Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Plzeň, 2023 **Filip Majer**

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Filip MAJER**
Osobní číslo:	**A20B0334P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Umělá inteligence a automatizace**
Téma práce:	**Detekce objektů s využitím textových dotazů**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Nastudujte problematiku neuronových sítí, architekturu Transformer a Vision Transformer, techniku transfer-learning.
2. Proveďte rešerši dostupných datasetů pro úlohu detekce objektů.
3. Navrhněte architekturu modelu pro detekci objektů v obrazu s využitím textových dotazů.
4. Model natrénujte, proveďte zhodnocení a navrhněte případná vylepšení.

Rozsah bakalářské práce: **30-40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

VASWANI, Ashish, et al. Attention is all you need. Advances in neural information processing systems, 2017, 30.
DOSOVITSKIY, Alexey, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.
DEVLIN, Jacob, et al. Bert: Bidirectional Encoder Representations from Transformers. 2016.

Vedoucí bakalářské práce: **Ing. Jan Švec, Ph.D.**
Katedra kybernetiky

Datum zadání bakalářské práce: **17. října 2022**
Termín odevzdání bakalářské práce: **22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 17. října 2022

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 17. května 2023

.....

podpis

Poděkování

Tímto bych rád poděkoval vedoucímu bakalářské práce panu Ing. Janu Švecovi, PhD. za skvělé vedení práce a poskytování cenných rad a věcných připomínek. Mé děkování patří také organizaci MetaCentrum (MetaVO) za poskytnutí výpočetních zdrojů.

Anotace

V teoretické části práce je představena obecná teorie dopředných, rekurentních a konvolučních neuronových sítí. Následuje podrobný popis Transformer architektury, použitých modelů a datasetů pro detekci objektů. V praktické části práce byla navržena architektura modelu pro detekci objektů v obrazu s využitím textových dotazů. Dotazy mohly být jednoslovné nebo ve formě vět. Pro obě varianty bylo natrénováno několik modelů s různými kombinacemi parametrů. Na závěr byly tyto modely vyhodnoceny a byla navržena některá vylepšení.

Klíčová slova: neuronové sítě, Transformer, BERT, RoBERTa, Vision Transformer, transfer learning, COCO, detekce objektů, textové dotazy, detekce objektů s využitím textových dotazů

Abstract

In the theoretical part of the thesis, the general theory of feedforward, recurrent and convolutional neural networks is presented. This is followed by a detailed description of the Transformer architecture, important models and datasets used for object detection. In the practical part of the thesis, a model architecture has been proposed for object detection using textual queries. The queries could be a single word or in the form of sentences. For both variants, several models were trained with different parameter combinations. Lastly, these models were evaluated and improvements were proposed.

Keywords: neural networks, Transformer, BERT, RoBERTa, Vision Transformer, transfer learning, COCO, object detection, textual queries, object detection using textual queries

Obsah

1 Úvod

Pro celý obor strojového učení jsou naprosto klíčová data, která mohou nabývat různých forem jako například text, obraz nebo zvukový záznam. Každá forma dat vyžaduje odlišné techniky zpracování, čímž vznikla jednotlivá odvětví strojového učení jako například Zpracování přirozeného jazyka (angl. Natural Language Processing - zkr. NLP) nebo Počítačové vidění (angl. Computer Vision - zkr. CV). Tato dvě odvětví se dlouhá léta vyvíjela odděleně a vytvořila si řadu vlastních modelů a architektur. Pro NLP byly nejrozšířenější rekurentní neuronové sítě, pro CV konvoluční neuronové sítě. Popisu těchto sítí je věnována první část bakalářské práce. Jsou zde vysvětleny základní principy jejich fungování s důrazem na některé nedokonalosti a problémy, které tyto sítě mají.

V roce 2017 byla představena nová architektura neuronových sítí - Transformer architektura. Modely využívající tuto architekturu začaly velice rychle dominovat v celé oblasti NLP a následně i CV, i když v menší míře. Hlavními výhodami těchto modelů je efektivní trénování na velkém množství dat bez anotací a schopnost naučit se z dat velice složité závislosti. Popisu Transformer architektury je věnována druhá část. Dále jsou také zmíněny důležité Transformer modely, které jsou využity v praktické části práce.

Spolu s Transformer modely byl v posledních letech zaznamenán také velký vývoj v oblasti tzv. multimodálního strojového učení. Multimodální strojové učení se zabývá algoritmy, které jsou schopny učit se z dat využívajících více modalit jako je text, obraz nebo zvuk. Inspiraci nachází v tom, jakým způsobem lidé získávají informace ze světa. Ti současně sbírají data ze svého okolí pomocí různých smyslů (zrak, sluch, chuť...) a tato data pak kombinují a vyvozují z nich závěry.

Do kategorie multimodálních úloh spadá také hlavní úloha této bakalářské práce, a sice navrhnout architekturu modelu, který bude detekovat objekty s využitím textových dotazů. Architektura bude mít dva vstupy. První vstup bude obrázek, druhý vstup bude textový dotaz na daný obrázek. Dotazy budou v angličtině a mohou být v jednoslovném tvaru ("dog") nebo ve tvaru věty ("Find a dog."). Výstupem bude označení hledaného objektu, pokud se na obrázku vyskytuje. Jelikož se bude jednat o originální architekturu a přístup k úloze, nebude hlavním cílem natrénovat co nejlepší model. Důležitější bude ověřit, že se model učí požadované závislosti a je schopný úlohu řešit. V závěru proběhne vyhodnocení a bude zkoumán vliv parametrů modelu na jeho úspěšnost řešení úlohy.

2 Neuronové sítě

Neuronová síť je systém strojového učení, který používá síť funkcí k mapování vstupních dat na požadovaný výstup. Inspiraci nachází v biologickém fungování neuronů v lidském mozku. Oproti dřívějším tradičním přístupům strojového učení, jako je logistická regrese nebo SVM (Support Vector Machines), jsou neuronové sítě schopné naučit se daleko složitější závislosti ve vstupních datech (jsou schopné aproximovat silně nelineární funkce). To ale pouze za předpokladu, že mají dostatečné množství parametrů.

Základním stavebním blokem neuronových sítí je tzv. perceptron. Jedná se o nejpoužívanější model neuronu. Funkci perceptronu lze popsat následujícím vzorcem:

$$\hat{y}(k+1) = f(\sum_{i=1}^{n} w_i x_i(k) + b) = f(\mathbf{w}^T \mathbf{x}(k) + b), \tag{1}$$

kde f je aktivační funkce, $\mathbf{w} = [w_1, w_2, ..., w_n]^T$ je váhový vektor, $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ je vstupní vektor a b je práh (angl. bias). V případě, kdy máme N trénovacích dvojic $(\mathbf{x}, y)$, kde y je požadovaný výstup, můžeme pro každou trénovací dvojici porovnat výstup perceptronu s požadovaným výstupem a upravit váhový vektor spolu s prahem tak, aby byly výstupy co nejpodobnější. Jinými slovy zadefinujeme tzv. ztrátovou funkci L a proces trénování bude spočívat v minimalizaci této funkce. Existuje více způsobů, jak nadefinovat ztrátovou funkci. Následující vzorec (2) je příkladem tzv. střední kvadratické chyby (angl. mean squared error, zkr. MSE) [1].

$$L = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 \tag{2}$$

Umělá neuronová síť vznikne spojením jednotlivých modelů neuronů. Výsledná funkce sítě je určena způsobem propojení jednotlivých neuronů (tzv. topologií sítě), váhami těchto spojení a způsobem činnosti jednotlivých neuronů (tj. aktivační funkcí).

2.1 Dopředné neuronové sítě

Dopředné neuronové sítě, nazývané též vícevrstvé perceptrony (angl. multilayer perceptrons, zkr. MLPs), jsou základním typem neuronových sítí. Jednotlivé neurony jsou uspořádány do vrstev, přičemž výstup jedné vrstvy je připojen na vstup následující vrstvy. Tím vzniká řetězová struktura. Rozšířením vzorce (1) na více neuronů v jedné vrstvě získáme vztah pro první vrstvu neuronové sítě

$$\mathbf{h}^{(1)} = g^{(1)}(\mathbf{W}^{(1)T} \mathbf{x} + \mathbf{b}^{(1)}), \tag{3}$$

kde $g^{(1)}$ je aktivační funkce první vrstvy, $\mathbf{W}^{(1)T}$ je váhová matice první vrstvy a $\mathbf{b}^{(1)}$ je prahový vektor první vrstvy. Každá další i-tá vrstva je pak definována jako

$$\mathbf{h}^{(i)} = g^{(i)}(\mathbf{W}^{(i)T} \mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}). \tag{4}$$

Jak název napovídá, signál se u dopředných sítí šíří pouze ze vstupu sítě $\mathbf{x}$ na její výstup $\mathbf{y}$ a cestou může procházet přes libovolné množství tzv. skrytých vrstev. Poslední

vrstvě se říká výstupní vrstva. Počet vrstev určuje tzv. hloubku modelu a počet neuronů ve vrstvě určuje šířku modelu. Jednotlivé vrstvy definují výsledné zobrazení $\mathbf{y} = f(\mathbf{x}; \theta)$, kde θ jsou parametry sítě (váhové matice a prahové vektory) [1] [2].

Cílem trénování je nastavit prostřednictvím trénovacích dat parametry θ tak, aby síť co nejlépe aproximovala nějakou funkci f^* danou testovacími daty. Při trénování určujeme, jaké výsledky chceme po výstupní vrstvě, ale nespecifikujeme chování skrytých vrstev. Učící algoritmus se musí sám rozhodnout, jak tyto vrstvy co nejlépe využít k získání požadovaného výstupu [2].

2.2 Rekurentní neuronové sítě

Rekurentní neuronové sítě (angl. recurrent neural networks, zkr. RNNs) jsou sítě se zpětnou vazbou, určené pro zpracování sekvenčních dat, tj. sekvence hodnot $\mathbf{x}^{(1)}$, ..., $\mathbf{x}^{(t)}$, kde $\mathbf{x}^{(t)}$ je vzorek získaný v časovém kroku t. Mezi úlohy vhodné pro RNN tedy patří například rozpoznávání řeči, strojový překlad nebo analýza DNA sekvencí. Důležitou vlastností sekvencí je, že prvek $\mathbf{x}^{(t)}$ v čase t závisí na prvcích předchozích. Z tohoto důvodu je vhodné jej přivést na vstup RNN až v okamžiku, kdy byly zpracovány předchozí prvky a informace z nich byla zakódována do tzv. skrytého stavu $\mathbf{h}^{(t)}$. Skrytý stav je tedy funkcí $f(\cdot, \cdot)$ vstupního vektoru v čase t a skrytého stavu v čase $t-1$

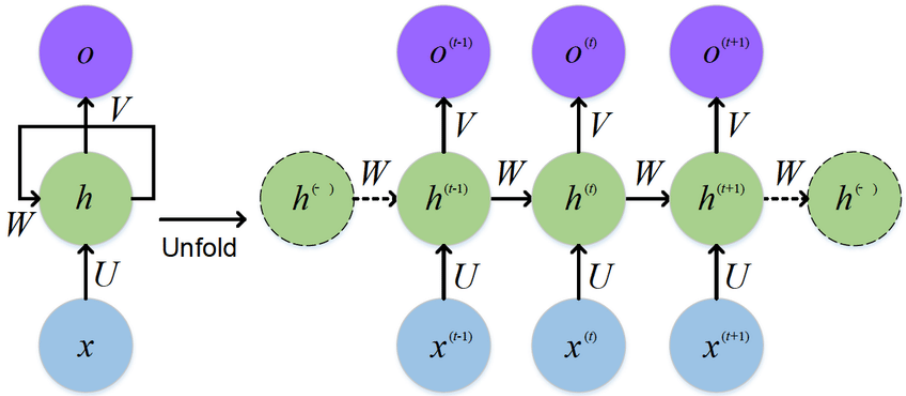
$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}). \tag{5}$$

Výstupní vektor $\mathbf{o}^{(t)}$ je získán odlišnou funkcí $g(\cdot)$ skrytého stavu $\mathbf{h}^{(t)}$. Při rozepsání funkcí f a g lze chování RNN popsat vztahy

$$\begin{aligned} \mathbf{h}^{(\mathbf{t})} &= tanh(\mathbf{W_{xh}}\mathbf{x}^{(\mathbf{t})} + \mathbf{W_{hh}}\mathbf{h}^{(\mathbf{t-1})}), \\ \mathbf{o}^{(\mathbf{t})} &= \mathbf{W_{hy}}\mathbf{h}^{(\mathbf{t-1})}. \end{aligned} \tag{6}$$

Dopředná síť není na úlohy se sekvenčními daty vhodná, neboť umí na vstup přivést pouze celou sekvenci naráz. Navíc je architektura dopředné sítě závislá na velikosti vstupu a výstupu. Naproti tomu většina RNN umí zpracovávat sekvence proměnné délky [1].

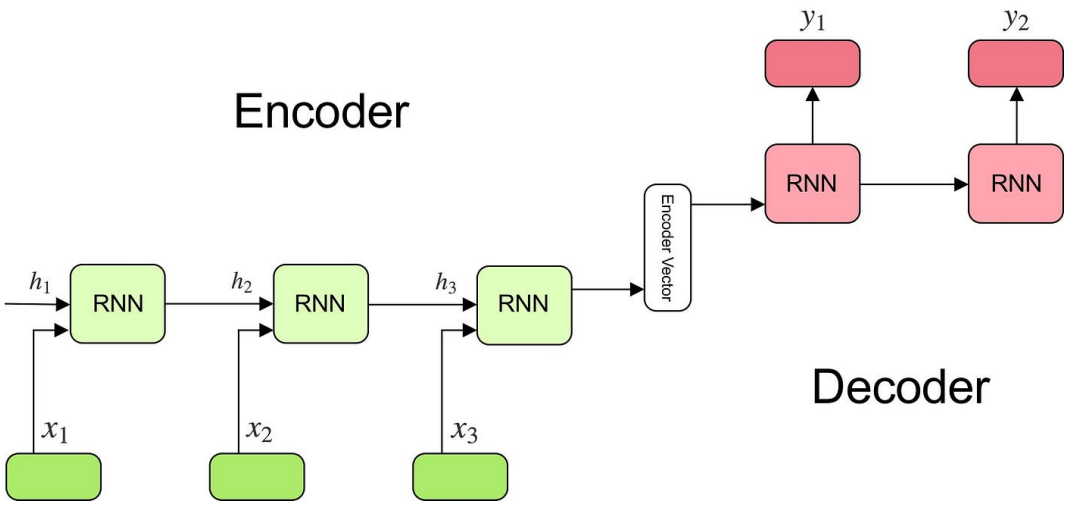
Na obrázku 1 je dvěma způsoby zobrazeno schéma RNN. Zpětná vazba v levé části obrázku lze rozvinout a rekurentní síť zobrazit jako acyklický orientovaný graf bez rekurence. Z obrázku si lze všimnout další důležité vlastnosti rekurentních sítí, a sice sdílení parametrů. RNN používá pro výpočet v každém časovém kroku stejnou sadu matic parametrů. To jí umožňuje generalizovat na různě dlouhé sekvence neviděné při procesu trénování. Sdílení parametrů je obzvláště důležité v případě, kdy se stejná informace vyskytuje v textu na více místech [2].



Obrázek 1: Schéma rekurentní neuronové sítě. Převzato z [3]

2.2.1 Enkodér-dekodér architektura

Existují různé typy RNN architektur, jejichž názvy se odvíjejí od toho, jestli vstupní a výstupní sekvence jsou jednoprvkové nebo víceprvkové, například one-to-many, many-to-one nebo many-to-many. Příkladem many-to-many architektury je tzv. enkodér-dekodér architektura, která je velmi rozšířená hlavně u úlohy strojového překladu. Enkodér-dekodér architektura se skládá ze dvou rekurentní sítí, kde první RNN má za úkol zakódovat postupně vstupní sekvenci do vektoru pevné délky (vektor kontextu) a druhá RNN má za úkol tento vektor dekódovat do výstupní sekvence. Z obrázku 2 je zřejmé, že vektor kontextu je vlastně poslední skrytý stav enkodéru [4] [5].



Obrázek 2: Enkodér-dekodér architektura RNN. Převzato z [5]

V případě dlouhých sekvencí se ukázalo jako obtížné pokusit se veškerou informaci zakódovat do jediného vektoru kontextu. Jako řešení byl v roce 2014 představen mechanismus *attention*, který umožňoval modelu soustředit se při dekódování na důležité části vstupní sekvence [6] [7].

Attention mechanismus zavádí do enkodér-dekodér architektury několik změn. Dekodéru jsou v každém okamžiku predikce výstupního prvku předány všechny skryté stavy enkodéru.

Každý skrytý stav enkodéru je asociován s jedním prvkem ve vstupní sekvenci. V daném časovém kroku přiřadí dekodér každému skrytému stavu skóre podle jeho důležitosti při predikci výstupního prvku. Na skóre se aplikuje funkce softmax. Těmito výstupy se váží skryté stavy enkodéru, které se dále sečtou, čímž vznikne vektor kontextu. Vektor kontextu se spojí se skrytým stavem dekodéru a nechá se projít dopřednou neuronovou sítí. Výstup dopředné neuronové sítě je výstupem dekodéru v daném časovém kroku [6] [7].

2.2.2 Obousměrné rekurentní neuronové sítě

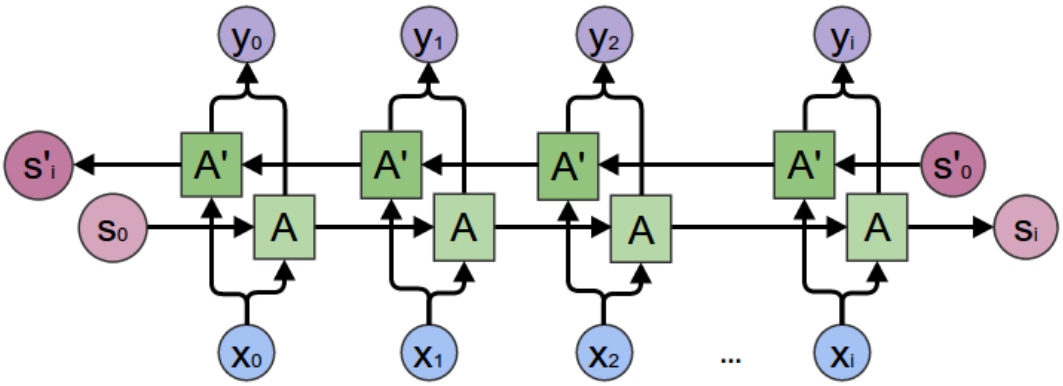
Jednou z nevýhod rekurentních sítí je, že aktuální skrytý stav $\mathbf{h}_t$ má znalost pouze o předchozích zpracovaných hodnotách v sekvenci. V určitých aplikacích je docíleno lepších výsledků, pokud má skrytý stav znalost také o budoucích hodnotách v sekvenci. To je dosaženo tzv. obousměrnou rekurentní neuronovou sítí.

Obousměrné rekurentní neuronové sítě (angl. bidirectional recurrent neural networks, zkr. BRNN) jsou speciálním typem rekurentních neuronových sítí, které umožňují informacím plynout jak dopředu, tak i dozadu v čase. Díky tomu má model přístup k informacím z budoucnosti a minulosti současně, což mu umožňuje daleko lépe porozumět kontextu vstupních dat. Mezi úlohy vhodné pro BRNN patří například rozpoznávání ručně psaného textu nebo rozpoznávání řeči, kde interpretace aktuálního zvuku záleží také na zvuku, který následuje, a to z důvodu koartikulace [2].

BRNN se skládá ze dvou rekurentních sítí, kdy jedna zpracovává vstupní data v tradičním dopředném pořadí a produkuje skrytý stav $\mathbf{h}_\mathbf{t}^{(\mathbf{f})}$, zatímco druhá zpracovává stejná data v opačném pořadí a produkuje skrytý stav $\mathbf{h}_\mathbf{t}^{(\mathbf{b})}$. Výstupy obou sítí jsou následně spojeny a použity pro predikci výstupního vektoru $\mathbf{y_t}$. Obě rekurentní sítě mají svoje vlastní matice parametrů. Matice dopředné RNN jsou značeny $\mathbf{W}_\mathbf{xh}^{(\mathbf{f})}$, $\mathbf{W}_\mathbf{hh}^{(\mathbf{f})}$ a $\mathbf{W}_\mathbf{hy}^{(\mathbf{f})}$. Matice zpětné RNN jsou značeny $\mathbf{W}_\mathbf{xh}^{(\mathbf{b})}$, $\mathbf{W}_\mathbf{hh}^{(\mathbf{b})}$ a $\mathbf{W}_\mathbf{hy}^{(\mathbf{b})}$. Při použití *tanh* aktivační funkce můžeme činnost BRNN popsat vztahy

$$\begin{aligned}
\mathbf{h}_\mathbf{t}^{(\mathbf{f})} &= tanh(\mathbf{W}_\mathbf{xh}^{(\mathbf{f})}\mathbf{x_t} + \mathbf{W}_\mathbf{hh}^{(\mathbf{f})}\mathbf{h}_\mathbf{t-1}^{(\mathbf{f})}), \\
\mathbf{h}_\mathbf{t}^{(\mathbf{b})} &= tanh(\mathbf{W}_\mathbf{xh}^{(\mathbf{b})}\mathbf{x_t} + \mathbf{W}_\mathbf{hh}^{(\mathbf{b})}\mathbf{h}_\mathbf{t-1}^{(\mathbf{b})}), \\
\mathbf{y_t} &= \mathbf{W}_\mathbf{hy}^{(\mathbf{f})}\mathbf{h}_\mathbf{t}^{(\mathbf{f})} + \mathbf{W}_\mathbf{hy}^{(\mathbf{b})}\mathbf{h}_\mathbf{t}^{(\mathbf{b})}.
\end{aligned} \tag{7}$$

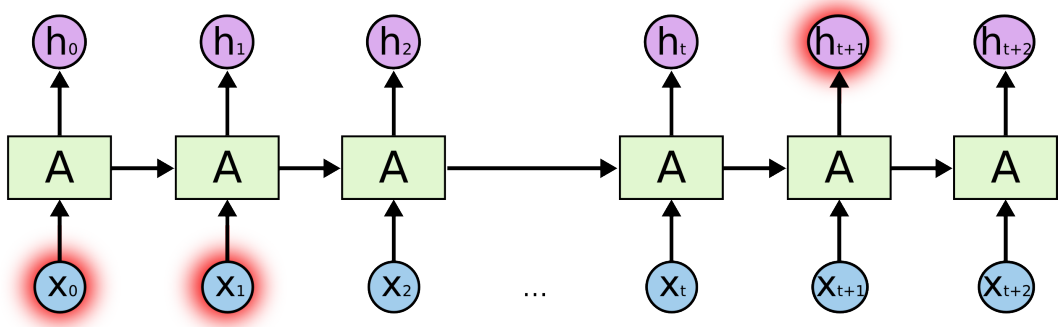
Ze vztahů si lze všimnout, že dopředné a zpětné skryté stavy spolu interagují jen při výpočtu výstupního vektoru $\mathbf{y_t}$. To znamená, že je možné spočítat si dopředné a zpětné skryté stavy odděleně a až poté je využít na výpočet výstupního vektoru [1].



Obrázek 3: Schéma obousměrné rekurentní neuronové sítě. Převzato z [8]

2.2.3 LSTM

Jak bylo zmíněno v kapitole 2.2, při zpracování aktuálního prvku sekvence hodnot umí RNN využít znalost o předchozích prvcích v sekvenci. Například při predikci posledního slova ve větě "Na obloze jsou ..." zvládne bez problému RNN doplnit slovo *mraky*. Problém nastává, pokud jsou například v textu relevantní informace od sebe hodně vzdálené. Pokud by věta začínala "Vyrůstal jsem ve Francii...", následovalo by několik dalších slov nebo vět a až poté by měla RNN doplnit poslední slovo ve větě "mluvím plynně ...", mohla by mít problémy, i když je zřejmé, že správná odpověď je *francouzsky*. Čím více slov je mezi relevantní informací a místem, kde je tato informace potřeba, tím horší je úspěšnost predikce [9].

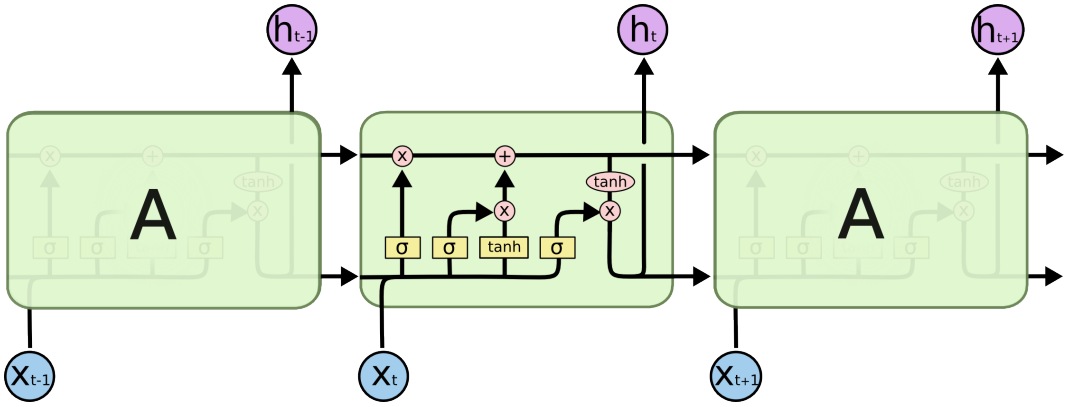


Obrázek 4: Ilustrace problému vzdálených souvislostí. Převzato z [9]

Long Short-Term Memory (zkr. LSTM) sítě jsou speciální typem rekurentních neuronových sítí, které byly navrženy pro řešení výše zmíněného problému zachycení vzdálených souvislostí v dlouhých sekvencích. Kromě toho řeší další problém klasických RNN při trénování na dlouhých sekvencích, a sice problém zanikajícího gradientu. LSTM sítě byly poprvé představeny v práci Hochreitera a Schmidhubera z roku 1997 [10] a rychle se staly velmi oblíbeným nástrojem pro zpracování sekvencí dat v mnoha aplikacích, jako je rozpoznávání řeči a strojový překlad.

Princip fungování LSTM sítí spočívá v použití speciálních jednotek nazvaných buňky (angl. cell), které umožňují zachovat informace v průběhu trénování sítě. Buňky obsahují tři brány (angl. gates): forget gate, input gate a output gate. Forget gate umožňuje síti

“zapomenout“ nechtěné informace (zapomenout svůj uchovaný stav), input gate umožňuje novým informacím vstoupit do buňky a output gate umožňuje síti “rozhodnout“, jestli informace má být vydána ven z buňky. Díky těmto bránám jsou LSTM sítě schopné efektivně pracovat se sekvencemi dat různé délky a udržovat si informace v dlouhodobé paměti. Schéma LSTM buňky je zobrazeno na obrázku 5, kde si lze také všimnout horizontální cesty napříč buňkami. Tato cesta zajišťuje volný tok gradientu při trénování [2] [9].



Obrázek 5: Schéma LSTM buněk. Převzato z [9]

2.3 Konvoluční neuronové sítě

Konvoluční neuronové sítě (angl. convolutional neural networks, zkr. CNNs) jsou speciálním typem neuronových sítí pro zpracovávání dat uspořádaných do mřížky. Typickým příkladem takových dat jsou obrázky, na které lze nahlížet jako na 2-D mřížku pixelů [2].

Podobně jako dopředné neuronové sítě nejsou samotné vhodné na zpracování sekvencí, nejsou příliš vhodné ani na zpracování obrázků. Pokud bychom uvažovali, že každý pixel obrázku je jeden prvek vstupního vektoru $\mathbf{x}$, pak pro RGB obrázek velikosti 640x480 bychom měli hned v první vrstvě téměř milion parametrů. V obrázcích se navíc vyskytují opakující se vzory jako například hrany. Bylo by vhodné, aby bylo možné detekovat tyto hrany za pomocí jedné relativně malé sady parametrů, kterou bychom aplikovali postupně na části obrázku. Stejně jako v případě RNN vede tato myšlenka na sdílení parametrů [1] [4].

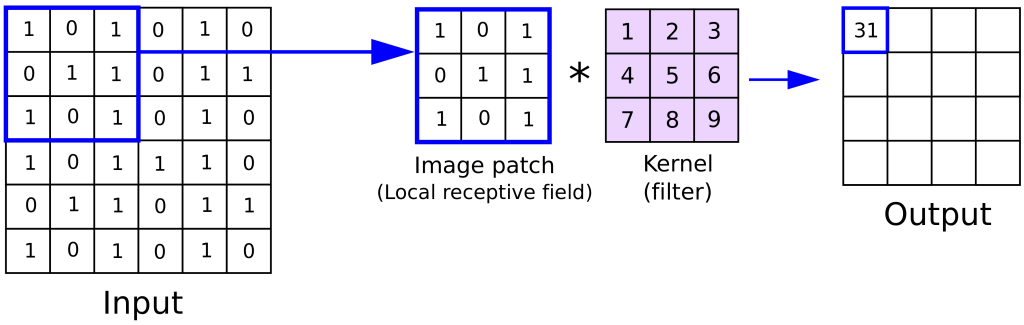
2.3.1 Popis činnosti

Konvoluční sítě využívají v případě černobílých obrázků operaci diskrétní konvoluce 2-D obrázku I s 2-D jádrem K popsanou vztahem

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{8}$$

Jádro K (nazývané také angl. filter) obsahuje trénovatelné parametry a bývá daleko menší než zpracovávaný obrázek. Při konvoluci se jádrem prochází vstupní obrázek, sečtou se výsledky součinu po prvcích a výsledná hodnota se uloží na odpovídající místo do výstupní matice. Tento proces je znázorněn na obrázku 6. Pro CNN je důležitá lokalita, což znamená,

že jelikož je jádro menší než zpracovávaný obrázek, CNN se na začátku soustředí na (lokální) části obrázku.

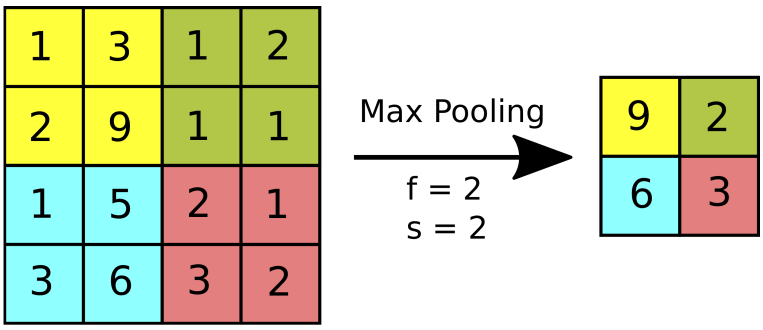


Obrázek 6: Operace 2-D diskrétní konvoluce. Převzato z [11]

Na obrázku 6 si lze všimnout, že konvoluce zmenšuje vstupní matici a znevýhodňuje pixely na kraji, které jsou využity méně než pixely uprostřed. Řešením takového problému je tzv. padding, kdy se rozšíří vstupní matice ve všech směrem o určitých počet řádek a sloupců. Výsledky konvoluce lze také upravit zvolením tzv. stride, který nám říká, o kolik pixelů se při konvoluci budeme posouvat s jádrem [4].

V případě RGB obrázků je nutné v konvoluci použít 3-D jádro. Konvoluce RGB obrázku s 3-D jádrem ale zredukuje výsledek na 2-D matici, čímž ztratíme jednu dimenzi. Řešením je použití více konvolucí s více jádry a spojení výsledných 2-D matic do jediné 3-D matice, kde velikost poslední dimenze je určena počtem použitých jader. Samozřejmě je nutné, aby měla jádra stejnou velikost a byl použit stejný stride [4].

Kromě konvoluce využívají CNN také operaci, která se nazývá anglicky pooling. Pooling slouží ke snížení velikosti dimenzí při zachování důležitých informací. Používají se dva typy - max pooling a average pooling. Používanější je max pooling, který podobně jako konvoluce prochází po částech obrázek a z určitého okolí vždy vybere největší hodnotu a tu uloží do výsledné matice. Při poolingu se tedy neučí žádné parametry [4].



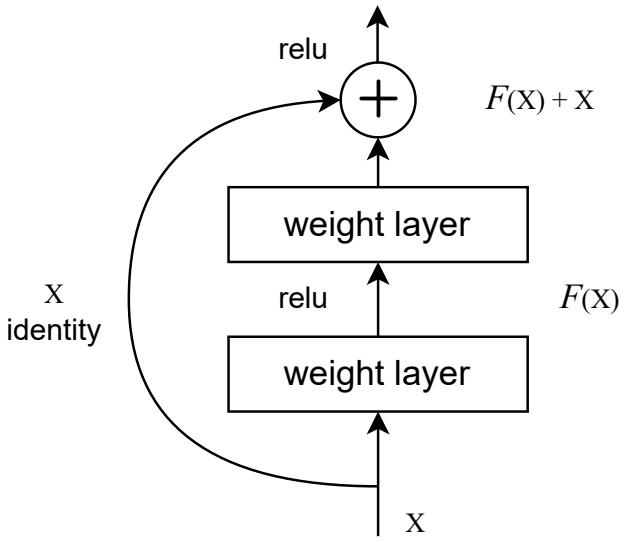
Obrázek 7: Max pooling s jádrem velikosti 2x2 a stride=2. Převzato z [11]

V případě RGB obrázku umí pooling zmenšit pouze první dvě dimenze. Pro snížení velikosti třetí dimenze se využívá operace 1D konvoluce, tedy konvoluce s jádrem velikosti 1x1. Při 1D konvoluci se také někdy používá ReLU aktivační funkce, která zavádí do CNN další nelinearitu a umožňuje síti naučit se složitější závislosti [4].

2.3.2 ResNet

ResNet (Residual Network) je hluboká konvoluční neuronová síť, která byla představena v roce 2015 v článku *Deep Residual Learning for Image Recognition* [12]. V té době přišla s řešením několika problémů hlubokých neuronových sítí. Očekávalo se, že čím více vrstev budou sítě mít, tím lepší a složitější závislosti se dokáží naučit. Bohužel se ukázalo, že od určitého bodu se sítě s větším počtem vrstev spíše zhoršovaly. Kromě toho se u hlubokých neuronových sítí projevoval problém s mizejícím gradientem [12].

V síti ResNet byly tyto problémy odstraněny s využitím reziduálních bloků a tzv. skip connections. Jedná se o přičtení původního vstupu vrstvy na jejím konci a aplikaci ReLU aktivační funkce. Hlavní přínos skip connections je při trénování, kdy toto spojení umožňuje přímý tok gradientů napříč sítí.



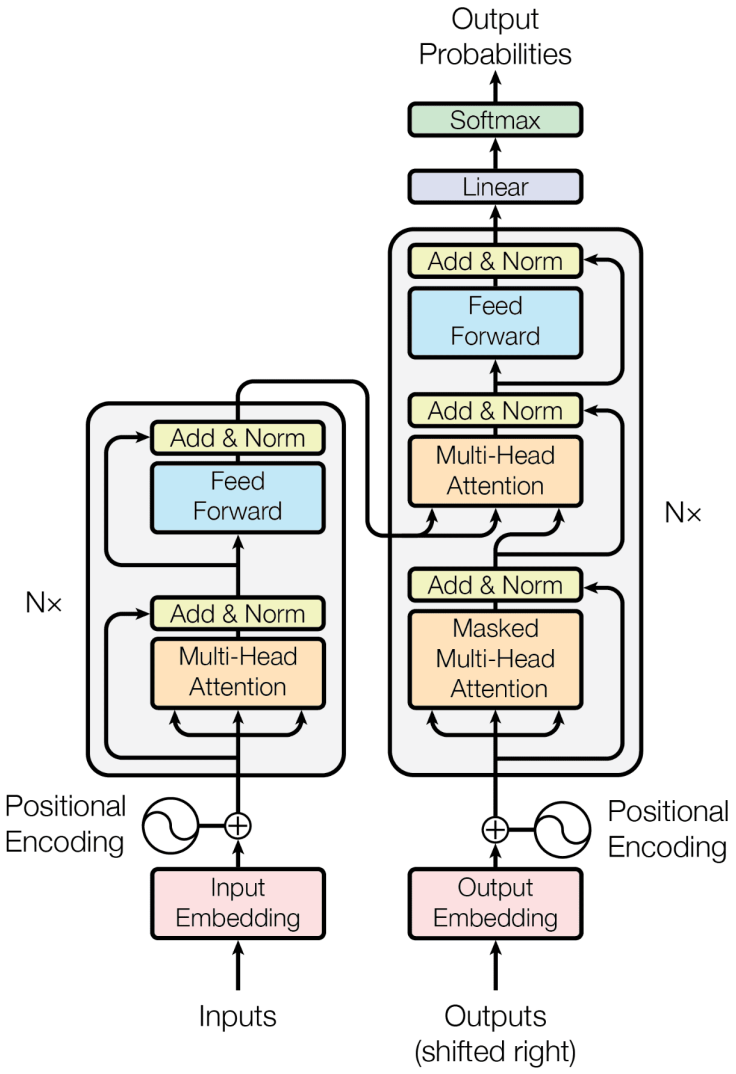
Obrázek 8: Schéma skip connection

3 Transformer architektura

Transformer architektura byla poprvé představena v roce 2017 v článku *Attention Is All You Need* [13]. Vychází z enkodér-dekodér architektury, která byla do té doby považována za state-of-the-art přístup při zpracování sekvencí (například strojový překlad řeči). Transformer architektura nevyužívá rekurenci, jako je to v případě rekurentních neuronových sítí, a spoléhá se pouze na attention mechanismus, který je popsán v kapitole 2.2.1. Tím se výrazně zlepšila paralelizace výpočtů a zkrátil čas nutný pro trénování [13].

3.1 Popis architektury

Původní architektura z článku *Attention Is All You Need* je zobrazena na obrázku 9. V následujících kapitolách je podrobně popsána. Při popisu byly informace čerpány z [13] a [14].



Obrázek 9: Transformer architektura. Převzato z [13]

3.1.1 Enkodér a dekodér

Enkodér se skládá z N identických bloků. Dimenze vektorů mezi bloky zůstávají zachovány. Každý tento blok se skládá z multi-head self-attention vrstvy a dopředné neuronové sítě. Kolem obou těchto vrstev je zavedeno reziduální spojení (kapitola 2.3.2), které zlepšuje tok gradientů napříč sítí. Výsledek je normalizován přes vrstvy (angl. layer normalization), což znamená, že se normalizace provádí s ohledem na statistiky neuronů v rámci jedné vrstvy namísto statistik napříč dávkami dat (angl. batch normalization). Obecně lze říci, že normalizace zlepšuje stabilitu a konvergenci trénování neuronových sítí. Layer normalization je podrobněji popsána v článku [15].

Dekodér se skládá ze stejného počtu identických bloků jako enkodér. Dekodér využívá na rozdíl od enkodéru dvě multi-head self-attention vrstvy, přičemž první z nich počítá self-attention z výstupní sekvence a sice tak, že dekodér nevidí budoucí prvky ve výstupní sekvenci při trénování. Této vrstvě se říká masked multi-head self-attention. Druhá z nich počítá cross-attention mezi vstupní sekvencí zpracovanou enkodérem a výstupní sekvencí.

3.1.2 Vektorová reprezentace vstupu

V případě, že na vstupu Transformer architektury je posloupnost slov, je nutné tato slova převést na vektory. Předpokládáme, že máme nějaký slovník, ze kterého tato slova pochází. Slovo by bylo možné reprezentovat například vektorem samých nul, který by měl jedničku na pozici, která odpovídá pozici slova ve slovníku. Takové reprezentaci se říká one-hot vektor. Problém s one-hot vektory je ten, že u podobných slov (například jablko a pomeranč) si vektory nejsou nijak blízké. Daleko vhodnější je použít prvky vektoru k vyjádření podobností a odlišností mezi slovy. Takovému vektorovému vyjádření slov se říká anglicky (word) embedding [4].

Existuje více technik, jakým způsobem tyto embedding vektory vytvářet. Jednou z možností je učit jednovrstvou neuronovou síť tak, aby na základě slova na vstupu (reprezentovaného one-hot vektorem) zkusila predikovat slova v jeho blízkosti ve větě. Této technice se více věnuje článek [16]. Sloupce, resp. řádky matice parametrů natrénované jednovrstvé sítě pak reprezentují embedding vektory slov. Přivedením one-hot vektoru na vstup takové sítě pak dojde pouze k vybrání odpovídajícího sloupce, resp. řádku.

V rámci Transformer architektury je síť pro tvorbu embedding vektorů na začátku náhodně inicializována a je trénována společně s ostatními částmi architektury.

3.1.3 Poziční vektor

Jelikož Transformer architektura nevyužívá rekurenci, je nutné nějakým způsobem reprezentovat pozici slova v sekvenci. To je řešeno tak, že ke vstupnímu embedding vektoru je pro vyjádření pozice přičten poziční vektor (angl. positional encoding) stejné dimenze. Poziční vektor je v článku [13] získán následujícími vzorci

$$PE_{pos,2i} = sin\left(\frac{pos}{1000^{\frac{2i}{d}}}\right), \tag{9}$$

$$PE_{pos,2i+1} = cos\left(\frac{pos}{1000^{\frac{2i}{d}}}\right), \tag{10}$$

kde *pos* je pozice slova v sekvenci, d je dimenze vstupního embedding vektoru a zároveň dimenze modelu a i je prvek vektoru, což znamená, že první rovnice je použita pro výpočet sudých prvků, druhá rovnice pro výpočet lichých prvků. Důvodem, proč nevyužít pro reprezentaci pozice slova ve větě například one-hot vektor, může být například snaha o rozložení informace o pozici do více, resp. všech dimenzí [4].

3.1.4 Self-Attention

Self-attention vrstva pomáhá enkodéru, resp. dekodéru při zpracování daného slova zachytit jeho závislost na slovech okolních. Vstupem této vrstvy je tedy embedding vektor a výstupem je vektor se stejnou dimenzí, který v sobě ale obsahuje informaci o okolních slovech. Pokud by například enkodér měl na vstupu postupně věty "V září začíná škola." a "Na obloze září hvězdy." a v obou případech by se snažil zakódovat slovo "září", self-attention vrstva mu umožní ho v obou případech reprezentovat jinými vektory, kdy v první větě se jedná o měsíc v roce, zatímco v druhé větě se jedná o synonymum svícení.

Vektorová implementace

Prvním krokem při výpočtu self-attention je spočítat pro každý vstupní vektor tři vektory - query, key a value. Ty jsou získány vynásobením vstupního vektoru třemi různými trénovatelnými maticemi $\mathbf{W^Q}$, $\mathbf{W^K}$ a $\mathbf{W^V}$, které jsou na začátku náhodně nainicializovány. Matice lze jinak reprezentovat jednovrstvými lineárními sítěmi. Vektory query, key a value mají výrazně nižší dimenzi, než je dimenze vstupního vektoru.

V dalším kroku se spočítají skóre, neboli jak velkou vazbu má zpracovávané slovo na všechna ostatní slova. Skóre pro danou dvojici slov je spočteno vynásobením query vektoru zpracovávaného slova s key vektorem druhého slova. Skóre je dále vyděleno druhou mocninou dimenze key vektoru d_k. Tato operace vede ke stabilnějšímu procesu trénování. Výsledek je zpracován funkcí softmax, která normalizuje všechny skóre tak, aby byly kladné a jejich součet byl roven jedné.

V následujícím kroku jsou skóre vynásobeny s value vektory jednotlivých slov. Cílem je zvýraznit hodnoty value vektorů u slov, které mají velkou vazbu na zpracovávané slovo a jeho query vektor, a naopak upozadit value vektory s malou vazbou. Výsledné value vektory se sečtou a vznikne výstupní vektor self-attention vrstvy pro jedno zpracované slovo.

Pojmy query, key a value pocházejí z terminologie databází. Na query lze nahlížet jako na otázku, na kterou se pro jednotlivá slova ptáme. Vynásobením query a key vektorů získáme závislost mezi slovy pro daný dotaz. Jedná se o jedno z možných vysvětlení toho, jakým způsobem Transformer architektura pracuje.

Maticová implementace

Vstupní embedding vektory lze poskládat do matice $\mathbf{X}$. Vynásobením vstupní matice $\mathbf{X}$ s maticemi $\mathbf{W^Q}$, $\mathbf{W^K}$ a $\mathbf{W^V}$ získáme matice $\mathbf{Q}$, $\mathbf{K}$ a $\mathbf{V}$. Veškeré kroky z předchozí kapitoly lze tedy shrnout v jediném vzorci, který je v článku [13] nazván scaled dot-product attention.

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{11}$$

3.1.5 Multi-Head Attention

V článku [13] bylo zjištěno, že místo jediného attention vektoru pro jedno slovo je přínosnější spočítat těchto vektorů více a ty následně spojit do jednoho. Transformer architektura si tedy pro každou self-attention vrstvu nainicializuje náhodně několik trénovatelných matic $\mathbf{W_i^Q}$, $\mathbf{W_i^K}$, $\mathbf{W_i^V}$, které použije k výpočtu více matic $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ a podle vzorce (11) spočítá více attention vektorů. Ty poskládá za sebe do jediného vektoru a pomocí trénovatelné jednovrstvé lineární sítě (reprezentované maticí parametrů $\mathbf{W^O}$) upraví dimenzi zpět na dimenzi modelu (dimenzi původních vektorů). Funkci multi-head attention lze popsat následujícím vzorcem.

$$\begin{aligned} MultiHead(\mathbf{W_i^Q}, \mathbf{W_i^K}, \mathbf{W_i^V}) &= Concat(head_1, ..., head_h)\mathbf{W^O} \\ where\ head_i &= Attention(\mathbf{XW_i^Q}, \mathbf{XW_i^K}, \mathbf{XW_i^V}) \end{aligned} \tag{12}$$

Výsledný vektor je díky většímu množství spočtených attention vektorů daleko bohatší reprezentací původní slova s ohledem na slova okolní. Pokud by navíc model nepoužíval multi-head attention, mohlo by se stát, že slovo bude mít přirozeně velkou vazbu (skóre) samo na sebe, což ale modelu ve snaze zjistit závislosti mezi slovy nijak nepomůže.

Existuje více možných způsobů, jak si představit multi-head attention. Například můžeme na multi-head attention nahlížet tak, že různé matice $\mathbf{Q}$ reprezentují různé dotazy na slova (například "kdy?", "kdo?") a matice $\mathbf{K}$ a $\mathbf{V}$ reprezentují, jak dobré odpovědi jsou jednotlivá slova na dané dotazy. Tím získá model daleko lepší porozumění vstupní větě.

3.1.6 Masked Multi-Head Attention

Výstup posledního enkodér bloku je přiveden do self-attention vrstev všech bloků dekodéru, kde se spočítají matice $\mathbf{K}$ a $\mathbf{V}$ za pomocí trénovatelných matic $\mathbf{W^K}$ a $\mathbf{W^V}$. To umožňuje dekodéru soustředit se při predikci výstupní prvku na důležité souvislosti vstupní sekvence. Matice $\mathbf{Q}$ se počítá ze zpracované výstupní sekvence.

Pro zachycení vzájemných souvislostí ve výstupní sekvenci používá dekodér vlastní multi-head attention vrstvu. Ta je na rozdíl od self-attention vrstvy enkodéru maskovaná, což při trénování zajišťuje, že dekodér nevidí budoucí prvky výstupní sekvence při trénování. Maskování je provedeno tak, že po vynásobení matic $\mathbf{Q}$ a $\mathbf{K}$ je přičtena matice samých 0, která má záporné nekonečno na pozicích slov, které mají být skryty. Následný softmax prvky s nekonečny vynuluje a vznikne dolní trojúhelníková matice.

3.1.7 Dopředné neuronové sítě

Poslední nezmíněnou částí Transformer architektury jsou dopředné neuronové sítě. Ty se nachází v každém enkodér/dekodér bloku za vrstvou multi-head attention. V každém bloku je trénována jedna dopředná síť, která se se stejnými parametry aplikuje odděleně na všechny vektory jdoucí z multi-head attention vrstvy. Dopředná síť se skládá ze dvou lineárních vrstev s ReLU aktivační funkcí mezi nimi. Dimenze vstupních a výstupních vektorů je zachována, dimenze mezi dvěma lineárními vrstvami bývá daleko větší, což umožňuje modelu naučit se složitější závislosti.

Dopředná síť je také na výstupu celé Transformer architektury na straně dekodéru. Zde má za úkol převést výstupní vektor dekodéru na daleko větší vektor, jehož dimenze odpovídá velikosti výstupního slovníku. Prvky takového vektoru odpovídají skóre slov ve

slovníku. Vrstva softmax vytvoří ze skóre pravděpodobnosti a na výstupu se pak vybere nejpravděpodobnější slovo.

3.2 Transfer Learning

Transformer modely mají velké množství parametrů a vyžadují velké množství dat pro natrénování. Může se tedy zdát, že jejich použití je nepraktické na úlohy s malými datasety, což může v některých případech být pravda. Ve strojovém učení platí princip Occamovy břitvy, kdy se snažíme preferovat jednodušší modely a přístupy. S rozvojem Transformerů se ovšem více rozvinuly také metody transfer learning a self-supervised learning.

Transfer learning je metoda ve strojovém učení, kde naučený model pro určitou úlohu je použit jako počáteční model pro jinou úlohu. Modely pro jednotlivé úlohy totiž mohou sdílet velké množství naučených závislostí. V případě zpracování obrazu je dokázáno, že u CNN se v počátečních vrstvách hledají například hrany a jednotlivé objekty se z těchto hran skládají až v pozdějších vrstvách.

Self-supervised learning (SSL) je způsob učení sítí, který využívá pouze data bez anotací, ze kterých si sám vytváří požadované výstupy. Základní myšlenkou self-supervised learning je predikovat skryté části vstupu. V případě oblasti Zpracování přirozeného jazyka se může jednat o úlohu predikce skrytých slov ve větě na základě slov okolních.

Self-supervised learning umožňuje použít velké množství dat z internetu bez anotací a předtrénovat Transformer modely tak, že v sobě mají uchované složité jazykové/obrazové závislosti, které by u menších modelů nebylo možné zachytit. Předtrénované Transformer modely je pak možné využít jako inicializaci a dotrénovat je na konkrétní úlohu pro data s anotacemi. Takovému procesu se říká fine-tuning.

3.3 Bidirectional Encoder Representations from Transformer (BERT)

Model BERT byl představen v roce 2018 v článku *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [17]. Architektura tohoto modelu využívá pouze enkodér část Transformer archiktury. V článku byly představeny dva různě velké modely.

- $\text{BERT}_{\text{BASE}}$ - využívá 12 enkódovacích bloků s dimenzí vektorů 768 mezi jednotlivými bloky. Každá self-attention vrstva má 12 attention heads.
- $\text{BERT}_{\text{LARGE}}$ - využívá 24 enkódovacích bloků s dimenzí vektorů 1024 mezi jednotlivými bloky. Každá self-attention vrstva má 16 attention heads.

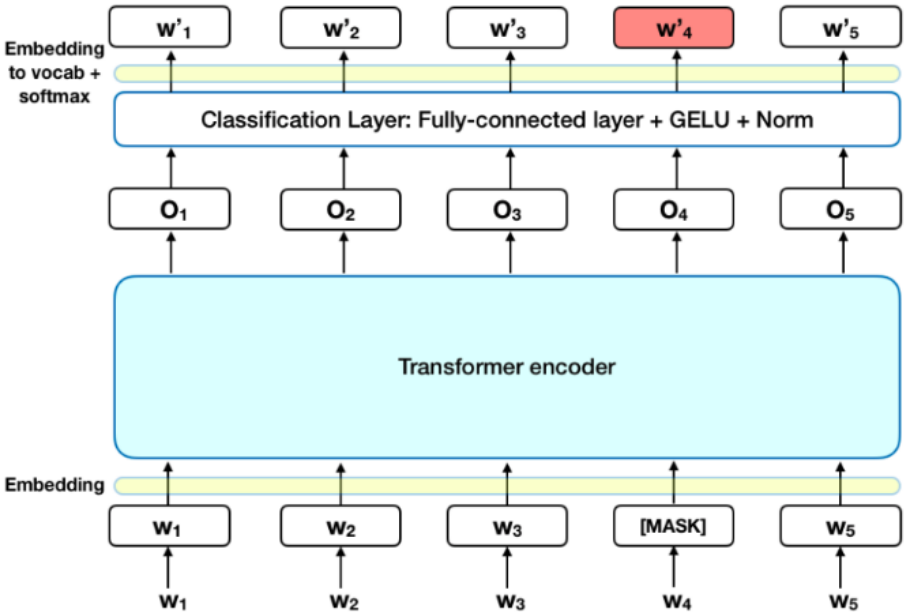
Princip zpracování vstupní sekvence je stejný jako v případě Transformer architektury z článku [13] popsané v dřívějších kapitolách. Jeden z hlavních rozdílů je způsob, jakým je vstupní sekvence vytvářena.

Na začátku se provede tokenizace vstupu, která rozdělí vstupní větu na jednotlivá slova, znaky diakritiky, případně rozdělí i slova na menší části. BERT k tomu využívá algoritmus WordPiece. Ten patří mezi tzv. subword-based tokenizační algoritmy, které umožňují mít rozumně velký slovník tokenů při zachování jejich smysluplnosti[1].

[1]Existují také word a character-based tokenizační algoritmy. Word-based tokenizace mívá velké množství tokenů odpovídajících slovům ve slovníku, kde podobná slova můžou mít zcela odlišné tokeny. Character-based tokenizace mívá menší množství tokenů, které ale bývají méně smysluplné.

Na první pozici vstupní sekvence se přidává token [CLS]. Pokud je na vstupu více vět, jsou mezi sebou odděleny tokenem [SEP]. BERT očekává vektory fixní délky a pokud je délka vstupní sekvence kratší, je zbytek sekvence doplněn o tokeny [PAD]. Jednotlivé tokeny se dále převedou na sekvenci odpovídajících celočíselných identifikátorů. K sekvenci se ještě přičítá vektor segmentů, který označuje, jaké tokeny patří k jaké větě, a dále se ještě přičítá poziční vektor, který je na rozdíl od původní Transformer architektury z článku [13] trénovatelný.

BERT byl předtrénován na dvou úlohách na textech z Wikipedie a datasetu BookCorpus. V jednom případě bylo na vstupu skryto (maskováno) 15% vstupní sekvence. Na výstupu pak byla přidána MLP vrstva se softmax funkcí s cílem doplnit maskované slovo. V druhém případě byly na vstup přivedeny dvě věty, kde cílem MLP a softmax vrstvy bylo určit, jestli druhá věta patří za první větu. BERT lze také využít pro extrakci word embedding vektorů obohacených o kontext. Nelze jistě říci, které embedding vektory budou pro konkrétní úlohu nejlepší. Článek [17] prozkoumává několik možností jako například použít vektory z posledního enkodér bloku nebo sečíst vektory posledních čtyř enkodér bloků.



Obrázek 10: Model BERT. Převzato z [18]

3.4 A Robustly Optimized BERT Pretraining Approach (RoBERTa)

V roce 2019 byl v článku *RoBERTa: A Robustly Optimized BERT Pretraining Approach* [19] představen model RoBERTa. U původního modelu BERT [17] bylo zjištěno, že byl značně podtrénovaný a mohl dosahovat daleko lepší výsledků. RoBERTa model je tedy založen na BERT architektuře s řadou změn při procesu trénování.

Jako je uvedeno v [19], mezi změny při procesu trénování patří: (1) delší trénování, s vetšími dávkami dat, na větším množství dat; (2) odstranění trénování pro predikci pořadí vět; (3) trénování na delších sekvencích; (4) dynamické maskování vstupní sekvence.

Oproti BERT modelu se RoBERTa liší ještě v použitém algoritmu tokenizace. RoBERTa používá tzv. Byte-Pair Encoding (BPE) tokenizaci, která stejně jako WordPiece patří mezi

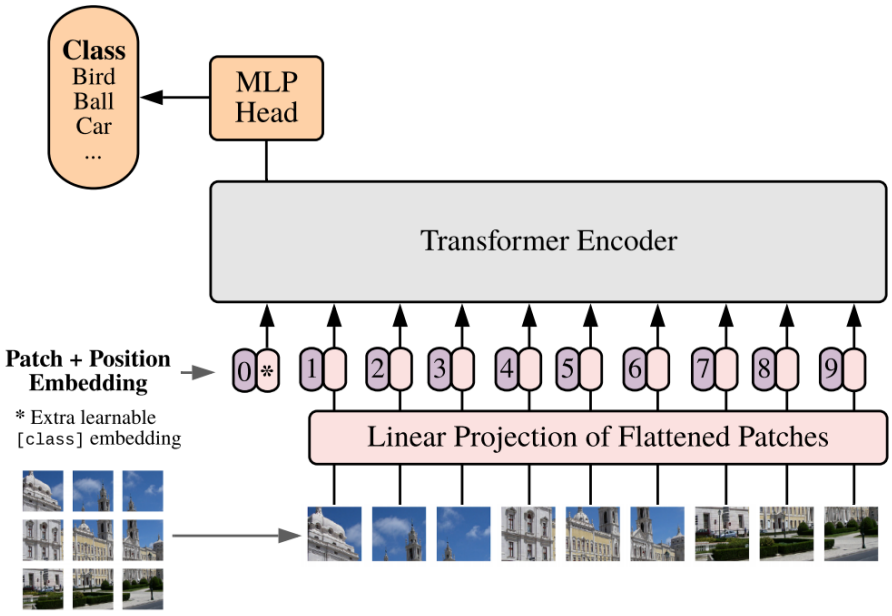
subword-based tokenizační algoritmy. Metoda je založena na kompresi dat. Na začátku je každé slovo rozděleno na jednotlivé znaky. Každý znak odpovídá jednomu tokenu a ke každému znaku je spočten jeho výskyt. Znaky, které se nejčastěji vyskytují vedle sebe, jsou spojeny do nového tokenu. Takové spojování končí při dosažení stanovené hranice počtu tokenů nebo dosažení maximálního počtu iterací spojování [20].

3.5 Vision Transformer (ViT)

Model Vision Transformer (ViT) byl představen v roce 2020 v článku *An Image is Worth 16x16 Words* [21]. Jedná se o aplikaci Transformer architektury na obrazových datech bez použití konvoluce a CNN. Stejně jako BERT používá i ViT pouze enkodér část Transformer archiktetury. Základní varianty ViT_{BASE} a $\text{ViT}_{\text{LARGE}}$ se inspirují modelem BERT a mají tedy stejné parametry, jako je uvedeno v kapitole 3.3.

Transformer architektura vyžaduje na svém vstupu 1D sekvenci hodnot. Vision Transformer tedy nejdříve rozdělí 2D obrázek na 16x16 výřezů (angl. patch) a každý výřez následně rozvine do 1D vektoru. Z těchto vektorů se trénovatelnou lineární vrstvou vytvoří vektory požadované dimenze použitého ViT modelu. K vektorům je dále přičten trénovatelný poziční vektor. Stejně jako u BERT modelu se k sekvenci vektorů přidává ještě klasifikační token. Z posledního enkodér bloku se použije pouze první výstupní vektor, který se MLP sítí u úlohy klasifikace převede na třídu, která se na obrázku nachází. ViT byl předtrénován na ImageNet a ImageNet-21k datasetech.

Při trénování na středně velkých datasetech nedosahoval ViT tak dobrých výsledků jako některé state-of-the-art CNN. To je způsobeno tím, že Transformer modelům chybí některé vlastnosti CNN, jako je například invariance vůči posunu a lokalita, zmíněná v kapitole 2.3.1. Pokud je však ViT trénován na větších datasetech (15-300M obrázků), je schopný dosahovat velice dobrých výsledků na řadě testovacích úloh (angl. computer vision benchmark).



Obrázek 11: Model Vision Transformer. Převzato z [21]

4 Datasety

Pro účely této práce byly popsány dva obrazové datasety - COCO a Objects 365. V praktické části byl následně využit dataset COCO, konkrétně COCO Detection 2017, a to zejména kvůli jeho rozumné velikosti, rozšířenosti a uživatelské přívětivosti. Hlavním důvodem, proč nebyl v této práci využit dataset Objects 365, je jeho velikost. Epochy by trvaly mnohonásobně déle než u datasetu COCO Detection 2017, což při testování vlivu parametrů na výsledky není vhodné.

4.1 COCO

The Common Object in Context (COCO) je jeden z nejrozšířenějších obrazových datasetů přístupný veřejnosti, který se běžně používá pro trénování a následné porovnání modelů v oblasti počítačového vidění. Dataset obsahuje více než 330 tisíc obrázků, kde 200 tisíc z nich je s anotacemi (angl. labeled). Medián rozlišení obrázků je 640 x 480. Celkem je na obrázcích označeno přibližně 1.5 milionů objektů, které pochází z 12 nadkategorií (angl. supercategories). Ty se nadále dělí na 80 kategorií s unikátními celočíselnými identifikátory. Kategorie tvoří například různé dopravní prostředky, zvířata, základní potraviny a běžné vybavení domácnosti.

Identifikátory kategorií v COCO datasetu jsou tvořeny čísly 1 až 91. Původní dataset oficiálního článku z roku 2014 *Microsoft COCO: Common Objects in Context* [22] totiž obsahoval 91 kategorií, ale pouze 80 kategorií bylo použito v datasetu pro veřejnost. Pro zjednodušení práce byl tedy rozsah identifikátorů v této práci zmenšen na rozsah 0 až 80, kde 0 přísluší pozadí, které v této práci nebylo využito.

COCO lze využít pro velké množství úloh jako je detekce objektů, segmentace nebo automatický popis obrázků. Podle dané úlohy se odvíjí název konkrétního datasetu. Jak již bylo zmíněno, v této práci byl použit dataset COCO Detection 2017, který obsahuje 118 tisíc trénovacích obrázků a 5000 validačních. Výskyt kategorií v datasetu není rovnoměrně rozdělený. V trénovací části COCO Detection 2017 má čtvrtina kategorií méně než 2000 anotací, polovina kategorií 2000-4300 anotací a zbývající čtvrtina více než 4300. Nerovnoměrnost je dobře vidět na příkladu kategorií člověk a fén na vlasy, kde člověk má 64115 anotací, zatímco fén na vlasy pouze 189. Podobné rozložení anotací lze sledovat i ve validační části datasetu, kde polovina kategorií má 100-300 anotací. Z druhé poloviny má 85% kategorií méně než 100 anotací a zbylých 15% více než 300. Počet anotací trénovací části datasetu pro jednotlivé kategorie je zobrazen v posledním sloupci tabulky 10 v dodatku A na konci práce. Je každopádně zřejmé, že bylo obtížné natrénovat model tak, aby dobře rozpoznával kategorie, které jsou v datasetu málo zastoupené.

Anotace jsou u COCO datasetu uloženy ve formátu JSON s klíči info, licenses, categories, image a annotations. Důležité jsou zejména poslední dva klíče, které vypadají následovně:

$$
\begin{aligned}
&image : \{ \\
&\quad id && : int, \\
&\quad width && : int, \\
&\quad height && : int, \\
&\quad file\_name && : str, \\
&\quad license && : int, \\
&\quad flickr\_url && : str, \\
&\quad coco\_url && : str, \\
&\quad data\_captured && : datetime \\
&\}
\end{aligned}
\qquad
\begin{aligned}
&anotations : \{ \\
&\quad id && : int, \\
&\quad image\_id && : int, \\
&\quad category\_id && : int \\
&\quad segmentation && : RLE \ or \ [polygon] \\
&\quad area && : float, \\
&\quad bbox && : [x, y, width, height] \\
&\quad iscrowd && : 0 \ or \ 1 \\
&\}
\end{aligned}
$$

Při práci s datasetem byly využity anotace ve formě segmentací. Segmentace mohou být popsány dvěma způsoby. První způsob je popis ve tvaru polygonu, tedy listu ve tvaru $[x_1, y_1, x_2, y_2...]$, kde jednotlivé body $[x_i, y_i]$ tvoří body polygonu. Druhý způsob je popis ve formátu RLE (Run Length Encoding), který využívá počty pixelů k vymezení objektů. Formát RLE obsahuje list s názvem *counts*, kde první hodnota obsahuje počet pixelů po řádkách, než se narazí na pixel, který patří k objektu. Následuje počet pixelů náležící objektu, dále opět počet pixelů nenáležící objektu a tak dále. Tento způsob popisu je zejména vhodný pro velké množství objektů stejné kategorie, které by bylo zbytečně obtížné ohraničovat polygonem (například lidé na tribuně).

Obrázek 12: Ukázka anotací ve tvaru polygonu

4.2 Objects 365

Objects 365 je doposud největší zcela označený obrazový dataset. Původní dataset z článku *Objects365: A Large-scale, High-quality Dataset for Object Detection* [23] obsahoval více než 600 tisíc trénovacích obrázků s více než 10 miliony označenými objekty. Nyní má tento veřejně přístupný dataset více než 2 miliony trénovacích obrázků s více než 30 miliony označenými objekty. V obou případech obsahuje dataset 11 nadkategorií jako například oblečení, doprava, kuchyně nebo elektronika. Nadkategorie se dále dělí na celkem 365 kategorií, které blíže specifikují konkrétní objekty. Všechny kategorie v COCO jsou pro lepší porovnání obsaženy také v Objects 365.

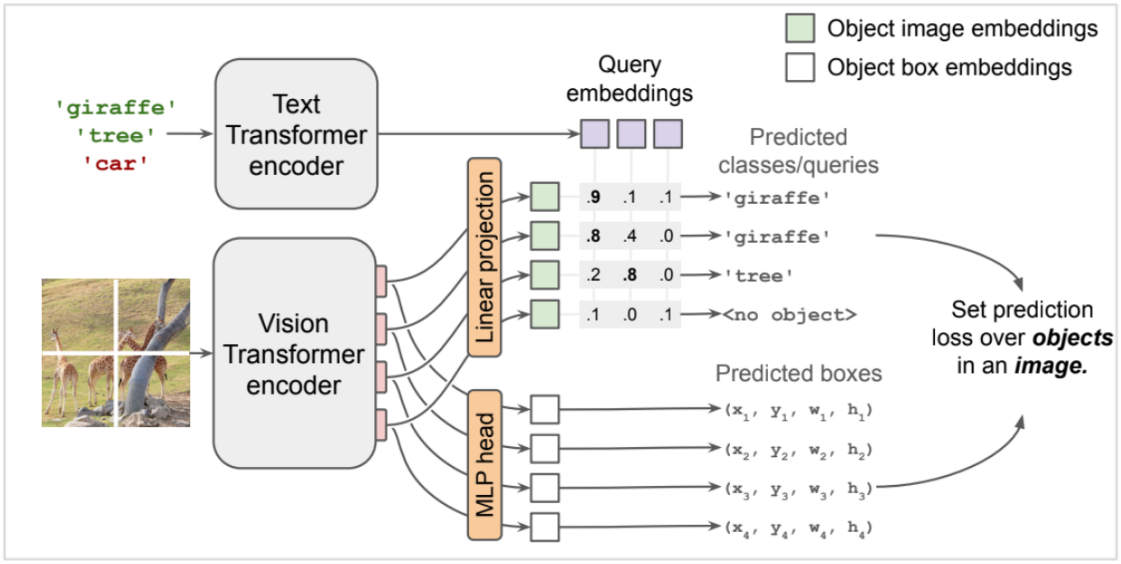
Kromě větší velikosti v porovnání s ostatními datasety se Objects 365 soustředí také na lepší kvalitu anotací. Pro zaručení kvality anotací byl anotační proces rozdělen do tří kroků. V prvním kroku se provádí filtrace. Jelikož se Objects 365 soustředí na detekci objektů, snaží se eliminovat obrázky, které jsou více vhodné na klasifikaci a obsahují pouze jeden objekt blízko středu obrázku. Podobný princip filtrace se uplatňuje také v COCO. Pokud je obrázek vhodný na detekci objektů a obsahuje alespoň jeden objekt z jedné nadkategorie, následuje druhý krok. Pro anotátora je téměř nemožné, aby si pamatoval všech 365 kategorií. V druhém kroku se tedy pouze vyberou nadkategorie, které se na obrázku vyskytují. V posledním kroku anotátor označí bounding-boxem pouze ty objekty, které patří pod jednu danou nadkategorii. Anotátorům tedy stačí, aby byli obeznámeni pouze s kategoriemi pro jednu nadkategorii, nikoliv pro všech 365 kategorií. Tím je zvýšena efektivita anotování a kvalita anotací [23].

5 Detekce objektů s využitím textových dotazů

5.1 Formulace problému

Cílem praktické části bakalářské práce je navrhnout architekturu modelu pro detekci objektů s využitím textových dotazů a vzniklý model natrénovat a vyhodnotit. Architektura bude mít dva vstupy. Prvním vstupem bude obrázek, druhým vstupem bude dotaz ve formě požadavku na nalezení určitého objektu. V případě, že dotazovaný objekt nebude na obrázku, nebude označeno nic.

Pro zpracování vstupů budou využity předtrénované Transformer modely. Podobný přístup lze vidět u modelu OWL-ViT [24], jehož architektura je zobrazena na obrázku 13. Oproti OWL-ViT dojde ke zjednodušení architektury, které nám umožní přivést na vstup kromě jednoslovných dotazů také dotazy ve formě vět, což model OWL-ViT neumožňuje. Transformery spolu s dalšími částmi modelu budou dotrénovány pomocí již zmíněného datasetu COCO Detection 2017.



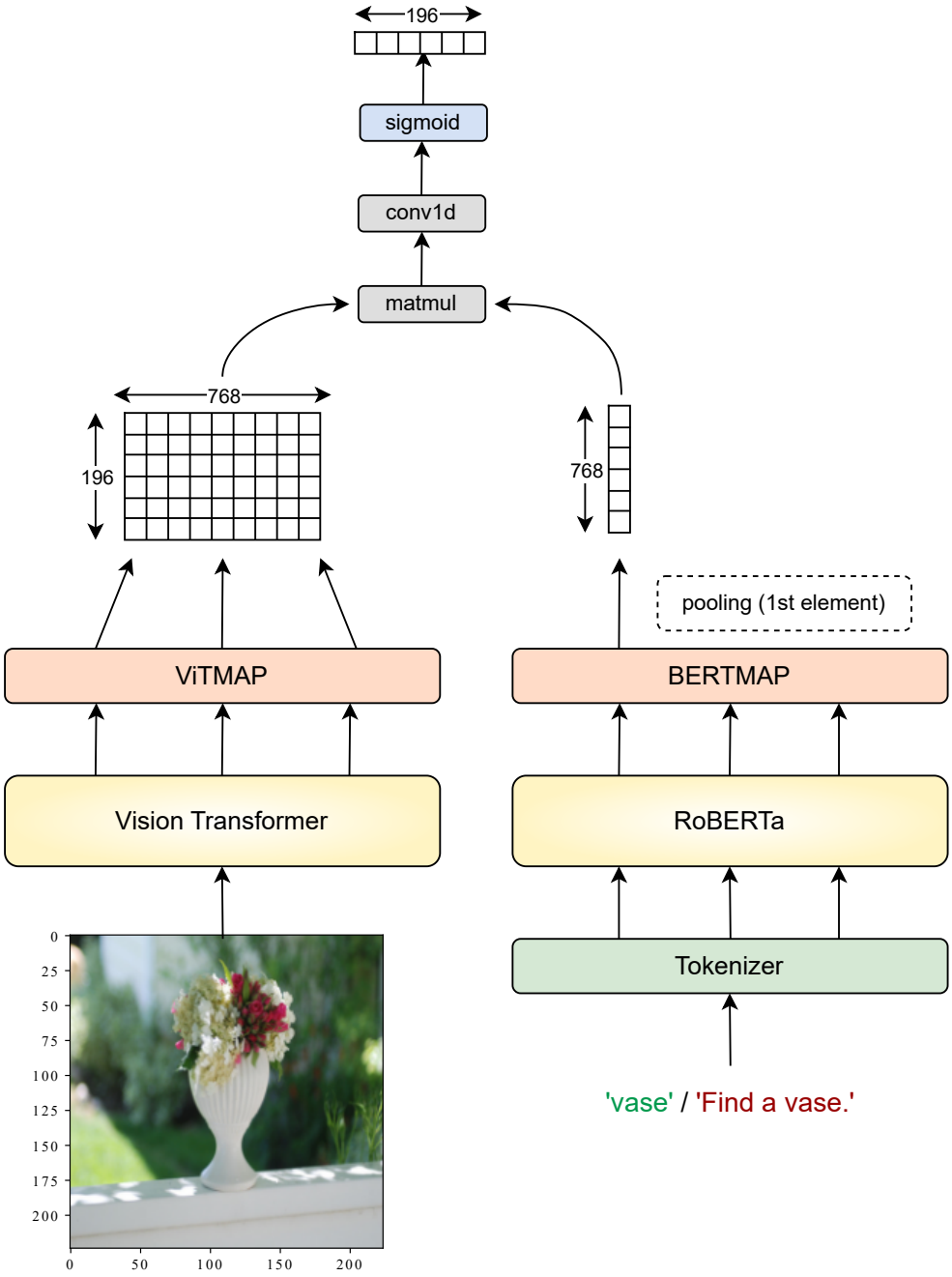
Obrázek 13: Model OWL-ViT. Převzato z [24]

Použití předtrénovaných modelů na velkých datasetech vedlo k velkému zlepšení v úloze klasifikace obrázku, tedy určení kategorie, která se na obrázku primárně vyskytuje. Na tomto zlepšení se podílely architektury jako například CLIP, ALIGN, Florence nebo SimVLM. V posledních letech vznikla řada dalších metod a modelů, které se tyto možnosti trénování na velkých datasetech snaží aplikovat na úlohu detekce objektů. Jedním z posledních modelů je právě již zmíněný OWL-ViT, který byl představen v roce 2022 v článku *Simple Open-Vocabulary Object Detection with Vision Transformers* [24].

Z výše uvedeného popisu je zřejmé, že o úlohu detekce objektů s využitím textu je v posledních letech velký zájem a jedná se o moderní problematiku. Níže navržená architektura představuje originální a jednoduchý přístup řešení této problematiky.

5.2 Návrh architektury

Na obrázku 14 je zobrazena navržená architektura modelu pro detekci objektů s využitím textových dotazů. Lze vidět inspiraci v modelu OWL-ViT s odděleným zpracováním obrazové a textové části.



Obrázek 14: Navržená archiktetura modelu

Pro zpracování vstupního obrázku byl použit předtrénovaný ViT_{BASE} z knihovny *timm*[2]. Konkrétně se jedná o verzi "vit_base_patch16_224", kde patch16 značí rozlišení výřezů, které je 16x16. Parametr 224 značí očekávanou velikost vstupního obrázku. Výřezů tedy

[2] `https://timm.fast.ai/`

v obrázku dimenze 224x224 bude 14x14. Navíc byla u ViT zahozena finální MLP vrstva a vypnutý globální pooling na výstupu. Jelikož je u ViT_{BASE} vnitřní dimenze vektorů 768, výstupem bude matice 196x768, neboli 196 vektorů dimenze 768, kde každý vektor odpovídá jednomu výřezu ve vstupním obrázku.

Pro zpracování vstupního dotazu byl použit předtrénovaný model $\text{RoBERTa}_{\text{BASE}}$ [19] s odpovídajícím natrénovaným Tokenizerem. Oboje bylo dostupné v knihovně *transformers*[3].

Výstupy RoBERTa a ViT modelů jsou dále přivedeny do vrstev ViTMAP, resp. BERTMAP. V obou případech se jedná o 1D konvoluční vrstvy s dimenzí vstupu 768, dimenzí výstupu 768 a velikostí jádra 1. Cílem těchto vrstev je transformace výstupů ViT a RoBERTa do sdíleného vektorového prostoru. Za BERTMAP vrstvou ještě následuje pooling. V navržené architektuře se jedná pouze o zahození všech ostatní výstupů kromě toho prvního, ale bylo by možné využít také average nebo max pooling (viz. kapitola 2.3.1).

Následuje hlavní myšlenka celé architektury. Výstupem ViTMAP je matice 196x768, kde každý řádek odpovídá jednomu výřezu ve vstupním obrázku. Výstupem BERTMAP je jediný vektor dimenze 768, který reprezentuje kategorii, kterou na obrázku hledáme. Skalárním součinem vektorů výřezů s vektorem z textové části získáme pro každý výřez ve vstupním obrázku skóre, které určuje, jestli se na daném výřezu hledaná kategorie vyskytuje. Následná kalibrační 1D konvoluční vrstva a sigmoida mají pouze za úkol toto skóre převést do rozsahu 0 až 1. Kalibrační konvoluční vrstva je vstupní dimenze 1, výstupní dimenze 1 a s jádrem velikosti 1. To znamená, že se aplikuje na každý prvek vektoru zvlášť. Cílem této vrstvy je snížit rozsah hodnot pro sigmoidu a tím stabilizovat proces trénování.

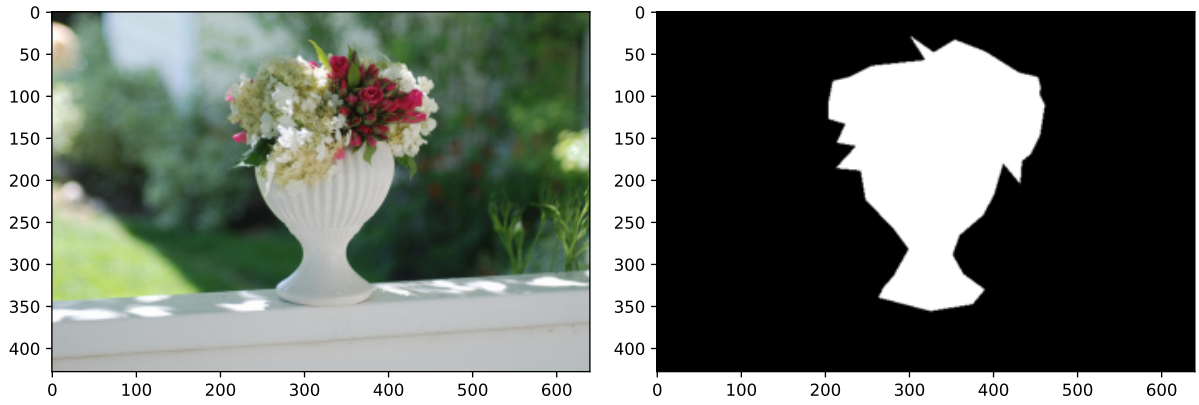
5.3 Předzpracování dat

Při předzpracování dat byl jako výchozí bod použit zdrojový kód PyTorch CocoDetection datasetu, který implementoval základní funkce načtení obrázků a anotací, ale bylo nutné značně změnit metodu *__getitem__()*, která vrací n-tý prvek datasetu. Požadovaný formát n-tého prvku byl (*obrázek, id kategorie, požadovaný výstup, dotaz*).

5.3.1 Generování požadovaného výstupu

Jako první se v metodě *__getitem__()* načte obrázek a jeho anotace ve formě segmentací. Pro každý obrázek byly segmentace seřazeny podle id kategorie. Segmentace se stejným id byly sloučeny a byla z nich vytvořena černobílá maska stejné dimenze jako vstupní obrázek.

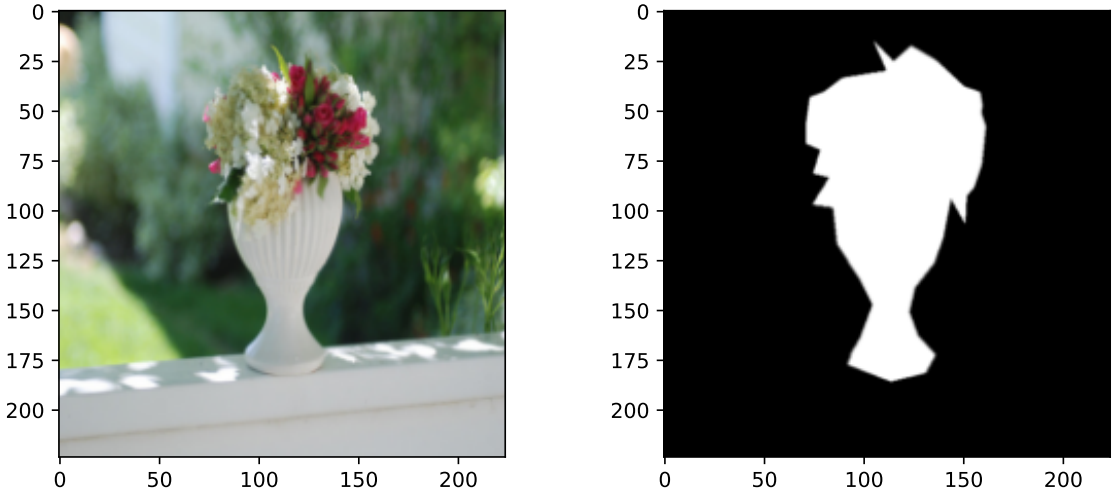
[3] `https://huggingface.co/docs/transformers/index`



(a) Obrázek z COCO datasetu (b) Maska kategorie "potted plant"

Obrázek 15: Vytvoření masky jedné z kategorií na obrázku

Na masky spolu s obrázkem byly aplikovány transformace, které odpovídaly požadavkům použitého modelu (ViT_{BASE}). V první řadě byly u transformace vypnuty augmentace. Dále bylo u transformace nastaveno zmenšení vstupního obrázku a masky na rozměry 224x224 při použití bilineární interpolace. Hodnoty obrázku byly normalizovány tak, aby střední hodnota a směrodatná odchylka odpovídaly těmto statistikám u ImageNet datasetu, na kterém byl ViT předtrénován. Pro tranformaci masky byla normalizace vypnuta (střední hodnota nastavena na nulu a směrodatná odchylka na jedničku).

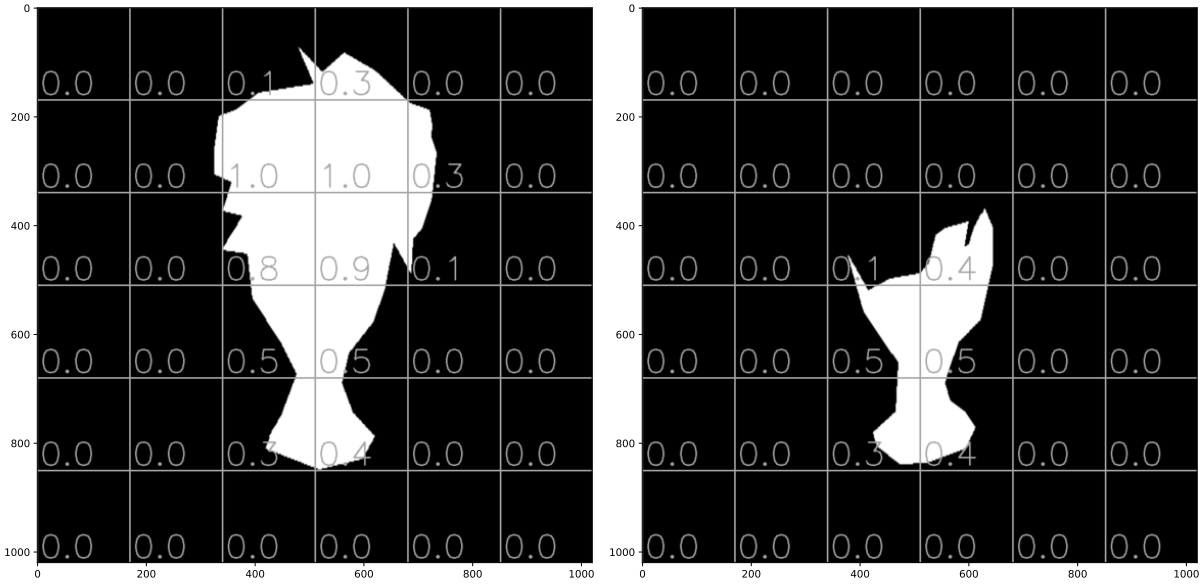


(a) Obrázek po transformaci (b) Maska po transformaci

Obrázek 16: Ukázka aplikovaných transformací

Pro velikost masky 224x224 byl dále vygenerován seznam 196 nepřekrývajících se výřezů (14x14) ve formátu $[x, y, width, height]$. Počet výřezů musel odpovídat použitému ViT modelu. Ve funkci *compute_score()* byly výřezy promítnuty na masku a pro každý výřez bylo spočítáno skóre v rozmezí 0 až 1, které určuje, jaká část výřezu je překryta maskou. Vynásobením skóre číslem 100 by se jednalo o procentuální vyjádření překryvu konkrétního výřezu a masky. Skóre bylo dále uloženo do vektoru výsledků. Výstupem funkce *compute_score()* je tedy vektor **u** dimenze 196, na který lze pohlížet jako na rozvinutou

matici dimenze 14x14. Maska s maticí je zobrazena na obrázku 17. Pro lepší čitelnost jsou matice pouze dimenze 6x6, její hodnoty byly zaokrouhleny a oba obrázky byly zvětšeny na velikost 1020x1020.



(a) Kategorie "potted plant" (b) Kategorie "vase"

Obrázek 17: Matice požadovaných výstupů pro jednotlivé kategorie

Vektory **u** byly spočítány pro všechny kategorie, které se v obrázku vyskytovaly. Následně byl jeden z nich náhodně vybrán s pravděpodobností $1 - coco\_random\_cat\_prob$. S pravděpodobností *coco_random_cat_prob* byla náhodně vybrána libovolná z kategorií, které se na obrázku nevyskytovaly a k této kategorii byl vygenerován vektor **u** samých nul jakožto požadovaný výstup modelu. Při prvním vyhodnocení natrénovaného modelu bez této funkcionality bylo totiž zjištěno, že model velice často predikoval falešné pozitivní výsledky. Jinými slovy nacházel a označoval objekty, i když se na obrázku nevyskytovaly. Náhodná kategorie spolu s vektorem **u** samých nul byly vygenerovány i v případě, že obrázek neměl žádné anotace.

Pro účely trénování modelu také na dotazy ve formě vět byl pro zvolenou kategorii náhodně vybrán dotaz ze slovníku ručně předpřipravených vět. Zde je výčet čtrnácti použitých vět:

- Find a {class}.
- Find me a {class}.
- Where is the {class}?.
- Mark a {class}.
- Can you mark a {class}?
- Could you mark a {class}?
- Detect a {class}.
- Could you detect a {class}?
- Where is the {class} located?
- Where is the {class} positioned?
- Is there a {class}?
- Look for a {class}.
- Where can I find a {class}?
- Could you pinpoint a {class}?

Další věty byly vytvořeny přidáním dovětků “in the picture“ a “in this image“, čímž se celkový počet předpřipravených vět ztrojnásobil a celkem jich bylo 42.

5.3.2 Statistiky požadovaného výstupu

Potom, co COCO dataset dával data v požadovaném formátu, bylo vhodné se podívat na nějaké základní statistiky požadovaného výstupu (vektoru **u**) pro jednotlivé kategorie. Postupně byly zpracovány všechny anotace datasetu a do slovníku s klíči id kategorií byly uloženy odpovídající vektory **u**.

Statistiky jsou zobrazeny v následující tabulce 1 a byly seřazeny podle průměru všech prvků vektorů pro danou kategorii (první sloupec tabulky). Pro naše účely stačí, když zde nebude zobrazeno všech 80 kategorií, ale pouze určitý počet ze začátku, z prostředku a z konce. Statistiky pro všech 80 kategorií jsou uvedeny v tabulce 10 v dodatku A na konci práce.

První sloupec tabulky zobrazuje průměr všech prvků vektorů **u** pro danou kategorii, druhý sloupec průměr pouze prvků nenulových, dále medián nenulových prvků a počet anotací v datasetu. Minimální nenulová hodnota vektorů je pro všechny kategorie $2 \cdot 10^{-5}$ a maximální hodnota je 1.

categories	avg	$\overline{\text{avg}}$	median	#anns
sports ball	0.005	0.218	0.088	4262
baseball bat	0.007	0.189	0.114	2506
skis	0.008	0.163	0.095	3082
baseball glove	0.009	0.290	0.149	2629
spoon	0.011	0.239	0.139	3529
...				
book	0.052	0.519	0.485	5332
bicycle	0.054	0.481	0.424	3252
car	0.059	0.556	0.573	12251
vase	0.059	0.555	0.567	3593
chair	0.060	0.480	0.419	12774
...				
train	0.236	0.740	0.992	3588
elephant	0.247	0.667	0.831	2143
pizza	0.288	0.758	1.0	3166
bed	0.301	0.786	1.0	3682
dining table	0.312	0.825	1.0	11837

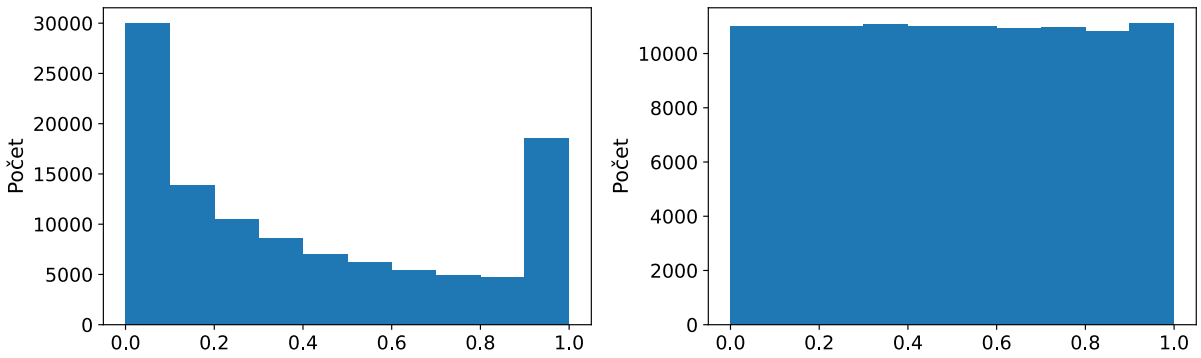
Tabulka 1: Statistiky vybraných kategorií

Z tabulky lze získat mnoho zajímavých poznatků o vektorech požadovaného výstupu. Při pohledu na první sloupec je zřejmé, že objekty (a jejich masky) průměrně zabírají spíše jednotky procent celého obrázku, jinými slovy vektory požadovaných výstupů jsou běžně tvořeny velkým množstvím nulových prvků.

Zajímavější je pro naše účely druhý a třetí sloupec, a sice průměr a medián nenulových prvků vektorů. V případě, že na obrázku je míč na sport (sports ball), tak jeho průměrná hodnota nenulových prvků je 0.218. To znamená, že prvek vektoru (odpovídající výřezu

obrázku), na kterém je kus míče, má průměrně tuto hodnotu. Takovou hodnotu požadujeme pro daný výřez na výstupu modelu. V případě postele (bed) je však tato hodnota 0.786. Pro účely lepšího trénování bylo tedy vhodné zvýšit hodnoty prvků vektorů pro kategorie s malou průměrnou hodnotou nenulových prvků a naopak snížit hodnoty prvků vektorů pro kategorie s vysokou průměrnou hodnotou nenulových prvků. Jinými slovy jsme chtěli hodnoty prvků vektorů pro kategorie rovnoměrněji rozložené.

Pro účely rovnoměrného rozložení byl pro každou kategorii natrénován *QuantileTransformer* z knihovny *sklearn.preprocessing*. Změna rozložení hodnot prvků vektorů byla zobrazena pomocí histogramu, kde na vodorovné ose jsou zaokrouhlené hodnoty prvků vektorů požadovaných výstupů pro konkrétní kategorii.



(a) Histogram před transformací

(b) Histogram po transformaci

Obrázek 18: Histogramy kategorie "bottle"

Natrénované *QuantileTransformery* byly následně uloženy do slovníku s klíči odpovídajících id kategorií. V samotném závěru metody *__getitem__()* popsané v kapitole 5.3.1 se tedy ještě pro jeden vektor **u** požadovaného výstupu provedla transformace příslušným *QuantileTransformer*.

5.4 Trénování

Trénování bylo rozděleno na trénování pro jednoslovné dotazy a trénování pro dotazy ve formě vět. V obou případech se jednalo o dávkové trénování (angl. batch training) s dávkami velikosti 16. Data byla pro každou epochu zamíchána. Jako optimizer byl použit ADAM [25] s learning rate nastaveným na 10^{-4}. Pro snižování learning rate byl využit StepLR scheduler, který každých 5 epoch snížil learning rate o 10%.

Důležitým parametrem celého procesu trénování byl binarizační práh *train_thr*. Prvky vektorů požadovaných výstupů byly při trénování binarizovány na hodnoty 1 pro prvky větší než *train_thr* a 0 pro hodnoty menší. V minulé kapitole bylo zmíněno, že rovnoměrné rozložení prvků vektorů požadovaného výstupu zajistí lepší trénování. Zde je nyní vhodné vysvětlit, proč tomu tak je. Při vyrovnání hodnot prvků vektorů požadovaného výstupu jsme docílili toho, že nyní po binarizaci budou mít všechny třídy skoro[4] stejné relativní zastoupení nulových a nenulových prvků. Pro binarizované vektory pak byla zvolena ztrátová funkce Binary Cross Entropy (BCE).

Binarizační práh *train_thr* společně s *coco_random_cat_prob* z kapitoly 5.3.1 byly dva hlavní parametry, které zásadně ovlivňovaly výsledky modelu. Vyšší hodnota *train_thr*

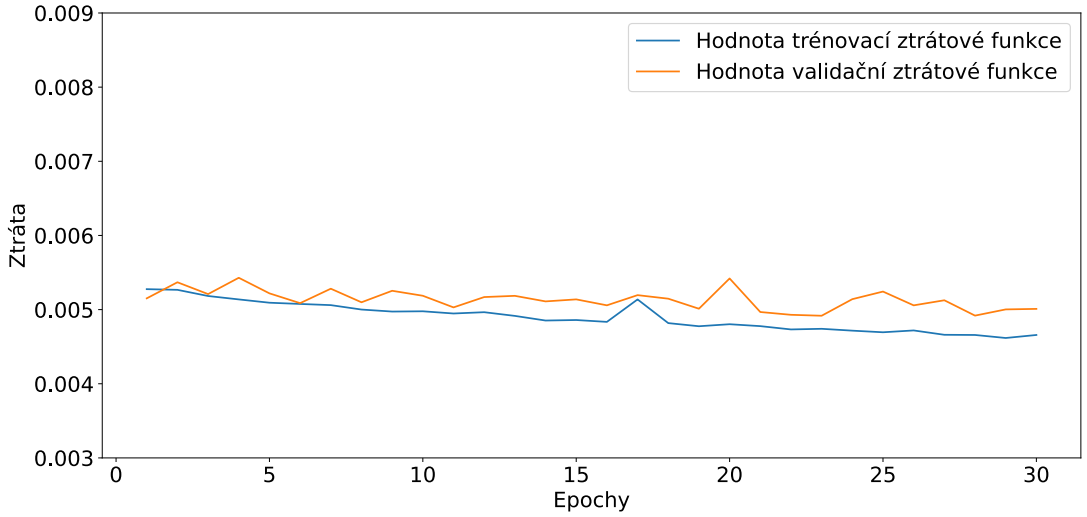
[4]U některých kategorií nedokázal QuantileTransformer hodnoty prvků zcela rovnoměrně rozložit.

učila model soustředit se na větší objekty a malé předměty mohly být prahem často vynulovány. Model se ale zároveň učil být si jistější při hledání objektů a nedávat vysoké skóre výřezu obrázku, kde nic hledaného není.

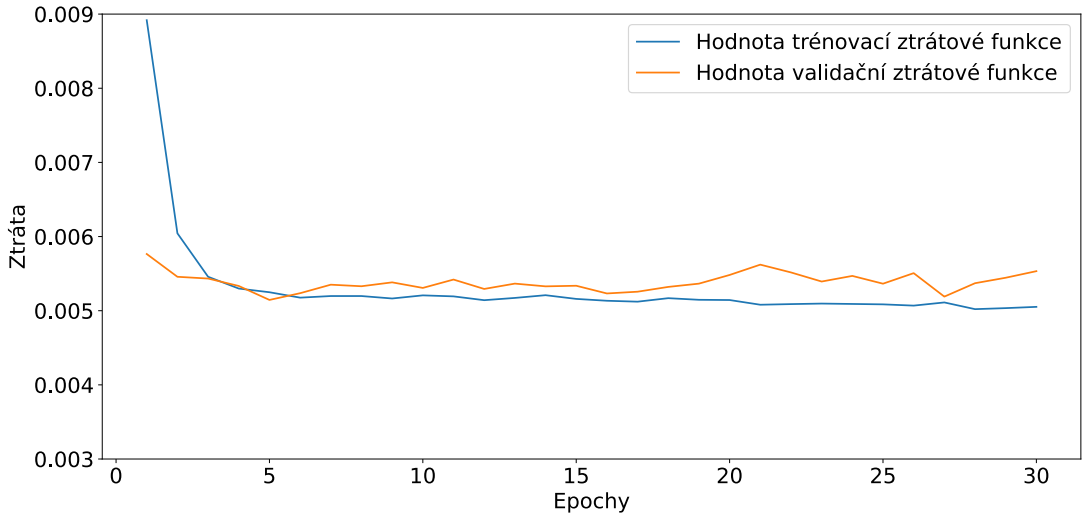
Hlavním cílem této části práci tedy bylo natrénovat několik modelů pro různá nastavení těchto dvou parametrů, aby bylo možné modely následně porovnat. Cílem tedy nebylo nalézt optimální nastavení těchto parametrů a natrénovat model dosahující nejlepších výsledků, ale spíše získat představu o tom, jak parametry ovlivňují výsledky, aby bylo možné na tyto poznatky v budoucí práci navázat.

5.4.1 Jednoslovné dotazy

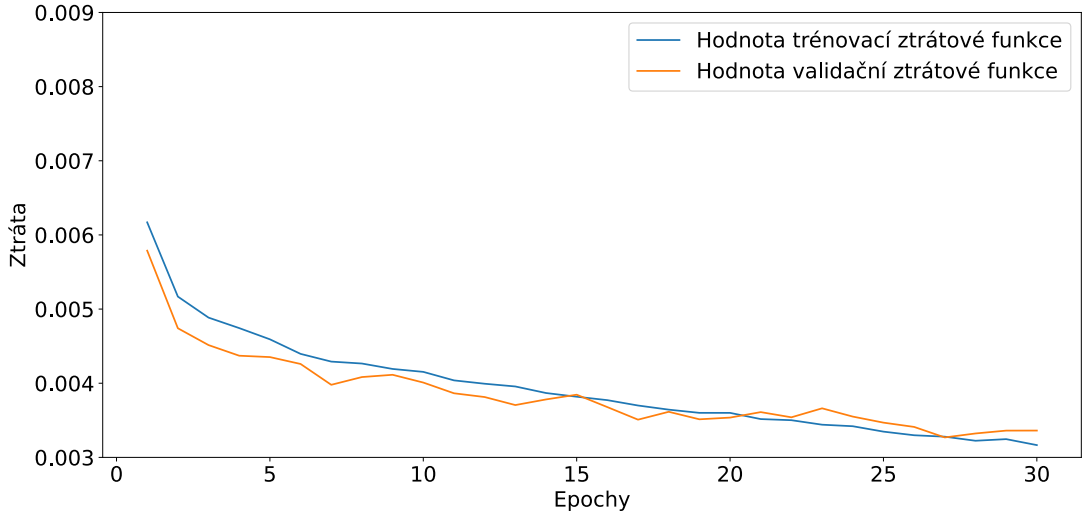
Jelikož byly v práci použity předtrénované Transformer modely s velkým množstvím parametrů, bylo na začátek trénování důležité zjistit, jestli má smysl trénovat oba modely nebo pouze jeden z nich. Všechny ostatní vrstvy a parametry architektury byly trénovány. Na následujícím obrázku lze vidět průběhy tří trénování, kdy byly trénovány buď oba Transformer modely nebo pouze jeden z nich. Ve všech třech případech byl použit binarizační práh $train\_thr = 0.5$ a náhodný sampling kategorií $coco\_random\_cat\_prob = 0.5$.



(a) Trénování RoBERTa i ViT



(b) Trénování pouze RoBERTa



(c) Trénování pouze ViT

Obrázek 19: Průběhy trénování pro různé kombinace zafixování parametrů

V případě, kdy byly trénovány oba Transformer modely, docházelo sice k poklesu hodnot validační i trénovací ztrátové funkce, ale pokles byl velice mírný. Pro validační ztrátovou funkci už model po 30 epochách pomalu dosahoval saturace a neučil se další závislosti. Průběh grafu lze pravděpodobně vysvětlit tak, že dohromady má architektura až příliš mnoho parametrů na množství dostupných dat a pro Transformer modely je obtížné nastavit své parametry tak, aby se navzájem doplňovaly a pomáhaly si k lepším výsledkům.

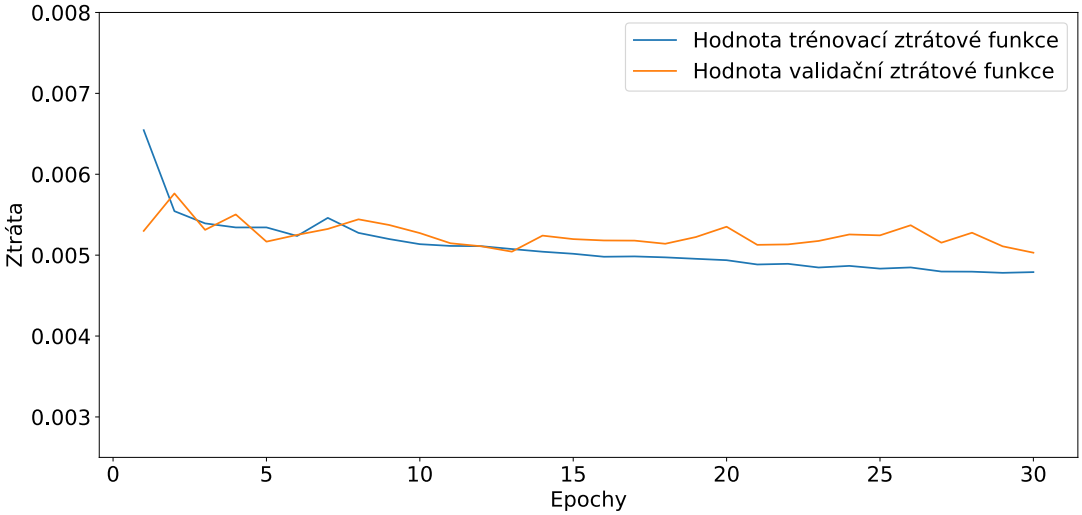
Velice zajímavý je případ, kdy byly zafixované parametry ViT a byl trénován RoBERTa model. Z grafu je vidět relativně rázný pokles obou hodnot ztrátových funkcí v prvních pěti epochách, ale nadále už se hodnoty trénovací ztrátové funkce téměř nezmenšovaly a hodnoty validační ztrátové funkce se spíše mírně zhoršovaly. To je pravděpodobně dáno tím, že jednoslovné dotazy nemají kontext. RoBERTa se naučí vytvářet pro 80 kategorií co nejlepší vektorové reprezentace, ale brzy dosáhne saturace.

Nejlepší průběh grafu je v případě zafixovaných parametrů RoBERTa a trénování ViT. Jelikož jsou oba modely předtrénované, RoBERTa dává i bez dalšího trénování dobrou a bohatou vektorovou reprezentaci dotazu. V obrázcích je daleko větší variabilita toho, jak mohou vypadat, a dává tedy smysl trénovat ViT, aby pro konkrétní výřez obrázku dokázal co nejlépe vyjádřit, co se na něm nachází.

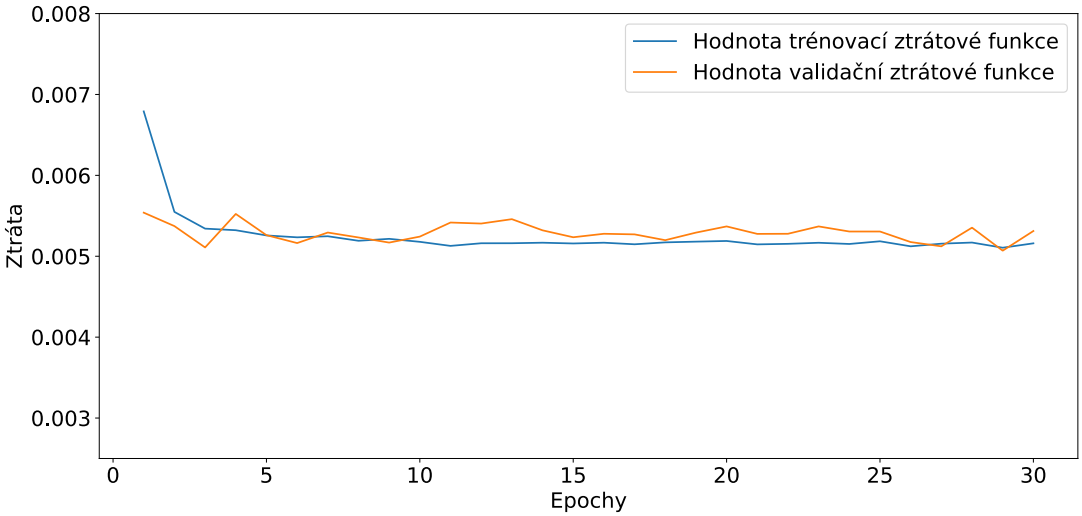
Parametry RoBERTa modelu byly tedy zafixovány a bylo natrénováno 9 modelů pro 9 různých kombinací parametrů *train_thr* a *coco_random_cat_prob*.

5.4.2 Dotazy ve formě vět

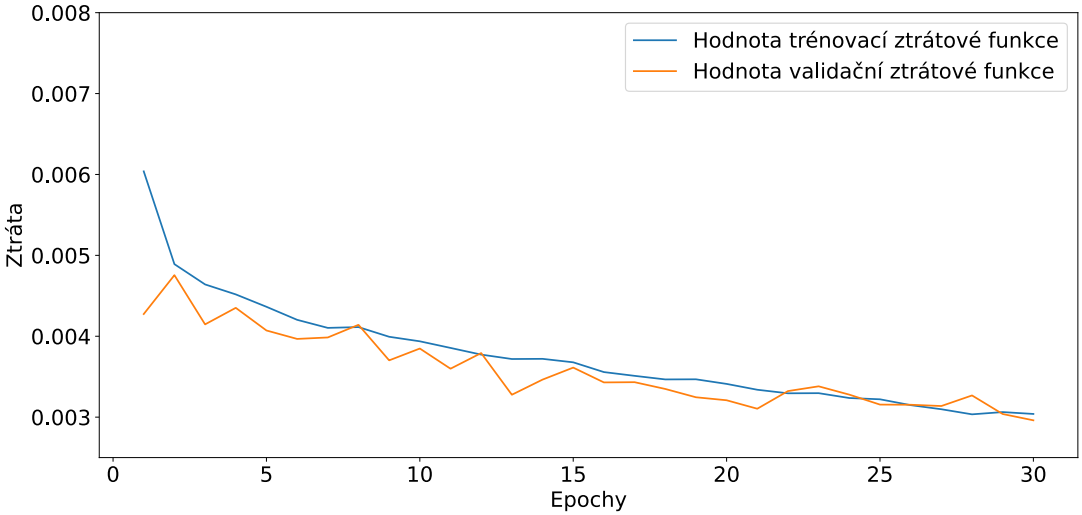
I pro dotazy ve formě vět bylo stejně jako v předchozí kapitole nutné zjistit, jestli má smysl trénovat oba Transformer modely. Opět bylo použito stejné nastavení parametrů $train\_thr = 0.5$ a $coco\_random\_cat\_prob = 0.5$.



(a) Trénování RoBERTa i ViT



(b) Trénování pouze RoBERTa



(c) Trénování pouze ViT

Obrázek 20: Průběhy trénování pro různé kombinace zafixování parametrů

Z grafů jsou zřejmé podobné poznatky, jaké byly popsány v předchozí kapitole pro jednoslovné dotazy. V případě trénování obou modelů opět docházelo k velice mírnému poklesu hodnot trénovací ztrátové funkce a hodnota validační ztrátové funkce dokonce téměř neklesala. To může být dáno tím, že jelikož už dotazy mohou nabývat více podob, je ještě obtížnější pro oba modely, aby se navzájem doplňovaly.

Pokud byly zafixovány parametry ViT a byl trénován pouze RoBERTa model, došlo opět k ráznému poklesu pouze v prvních pár epochách. Na rozdíl od jednoslovných dotazů už mohou být dotazy nyní trochu více variabilní, ale RoBERTa se je pravděpodobně stejně rychle naučí a dále už se nezlepšuje.

I pro dotazy ve formě vět je tedy nejlepší variantou zafixovat parametry RoBERTa a trénovat pouze ViT, aby se naučil pro konkrétní výřez obrázku co nejlépe vyjádřit, co se na něm nachází. Opět bylo natrénováno 9 modelů pro 9 různých kombinací parametrů *train_thr* a *coco_random_cat_prob*.

6 Vyhodnocení

Vyhodnocení natrénovaných modelů probíhalo několika způsoby. Ve všech případech byl pro každý obrázek z validační části datasetu COCO Detection 2017 vygenerován dotaz na každou z 80 kategorií. Kromě toho byly výstupy modelu a požadované výstupy opět binarizovány jako při trénování, ale jiným prahem s názvem *eval_thr*. Parameter *eval_thr* měl zásadní vliv na výsledky vyhodnocení, neboť pro vyšší hodnoty prahu si model musel být více jistý, že nějaký objekt našel. Pro vysoké hodnoty prahu *eval_thr* navíc stejně jako pro vysoké hodnoty prahu *train_thr* zanedbáváme malé objekty.

6.1 Intersection over Union

První způsob vyhodnocení využíval metriku Intersection over Union (IoU). Jedná se o poměr překryvu a sjednocení skutečného výstupu a predikce modelu. Lze ji použít pro výstupy ve formě bounding-boxů i segmentací. V našem případě se jedná o poměr překryvu a sjednocení dvou vektorů dimenze 196 s hodnotami 0, nebo 1. Pro lepší ilustraci je funkce výpočtu IoU zobrazena na obrázku 21.

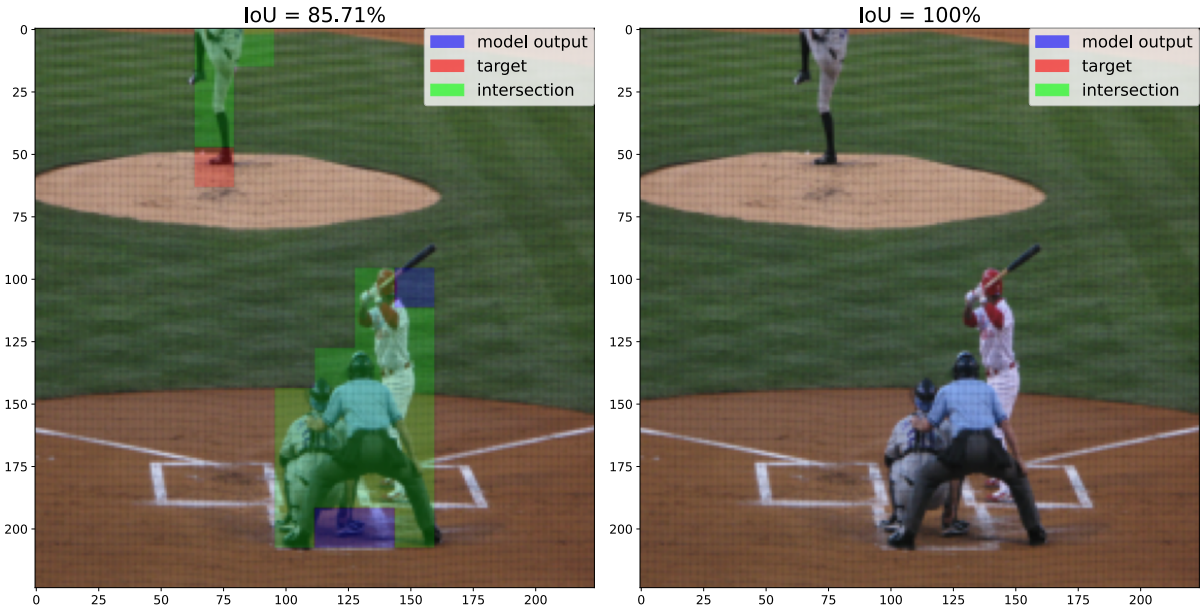
Na obrázku 21(b) byl záměrně ukázán případ, kdy model správně na obrázku nic neoznačil pro kategorii "airplane". Při výpočtu IoU pak vychází 0/0, což je nedefinovaná hodnota (NaN). Teoreticky by dávalo smysl přiřadit takovému případu hodnotu 1 (neboli 100%). Pokud bychom to však udělali, tak úspěšnost detekce modelu výrazně zvýšíme a to zejména u kategorií, které jsou v datasetu málo zastoupeny. Například fén na vlasy je v celé validační části datasetu (5000 obrázků) pouze 9 krát. Pro celou validační část datasetu tedy dostaneme pouze 9 konkrétních hodnot IoU, které říkají, jak dobře model detekuje fén na vlasy, pokud se na obrázku nachází. Předpokládejme, že tyto hodnoty jsou poměrně nízké, neboli model špatně detekuje fén na vlasy. Dále dostaneme 4991 hodnot pro případy, kdy na obrázku fén na vlasy není. Tyto hodnoty jsou buď 0, nebo 1 podle toho, jestli model něco zkusil označit. Pokud je model alespoň částečně dobře natrénovaný a nedetekuje fén na vlasy, pokud tam není, velice rychle se hodnoty IoU pro kategorii fén na vlasy zahltí čísly 1 a výsledná průměrná hodnota pak může vyjít velmi vysoká a to i přesto, že model původně fén na vlasy v 9 případech špatně detekoval.

Při výpočtu IoU pro jednotlivé obrázky a kategorie tedy bylo postupováno jinak. Výpočet průměrné hodnoty IoU pro jednu kategorii byl definován jako

$$IoU = \sum_{m=1}^{M} \sum_{k=1}^{K} \frac{\mathbf{i}_k^m}{\mathbf{u}_k^m}, \tag{13}$$

kde $\mathbf{i}$ je vektor průniku skutečného a požadovaného výstupu, $\mathbf{u}$ je vektor sjednocení skutečného a požadovaného výstupu, index k značí k-tý prvek vektoru a index m značí m-tý obrázek. Obrázků je ve validační části datasetu $M = 5000$ a vektory $\mathbf{i}$ a $\mathbf{u}$ jsou dimenze $K = 196$. Při výpočtu průměrné hodnoty IoU pro jeden obrázek lze postupovat podobně, akorát by index m značil m-tou kategorii a první suma by tedy byla přes všech 80 kategorií.

Tímto způsobem výpočtu zanedbáváme případ z obrázku 21(b) a naopak penalizujeme model, pokud by pro kategorii nenacházející se na obrázku něco označil.

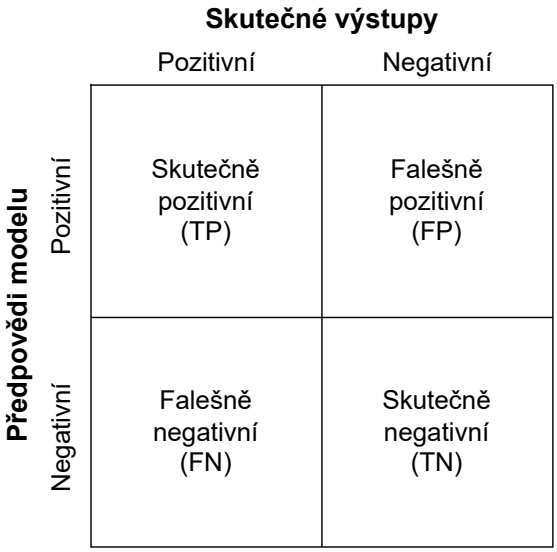


(a) Kategorie "person" (b) Kategorie "airplane"

Obrázek 21: Vizualizace výpočtu IoU přes výřezy obrázku

6.2 Metriky pro binární klasifikaci

Další způsob vyhodnocení využíval metriky pro binární klasifikaci. Bylo zkoumáno, jestli model našel nějaký objekt pro dotaz na danou kategorii, ale na rozdíl od IoU nebylo řešeno, co označil. Vektory výstupu modelu a požadovaného výstupu byly nahrazeny hodnotami True/False podle toho, jestli obsahovaly alespoň jeden prvek s vyšší hodnotou než *eval_thr*. Výsledky binární klasifikace lze zobrazit pomocí matice záměn, která je zobrazena na obrázku 22. Řádky matice reprezentují předpovědi modelu a sloupce reprezentují skutečné (požadované) výstupy.



Obrázek 22: Matice záměn

Z jednotlivých prvků matice záměn je možné spočítat různé metriky. V této práci byly použity metriky přesnost (angl. precision), úplnost (angl. recall) a F1 skóre (angl. F1 score), které lze vyjádřit následujícími vzorci.

$$Precision = \frac{TP}{TP + FP} \tag{14}$$

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

$$F1score = \frac{2 \cdot Precision \cdot Recall}{(Precision + Recall)} \tag{16}$$

Ze vzorců lze vidět, že je obtížné nastavit model tak, aby měl vysokou hodnotu recall a zároveň precision. Bude se jednat o kompromis, což vzbuzuje otázku, kterou z těchto metrik upřednostnit. V případě, že nás zajímají obě metriky stejně, je vhodné sledovat metriku F1 score.

Jelikož bylo vyhodnocení provedeno na každém obrázku pro všech 80 kategorií, výsledkem průchodu přes všechny obrázky validační části datasetu je 80 matic záměn a tím i 80 hodnot precision, recall a F1 score pro jednotlivé kategorie. Jsou různé způsoby, jak tyto hodnoty, konkrétně třeba hodnotu F1 score, zprůměrovat do jediné - macro average, weighted average a micro average. Macro average F1 score je spočteno jako arimetický průměr F1 score jednotlivých kategorií. Neuvažuje tedy absolutní četnost objektů v dané kategorii (angl. support). Weighted average F1 score je spočteno jako vážený součet F1 score jednotlivých kategorií. Váhy jsou získány jako poměr množství objektů v dané kategorii ku celkovému množství objektů. To znamená, že kategorie, které jsou v datasetu málo zastoupené, budou mít menší váhu než více zastoupené kategorie. Micro average sečte hodnoty TP, TN, FP a FN od jednotlivých kategorií a ze součtů spočte F1 score.

Metriky na úrovni jednotlivých výřezů

Poslední způsob vyhodnocení opět využíval metriky pro binární klasifikaci, ale na úrovni jednotlivých výřezů obrázku. Pro vektory výstupu modelu a požadovaného výstupu byly spočteny hodnoty matice záměn tak, že spolu byly porovnávány odpovídající binarizované prvky vektorů. Z hodnot matice záměn pak bylo možné spočítat metriky precision, recall a F1 score pro jednotlivé kategorie, které ale navíc uvažovaly přesnost detekce objektů.

6.3 Jednoslovné dotazy

Pro 9 natrénovaných modelů pro jednoslovné dotazy z kapitoly 5.4.1 proběhlo vyhodnocení s různými hodnotami prahu *eval_thr*, který je v tabulce 2 značen malým indexem u názvu IoU (například $\text{IoU}_{0.25}$ pro $eval\_thr = 0.25$). Trénovací práh *train_thr* je značen *tthr* a náhodný sampling *coco_random_cat_prob* je značen *coco*. Při vyhodnocení vlivu parametrů byla nejdříve sledována průměrná hodnota IoU pro všechny kategorie a celou validační část datasetu.

		$IoU_{0.25}$	$IoU_{0.5}$	$IoU_{0.75}$
tthr = 0.25	coco = 0.25	0.22	0.25	0.18
	coco = 0.5	0.30	0.29	0.19
	coco = 0.75	**0.32**	0.28	0.16
tthr = 0.5	coco = 0.25	0.23	**0.33**	0.24
	coco = 0.5	0.23	0.29	0.19
	coco = 0.75	0.24	0.27	0.17
tthr = 0.75	coco = 0.25	0.14	0.19	**0.25**
	coco = 0.5	0.14	0.17	0.20
	coco = 0.75	0.14	0.15	0.14

Tabulka 2: Vyhodnocení natrénovaných modelů pro metriku IoU

Hodnoty v tabulce 2 dosahují průměrně hodnoty 0.22, v nejlepším případě hodnoty 0.33. Tyto hodnoty nejsou nikterak vysoké, ale to je částečně dáno velkým množstvím falešně positivních výsledků, které se promítnou ve jmenovateli při výpočtu IoU (způsobem popsaným v kapitole 6.1). Pokud bychom výpočet prováděli pouze pro kategorie, které se vyskytují na obrázku, tak je průměrná hodnota IoU 0.37.

Z tabulky také plynou některé očekávané závislosti, a sice pokud byl model trénován při určité hodnotě prahu *tthr*, pak dosahuje nejlepších výsledků při evaluaci na stejné hodnotě prahu *eval_thr*. Naopak pokud byl model trénován s prahem $tthr = 0.25$, tak nejhorších výsledků dosahuje pro $eval\_thr = 0.75$. Je překvapivé, že model nedosahoval nejlepších výsledků pro $coco = 0.5$ a $coco = 0.75$. Bylo očekáváno, že model bude daleko více benefitovat z toho, že mu byly ukazovány i kategorie, které na obrázku nejsou.

V dalším vyhodnocení byla sledována metrika macro average F1 score, při jejímž výpočtu byla zanedbána přesnost detekce objektů. Jedná se tedy o případ, kdy nás zajímalo, jestli model něco označil, pokud něco označit měl. Navíc byla zanedbána absolutní četnost objektů u kategorií (viz. kapitola 6.2).

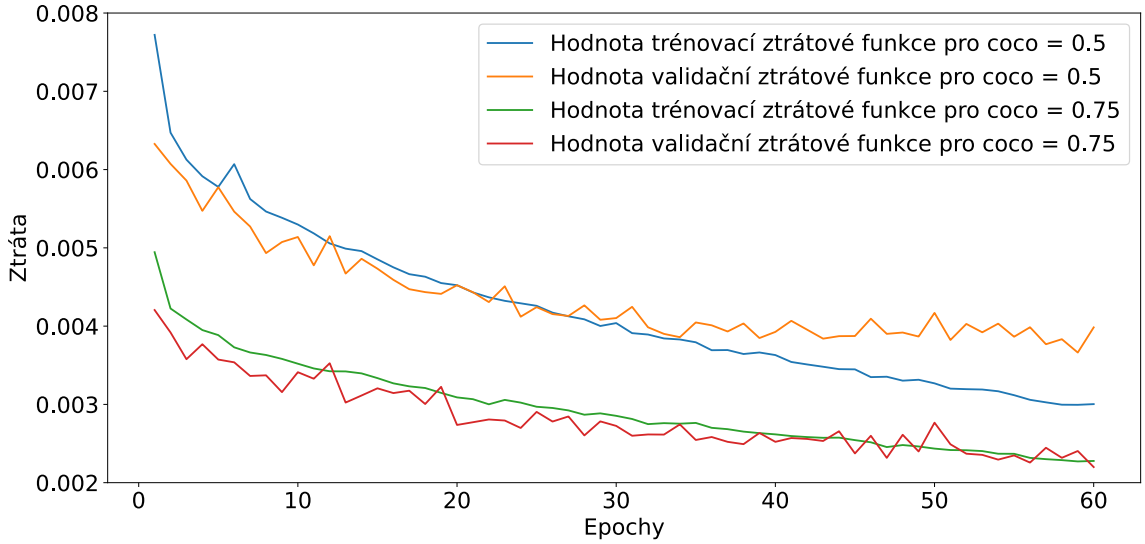
		$F1_{0.25}$	$F1_{0.5}$	$F1_{0.75}$
tthr = 0.25	coco = 0.25	0.12	0.21	0.27
	coco = 0.5	0.21	**0.30**	0.33
	coco = 0.75	**0.26**	0.29	0.24
tthr = 0.5	coco = 0.25	0.18	0.29	**0.34**
	coco = 0.5	0.18	0.25	0.25
	coco = 0.75	0.20	0.22	0.19
tthr = 0.75	coco = 0.25	0.13	0.21	0.27
	coco = 0.5	0.17	0.22	0.24
	coco = 0.75	0.18	0.18	0.15

Tabulka 3: Vyhodnocení natrénovaných modelů pro metriku macro avg F1 score

Průměr hodnot v tabulce 3 je 0.22. V nejlepším případě bylo dosaženo hodnoty 0.34. Hodnoty F1 score snižuje množství falešně pozitivních výsledků. Dále lze vidět, že nejlépe

vychází F1 score pro vyšší hodnoty prahu *eval_thr*. To je dáno tím, že při evaluaci neprojdou prahem nízké hodnoty na výstupu modelu. Další zajímavý poznatek je ten, že modely s prahem $tthr = 0.25$ dosahují lepších výsledků pro $coco = 0.5$ a $coco = 0.75$, a naopak modely s prahem $tthr = 0.75$ dosahují lepších výsledků pro $coco = 0.25$ a $coco = 0.5$. To lze vysvětlit tak, že modely s nižší hodnotou prahu *tthr* se učí hledat daleko více objektů a více benefitují z toho, že jim jsou častěji ukazovány kategorie nevyskytující se na obrázku.

Je překvapivé, že všechny nejvyšší hodnoty sloupců v tabulce 3 mají rozdílné hodnoty parametru *coco*. Vliv tohoto parametru na výsledky je tedy nejasný. Když se ovšem podíváme na průběhy grafů pro trénování modelů s hodnotou prahu například $train\_thr = 0.25$ a hodnotami parametrů $coco = 0.5$ a $coco = 0.75$, je zřejmé, že lépe se požadované závislosti učí model s větší hodnotou parametru *coco*, neboť déle klesá hodnota validační ztrátové funkce. Zároveň je ovšem mírnější pokles hodnoty trénovací ztrátové funkce.



Obrázek 23: Průběh trénování pro různé hodnoty parametru *coco*

V případě výběru nejlepší kombinace parametrů a tedy nejlepšího natrénovaného modelu byla rozhodující metrika IoU v tabulce 2. Při dalším vyhodnocení byl tedy zvolen model s kombinací parametrů $tthr = 0.5$ a $coco = 0.25$, který měl kromě nejlepší hodnoty IoU v celé tabulce 2 také nejlepší průměrnou hodnotu IoU pro všechny evaluační prahy (průměr řádky tabulky 2). Dále také dosahoval nejlepší hodnoty macro average F1 score v tabulce 3. Při dalším vyhodnocení byly použity výsledky pro $eval\_thr = 0.5$.

Použitá metrika macro average F1 score v tabulce 3 byla poněkud přísná a to z důvodu nerovnoměrného zastoupení kategorií v datasetu. V následující tabulce 4 jsou tedy zobrazeny také micro a weighted average hodnoty precision, recall a F1 score, které už vycházejí o něco lépe.

	precision	recall	F1 score
micro avg	0.25	0.58	0.35
macro avg	0.22	0.51	0.29
weighted avg	0.34	0.58	0.40

Tabulka 4: Hodnoty metrik pro nejlepší model

V poslední části vyhodnocení proběhl výpočet průměrné hodnoty IoU pro jednotlivé kategorie. Kromě toho byly spočteny také metriky precision, recall a F1 score na úrovni jednotlivých výřezů (přes jednotlivé prvky vektorů výstupu modelu a požadovaného výstupu). V posledním sloupci je zobrazena průměrná velikost masky jednotlivých kategorií. V tabulce 5 je ukázán výběr kategorií ze začátku, z prostředku a z konce. Výsledky pro všech 80 kategorií jsou zobrazeny v dodatku B na konci práce. Hodnoty jsou seřazeny podle prvního sloupce, a sice průměrné hodnoty IoU.

categories	$\overline{\text{IoU}}$	F1 score	precision	recall	$\overline{\text{mask}}$
spoon	0.0	0.0	0.0	0.0	0.011
toaster	0.0	0.0	0.0	0.0	0.035
hair drier	0.0	0.0	0.0	0.0	0.020
handbag	0.0	0.0	0.0	0.0	0.019
backpack	0.0	0.0	0.0	0.0	0.022
...					
bowl	0.14	0.24	0.17	0.43	0.120
vase	0.14	0.24	0.19	0.34	0.059
potted plant	0.14	0.24	0.18	0.37	0.066
frisbee	0.14	0.24	0.21	0.29	0.017
microwave	0.14	0.25	0.28	0.22	0.073
...					
elephant	0.52	0.69	0.62	0.77	0.247
giraffe	0.53	0.69	0.63	0.76	0.144
person	0.64	0.78	0.75	0.81	0.164
stop sign	0.67	0.80	0.81	0.80	0.074
zebra	0.76	0.86	0.86	0.86	0.198
avg	0.19	0.29	0.26	0.40	0.095

Tabulka 5: Hodnoty metrik nejlepšího modelu pro jednotlivé kategorie

Z výsledků posledních kategorií tabulky 5 a také z průběhů grafu trénování na obrázku 23 lze dobře vidět, že architektura funguje a učí se požadované závislosti, což bylo hlavním cílem této práce. Bohužel má model problémy s malými objekty a s kategoriemi, které jsou v datasetu málo zastoupené. S malými objekty má model problém i při použití menšího trénovacího a evaluačního prahu.

6.4 Dotazy ve formě vět

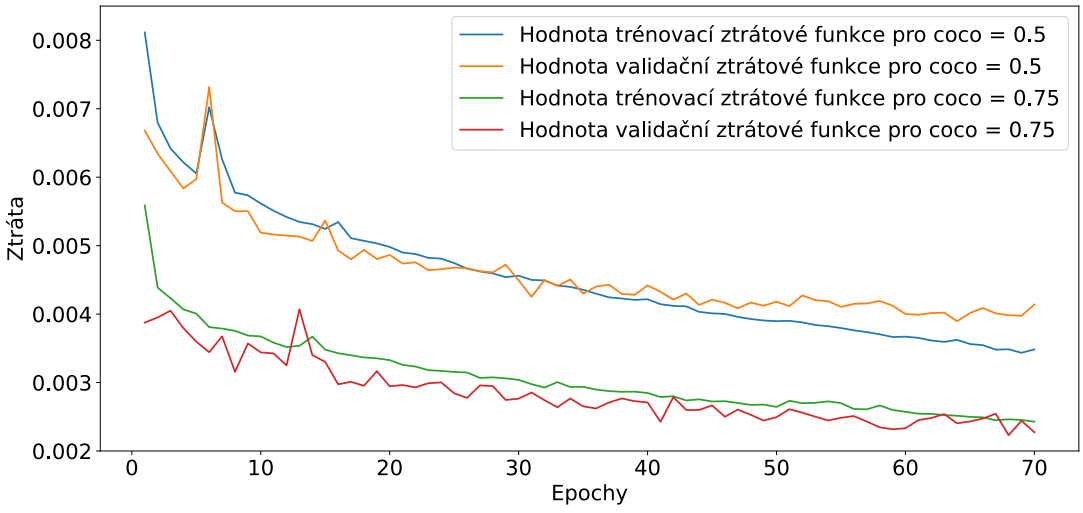
Podobné vyhodnocení jako v předchozí kapitole proběhlo také pro 9 natrénovaných modelů pro dotazy ve formě vět. Opět je použito stejné značení všech parametrů a v prvním způsobu vyhodnocení je v tabulce 6 zobrazena průměrná hodnota IoU pro všechny kategorie a celou validační část datasetu.

		$IoU_{0.25}$	$IoU_{0.5}$	$IoU_{0.75}$
tthr = 0.25	coco = 0.25	0.19	0.25	0.17
	coco = 0.5	0.25	0.24	0.14
	coco = 0.75	**0.29**	0.24	0.14
tthr = 0.5	coco = 0.25	0.17	0.27	0.19
	coco = 0.5	0.24	**0.32**	**0.21**
	coco = 0.75	0.22	0.25	0.15
tthr = 0.75	coco = 0.25	0.09	0.11	0.13
	coco = 0.5	0.11	0.12	0.13
	coco = 0.75	0.11	0.10	0.09

Tabulka 6: Vyhodnocení natrénovaných modelů pro metriku IoU

Průměr hodnot v tabulce 6 je 0.18. V nejlepším případě byla dosažena hodnota 0.32. Jedná se tedy o menší čísla než v případě modelů pro jednoslovné dotazy, což bylo očekávané, neboť už dotazy mohou být složitější. Nízké hodnoty jsou opět dány velkým množstvím falešně pozitivních výsledků. Pokud bychom uskutečnili výpočet pouze pro kategorie vyskytující se na obrázku, průměrná hodnota IoU by vyšla 0.36.

Z tabulky je zřejmé, že nemá smysl trénovat model pro $tthr = 0.75$, neboť nedosahuje dobrých výsledků. Zajímavé je, že nejlepších výsledků bylo dosaženo pro $tthr = 0.5$, stejně jako v případě modelů pro jednoslovné dotazy. Vliv parametru *coco* opět není jednoznačný, ale nejlepších výsledků bylo ve dvou případech dosaženo pro $coco = 0.5$. Pokud se ovšem podíváme na průběhy trénování modelů s hodnotou prahu $train\_thr = 0.25$ a hodnotami parametrů $coco = 0.5$ a $coco = 0.75$, platí stejně jako v případě jednoslovných dotazů, že lépe se požadované závislosti učí model s větší hodnotou parametru *coco*.



Obrázek 24: Průběh trénování pro různé hodnoty parametru *coco*

V dalším vyhodnocení byla opět sledována metrika macro average F1 score se zanedbanou přesností detekce objektů.

		$F1_{0.25}$	$F1_{0.5}$	$F1_{0.75}$
tthr = 0.25	coco = 0.25	0.12	0.22	0.28
	coco = 0.5	0.18	0.22	0.20
	coco = 0.75	0.20	0.21	0.17
tthr = 0.5	coco = 0.25	0.13	0.20	0.23
	coco = 0.5	**0.22**	**0.29**	**0.28**
	coco = 0.75	0.18	0.18	0.15
tthr = 0.75	coco = 0.25	0.10	0.13	0.13
	coco = 0.5	0.13	0.15	0.14
	coco = 0.75	0.13	0.10	0.07

Tabulka 7: Vyhodnocení natrénovaných modelů pro metriku macro avg F1 score

Průměr hodnot v tabulce 7 je 0.17, přičemž nejvyšší dosažená hodnota je 0.29. Nižší hodnoty jsou opět způsobeny velkým množstvím falešně pozitivních výsledků. Opět platí, že lepších výsledků bylo dosaženo pro vyšší hodnoty prahu *eval_thr* (vysvětlení viz. kapitola 6.3).

Nejzajímavější na celé tabulce 7 je fakt, že nejlepší hodnoty pro všechny prahy vykazoval jeden konkrétní model s nastavením parametrů $tthr = 0.5$ a $coco = 0.5$. Toto nastavení dalo také dva ze tří nejlepších výsledků IoU v tabulce 6. Výběr nejlepšího modelu je tedy jednoznačný. Při dalším vyhodnocení byl uvažován práh $eval\_thr = 0.5$.

	precision	recall	F1 score
micro avg	0.39	0.48	0.43
macro avg	0.27	0.40	0.29
weighted avg	0.42	0.48	0.41

Tabulka 8: Hodnoty metrik pro nejlepší model

V tabulce 8 lze opět sledovat lepší výsledky pro jiné způsoby výpočtu metrik precision, recall a F1 score. Na závěr jsou v tabulce 9 uvedeny výsledky pro pár vybraných kategorií. Tabulka zobrazuje průměrnou hodnotu IoU pro jednotlivé kategorie a zároveň metriky precision, recall a F1 score spočítané na úrovni jednotlivých výřezů. V posledním sloupci je zobrazena průměrná velikost masky jednotlivých kategorií. Tabulka byla seřazena podle prvního sloupce, tedy průměrné hodnoty IoU. Výsledky pro všech 80 kategorií jsou zobrazeny v dodatku C.

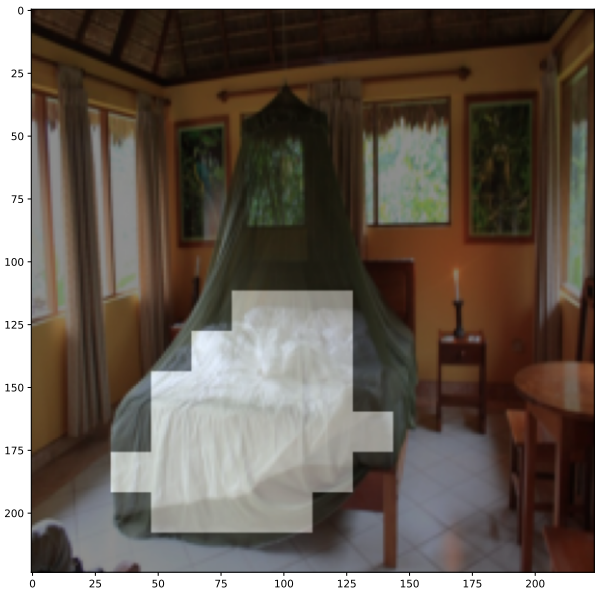
categories	$\overline{\text{IoU}}$	F1 score	precision	recall	$\overline{\text{mask}}$
backpack	0.0	0.0	0.0	0.0	0.022
handbag	0.0	0.0	0.0	0.0	0.019
baseball bat	0.0	0.0	0.0	0.0	0.007
fork	0.0	0.0	0.0	0.0	0.012
knife	0.0	0.0	0.0	0.0	0.013
...					
potted plant	0.14	0.25	0.18	0.41	0.066
sandwich	0.14	0.25	0.18	0.41	0.179
traffic light	0.15	0.26	0.21	0.34	0.019
cup	0.15	0.26	0.36	0.20	0.037
oven	0.15	0.26	0.42	0.19	0.141
...					
orange	0.39	0.56	0.53	0.59	0.102
zebra	0.45	0.62	0.49	0.84	0.198
fire hydrant	0.45	0.62	0.57	0.69	0.077
toilet	0.53	0.69	0.81	0.60	0.108
person	0.67	0.8	0.85	0.75	0.164
avg	0.17	0.27	0.26	0.33	0.095

Tabulka 9: Hodnoty metrik nejlepšího modelu pro jednotlivé kategorie

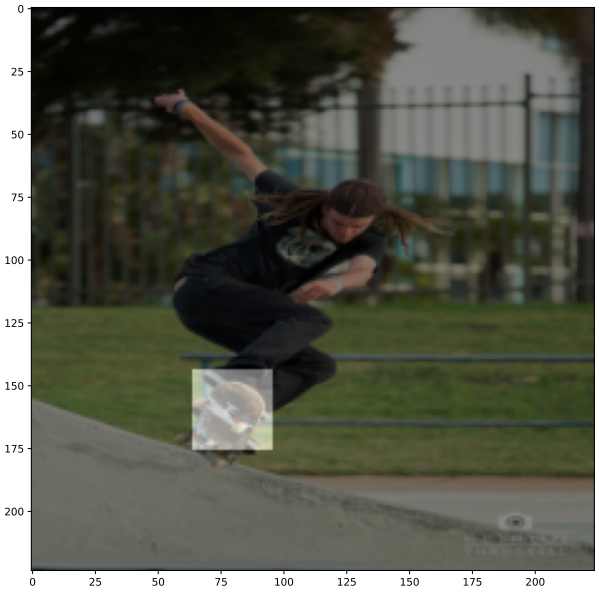
Z výsledků posledních kategorií tabulky 9 a také z průběhu grafu trénování na obrázku 24 lze dobře vidět, že architektura se učí požadované závislosti i pro dotazy ve formě vět. Opět má architektura problémy s malými objekty a s kategoriemi, které jsou v datasetu málo zastoupené.

Že se architektura učí požadované závislosti lze navíc ověřit položením dotazu, který nebyl součástí předpřipravených vět v kapitole 5.3.1. V několika případech byl model schopný objekt správně detekovat a byl schopný detekovat také více objektů, pokud byly v dotazu zahrnuty. Výstupy testovaných dotazů jsou zobrazeny na následujících obrázcích.

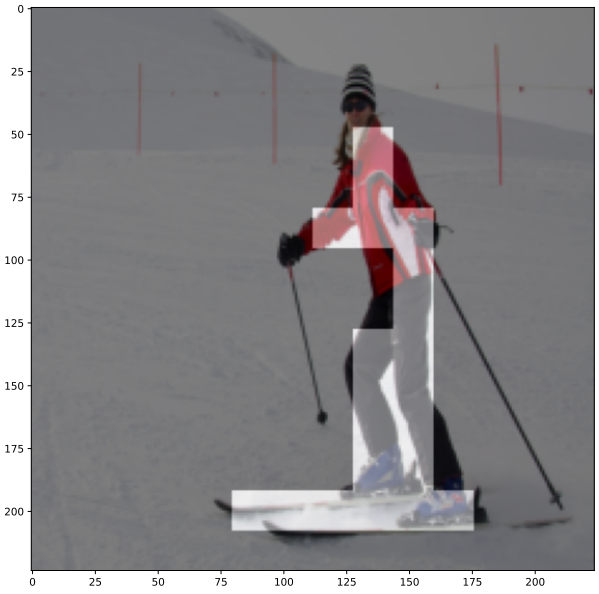
(a) Show me a motorcycle in this picture.



(b) There should be a bed in this image, where?



(c) Find a person and a skateboard.



(d) Could you detect a person with skis?

Obrázek 25: Výstupy natrénovaného modelu pro dotazy neviděné při procesu trénování

7 Závěr

V první části práce byly představeny tři základní typy neuronových sítí, a sice dopředné, rekurentní a konvoluční. U všech byla stručně popsána jejich činnosti. U rekurentních neuronových sítí byly navíc představeny tři další druhy RNN, na jejichž principech je mimo jiné založena Transformer architektura. V závěru první části byla ještě zmíněna konvoluční síť ResNet, která zavedla tzv. skip connections, které se využívají ve velkém množství moderních architektur neuronových sítí.

V druhé části pak byla představena Transformer architektura. Byla podrobně popsána její činnost a byly představeny tři důležité modely, pričemž dva z nich pak byly v využity v praktické části práce. V závěru teoretické části byly ještě popsány datasety pro detekci objektů, ze kterých byl použit dataset COCO Detection 2017.

V praktické části byla navržena architektura pro detekci objektů s využitím textových dotazů. Bylo popsáno předzpracování dat a natrénováno několik modelů s různými kombinacemi parametrů. Ukázalo se, že při trénování bylo nejlepší zafixovat model RoBERTa a trénovat model ViT. Na závěr proběhlo vyhodnocení pomocí metriky Intersection over Union a metrik pro binární klasifikaci. Bylo zjištěno, že architektura se učí požadované závislosti a některé kategorie hledá relativně dobře. čímž byl splněn hlavní cíl této práce. Velký problém má model s kategoriemi, které jsou v datasetu málo zastoupené a běžně zabírají malou část obrázku. Dále model velice často poskytoval falešné pozitivní výsledky.

Při vyhodnocení byly zjištěny některé závislosti na třech důležitých parametrech, a sice *train_thr*, *coco* a *eval_thr*. Optimální nastavení prahu *train_thr* by se mohlo nacházet někde mezi hodnotami 0.25 a 0.5. Relativně neznámý zůstává vliv parametru *coco*, který s určitou pravděpodobností ukazoval modelu kategorie nevyskytující se na obrázku. Modely dosahující nejlepších výsledků totiž často měly odlišné hodnoty tohoto parametru. Pokud jsme se ovšem podívali na průběhy trénování, bylo zřejmé, že lépe se učily požadované závislosti modely s větší hodnotou *coco* (například $coco = 0.75$).

Nejlepší natrénovaný model pro dotazy ve formě vět je možné si vyzkoušet na adrese https://huggingface.co/spaces/fmajer/T-BOD.

7.1 Budoucí práce

V budoucí práci by bylo zajímavé použít dataset Objects 365, díky čemuž by byl model schopen rozpoznávat více kategorií a pravděpodobně i s větší přesností, což by bylo dáno velikostí datasetu a kvalitou anotací. Kromě většího obrazového datasetu by bylo vhodné zaměřit se také na množství a kvalitu předpřipravených dotazů ve formě vět. Ty by mohly navíc obsahovat věty, které by nepožadovaly žádnou detekci a model by se učil pro takové dotazy nic neoznačovat.

Použitím většího datasetu a většího množství předpřipravených dotazů by mohl mít lepší průběh graf na obrázku 20(a), kde byl trénován zároveň RoBERTa i ViT model. I při použití datasetu COCO Detection 2017 by bylo možné prozkoumat více některé možnosti architektury, například porovnat různé varianty poolingu za vrstvou BERTMAP, nebo zafixovat pouze parametry prvních pár vrstev předtrénovaných Transformer modelů. Nebyly také prozkoumány možnosti použití různých variant RoBERTa a ViT. Za zmínku stojí například model "vit_base_patch8_224", který by umožňoval mít větší množství výřezů v obrázku a tím i větší přesnost detekce. Závěrem by bylo vhodné více prozkoumat vliv parametru *coco* a pokusit se zmenšit množství falešně pozitivních výsledků a to například za pomoci speciální ztrátové funkce.

Reference

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer Publishing Company, Incorporated, 1st edition, 2018.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. Dostupné z: `http://www.deeplearningbook.org`.

[3] Weijiang Feng, Naiyang Guan, Yuan Li, Xiang Zhang, and Zhigang Luo. Audio visual speech recognition with multimodal recurrent neural networks, 2017.

[4] Andrew Ng. Deep learning course, 2017.

[5] Jain Neelesh. Understanding encoder-decoder sequence-to-sequence model. 5.2.2019 [citace 17.3.2023]. Dostupné z: `https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346`.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[7] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation, 2015.

[8] Christopher Olah. Neural networks, types, and functional programming. 3.9.2015 [cit. 19.3.2023]. Dostupné z: `http://colah.github.io/posts/2015-09-NN-Types-FP/`.

[9] Christopher Olah. Understanding lstm networks. 27.8.2015 [cit. 20.3.2023]. Dostupné z: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[11] Anh H. Reynolds. Convolutional neural networks (CNNs). 2019 [cit. 26.3.2023]. Dostupné z: `https://anhreynolds.com/blogs/cnn.html`.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[14] Jay Alammar. The illustrated transformer. 27.6.2018 [cit. 2.4.2023]. Dostupné z: `http://jalammar.github.io/illustrated-transformer/`.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019.

[18] Rani Horev. BERT explained: State of the art language model for NLP. 10.11.2018 [cit. 17.4.2023]. Dostupné z: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach, 2019.

[20] Philip Gage. A new algorithm for data compression. *C Users J.*, 1994.

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common objects in context, 2015.

[23] Shuai Shao, Zeming Li, Tianyuan Zhang, Chao Peng, Gang Yu, Xiangyu Zhang, Jing Li, and Jian Sun. Objects365: A large-scale, high-quality dataset for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8429–8438, 2019.

[24] Matthias Minderer and Alexey Gritsenko. Simple open-vocabulary object detection with vision transformers. *ECCV*, 2022.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Dodatek A

Tento dodatek zobrazuje statistiky z kapitoly 5.3.2 pro všech 80 kategorií v trénovací části datasetu COCO Detection 2017. Statistiky byly seřazeny podle prvního sloupce tabulky, tedy podle průměru všech prvků vektorů požadovaného výstupu pro danou kategorii.

categories	avg	$\overline{\text{avg}}$	median	#anns
sports ball	0.005	0.218	0.088	4262
baseball bat	0.007	0.189	0.114	2506
skis	0.008	0.163	0.095	3082
baseball glove	0.009	0.290	0.149	2629
spoon	0.011	0.239	0.139	3529
fork	0.012	0.242	0.153	3555
knife	0.013	0.275	0.170	4326
mouse	0.015	0.400	0.262	1876
frisbee	0.017	0.401	0.267	2184
handbag	0.019	0.354	0.207	6841
tennis racket	0.019	0.355	0.231	3394
traffic light	0.019	0.336	0.188	4139
hair drier	0.020	0.399	0.280	189
skateboard	0.021	0.326	0.210	3476
backpack	0.022	0.398	0.268	5528
remote	0.022	0.417	0.266	3076
toothbrush	0.023	0.367	0.253	1007
tie	0.024	0.370	0.249	3810
snowboard	0.026	0.374	0.260	1654
bottle	0.027	0.408	0.307	8501
cell phone	0.028	0.500	0.439	4803
clock	0.032	0.511	0.459	4659
surfboard	0.035	0.437	0.343	3486
toaster	0.035	0.551	0.545	217
cup	0.037	0.485	0.418	9189
kite	0.039	0.331	0.179	2261
sink	0.043	0.534	0.524	4678
scissors	0.045	0.428	0.355	947
wine glass	0.045	0.453	0.369	2533
book	0.052	0.519	0.485	5332
bicycle	0.054	0.481	0.424	3252
car	0.059	0.556	0.573	12251
vase	0.059	0.555	0.567	3593
chair	0.060	0.480	0.419	12774
bird	0.060	0.446	0.348	3237
carrot	0.061	0.446	0.367	1683
potted plant	0.066	0.533	0.523	4452
bench	0.068	0.560	0.577	5570
microwave	0.073	0.662	0.838	1547

stop sign	0.074	0.645	0.827	1734
fire hydrant	0.077	0.594	0.670	1711
keyboard	0.081	0.621	0.740	2115
umbrella	0.086	0.554	0.569	3968
apple	0.089	0.591	0.669	1586
tv	0.092	0.661	0.831	4561
orange	0.102	0.628	0.768	1699
boat	0.103	0.573	0.614	3025
toilet	0.108	0.641	0.785	3353
truck	0.115	0.668	0.875	6127
parking meter	0.116	0.680	0.902	705
bowl	0.120	0.689	0.938	7111
broccoli	0.124	0.563	0.597	1939
dog	0.124	0.622	0.742	4385
airplane	0.126	0.547	0.554	2986
laptop	0.128	0.670	0.857	3524
horse	0.129	0.553	0.557	2941
banana	0.132	0.617	0.728	2243
oven	0.141	0.725	0.965	2877
couch	0.142	0.687	0.871	4423
giraffe	0.144	0.525	0.507	2546
hot dog	0.144	0.658	0.845	1222
sheep	0.150	0.565	0.592	1529
cow	0.150	0.586	0.635	1968
motorcycle	0.151	0.615	0.721	3502
suitcase	0.154	0.685	0.894	2402
donut	0.157	0.652	0.808	1523
cake	0.159	0.680	0.894	2925
person	0.164	0.598	0.672	64115
teddy bear	0.179	0.662	0.833	2140
sandwich	0.179	0.696	0.938	2365
cat	0.184	0.678	0.886	4114
refrigerator	0.191	0.757	0.998	2360
zebra	0.198	0.601	0.660	1916
bear	0.210	0.696	0.922	960
bus	0.215	0.740	0.999	3952
train	0.236	0.740	0.992	3588
elephant	0.247	0.667	0.831	2143
pizza	0.288	0.758	1.0	3166
bed	0.301	0.786	1.0	3682
dining table	0.312	0.825	1.0	11837

Tabulka 10: Statistiky pro všech 80 kategorií

Dodatek B

Tento dodatek zobrazuje hodnoty zvolených metrik z kapitoly 6.3 pro všech 80 kategorií ve validační části datasetu COCO Detection 2017. Byl použit model pro jednoslovné dotazy s parametry $train\_thr = 0.5$, $coco = 0.25$ a $eval\_thr = 0.5$. Hodnoty jsou seřazeny podle prvního sloupce, tedy průměrné hodnoty IoU pro danou kategorii.

categories	$\overline{\text{IoU}}$	F1 score	precision	recall	$\overline{\text{mask}}$
spoon	0.0	0.0	0.0	0.0	0.011
toaster	0.0	0.0	0.0	0.0	0.035
hair drier	0.0	0.0	0.0	0.0	0.020
handbag	0.0	0.0	0.0	0.0	0.019
backpack	0.0	0.0	0.0	0.0	0.022
toothbrush	0.0	0.01	0.0	0.01	0.023
fork	0.01	0.02	0.01	0.05	0.012
mouse	0.01	0.02	0.01	0.13	0.015
knife	0.01	0.02	0.03	0.02	0.013
remote	0.01	0.03	0.02	0.06	0.022
snowboard	0.03	0.06	0.03	0.20	0.026
wine glass	0.03	0.06	0.04	0.10	0.045
baseball bat	0.03	0.06	0.06	0.06	0.007
apple	0.04	0.08	0.05	0.19	0.089
chair	0.04	0.08	0.08	0.08	0.060
oven	0.05	0.09	0.05	0.28	0.141
clock	0.06	0.12	0.06	0.52	0.032
bench	0.07	0.13	0.08	0.28	0.068
tie	0.07	0.13	0.07	0.48	0.024
sports ball	0.07	0.13	0.26	0.09	0.005
bird	0.07	0.14	0.08	0.46	0.060
book	0.08	0.14	0.13	0.16	0.052
baseball glove	0.08	0.15	0.22	0.11	0.009
donut	0.08	0.15	0.08	0.73	0.157
skateboard	0.08	0.15	0.13	0.19	0.021
boat	0.09	0.17	0.11	0.41	0.103
skis	0.10	0.19	0.20	0.17	0.008
cow	0.10	0.19	0.11	0.58	0.150
cake	0.11	0.19	0.13	0.39	0.159
bear	0.11	0.19	0.11	0.87	0.210
parking meter	0.11	0.20	0.17	0.24	0.116
bottle	0.12	0.21	0.19	0.24	0.027
cup	0.12	0.22	0.17	0.31	0.037
tennis racket	0.13	0.23	0.24	0.21	0.019
suitcase	0.13	0.23	0.19	0.28	0.154
dog	0.13	0.23	0.15	0.49	0.124
scissors	0.14	0.24	0.29	0.20	0.045
horse	0.14	0.24	0.15	0.62	0.129
bowl	0.14	0.24	0.17	0.43	0.120

vase	0.14	0.24	0.19	0.34	0.059
potted plant	0.14	0.24	0.18	0.37	0.066
frisbee	0.14	0.24	0.21	0.29	0.017
microwave	0.14	0.25	0.28	0.22	0.073
bed	0.15	0.26	0.19	0.45	0.301
hot dog	0.16	0.28	0.21	0.41	0.144
orange	0.17	0.28	0.18	0.68	0.102
train	0.17	0.30	0.19	0.66	0.236
couch	0.18	0.30	0.22	0.46	0.142
sandwich	0.18	0.30	0.19	0.69	0.179
surfboard	0.21	0.34	0.31	0.37	0.035
carrot	0.21	0.35	0.27	0.50	0.061
bus	0.22	0.36	0.24	0.71	0.215
bicycle	0.22	0.36	0.29	0.47	0.054
tv	0.23	0.38	0.43	0.34	0.092
dining table	0.24	0.38	0.31	0.50	0.312
teddy bear	0.24	0.39	0.29	0.57	0.179
truck	0.25	0.39	0.41	0.38	0.115
traffic light	0.25	0.39	0.45	0.35	0.019
sink	0.26	0.41	0.50	0.35	0.043
banana	0.26	0.41	0.35	0.51	0.132
kite	0.28	0.43	0.53	0.36	0.039
car	0.30	0.46	0.52	0.41	0.059
refrigerator	0.30	0.46	0.46	0.47	0.191
cat	0.31	0.47	0.35	0.75	0.184
sheep	0.31	0.47	0.37	0.65	0.150
cell phone	0.33	0.49	0.53	0.46	0.028
umbrella	0.33	0.50	0.41	0.63	0.086
keyboard	0.36	0.53	0.51	0.55	0.081
laptop	0.39	0.56	0.51	0.61	0.128
pizza	0.39	0.56	0.43	0.79	0.288
broccoli	0.39	0.56	0.51	0.62	0.124
motorcycle	0.41	0.58	0.57	0.59	0.151
fire hydrant	0.42	0.60	0.64	0.56	0.077
toilet	0.44	0.61	0.69	0.55	0.108
airplane	0.44	0.61	0.57	0.66	0.126
elephant	0.52	0.69	0.62	0.77	0.247
giraffe	0.53	0.69	0.63	0.76	0.144
person	0.64	0.78	0.75	0.81	0.164
stop sign	0.67	0.80	0.81	0.80	0.074
zebra	0.76	0.86	0.86	0.86	0.198
avg	0.19	0.29	0.26	0.40	0.095

Tabulka 11: Hodnoty metrik nejlepšího modelu pro jednotlivé kategorie

Dodatek C

Tento dodatek zobrazuje hodnoty zvolených metrik z kapitoly 6.4 pro všech 80 kategorií ve validační části datasetu COCO Detection 2017. Byl použit model na dotazy ve formě vět s parametry $train\_thr = 0.5$, $coco = 0.5$ a $eval\_thr = 0.5$. Hodnoty jsou seřazeny podle prvního sloupce, tedy průměrné hodnoty IoU pro danou kategorii.

categories	$\overline{\text{IoU}}$	F1 score	precision	recall	$\overline{\text{mask}}$
backpack	0.0	0.0	0.0	0.0	0.022
handbag	0.0	0.0	0.0	0.0	0.019
baseball bat	0.0	0.0	0.0	0.0	0.007
fork	0.0	0.0	0.0	0.0	0.012
knife	0.0	0.0	0.0	0.0	0.013
spoon	0.0	0.0	0.0	0.0	0.011
mouse	0.0	0.0	0.0	0.0	0.015
microwave	0.0	0.0	0.0	0.0	0.073
toaster	0.0	0.0	0.0	0.0	0.035
scissors	0.0	0.0	0.0	0.0	0.045
hair drier	0.0	0.0	0.0	0.0	0.020
toothbrush	0.0	0.0	0.0	0.0	0.023
bottle	0.0	0.01	0.08	0.0	0.027
carrot	0.01	0.01	0.02	0.01	0.061
baseball glove	0.01	0.01	0.06	0.01	0.009
wine glass	0.01	0.03	0.03	0.03	0.045
book	0.03	0.06	0.22	0.03	0.052
skis	0.03	0.06	0.04	0.11	0.008
remote	0.03	0.07	0.07	0.06	0.022
snowboard	0.03	0.07	0.04	0.22	0.026
refrigerator	0.04	0.07	0.61	0.04	0.191
apple	0.04	0.08	0.07	0.10	0.089
chair	0.04	0.08	0.17	0.05	0.060
sports ball	0.04	0.08	0.07	0.10	0.005
skateboard	0.05	0.09	0.07	0.12	0.021
tennis racket	0.07	0.13	0.20	0.10	0.019
bicycle	0.07	0.14	0.24	0.09	0.054
hot dog	0.08	0.15	0.09	0.38	0.144
sink	0.09	0.17	0.22	0.14	0.043
frisbee	0.10	0.18	0.12	0.35	0.017
donut	0.10	0.18	0.11	0.46	0.157
suitcase	0.10	0.19	0.24	0.15	0.154
keyboard	0.11	0.20	0.19	0.21	0.081
bench	0.12	0.21	0.33	0.16	0.068
cake	0.12	0.22	0.16	0.33	0.159
parking meter	0.13	0.23	0.19	0.29	0.116
couch	0.14	0.24	0.30	0.20	0.142
kite	0.14	0.25	0.16	0.50	0.039
potted plant	0.14	0.25	0.18	0.41	0.066

sandwich	0.14	0.25	0.18	0.41	0.179
traffic light	0.15	0.26	0.21	0.34	0.019
cup	0.15	0.26	0.36	0.20	0.019
oven	0.15	0.26	0.42	0.19	0.141
bowl	0.18	0.31	0.52	0.22	0.120
horse	0.18	0.31	0.21	0.62	0.129
truck	0.19	0.32	0.25	0.47	0.115
surfboard	0.19	0.32	0.24	0.49	0.035
cow	0.20	0.34	0.23	0.61	0.150
vase	0.22	0.36	0.41	0.32	0.059
dog	0.22	0.36	0.27	0.54	0.124
bird	0.23	0.37	0.33	0.42	0.060
tv	0.23	0.38	0.43	0.33	0.092
sheep	0.24	0.39	0.28	0.64	0.150
banana	0.24	0.39	0.39	0.39	0.132
bed	0.25	0.40	0.39	0.41	0.301
dining table	0.25	0.40	0.42	0.38	0.312
boat	0.25	0.40	0.36	0.46	0.103
tie	0.26	0.41	0.36	0.47	0.024
cell phone	0.26	0.41	0.49	0.35	0.028
laptop	0.27	0.42	0.43	0.41	0.128
giraffe	0.27	0.42	0.29	0.76	0.144
train	0.28	0.44	0.33	0.67	0.236
car	0.30	0.46	0.50	0.43	0.059
stop sign	0.30	0.46	0.32	0.81	0.074
airplane	0.30	0.47	0.40	0.55	0.126
teddy bear	0.31	0.48	0.40	0.60	0.179
bus	0.32	0.49	0.38	0.68	0.215
broccoli	0.32	0.49	0.44	0.56	0.124
bear	0.35	0.52	0.39	0.79	0.210
clock	0.36	0.53	0.52	0.53	0.032
pizza	0.36	0.53	0.39	0.80	0.288
cat	0.37	0.55	0.43	0.74	0.184
elephant	0.38	0.55	0.47	0.67	0.247
umbrella	0.38	0.55	0.57	0.54	0.086
motorcycle	0.39	0.56	0.55	0.56	0.151
orange	0.39	0.56	0.53	0.59	0.102
zebra	0.45	0.62	0.49	0.84	0.198
fire hydrant	0.45	0.62	0.57	0.69	0.077
toilet	0.53	0.69	0.81	0.60	0.108
person	0.67	0.80	0.85	0.75	0.164
avg	0.17	0.27	0.26	0.33	0.095

Tabulka 12: Hodnoty metrik nejlepšího modelu pro jednotlivé kategorie