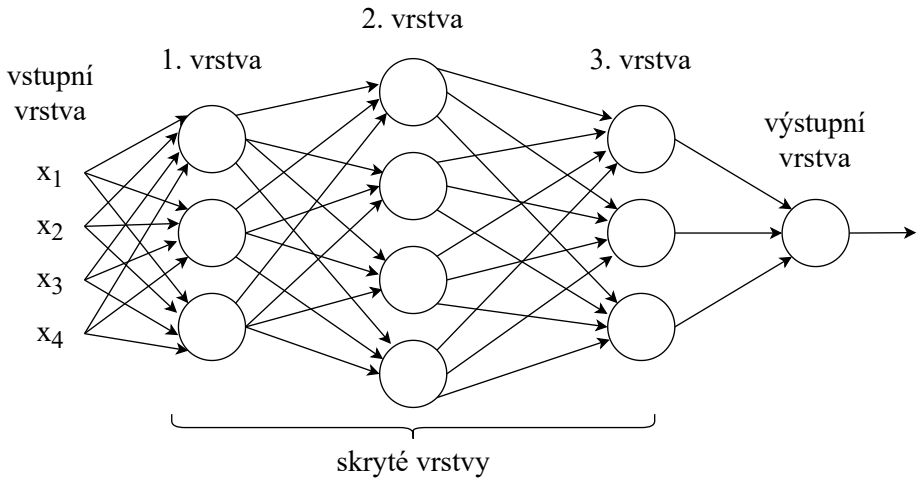
Obrázek 2.1: Obrázek popisující architekturu preceptronu s několika vstupy a jedním binárním výstupem

Preceptron má čtyři vstupy x_1, x_2, x_3, x_4. Obecně by mohl mít více či méně vstupů. Rosenblatt [4] navrhl jednoduché pravidlo pro výpočet výstupu, kde představil váhy $w_1, w_2, \ldots$, které určují důležitost příslušných vstupů na výstup. Každý neuron má parametr určující práh (threshold), který je stejně jako váhy zastoupen reálným číslem. Tento práh byl nahrazen biasem a platí rovnice: $b \equiv -t$, kde b je bias a t je práh. Výstup neuronu je určen:

$$\text{výstup} = \begin{cases} 0 & \text{pro } \sum_j^n w_j x_j + b \leq 0 \\ 1 & \text{pro } \sum_j^n w_j x_j + b > 0. \end{cases} \tag{2.2}$$

Celý princip, jak preceptron funguje, je založen na základě zvážení vstupních hodnot, díky kterým ví jak se rozhodnout. Bias nám určuje, jak moc těžké je dostat preceptron k hodnotě 1.

Celá NN nestojí na jednom neuronu, ale je tvořena několika Obr. 2.2. Závažnost rozhodování neuronu bychom mohli rozdělit do vrstev. První vrstva provede rozhodnutí ze získaných vstupních dat. Druhá vrstva se rozhoduje na základě zvážení výsledků první vrstvy. Tímto způsobem může perceptron ve druhé vrstvě učinit rozhodnutí na složitější a abstraktnější úrovni než perceptrony v první vrstvě. Ještě složitější rozhodnutí muže učinit preceptron ve třetí vrstvě.



Obrázek 2.2: Vícevrstvé preceptrony (MLP)

Sigmoid

Architekura preceptronu je postavená na výstupech, které se odvádí od biasu a váhy důležitosti nabývající 0 a 1. Pokud by malá změna těchto hodnot ovlivnila změnu výstupu, mohl by být tento faktor využít k modifikaci váh a biasů k tomu, aby NN zlepšila svoje výsledky. Avšak preceptron funguje pouze ve škále 0 a 1, kde není možné provést menší změnu váhy, která by zlepšila výsledky dané NN. Změna v preceptronu by mohla nastat v momentě převrácení hodnoty z 0 na 1, nebo naopak. Tato změna by mohla vést k celkové změně chování sítě. Tím se vylučuje možnost učení neuronu.

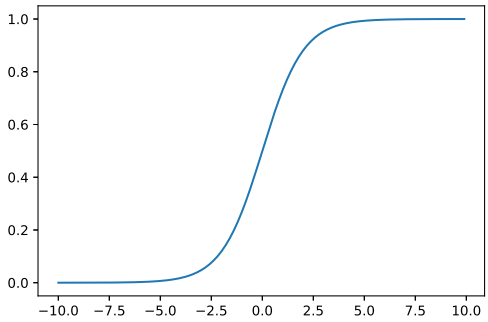
Tento problém lze vyřešit zavedením nového typu umělého neuronu zvaného `sigmoid`. Sigmoid je podobný architekurou preceptronu Obr. 2.1, ale umožňuje provádět malé změny váh a biasu, které zlepší výstup NN. Místo hodnot 0 a 1 může neuron nabývat hodnot kdekoliv v rozmezí $< 0, 1 >$. Hodnoty jsou dosaženy díky přidání `sigmoid funkce`:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}. \tag{2.3}$$

Pro přesnější představu v případě váh $w_1, w_2, \ldots$, hodnot $x_1, x_2, \ldots$ a biasu b je výstup sigmoid neuronu definován následovně:

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}. \tag{2.4}$$

Graf sigmoid funkce.



Obrázek 2.3: Graf funkce sigmoidy

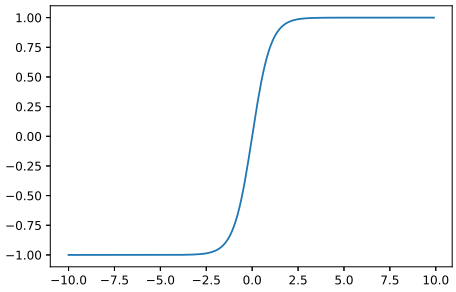
Sigmoid funkce je nazývána také jako aktivační funkce. V modernějších sítích byla sigmoid funkce nahrazena jinými aktivačními funkcemi, jako je `tanh` funkce 2.4, jejíž definice je

$$\sigma(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \tag{2.5}$$

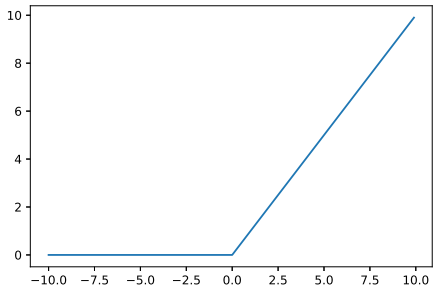
nebo `ReLU` funkce 2.5, jejíž definice je

$$\sigma(z) = max(0, z). \tag{2.6}$$

Funkce `tanh` a `ReLU` jsou vykresleny na níže uvedných grafech:



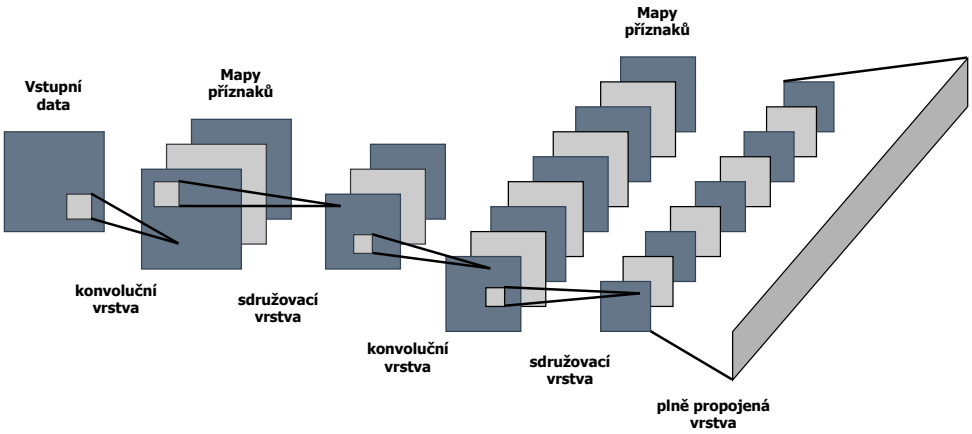
Obrázek 2.4: Graf funkce `tanh`



Obrázek 2.5: Graf funkce `ReLU`

Konvoluční Neuronová Síť

Vícevrstvé preceptrony (MLP) Obr. 2.2 jsou velmi omezené v klasifikaci obrázků. Neumí redukovat vstupní obraz na menší segmenty a detekovat menší struktury. Konvoluční Neuronová síť (CNN) Obr. 2.6 vylepšuje MLP přidáním několika nových komponent, jako jsou konvoluční a sdružovací vrstvy. Díky nim dokáží vstupní obraz zmenšit a detekovat menší a přesnější struktury.



Obrázek 2.6: Architektura konvoluční neuronové sítě

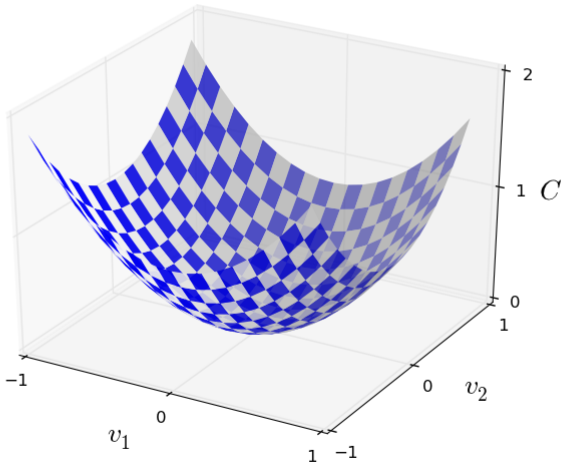
2.3 Gradient Descent

Aby bylo možné najít váhy a biasy k požadovaným výstupům, provádí se proces trénování na trénovacích datech. Trénovací data jsou předávána na vstup MLP. Skutečný výstup je porovnán s požadovaným výstupem. Na základě porovnání výstupů lze upravit parametry sítě tak, aby lépe odpovídaly požadovanému výsledku. Zlepšení lze kvantifikovat pomocí funkce ztráty (loss function):

$$L(w,b) \equiv \frac{1}{2n} \sum_x ||y(x) - a||^2, \tag{2.7}$$

kde w označuje všechny váhy v MLP, b označuje všechny biasy, n je celkový počet vstupních trénovacích dat, a je vektor označující všechny vstupy sítě, x označuje vstupní data. Výstup a závisí na trojici (x, w, b), ale pro jednodušší zápis zde nejsou uvedeny konkrétní závislosti. $y(x)$ odpovídá správnému výstupu/třídě konkrétního vstupu x. L můžeme nazývat kvadratickou ztrátovou funkcí (quadratic loss function) Obr. 2.7, také známá jako střední čtvercová chyba (mean squared error neboli MSE). Ztráta $L(w, b)$ se snižuje $L(w, b) \approx 0$ v momentě, kdy $y(x)$ se blíží k požadované hodnotě a, pro všechna trénovaná data x, což je požadovaný výsledek trénování.

Pro lepší představu jsou nahrazeny váhy a biasy $v = (w, b)$. Pro minimalizaci funkce $L(v)$, si lze vypomoci představením si funkce L pouze pro dvě proměnné (v_1, v_2).



Obrázek 2.7: Graf kvadratické funkce ztráty (quadratic loss function) [3]

Pro nalezení správných parametrů w, b, muže být využity techniky gradient descent. Cílem techniky gradient descent je nalezení globálního minima funkce L. Nalezení minima pomocí derivací funkce L je příliš složité pro více parametrů funkce. Další způsob je představit si kvadratickou ztrátovou funkci jako údolí Obr. 2.7 a někde na okraji tohoto údolí míček, který se kutálí dolů. Podle toho, co je známo z každodenního života by se měl míček skutálet do údolí.

Vybere se tedy náhodný startovní bod pro míček a provede se simulace pohybu míčku. Tato simulace by se dala jednoduše spočítat pomocí derivací (případně druhých derivací), které nám určí vše o lokálním tvaru údolí. Pro přesnější pochopení je potřeba se podívat, co se stane při malém posunu míčku Δv_1 ve směru v_1 a malém posunu Δv_2 ve směru v_2. Výpočet nám řekne, že L se změní následovně:

$$\Delta L \approx \frac{\partial L}{\partial v_1}\Delta v_1 + \frac{\partial L}{\partial v_2}\Delta v_2. \tag{2.8}$$

Je zapotřebí najít cestu určování hodnot Δv_1 a Δv_2 tak, aby hodnota ΔL byla záporná. Při správném určení docílíme pohybu míčku do údolí. Pro lepší pochopení je určen vektor změn

$$\Delta\ v\ \equiv (\Delta v_1, \Delta v_1)^T, \tag{2.9}$$

kde T je operátor transpozice, který převádí vektor řádků na vektor sloupců. Také je určen gradient funkce L, jako vektor parciálních derivací

$$\nabla L \equiv \left(\frac{\partial L}{\partial v_1}, \frac{\partial L}{\partial v_2}\right)^T. \tag{2.10}$$

S těmito definicemi můžeme rovnici 2.8 přepsat na

$$\Delta L \equiv \nabla L \cdot \Delta v. \tag{2.11}$$

Aby se snížila celková cena L, je třeba nahrazovat vektor v správnými hodnotami v každé iteraci tréninku. Aby toho bylo dosaženo, je zde přidán malý kladný parametr η, která je známý jako rychlost učení (learning rate). Vektor v pak dostaneme rovnicí

$$\Delta v = -\eta \nabla L. \tag{2.12}$$

Dosazením do rovnice 2.11, dostaneme:

$$\Delta C \approx -\eta \nabla L \cdot \nabla L = -\eta ||\nabla L||^2. \tag{2.13}$$

Hodnoty $||\nabla L||^2$ a η jsou vždy kladné a z toho hlediska změna funkce L je pokaždé negativní. To zaručuje, že při každé iteraci se ztráta snižuje. Dosazením zpátky do rovnice, parametry w a b na místo $v1$ a $v2$, je pravidlo, pro aktualizaci váh w_k definováno následovně:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial L}{\partial w_k}, \tag{2.14}$$

a pro aktualizaci biasu b_l:

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial L}{\partial b_l}. \tag{2.15}$$

Při opětovném aplikování těchto pravidel je možné se dostat do minima ztrátové funkce. Jinými slovy, tento postup může být aplikován pro učení neuronových sítí.

Pro urychlení učení NN je možné využít techniku zvanou `stochastic gradient descent` (SGD). Hlavní myšlenkou je odhadnout gradient ∇L vypočítáním ∇L_x náhodných malých vzorků vybraných z tréninkových vstupů. Zprůměrováním tohoto malého vzorku lze rychle získat dobrý odhad skutečného gradientu ∇L.

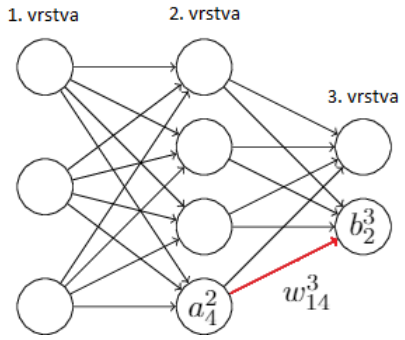
2.4 Backpropagation

V předchozí sekci bylo zmíněno učení NN pomocí metody gradient descent, kde NN se dokázala naučit svoje váhy a biasy. V kapitole nebylo zmíněno, jak lze vypočítat gradient ztrátové funkce. Algoritmem `backpropagation` [5] lze tyto gradienty rychle vypočítat. Cílem backpropagation algoritmu je vypočítání parciálních derivací $\frac{\partial L}{\partial w}$ a $\frac{\partial L}{\partial b}$ ztrátové funkce L s respektem na váhy w a biasy b.

Algoritmus pojednává o porozumění jak moc změna váh nebo biasů v NN změní výstup ztrátové funkce. V následující notaci je třeba poznamenat

$$z_j^l \equiv \sum_k w_{jk}^l a_k^{l-1} + b_j^l, \tag{2.16}$$

kde z_j^l je pouze vážený vstup do ztrátové funkce pro neuron j ve vrstvě l. Zde w_{jk}^l značí váhu mezi neuronem k ve vrstvě $l-1$ a neuronem j ve vrstvě l. a_k^{l-1} označuje aktivaci neuronu k ve vrstvě $l-1$ a b_j^l označuje bias neuronu j ve vrstvě l.



Obrázek 2.8: Ukázka spojitosti mezi a_k^{l-1}, b_j^l a w_{jk}^l z rovnice 2.16

Dále je definována nějaká malá změna Δz_j^l, která je přidána do vážených vstupů neuronů a tato změna způsobuje celkový dopad $\frac{\partial L}{z_j^l}\Delta z_j^l$ na výstup. Z toho můžeme definovat chybu

$$\delta_j^l \equiv \frac{\partial L}{\partial z_j^l} \tag{2.17}$$

pro neuron j ve vrstvě l. Pro výstupní vrstvu V sítě NN je chyba definována jako

$$\delta_j^V \equiv \frac{\partial L}{\partial a_j^V}\sigma'(z_j^V). \tag{2.18}$$

První term na pravo, $\partial L/\partial a_j^V$, pouze měří jak rychle se ztrátová funkce mění jakožto funkce aktivace výstupního neuronu j. Pokud funkce L tolik nezávisí na výstupu neuronu j, tak chyba σ_j^V daného neuronu bude malá. Druhý term napravo, $\sigma'(z_J^V)$, měří jak rychle se aktivační funkce σ mění na z_j^V.

Rovnice 2.18 je v nematicové formě, kterou chceme pro backpropagation. Rovnici přepíšeme do maticové podoby pomocí využití Hadamardového součinu jako

$$\delta^V = \nabla_a L \odot \sigma'(z^V), \tag{2.19}$$

kde $\nabla_a L$ je definována jako vektor jehož komponenty jsou parciální derivace $\partial L / \partial a_j^V$. Jako příklad jde uvést kvadratickou ztrátu $\nabla_a L = (a^V - y)$, kde y je odpovídající požadovaný výstup. Takže plně maticová forma rovnice 2.18 je

$$\delta^V = (a^V - y) \odot \sigma'(z^V). \tag{2.20}$$

Rovnici 2.20 lze použít pro výpočet chyb v předchozích vrstvách předáním již vypočítaných chyb zpět. Chyba δ^l ve vrstvě l se vypočítá rovnicí

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^V), \tag{2.21}$$

kde $(w^{l+1})^T$ je transponovaná matice váh w^{l+1} pro vrstvu $(l+1)$. δ^{l+1} je chyba která je převzata z vyšší vrstvy. Při aplikování transponované matice váh, $(w^{l+1})^T$, výsledek poskytne míru chyby na výstupu vrstvy l. Poté se aplikuje Hadamardův součin $\odot\sigma'(z^V)$, který posune chybu zpět přes aktivační funkci vrstvy l. Výsledkem je chyba δ^l ve váženém vstupu do vrstvy l.

Při spojení funkcí 2.19 a 2.21 lze spočítat chybu δ^l pro jakoukoliv vrstvu NN. Nejdříve se využije funkce 2.19 pro výpočet chyby δ^V a poté opětovným využitím funkce 2.21 se postupně vypočítávají chyby $(\delta^{V-1}, \delta^{V-2}, \dots)$, v jednotlivých nižších vrstvách.

Závěr

V pořadí posouvání chyb přes NN a počítání gradientů ztrátové funkce L s respektem na biasy b a váhy w musíme aplikovat řetězové pravidlo, které je definováno tak, že pro každou proměnnou z závisející na proměnné y, která sama o sobě závisí na proměnné x, pravidlo říká

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}. \tag{2.22}$$

S pomocí řetězového pravidla z rovnice 2.22 a za pomocí rovnic 2.16, 2.17, můžeme zderivovat změnu ztrátové funkce s respektem na bias b_j^l v NN

$$\frac{\partial L}{\partial b_j^l} = \frac{\partial L}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l. \tag{2.23}$$

Z rovnice 2.23 se ukázalo, že $\partial L/\partial b_j^l$ je chyba neuronu, kterému bias b_j^l náleží. Se stejnými rovnicemi změna ztrátové funkce s respektem na váhu w_{jk}^l je derivována jako

$$\frac{\partial L}{\partial w_{jk}^l} = \frac{\partial L}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}. \tag{2.24}$$

Shrnutí všech čtyř základních rovnic pro algoritmus backpropagation:

$$\delta^V = \nabla_a L \odot \sigma'(z^V), \tag{2.19}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^V), \tag{2.21}$$

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l. \tag{2.23}$$

$$\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}. \tag{2.24}$$

S těmito rovnicemi je možné spočítat gradient ztrátové funkce, ve směru změny parametrů váh w a biasu b v NN, k docílení celkovému snížení výsledku ztrátové funkce.

Formou algoritmu lze backpropagation zapsat:

1. **Vstup x:** Nastav odpovídající aktivaci a^1 pro vstupní vrstvu.
2. **Dopředný průchod:** Pro každé $l = 2, 3, \ldots, V$ vypočítej $z^l = w^l a^{l-1} + b^l$ a $a^l = \sigma(z^l)$.
3. **Výstupní chyba δ^V:** Vypočítej vektor $\delta^V = \nabla_a L \odot \sigma'(z^V)$
4. **Zpětné šíření chyby:** Pro každé $l = V-1, V-2, \ldots, 2$ vypočítej $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^V)$
5. **Výstup:** Gradient ztrátové funkce je dán funkcemi $\frac{\partial L}{\partial b_j^l} = \delta_j^l$ a $\frac{\partial L}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$

Kapitola 3

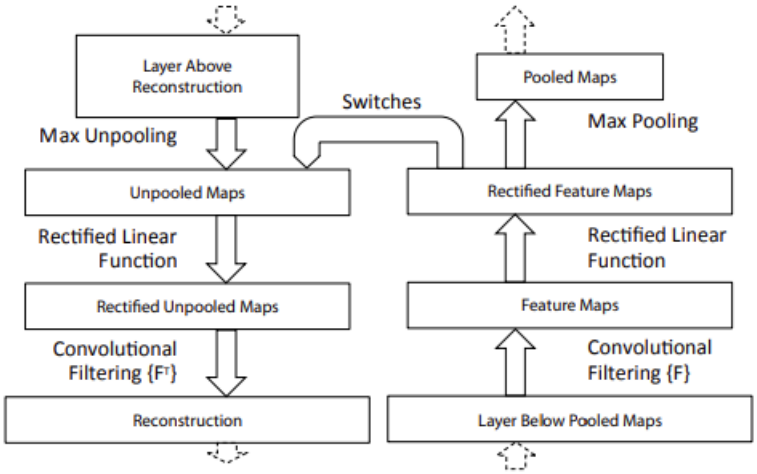
Metody vizualizace

Vizualizace příznaků pro získání intuice, co se v CNN děje je běžnou praxí, ale často limitováno na první vrstvu, kde je projekce na vstupní vrstvu pixelů možná. Korelace mezi vstupní vrstvou a mapou příznaků je ve vyšších vrstvách mnohem komplexnější. Vyšší vrstvy obsahují mapy, které detekují komplexnější příznaky a je těžší je promítnout na vstupní pixelový prostor. Přesto existují metody, které umožňují vizualizaci těchto vrstev. Metody, které jsou zde zmíněny jsou všechny založeny na gradientu cílové klasifikace.

3.1 Deconvolution

Jednou z metod je dekonvoluce [8]. Metoda mapuje aktivace ve středních vrstvách zpátky na vstupní pixelový prostor, které nám ukazují, jaký vstupní vzor způsobil danou aktivaci. Architekury metody deconvnet a konvoluční neuronové sítě jsou podobné a obě se skládají ze stejných komponent (filtering, pooling) v opačném pořadí. Metoda místo mapování pixelů do mapy příznaků, mapuje příznaky na vstupní vrstvu pixelů.

K prozkoumání CNN je metoda propojena s každou její vrstvou, což je ukázáno ve vrchní části obrázku 3.1. Propojení poskytuje cestu zpět k vstupní pixelové vrstvě. Pro začátek se vypočítá reprezentace vstupního obrazu pro CNN. Pro prozkoumání požadované aktivace dané třídy jsou vynulovány všechny ostatní aktivace v dané vrstvě l. Aktivace se předá jako vstup do metody DeconvNet. Pro vypočítání aktivací vrstvy $l-1$, která se nachází pod vrstvou l, která způsobila danou aktivaci, je zrekonstruována postupem `unpool`, `rectify` a `filter`. Tím jsou získány aktivace vrstvy $l-1$, které způsobyly aktivaci vrstvy l. Postup je opakován, dokud nedosáhne vstupní vrstvy pixelů.

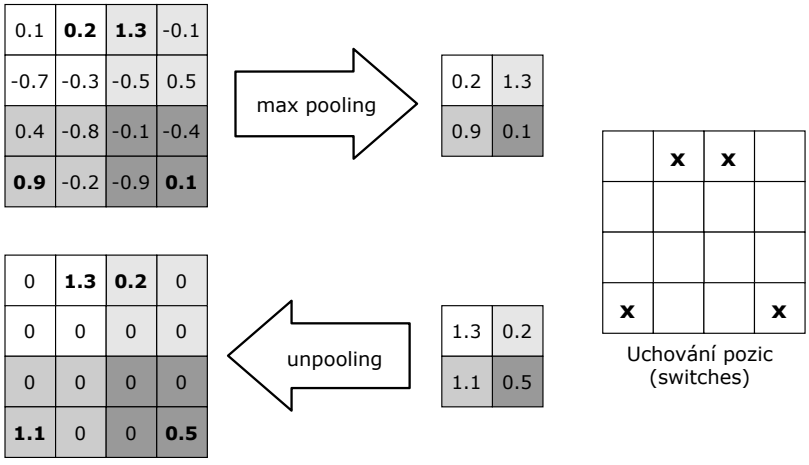


Obrázek 3.1: Architektura DeconvNet [8].

Unpooling: V konvolučních sítích je `max pooling` Obr. 3.2 nezvratný. V metodě DeconvNet jsou uchovány pozice nejvyšších hodnot, pro každý pooling region. Pozice hodnoty je uložena do takzvané *switch* proměnné. Při zpětném průchodu unpooling operace využívá této proměnné pro umístění rekonstrukce z vyšší vrstvy na správné místo, aby zachoval správnou strukturu.

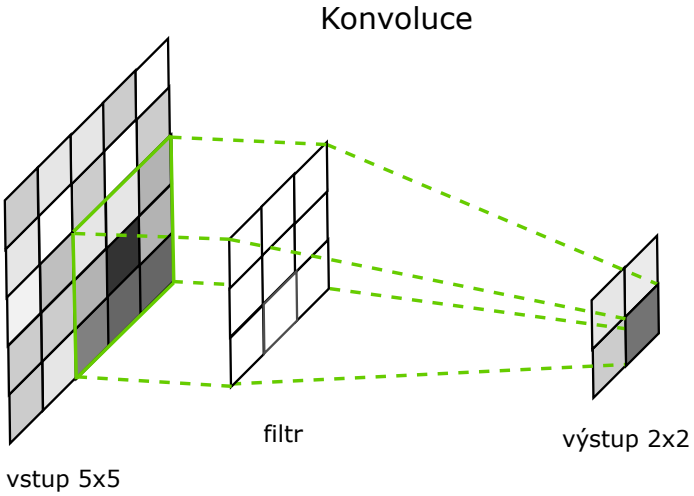
Rectification: Konvoluční sítě využívají funkci ReLU Obr. 2.5 pro zavedení nelinearity. Tato funkce zaručuje, že mapy příznaků jsou vždy kladné. Pro obdržení validní rekonstrukce je i v opačném směru zapotřebí aplikovat ReLU funkci.

Filtering: Konvoluční sítě využívají naučené filtry pro konvoluci s mapou příznaků z předešlé vrstvy. Pro přibližný invertování je v DeconvNet využita konvoluce se stejným, ale transponovaným, filtrem. Filtr je aplikován na rektifikované mapy, a ne na výstup nižší vrstvy.



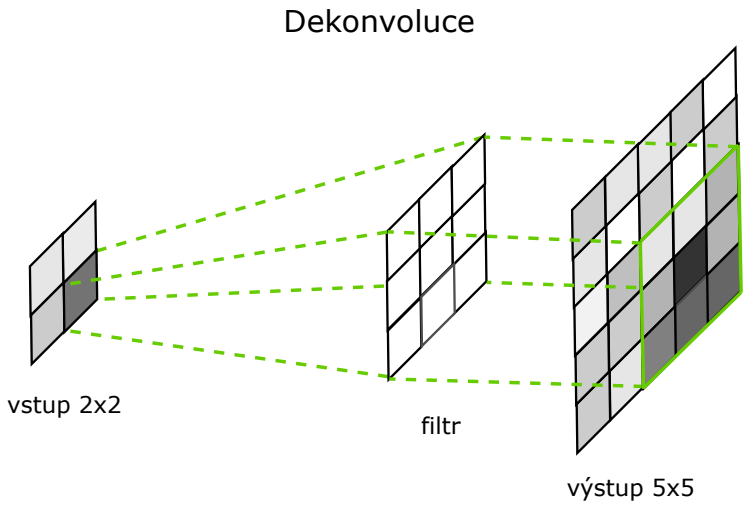
Obrázek 3.2: Max Pooling.

Na obrázku 3.3 je ukázána konvoluce s 3x3 kernelem a nastaveným krokem 2. Všechny hodnoty vyskytující se v zeleném čtverci jsou vynásobeny filtrem konvoluce a poté sečteny do výsledného čísla, které je zapsáno do výstupu. Poté je kernel posunut dále o předem definovaný krok a další hodnota mapy příznaků je vypočtena.



Obrázek 3.3: Konvoluce u `Forward-pass`.

Na obrázku 3.4 je ukázána dekonvoluce. Vstup 2x2 a je rozvzorkován do výstupu 5x5. Konvoluční kernel je vážen s každou vstupní hodnotou zvlášť a poté je zapsán do výsledné mapy příznaků, kde přesahující hodnoty jsou sečteny.

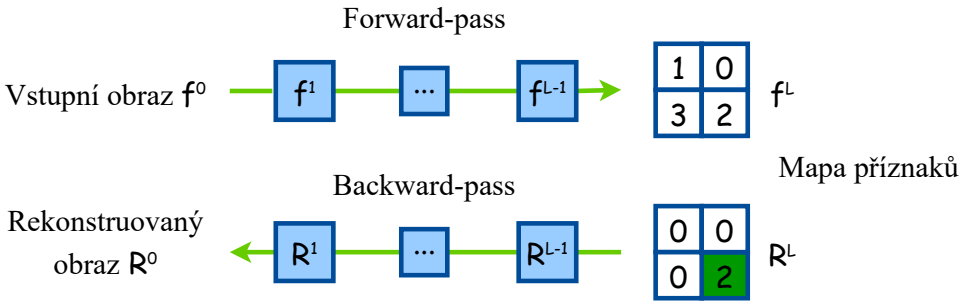


Obrázek 3.4: Konvoluce u `Backward-pass`.

3.2 Guided Backpropagation

Metoda Guided Backpropagation [7] vychází z metody Deconvolution [8] a to lehkou modifikací. Tento přístup lze aplikovat na širší škálu struktur neuronových sítí a vede k přesnějším rekonstrukcím, zejména z vyšších vrstev.

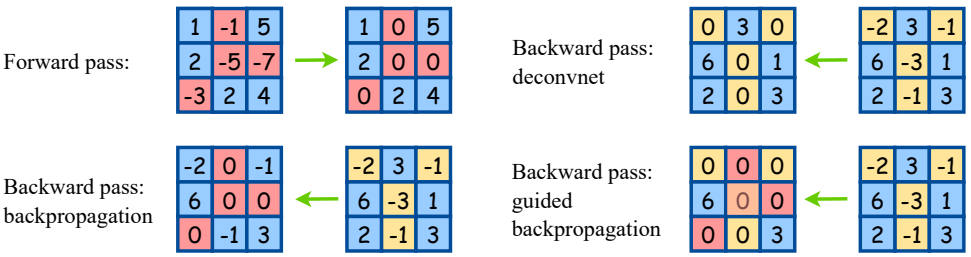
Vizualizační koncept dekonvoluce ve vyšších vrstvách CNN lze shrnout následovně. Předaná vyšší vrstva mapy příznaků jde, díky obrácenému toku dat, přes neurony aktivace v dané vrstvě, až k vstupní vrstvě pixelů. Typicky je jeden neuron ve vyšší vrstvě nenulový. Poté výsledná rekonstrukce, vyobrazena na obrázku, ukáže, v jakých místech obrazu byl neuron nejvíce aktivován. Pro rekonstrukci obrazu přes max pooling musí být nejdříve proveden `Forward-pass` pro uchování *switches*, pozice maxima každého pooling regionu. Tyto hodnoty jsou poté využívány pro dekonvoluci. Tím se metoda více zaměřuje na obraz než na naučené příznaky.



Obrázek 3.5: Schéma `Forward-pass` a `Backward-pass` [7].

Architektura Guided Backpropagation neobsahuje max-pooling a tím dokáže "dekonvolovat" bez uložených *switches* a tím pádem výstup není podmíněn vstupním obrazem. Tímto způsobem lze dokázat co se vrstvy skutečně naučily a výsledná rekonstrukce je mnohem přesnější, hlavně z vyšších vrstev. Max-pooling je nahrazeno konvoluční vrstvou se zvýšeným krokem bez ztráty přesnosti u rozpoznávání obrazů. Pro rekonstrukci vizualizace příznaků obrazu je u obou metod zapotřebí předat vstup CNN až po zvolenou vrstvu Obr. 3.5. Všechny negativní hodnoty jsou nastaveny na 0 a poté je mapa příznaku předána `Backward-pass`, pro celkovou rekonstrukci.

Důležitou roli hraje v rekonstrukci `Backward-pass`, kde se musí vyžít ReLU funkce nelinearity. Ve `Forward-pass` byly všechnu negativní hodnoty nastaveny na 0, což je vidět v levém horním rohu obrázku 3.6.



Obrázek 3.6: Porovnání `Backward-pass` průchodů [7].

Při použití metody `backpropagatin`, popsána v sekci 2.4, jsou použity pouze gradienty z `Forward-pass` pro maskování hodnot v `Backward-pass`. Díky tomu jsou nastaveny hodnoty na nulu pouze na pozicích, které byly negativní při `Forward-pass`, jak lze vidět ve levém spodním rohu obrázku 3.6. Poté se gradient příznaku i v l vrstvě značen jako R_i^l vypočítá rovnicí

$$R_i^l = \sigma(z_i^l) \cdot R_i^{l+1}, \tag{3.1}$$

kde $R_i^{l+1} = \partial a^V / \partial a_i^{l+1}$ a a^V značí aktivaci nejvyšší vrstvy. $\sigma(z_i^l)$ je aktivační ReLU funkce.

Při použití metody `deconvolution`, popsána v sekci 3.1, je využita ReLU funkce na vrchní gradient. Hodnoty které jsou ve vrchním gradientu negativní jsou nastaveny na 0, jak lze vidět v pravém horním rohu obrázku 3.6. Gradient pro metodu deconvolution je pak vypočítán rovnicí

$$R_i^l = \sigma(R_i^{l+1}) \cdot R_i^{l+1}. \tag{3.2}$$

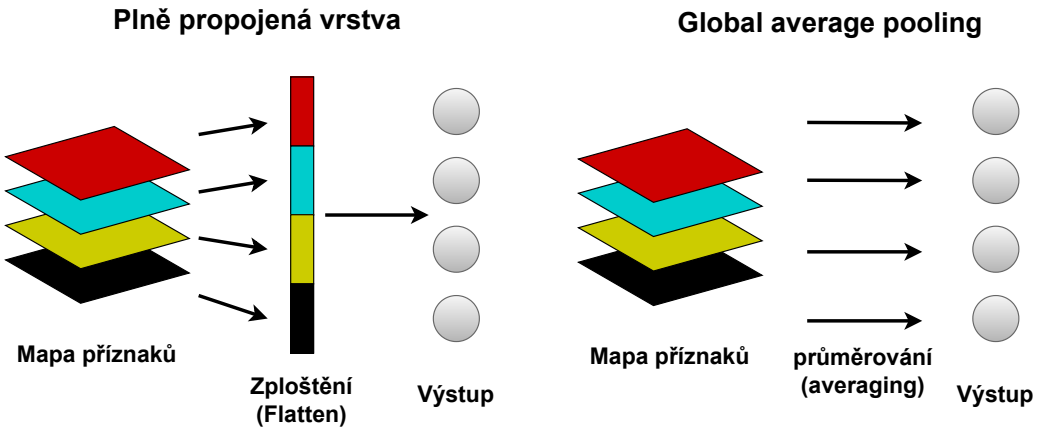
Pro metodu `guided Backpropagation` jsou tato dvě pravidla spojena do jednoho celku. Metoda využívá jak spodních, tak vrchních gradientů vypočítaných přes ReLU funkci nelinearity. Hodnoty jsou poté nastaveny na 0 pokud alespoň jeden z gradientu obsahuje hodnotu 0 na dané pozici, jak lze vidět v pravém spodním rohu obrázku 3.6. Gradient se poté spočítá rovnicí

$$R_i^l = \sigma(R_i^{l+1}) \cdot \sigma(z_i^l) \cdot R_i^{l+1}. \tag{3.3}$$

Metoda se nazývá `guided`, protože přidává další naváděcí signál z vyšších vrstev do obvyklé metody `backpropagation`.

3.3 Class Activation Mapping

V roce 2016 byla navržena nová metoda vizualizace pojmenována `Class Activation Mapping` (CAM) [9]. Metoda je schopná vytvořit tepelnou mapu označující důležité oblasti ve vstupním obrázku, které vedou k rozhodnutí sítě pro danou třídu. Tato metoda lze aplikovat pouze na plně propojené CNN, které využívají global average pooling a fungují bez jakékoliv plně propojené vrstvy. Ilustrace rozdílu lze vidět na obrázku 3.7.



Obrázek 3.7: Rozdíl mezi plně propojenou vrstvou a global average pooling.

Jak je znázorněno na obrázku 3.8, kde `global average pooling` je výstupem prostorového průměru mapy příznaků každé jednotky v poslední konvoluční vrstvě. Pro vytvoření konečného výstupu se použije vážený součet těchto hodnot. Podobně je vypočítaný vážený součet map funkcí poslední konvoluční vrstvy, pro získání aktivační mapy tříd.

Obrázek 3.8: Mapování skóre třídy metodou CAM [9].

Podrobnější vysvětlení je ukázáno na funkci `softmax`, kde výsledek `global average pooling` proveden po poslední konvoluční vrstvě je získán rovnicí

$$a_k = \frac{1}{Z} \sum_{x,y} f_k(x, y), \tag{3.4}$$

kde k je index určující neuron j v poslední konvoluční vrstvě a Z je počet pixelů. Pro daný obraz je $f_k(x, y)$ aktivace jednotky k v prostorovém umístění (x, y). Vstup do funkce `softmax`, S_c, je

$$S_c = \sum_k w_k^c a_k, \tag{3.5}$$

kde w_k^c je váha důležitosti odpovídající k třídě c pro jednotku k. V podstatě w_k^c indikuje míru důležitosti aktivace a_k pro třídu c. Výstup funkce `softmax` pro třídu c je dán rovnicí

$$P_c = \frac{\exp(S_c)}{\sum_c \exp(S_c)}. \tag{3.6}$$

Hodnota biasu má malý nebo žádný vliv na ovlivnění přesnosti klasifikace, a proto je nastaven na 0. Po připojení rovnice 3.4 do skóre třídy, S_c, je dáno rovnicí

$$\begin{aligned} S_c &= \sum_k w_k^c \frac{1}{Z} \sum_{x,y} f_k^V(x, y) \\ &= \frac{1}{Z} \sum_k \sum_{x,y} w_k^c f_k^V(x, y). \end{aligned} \tag{3.7}$$

Definice mapy aktivací třídy, M_c, pro třídu c, kde každý prostorový prvek je dán vztahem

$$M_c^{\text{CAM}}(x, y) = \sum_k w_k^c f_k(x, y), \tag{3.8}$$

kde M_c přímo ukazuje důležitost aktivace v prostorovém umístění (x, y) vedoucí ke klasifikaci obrazu do třídy c. Z toho vyplívá, že platí $S_c = (1/Z) \sum_{x,y} M_c(x, y)$.

3.4 Grad-CAM

V roce 2017 byla představena metoda Gradient-weighted Class Activation Mapping (Grad-CAM) [6], která zobecňuje metodu CAM [9], jelikož metoda CAM je omezena v použití pouze na plně konvoluční neuronové sítě. Grad-CAM lze použít na podstatně širší škálu neuronových sítí, včetně běžných konvolučních neuronových sítí s plně propojenými vrstvami na konci. Autoři metody Grad-CAM [6] uvedli:

> "Konvoluční vrstvy přirozeně zachovávají prostorové informace, které se v plně propojených vrstvách ztrácejí, takže lze očekávat, že poslední konvoluční vrstvy budou mít nejlepší kompromis mezi sémantikou na vysoké úrovni a detailními prostorovými informacemi. Neurony v těchto vrstvách hledají v obraze sémantické informace specifické pro danou třídu (řekněme části objektu). Grad-CAM využívá informace o gradientu proudící do poslední konvoluční vrstvy CNN k přiřazení hodnot důležitosti jednotlivým neuronům pro konkrétní zájmové rozhodnutí."

Za účelem získání třídně-diskriminační lokalizační mapy Grad-CAM $L^c_{\text{Grad-CAM}} \in \mathbb{R}^{u \times v}$ šířky u a výšky v pro libovolnou třídu c, vypočítáme nejprve gradient skóre pro třídu c, y_c (před softmaxem), vzhledem k aktivaci příznakové mapy f_k konvoluční vrstvy. Gradienty jsou globálně zprůměrovány (global average pooling) přes rozměry šířky a výšky, aby se získaly váhy důležitosti neuronů:

$$\alpha_k^c = \frac{1}{Z} \sum_{x,y} \frac{\partial y^c}{\partial f_k(x,y)}, \tag{3.9}$$

kde první část reprezentuje `global average pooling` a druhá část gradient skóre y_c (před softmaxem) vzhledem k hodnotě mapy příznaků f_k na pozici (x, y). Poté je provedena vážená kombinace dopředných aktivačních map a následně pomocí $ReLU$ získáme

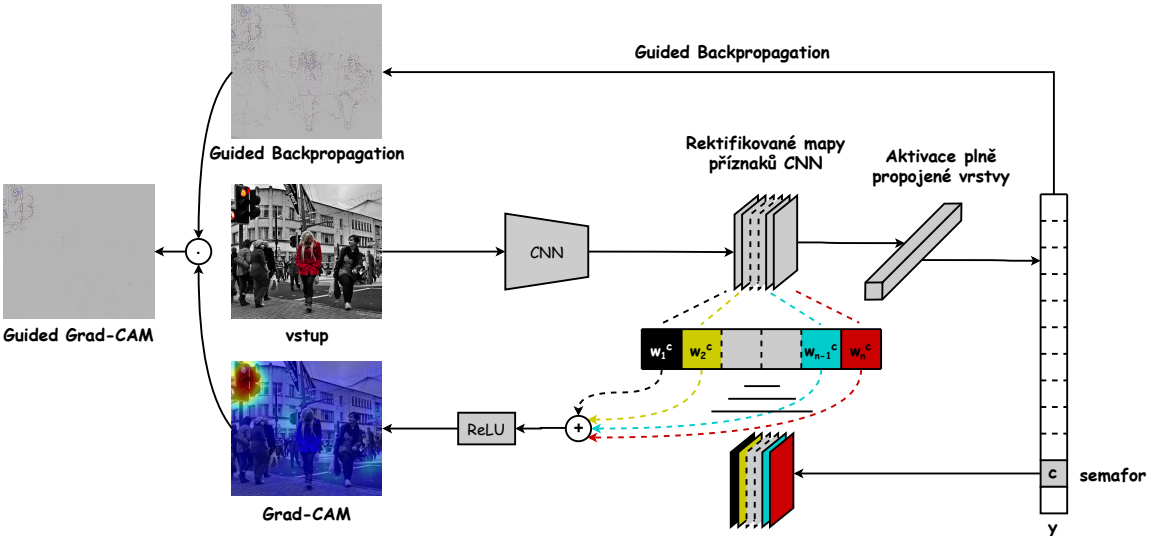
$$L^c_{\text{Grad-CAM}} = ReLU\left(\sum_k \alpha_k^c f_k\right). \tag{3.10}$$

Výsledkem je hrubá tepelná mapa stejné velikosti jako konvoluční mapy prvků.

3.5 Guided Grad-CAM

Grad-CAM sice rozlišuje třídy a lokalizuje relevantní oblasti obrazu, ale postrádá schopnost zvýraznit jemné detaily jako metody, vizualizující gradienty v pixelovém prostoru, Deconvolution popsána v sekci 3.1 a Guided Backpropagation popsána v sekci 3.2. Guided Backpropagation vizualizuje gradienty vzhledem k obrazu, kde jsou negativní gradienty při zpětném šíření přes vrstvy ReLU potlačeny. Intuitivně je cílem zachytit pixely detekované neurony, nikoliv ty, které neurony potlačují.

Proto byla představena metoda Guided Grad-CAM [6], která se snaží spojit nejlepší aspekty z Guided Backpropagation a Grad-CAM. Z hrubé tepelné mapy nelze detekovat proč síť předpovídá konkrétní případ dané třídy. Pro nejlepší výsledek jsou sloučeny metody Guided Backpropagation a Grad-CAM pomocí Hadamardového součinu pro získání nejlepších aspektů obou metod. Tepelná mapa $L^c_{\text{Grad-CAM}}$, před provedením Hadamardového součinu, je nejprve převzorkován na rozlišení vstupního obrazu pomocí bilineární interpolace. Obrázek 3.9 znázorňuje toto sloučení. Tato vizualizace má vysoké rozlišení a zároveň je třídně diskriminační.

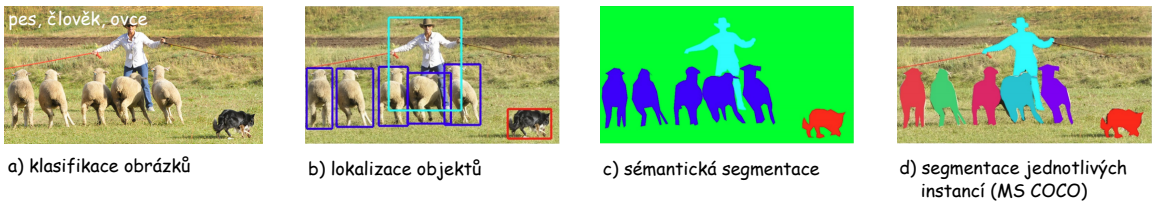


Obrázek 3.9: Přehled metody Guided Grad-CAM s vlastními výsledky z datasetu `MS COCO`.

Kapitola 4

Dataset Microsoft COCO

Jedním z hlavních cílů počítačového vidění je porozumět vizuálním scénám. Porozumění scény zahrnuje řadu úkolů včetně rozpoznání přítomných objektů, lokalizace objektů ve 2D a 3D, určení atributů objektů a scény, charakterizace vztahů mezi objekty a poskytnutí sémantického popisu scény. Microsoft COCO (MS COCO) [2] je rozsáhlý datový soubor, který se zabývá třemi základními výzkumnými problémy v porozumění scény: detekce neikonických pohledů na objekty, kontextové uvažování mezi objekty a přesná 2D lokalizace objektů. Dataset se také zaměřuje na segmentaci jednotlivých instancí Obr. 4.1.



Obrázek 4.1: Obrázek znázorňující klasifikaci, lokalizaci a segmentaci [2].

Obrázky

Obrázky jsou převzaty z datové sady *2017 Train images*. Jelikož datová sada je rozsáhlá, bylo vybráno 30 obrázků z pěti různých kategorií, nad kterými byly provedeny testy.

Anotace

V této práci byla využita anotace `instances_train2017.json`, která uchovává údaje o každém obrázku z tréninkové sady *2017 Train images*. COCO ukládá anotaci ve formátu JSON, která se skládá z pěti základních složek: `info, licenses, images, annotations, categories`.

`Info` obsahuje standardní informace/popis obrázků. `Images` všechny podrobnosti o každém jednotlivém obrázku. `Licenses` sestává z licence platné pro tento obraz. `Categories` obsahují seznam všech obrázků s jedinečným id a jeho superkategorií. Datová sada COCO obsahuje celkem 91 kategorií a 11 superkategorií. `Annotations` se skládá z `segmentation, iscrowd, image_id, category_id, id, bbox` a `area`. Z anotace je využita segmentace, obsahující seznam vrcholů polygonu, která určuje umístění třídy v obrázku.

Kapitola 5

Vyhodnocení metod

5.1 Základní myšlenka

Z datové sady `MS COCO` je vybráno několik tříd. Experimenty jsou prováděny na předem natrénovaných CNN z knihovny `PyTorch`, kde jejich natrénování bylo provedeno na datové sadě `ImageNet`. Kvůli odlišnosti datových sad je zapotřebí zvolit takové třídy, které jsou zastoupeny v obou datových sadách. Z datové sady COCO byly vybrány obrázky, které měly v anotaci výskyt vybraných tříd. Každá anotace poskytuje segmentaci konkrétní třídy. Tato segmentace je hlavním klíčem k vyhodnocení, jelikož přesně definuje, v jakém místě se třída nachází. Díky tomu lze určit, zdali metoda zobrazuje aktivace správného místa na vstupní pixelové vrstvě.

Výpočet je založený na jednotlivých aktivacích (gradientů) pixelů vstupní vrstvy. Pro získání celkové energie jsou sečteny mocniny hodnot. Výpočet je ekvivalentní k výpočtu energie diskrétního signálu, $E = \sum_n |x[n]|^2$. Pro získání energie dané třídy je vynásobena vstupní pixelová vrstva maskou dané třídy. Tím jsou získány hodnoty gradientů aktivujících danou třídu. Pro porovnání, jak moc byla energie koncentrována do místa výskytu třídy, je vyjádřena podílem energie místa třídy s celkovou energií.

Testy jsou prováděny na více typech CNN aby výsledky nebyly ovlivněny konkrétní strukturou CNN. Proto byly vybrány 4 typy CNN, na kterých byly testy provedeny. Jelikož datová sada `MS COCO` poskytuje obrázky v libovolném rozlišení, byly provedeny dvě sady testování. První sada předala na vstup CNN originální velikost obrázku. Poté byly vypočteny jednotlivé metody a výsledky byly uchovány. Druhá sada byla testována na stejných obrázcích, ale jejich veliost a poměr stran byl nastaven na jednu hodnotu a to `1:1` s rozlišením `224x224` pixelů. Změna velikosti zkoumá vliv velikosti obrázku na výsledek.

5.2 Výpočet energie

Rovnice pro výpočet energie obrázku i je definována

$$E_{celk} = \sum_{x,y} |I_i(x,y)|^2, \tag{5.1}$$

$$E = \sum_{x,y} |I_i(x,y) M_i(x,y)|^2. \tag{5.2}$$

E_{celk} znační celkovou energii obrázku. E značí energii v daném místě třídy vyskytující se na obrázku, které je určeno segmentací získanou z anotace datové sady `MS COCO`. I vyjadřuje vstupní pixelový prostor obrázku i s vypočítanými gradienty dané metody. M vyjadřuje masku vstupního pixelového prostoru, která určuje kde se na obrázku nachází požadovaná třída (na místech kde se daná třída vyskytuje je hodnota nastavena na 1, jinde na 0). Maska byla dilatována, o přibližně 20 %, aby bylo docíleno zvětšení prostoru výskytu obrázku a zachycen i prostor oblastí hran. x a y jsou indexy prostorového umístění.

5.3 Vyhodnocení

Koncentrace energie je vypočtena rovnicí

$$P = \frac{E}{E_{celk}}, \tag{5.3}$$

vyjadřující jakou koncentraci z celkové energie aktivace bylo vynaloženo danou třídou. Při vynásobení koncentrace $P * 100$ [%] získáme kolik procent segmentace dané třídy je zastoupeno v rozhodování CNN o skutečnosti, že se daná třída na obrázku nachází. Výsledný hodnoty byly vyhodnoceny a vyobrazeny v grafu, kde lze vidět jaká metoda byla nejúspěšnější.

Kapitola 6

Implementace

6.1 Použité technologie

Pro implementaci byl využit programovací jazyk Python[1]. Verze jazyka Python je doporučena od verze 3.6 výše. Nižší verze nejsou podporované ostatními knihovnami, které jsou v práci využívány.

Celá práce stojí na knihovně `PyTroch`[2]. `PyTorch` je open source knihovna pro strojové učení založená na knihovně `Torch`, která se používá pro aplikace, jako je počítačové vidění a zpracování přirozeného jazyka, kterou vyvinula především laboratoř AI Research (FAIR) společnosti Facebook. Jedná se o svobodný software s otevřeným zdrojovým kódem vydaný pod upravenou licencí BSD. Další důležitou knihovnou je `NumPy`[3]. `NumPy` je knihovna, která poskytuje infrastrukturu pro práci s vektory, maticemi a obecně vícerozměrnými poli. Dále byla využita knihovna `OpenCV`[4]. `OpenCV` je další softwarová knihovna s otevřeným zdrojovým kódem pro počítačové vidění a strojové učení, která obsahuje více než 2500 optimalizovaných algoritmů. Dále bylo využito nástroje `pycocotools`[5]. `Pycocotools` je Python API, které pomáhá při načítání, analýze a vizualizaci anotací v COCO. Ostatní využité knihovny jsou poměrně známé a není zapotřebí k nim nic dalšího uvádět. Jedná se o knihovny `matplotlib`[6], `pandas`[7] a `seaborn`[8]. Také bylo využito několika Python modulů: `sys`, `os`, `json` a `argparse`.

[1] https://www.python.org/
[2] https://pytorch.org/
[3] https://numpy.org/
[4] https://opencv.org/
[5] https://github.com/cocodataset/cocoapi/blob/master/PythonAPI/pycocotools/coco.py
[6] https://matplotlib.org/
[7] https://pandas.pydata.org/
[8] https://seaborn.pydata.org/

6.2 Metody

Celkové řešení je provedeno v objektově orientované terminologii. Každá z metod je zastoupena jednou třídou, která dědí z třídy `Nets`. Všechny metody jsou vypočteny nad poslední vrstvou každé CNN. Poslední vrstvy jsou uloženy a získány z datového JSON souboru. Je nutno podotknout, že v rámci celého řešení této práce a v souladu s využitím stejných technologií, byla část implementace metod převzata od uživatele *kazuto1011*[9]. Do implementace převzatých částí bylo zasahováno a byly upraveny podle potřeb práce.

Nets

Třída `Nets` je základní třídou všech metod. Jelikož v práci se pracuje s předem nadefinovanými CNN z importovaného balíčku `models` knihovny `PyTorch`, které mají podobné architektury (všechny využívají aktivační ReLU funkci, obsahují average pooling vrstvu atd.), lze pro celý program využít jednu všeobecnou třídu, která definuje základní funkce každé metody.

```
1  class Nets(nn.Module):
2          def __init__(self, model):
3              super(Nets, self).__init__()
4              self.model = model
5              self.hooks = []
6
7          def get_tensor_with_activated_class(self, class_index):
8              activated_class = torch.zeros_like(self.inputs)
9              activated_class.scatter_(1, class_index, 1.0)
10             return activated_class
11
12         def forward(self, image):
13             self.inputs = self.model(image)
14             self.outputs = func.softmax(self.inputs, dim=1)
15             return self.outputs.sort(dim=1, descending=True)
16
17         def backward(self, class_index):
18             grad = self.get_tensor_with_activated_class(class_index)
19             self.model.zero_grad()
20             self.inputs.backward(gradient=grad, retain_graph=False)
21
22         def clear(self):
23             for hook in self.hooks:
24                 hook.remove()
25             del self.inputs, self.outputs
```

Inicializační funkce, __init__, uloží typ CNN do třídní proměnné a inicializuje proměnnou hooks. Funkce get_tensor_with_activated_class vrátí požadovanou aktivovanou třídu, podle které se bude provádět zpětná propagace. Index je určen podle názvu třídy z datové sady ImageNet, která obsahuje 1000 tříd. Třídy a jejich názvy jsou uloženy v datovém JSON souboru. Funkce forward provede dopředný průchod CNN pro získání aktivací tříd, které jsou seřazené sestupně podle předpovědi sítě. Funkce backward provede zpětný průchod sítí pro získání gradientů požadované třídy. Funkce clear vymaže paměť po vypočítání metody, aby mohla být vypočtena další.

[9] https://github.com/kazuto1011

Volání metod

Každá z metod je v modulu `methods.py` volána podobně. Nejdříve je provedena inicializace metody s předáním typu sítě. Poté je proveden dopředný průchod CNN pro získání předpovědi sítě aktivací tříd. Poté je proveden zpětný průchod pro vypočtení gradientů požadované třídy. Třídy metod se můžou lišit ve vrácení gradientů, a proto je u každé metody volána příslušná funkce pro vrácení. Poté jsou získané gradienty uloženy, pomocí funkcí save_result nebo build_grad_cam. Hodnoty jsou uloženy ve formátu `png` pro vizualizaci, kde jsou gradienty upraveny do odstínu šedé. Hodnoty jsou uloženy také ve formátu `npy`, kde gradienty nejsou nijak upraveny a slouží pro výpočty. V poslední řadě jsou vymazány data, která byla vytvořena metodou, pomocí funkce clear.

```
1  bp = BackPropagation(model=model)
2  p, img_classes = bp.forward(img)
3
4  numpy_img_classes = img_classes[0].cpu().numpy()
5
6  index = np.where(numpy_img_classes == int(class_dict[class_name]))[0][0]
7  class_index = img_classes[:, [index]]
8
9  bp.backward(class_index=class_index)
10 gradients = bp.return_gradients()
11
12 save_result(filename=filename,
13               img_name=img_name + "-back_propagation",
14               gradients=gradients[0], save=save)
15
16 bp.clear()
```

Prvním dopředným průchodem jsou získány data o aktivací tříd. Tato data jsou zapotřebí získat pouze jednou pro daný obrázek. Ze získaných dat je vybrán index, class_index, požadované třídy, který je použit pro funkci get_tensor_with_activated_class třídy `Nets`. Tento index je poté využit u všech metod.

Backpropagation

Třída `Backpropagation` je ze všech tříd nejjednodušší. Funkce forward provede dopředný průchod sítě s nastaveným parametrem requires_grad_, který zaznamenává všechny operace provedeny nad daným obrázkem. Zpětný průchod je definován v třídě `Nets`. Funkce return_gradients vrátí vypočtené gradienty a vynuluje třídní proměnnou, která je obsahovala.

```
1  class BackPropagation(Nets):
2      def forward(self, img):
3          self.img = img.requires_grad_()
4          return Nets.forward(self, self.img)
5
6      def return_gradients(self):
7          gradient = self.img.grad.clone()
8          self.img.grad.zero_()
9          return gradient
```

Deconvolution

Třída `Deconvolution` je téměř identická k třídě `Guided Backpropagation`. Využívá se registrace zpětného hooku. Hook může registrovat funkci na Modulu nebo Tensoru. V metodě je využit backward_hook, který je aktivován při zpětném průchodu. Hooky umožňují provádět věci během šíření. Jedná se v podstatě o funkci, která se vykoná při zpětném, nebo dopředném průchodu. Funkce standardně nemění gradienty. Je-li však vrácena pomocí příkazu return, vrácená hodnota bude výstupem gradientu.

```
1  class deconvnet(Nets):
2      def ___init___(self, model):
3          super(deconvnet, self).___init___(model)
4
5          def backward_hook(module, grad_in, grad_out):
6              if isinstance(module, nn.ReLU):
7                  return (func.relu(grad_out[0]),)
8
9          for module in self.model.named_modules():
10             hook = module[1].register_backward_hook(backward_hook)
11             self.hooks.append(hook)
12
13     def forward(self, img):
14         self.img = img.requires_grad_()
15         return Nets.forward(self, self.img)
16
17     def return_gradients(self):
18         gradient = self.img.grad.clone()
19         self.img.grad.zero_()
20         return gradient
```

Funkce backward_hook je definovaná v inicializaci metody. Funkce detekuje všechny `ReLU` moduly CNN v zpětném šíření. Při zaregistrování je využit parametr grad_out, který obsahuje gradienty ztráty vzhledem k výstupu vrstvy a tím jsou eliminovány negativní gradienty a přeskočen výpočet `Backpropagation` Obr. 3.6. Funkce forward provede dopředný průchod CNN. Funkce return_gradients vrátí vypočtené gradienty metody.

Guided Backpropagation

Třída `Guided Backpropagation` pracuje na stejném principu jako `Deconvolution`. Jediný rozdíl je ve funkci backward_hook, kde je místo parametru grad_out využit parametr grad_in, který obsahuje všechny informace potřebné pro dopředný průchod. Tím vymaže všechny negativní gradienty a provede výpočet `Backpropagation`. Tím jsou spojeny výstupy metod `Backpropagation` a `Deconvolution` Obr. 3.6.

```
1  class guided_back(Nets):
2      def __init__(self, model):
3          super(guided_back, self).__init__(model)
4
5          def backward_hook(module, grad_in, grad_out):
6              if isinstance(module, nn.ReLU):
7                  return (func.relu(grad_in[0]),)
8
9          for module in self.model.named_modules():
10             hook = module[1].register_backward_hook(backward_hook)
11             self.hooks.append(hook)
12
13     def forward(self, img):
14         self.img = img.requires_grad_()
15         return Nets.forward(self, self.img)
16
17     def return_gradients(self):
18         gradient = self.img.grad.clone()
19         self.img.grad.zero_()
20         return gradient
```

Funkce forward provede dopředný průchod CNN. Funkce return_gradients vrátí vypočítané gradienty metody a vynuluje třídní proměnnou.

Grad-CAM

Třída `Grad-CAM` je nejrozsáhlejší třídou. Třída generuje tepelnou mapu `cam`, která vyznačuje místo výskytu třídy. Při inicializaci metody jsou deklarovány proměnné self.gradients, která uchovává gradienty metody a self.activations, která uchovává aktivace. Proměnná self.target_layer určuje vrstvu/modul CNN, podle kterého má být provedena vizualizace. Funkce save_gradients a save_activations ukládají gradienty a aktivace. Cyklus prochází všechny moduly CNN a aplikuje hook pouze na modul, který byl zadán.

```
1  class grad_cam(Nets):
2      def __init__(self, model, target_layer=None):
3          super(grad_cam, self).__init__(model)
4          self.gradients = []
5          self.activations = []
6          self.target_layer = target_layer
7
8          def save_gradients():
9              def backward_hook(module, grad_in, grad_out):
10                 self.gradients = [grad_in[0]] + self.gradients
11             return backward_hook
12
13         def save_activations():
14             def forward_hook(module, input, output):
15                 self.activations.append(output)
16             return forward_hook
17
18         for name, module in self.model.named_modules():
19             if name == self.target_layer:
20                 self.hooks.append(module.register_forward_hook(save_activations()))
21                 self.hooks.append(module.register_backward_hook(save_gradients()))
22
23
24     def create(self, img_shape):
25         activs = self.activations[-1].cpu().data.numpy()[0, :]
26         grads = self.gradients[-1].cpu().data.numpy()[0, :]
27         cam = np.zeros(activs.shape[1:], dtype=np.float32)
28         weights = np.mean(grads, axis=(1, 2))
29
30         for i, w in enumerate(weights):
31             cam += w * activs[i, :, :]
32
33         cam = np.maximum(cam, 0)
34         cam = cv2.resize(cam, img_shape)
35         cam = cam - np.min(cam)
36         cam = cam / np.max(cam)
37
38         del self.gradients, self.activations
39
40         return cam
```

Funkce create vytvoří samotnou tepelnou mapu `cam` z hodnot gradientů a aktivací průchodů CNN. Jelikož tepelná mapa je stejné velikosti, jak vrstva, na kterou byla metoda aplikována, musí být změněna její velikost pomocí knihovny `cv2` na velikost vstupní pixelové vrstvy. Funkce poté vrací gradienty tepelné mapy metody.

Guided Grad-CAM

Metoda `Guided Grad-CAM` nepotřebuje vlastní třídu, jelikož výsledek metody je součin výstupů metod `Guided Backpropagation` a `Grad-CAM`. Jediný problém je v nastavení stejných dimenzí objektu tenzor. V implementaci je výstup `Grad-CAM` jiné dimenze a proto je zapotřebí, pomocí funkce reshape, dimenze upravit do stejného tvaru, jako má výstup metody `Guided Backpropagation`.

```
1  cam = torch.tensor(cam)
2  cam = torch.reshape(cam, (1, 1, img.shape[2], img.shape[3]))
3
4  guided_grad_cam_grads = torch.mul(cam.cpu(), guided_back_grads.cpu())
```

Po upravení dimenzí jsou gradienty metod vynásobeny pomocí Hadamardového součinu. Výsledkem součinu je výstup metody `Guided Grad-CAM`.

6.3 Vyhodnocení metod

Implementace vyhodnocení metod je rozdělena do dvou modulů: `experiments.py` a `graphs.py`. Tyto moduly obstarávají výpočet a vyhodnocení výsledků nad získanými gradienty metod.

experiments.py: Modul obstarává výpočet energie, popsán v sekci 5.2. Pro efektivnější práci s anotací `MS COCO` bylo využito knihovny `pycocotools`, která usnadňuje práci s anotací. Výpočet je proveden pro každou metodu a nad každým obrázkem datové sady. Výsledky jsou poté zapsány do JSON souboru `exp.json`. Struktura `exp.json` souboru je rozdělena podle velikosti obrázku a to na originální a změněnou (resized) velikost. O daném obrázku uchovává jeho `id`, `třídu`, `metodu` které se dané výsledky týkají, `CNN` nad kterou byla metoda obrázku vypočtena, `P` značící koncentraci energie, `E` pro energii místa třídy a `E_celk` pro celkovou energii obrázku.

graphs.py: Modul z vypočtených hodnot modulu experiments.py vytvoří grafy pro vizualizaci výsledků. Jelikož soubor s výsledky obsahuje několik set až tisíců výsledků, je využita knihovna `pandas` pro práci s datovými rámci. Do datového rámce jsou načteny všechny výsledky a pomocí knihovny `seaborn` a `matplotlib` jsou vykresleny grafy nad celou sadou výsledků. Grafy jsou uloženy ve formátu `png` a `eps`.

Kapitola 7

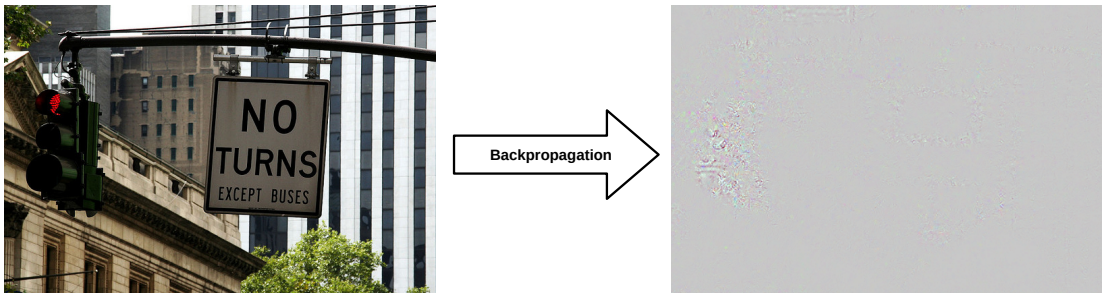
Experimenty

Cílem práce bylo porovnat jednotlivé metody vizualizace. Vyhodnocení bylo provedeno na zásadě výpočtu energie obrazu z dané datové sady a poté vypočítáním koncentrace energie v místě výskytu třídy. Hodnoty byly zprůměrovány a vyobrazenu na grafu 7.6. Vyhodnocení bylo provedeno i po vizualizační stránce a výsledky jednotlivých metod byly uloženy ve formě `.png`. Metody by měly dát člověku podnět toho co vidí a podle čeho se počítačové vidění rozhoduje. Z matematické stránky lze lépe usoudit která metoda je vizualizačně přesnější, ale nedokáže přesně určit, které části obrazu nejvíce ovlivnili výsledek.

Implementace byla provedena pro možnost využití několika různých CNN. Díky této flexibilitě lze vyloučit ovlivnění výsledků špatnou konstrukcí CNN. Dalším faktorem, který byl v práci brán v potaz je velikost obrazu. Jelikož obrázky z datové sady byly brány nahodile a každý z obrázků může mít jiné rozměry, byly metody počítány na dvou různých rozměrových sad vstupních obrázků. První sada byla počítána na základní velikosti obrázků. Druhá sada byla počítána na změněné velikosti vstupního obrázku na fixní rozměry 224x224 pixelů. Tímto byl zkoumán vliv velikosti vstupních dat na výsledných hodnotách.

Backpropagation

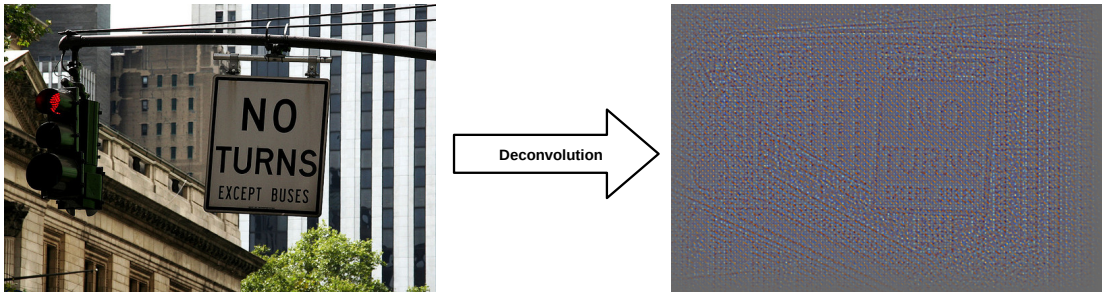
U metody `Backpropagation` lze vidět aktivaci v místě výskytu dané třídy, avšak nelze přesně určit z jakých faktorů se daná CNN rozhodla o vyhodnocení konkrétní třídy. Při zobrazení metody nelze rozpoznat jaký objekt se na obrázku vyskytoval a lze výsledek porovnávat pouze se vstupním obrázkem, a to s informací o třídě, které se výsledek týká. Na obrázku 7.1 lze vidět převaha šumu než konkrétních příznaků třídy. Vizualizací pomocí metody `Backpropagation` nelze dostat přívětivých výsledků podle kterých by se daly jednoduše a přesně určit příznaky důležité k rozhodnutí.



Obrázek 7.1: Grafická ukázka metody Backpropagation detekující třídu semafor.

Deconvolution

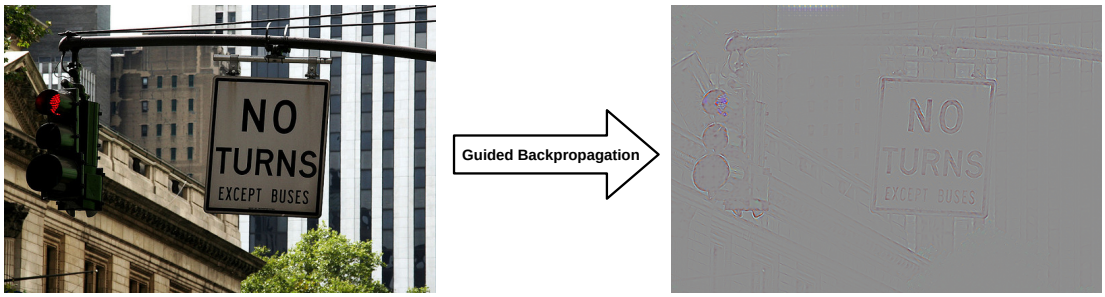
U metody `Deconvolution` jsou výsledky lepší a lze rozpoznat, co se na vstupním obrázku vyskytovalo. Stále ovšem převažuje šum a metoda přesně neukazuje příznaky, podle kterých se CNN rozhodovala o vyhodnocení dané třídy. Z výsledků na obrázku 7.2 by mohl pozorující usoudit o detekci značky, avšak metoda se snažila vizualizovat příznaky náležící třídě semaforu. V místě semaforu stále nelze určit konkrétní příznaky třídy, a tak vizualizací se toho moc dozvědět nedá.



Obrázek 7.2: Grafická ukázka metody Deconvolution detekující třídu semafor.

Guided Backpropagation

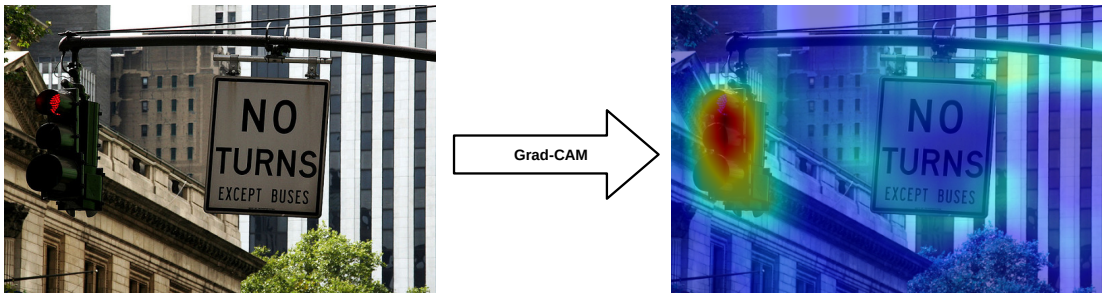
Metoda `Guided Backpropagation` přináší podstatně lepší výsledky. Ve vizualizaci výsledné metody již nepřevažuje šum a lze vidět konkrétní místa a detaily obrázku, které pro danou CNN byly důležité k rozhodnutí o dané třídě Obr. 7.3. Stále lze vidět i aktivace míst, které se dané třídy netýkají a pro třídu semafor se zvýraznila i místa v oblasti značky. Při opominutí aktivací na více místech lze z vizualizace poukázat na hlavní příznaky, podle kterých se CNN rozhodla o vyhodnocení třídy *semafor*.



Obrázek 7.3: Grafická ukázka metody guided Backpropagation detekující třídu semafor.

Grad-CAM

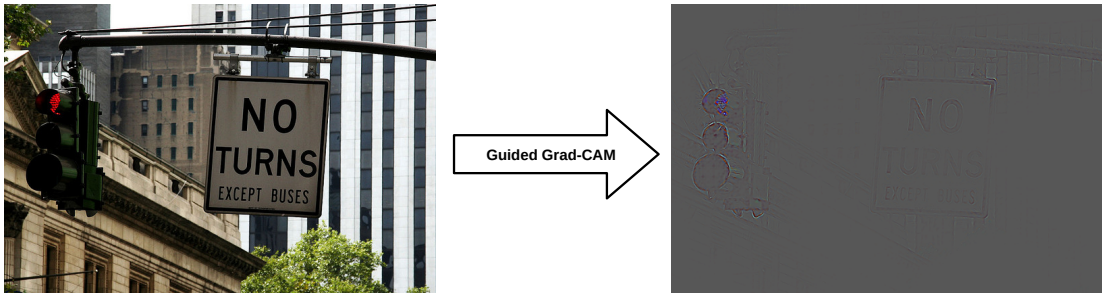
U metody `Grad-CAM` je vyhodnocení jiné. Metoda se nezaměřuje na detaily příznaků, podle kterých se CNN rozhodovala, ale na konkrétní místo výskytu dané třídy. Spojením tepelné mapy a vstupního obrazu lze vizualizovat vyhodnocení CNN o výskytu dané třídy Obr. 7.4. V oblasti, kde je tepelná mapa červená až rudá se daná třída vyskytuje a v ostatních oblastech se tepelná mapa zbarvuje do modré barvy. Z vizuální stránky tak lze nejlépe rozpoznat o správnosti vyhodnocení CNN o detekci dané třídy, ale nepřináší nám konkrétní informace o rozhodnutí, proč a podle čeho CNN danou třídu detekovala.



Obrázek 7.4: Grafická ukázka metody Grad-CAM detekující třídu semafor.

Guided Grad-CAM

Metoda `Guided Grad-CAM` vznikne spojením metod `Grad-CAM` a `Guided Backpropagation`. Spojením přesného výskytu třídy a zvýrazněním detailů lze nejlépe rozpoznat, jaké příznaky CNN využila k vyhodnocení třídy na obrázku. Díky tepelné mapě jsou ostatní místa, kde se třída nevyskytuje, potlačena a díky tomu je výsledek přesný a zřetelný Obr. 7.5.



Obrázek 7.5: Grafická ukázka metody guided Grad-CAM detekující třídu semafor.

Matematické vyhodnocení

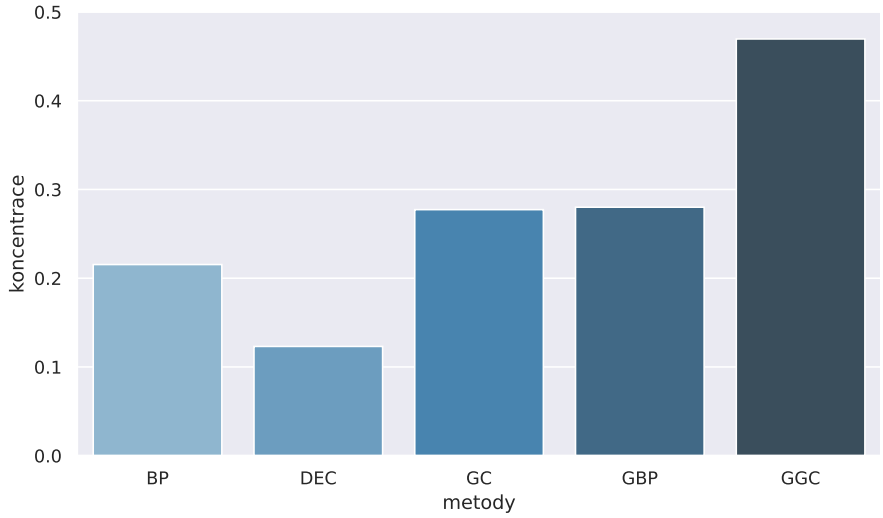
Díky výpočtu celkové energie aktivace a její koncentrace v místě třídy získanou pomocí segmentace lze vypočítat procentuální úspěšnost každé metody. V tabulce 7.1 lze vidět výsledky výpočtů provedené pro tuto práci.

metody	počet záznamů	suma koncentrací	průměrná koncentrace
Backpropagation	1440	310.17	0.215
Deconvolution	1440	177.30	0.123
Grad-CAM	1440	399.27	0.277
Guided Backpropagation	1440	403.22	0.280
Guided Grad-CAM	1440	676.19	0.470

Tabulka 7.1: Tabulka vyhodnocení koncentrace energie.

Z tabulky lze vyčíst názvy metod a k nim počet záznamů z kterých byly vypočítány výsledné hodnoty. Sloupec *suma koncentrací* ukazuje celkový součet všech hodnot P získaných z výsledků metod. Sloupec *průměrná koncentrace* vyjadřuje průměrnou koncentraci každé metody.

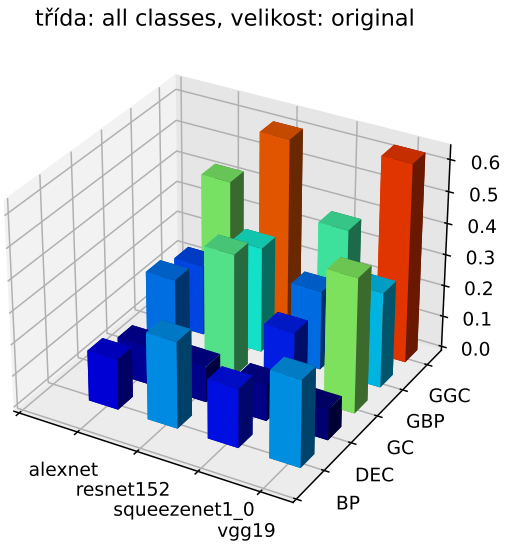
Na grafu 7.6 jsou tyto hodnoty graficky znázorněny. Z grafu jde vidět, že nejlepších výsledků dosáhla metoda `Guided Grad-CAM`, která vznikla spojením metod `Grad-CAM` a `Guided Backpropagation`, které dosáhly také velmi dobrých výsledků. Nejhůře se umístila metoda `Deconvolution`.



Obrázek 7.6: Graf koncentrace pro všechny vypočítané hodnoty.

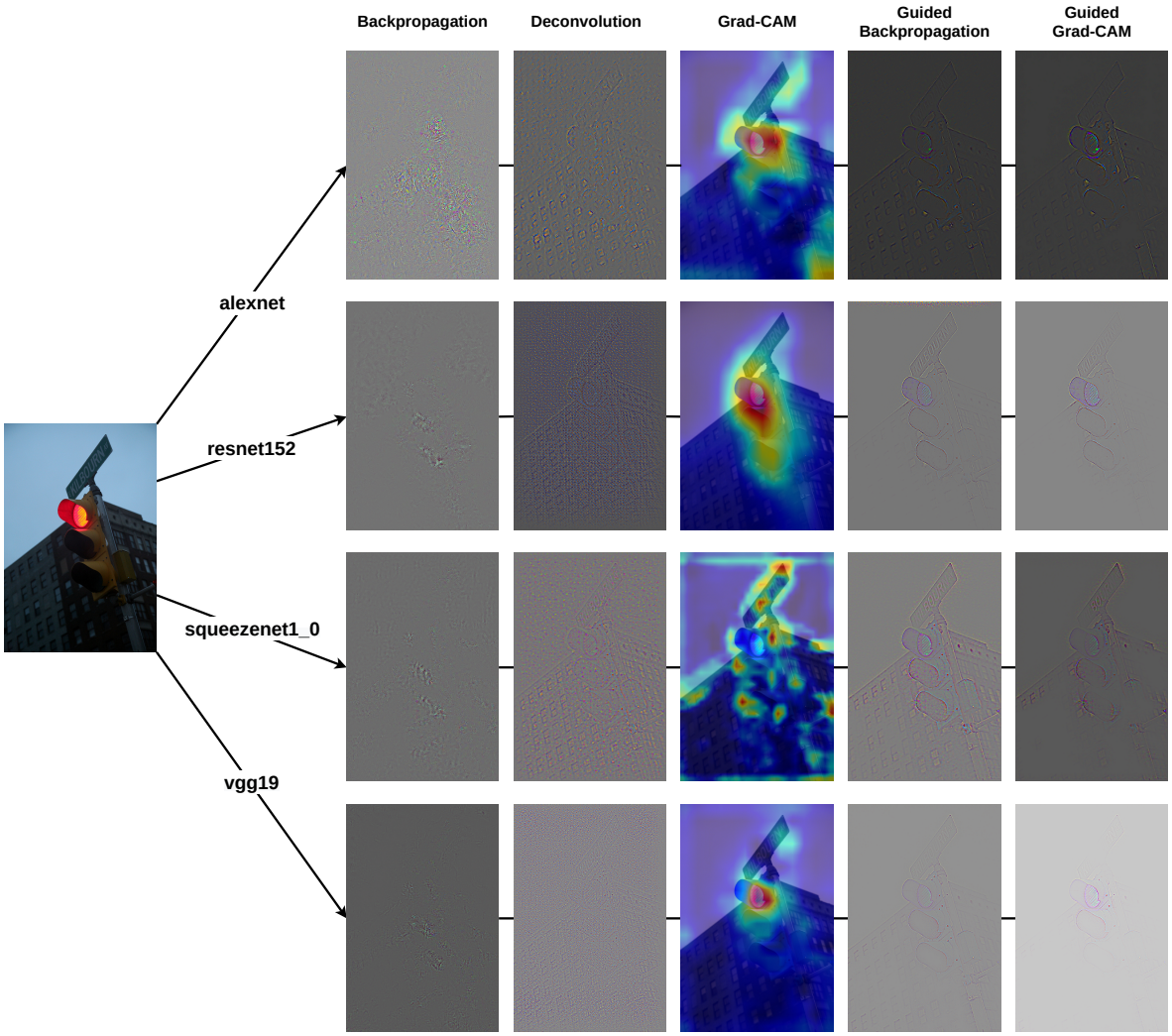
Závěr experimentů

Vliv architektury CNN na výsledek Vliv architektury CNN mělo velký dopad na hodnoty získaných výsledky, ale ne na pořadí jednotlivých metod. Na grafu 7.7 lze vidět, že u každé CNN byla nejlépe umístěna metoda `Guided Grad-CAM`. Největší odlišnosti mezi CNN byly u metod `Backpropagation` a `Deconvolution`.



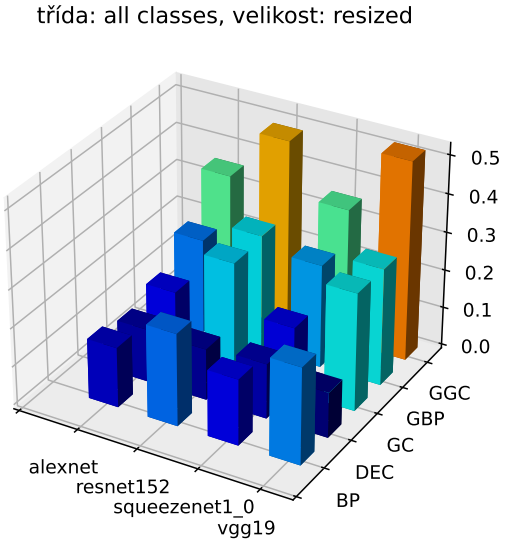
Obrázek 7.7: Graf koncentrace pro originální velikost.

Na obrázku lze vidět odlišnost architektur CNN. Každá síť je natrénována jinak a detekuje jiná místa obrazu pro danou třídu. Díky tomu jsou hodnoty odlišné pro každou třídu a je složité vyhodnotit metody nad jednou třídou. Na obrázku 7.8 lze vidět jak se jednotlivé CNN liší v detekci stejné třídy. Každá z použitých CNN dokáže detekovat stejné třídy, ale s rozlišnou úspěšností. Proto byly experimenty provedeny na několika různých třídách.



Obrázek 7.8: Vizualizace příznaků metodami pomocí různých CNN.

Vliv rozměru obrázku na výsledek Zkoumán byl také vliv velikosti vstupních dat na výsledcích daných metod. Na grafu 7.9 jsou ukázány výsledky metod pro stejnou datovou sadu jak na grafu 7.7, akorát vstupní data byla změněna na velikost 224x224 pixelů. Z porovnání lze vidět, že vliv na výsledné hodnoty, změnou velikosti, byl pouze negativní. Poměrově se hodnoty jednotlivých metod vůči sobě nijak nezměnily, pouze se zmenšily sumy a průměry koncentrací metod.



Obrázek 7.9: Graf koncentrace pro velikost 224x224 px.

Kapitola 8

Závěr

Cílem práce bylo porovnat metody vizualizace a získat povědomí o tom, co CNN vidí a podle čeho se rozhoduje při výpočtu predikce jednotlivé třídy. Jelikož existuje několik různých metod, které slouží pro vizualizaci, byly v této práci porovnány některé z nich. Všechny metody, které se v práci vyskytly, disponují společnou vlastností, a to výpočtem zpětné propagace na základě získaných gradientů. Na základě této vlastnosti byly metody vybrány, aby jejich porovnání bylo smysluplné.

Každá metoda byla naimplementována pomocí knihovny `PyTorch` s možností výpočtu hodnot nad různými typy CNN. Metody byly testovány na datové sadě o 6 třídách po 30 obrázcích získaných z datové sady `MS COCO`. Výsledné hodnoty jednotlivých metod byly uloženy do proměnné numpy s kterou se poté prováděli výpočty. Po provedení výpočtů metod nad všema obrázky z datové sady byla data připravena k porovnání.

Porovnání bylo provedeno pomocí získané segmentace, která byla získána ke každému obrázku z anotace `MS COCO`. Pomocí výpočtu celkové energie obrázku a energie koncentrace obrázku, získaná součtem energie vyskytující se v místě segmentace, bylo získáno procentuální zastoupení třídy z celkové energie. Díky tomu bylo možné prokázat, zdali jednotlivá metoda dokázala správně detekovat třídu na obrázku.

Pomocí výsledků získaných z výpočtů byla nejlépe vyhodnocena metoda `Guided Grad-CAM`, která vznikla spojením výsledků metod `Guided Backpropagation` a `Grad-CAM`. Její přesnost vykreslení příznaků na vstupní pixelovou vrstvu je znatelně nejlepší a detailně ukazuje podle čeho se CNN rozhodla o predikci třídy. V porovnání s ostatními metodami je její výkonnost znatelně nejlepší jak po vizuální stránce, tak po stránce vyhodnocení výpočtů.

Díky možnosti vizualizace jednotlivých metod lze vidět do CNN a původní `black box` se nyní otevírá a uživatel může nahlédnout do vnitřní struktury CNN. Díky tomu je možné zlepšovat výkonnost CNN a upravovat jejich strukturu k dosažení lepších výsledků.

Literatura

[1] Goodfellow, I., Bengio, Y. a Courville, A. *Deep Learning*. Illustrated edition. MIT Press, 2016. ISBN 978-0262035613. http://www.deeplearningbook.org.

[2] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P. et al. Microsoft COCO: Common Objects in Context. In:. Duben 2014, sv. 8693. DOI: 10.1007/978-3-319-10602-1_48. ISBN 978-3-319-10601-4.

[3] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com/.

[4] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain [J]. *Psychol. Review*. Prosinec 1958, sv. 65, s. 386 – 408. DOI: 10.1037/h0042519.

[5] Rumelhart, D., Hinton, G. a Williams, R. Learning Representations by Back Propagating Errors. *Nature*. Říjen 1986, sv. 323, s. 533–536. DOI: 10.1038/323533a0.

[6] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, s. 618–626. DOI: 10.1109/ICCV.2017.74.

[7] Springenberg, J., Dosovitskiy, A., Brox, T. a Riedmiller, M. Striving for Simplicity: The All Convolutional Net. Prosinec 2014.

[8] Zeiler, M. D. a Fergus, R. Visualizing and understanding convolutional networks. In: *In Computer Vision–ECCV 2014*. Springer, 2014, s. 818–833.

[9] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. a Torralba, A. Learning Deep Features for Discriminative Localization. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 2921–2929. DOI: 10.1109/CVPR.2016.319.