Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informačních technologií

Interpretační analýza hlubokých ANN

Diplomová práce

Autor: Daniel, Tesař, Bc.
Studijní obor: Aplikovaná informatika (AI2-k)

Vedoucí práce: Ing. Karel Mls, Ph.D.

Hradec Králové srpen 2023

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 6. 8. 2023 Bc. Daniel Tesař

Poděkování:

Děkuji Ing. Karlu Mlsovi, Ph.D. za jeho odborné rady a dobré vedení během zpracování mojí diplomové práce.

Anotace

Předmětem této diplomové práce je interpretovat chování umělé neuronové sítě. Prvně stručně vysvětlit, co je to umělá neuronová síť, neuron a jak se budují a k čemu se používají. V další řadě popsat problém černé skříňky a interpretace umělé neuronové sítě a jaké způsoby je možno využít v případě interpretace těchto umělých neuronových sítí. V praktické části bude vyzkoušeno několik způsobů interpretace umělé neuronové sítě pro již vytvořený a vytrénovaný model umělé neuronové sítě. Tyto způsoby interpretace a jejich výsledky budou prezentovány a zkoumány. Práce se bude snažit shrnout, zda použité interpretační metody skutečně napomáhají k tomu, aby byla umělá neuronová síť snadněji interpretovatelná.

Klíčová slova

Neuronová síť, interpretovatelné ANN, neuron, umělá, černá skříňka

Annotation

Title: Interpretation analysis of deep ANNs

The subject of this diploma thesis is to interpret the behaviour of an artificial neural network. Firstly, to briefly explain what an artificial neural network is, what a neuron is, what they are used for, and how they are constructed. Secondly, to describe the problem of the black box and interpretation of the artificial neural network, and what methods can be used to interpret these artificial neural networks. In the practical part, several methods of interpreting an artificial neural network will be tested for an already created and trained model of an artificial neural network. These interpretation methods and their results will be evaluated and examined. The thesis will try to summarize whether the interpretation methods used really help to make the artificial neural network more easily interpretable.

Key words

Neural network, interpretable ANN, neuron, artificial, black box

OBSAH

SEZNAM OBRÁZKŮ

1. ÚVOD

Cílem práce je pokusit se interpretovat a zanalyzovat chování umělé neuronové sítě. Vzhledem k tomu, že umělé neuronové sítě v posledních letech nabývají na popularitě, je porozumění jejich vnitřních procesů zajímavé a důležité téma. Chování uvnitř umělé neuronové sítě připodobňuje takzvaná černá skříňka neboli *black box*. Což znamená, že není možné jednoduše a jednoznačně vyjádřit, co se uvnitř umělé neuronové sítě děje. Naprostá většina populace tyto uměle vytvořené neuronové sítě běžně používá a vzhledem k výše zmíněnému problému černé skříňky toto téma v poslední době nabírá na důležitosti. Pro zpracování tohoto zadání je potřeba se v první řadě seznámit s tím, co je to neuron, umělá neuronová síť, jejich použití a jak se budují. Dále také popsat problém černé skříňky, a hlavně problematiku interpretace umělé neuronové sítě. Dále jaké principy se pro tuto problematiku dají uplatit.

Poté se práce bude zabývat vytvořením a použitím již navržených interpretačních postupů. Tyto postupy budou aplikovány na již vytvořenou a vytrénovanou umělou neuronovou síť. Výsledky těchto interpretačních metod poté budou prezentovány a zkoumány. Následně se bude práce snažit vysvětlit co, proč a jak bylo v dané sítí vyhodnoceno. Nakonec budou porovnávány výsledky interpretačních metod a bude se rozhodovat, zda opravdu napomáhají tomu, že se umělá neuronová síť dá skrze tyto metody interpretovat nebo je možné alespoň nějakým částečným způsobem pochopit její chování.

2. NEURON A NEURONOVÁ SÍŤ

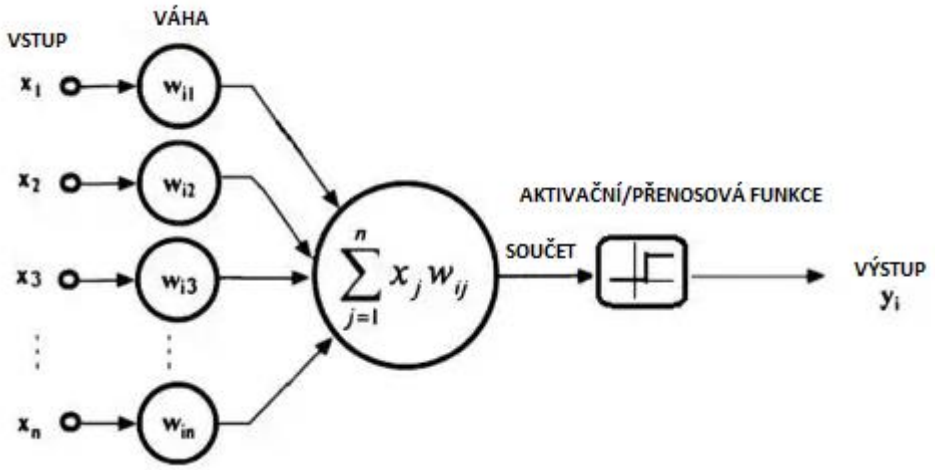
Umělá neuronová síť, též známá jako ANN (artificial neural network) nebo SNN (simulated neural network) je podmnožina strojového učení. Jde vlastně o výpočetní systémy inspirované biologickými neuronovými sítěmi, jako je lidský mozek. Umělé neuronové sítě jsou také jádro hlubokého učení. Podobně jako biologická neuronová síť se umělá neuronová síť skládá z neuronů. Podoba biologického neuronu je přesunuta do matematického prostředí a je z ní tak vytvořen umělý neuron. Jeho struktura je velice podobná biologickému neuronu.

Umělou neuronovou síť lze trénovat pomocí různých algoritmů. Po vyučení lze umělou neuronovou síť používat pro různé úkoly, včetně rozpoznávání obrazu, rozpoznávání řeči, zpracování přirozeného jazyka, a dokonce i hraní her.

Celkově jsou umělé neuronové sítě a umělé neurony mocnými nástroji pro zpracování složitých dat a vytváření predikcí nebo rozhodnutí na základě těchto složitých dat. [1] [2]

2.1. Neuron

Umělý neuron, také známý jako perceptron, je vlastně matematická funkce založená na fungování biologického neuronu. Je to základní stavební jednotka umělé neuronové sítě. Každý umělý neuron přijímá alespoň jeden nebo více vstupů. Všechny tyto vstupy mají nějakou svoji váhu, která váží daný vstup. To znamená, že váha vstupu zdůrazňuje, jak moc daný vstup přispívá a do jaké míry je prospěšný oproti ostatním vstupům. Vstup je tedy vynásoben váhou – nějakým číslem. Je důležité zdůraznit, že váha není jen tak nějaké náhodné číslo, ale je určená tréninkem neuronové sítě. Poté jsou tyto vynásobené vstupy sečteny a vpuštěny do neuronu, kde prochází nějakou nelineární funkcí známou jako aktivační nebo přenosová funkce. [3] [4] [5]



Obrázek 1 - Neuron

Pokud je výsledek aktivační funkce větší než práh umělého neuronu, tak je umělý neuron aktivován, což znamená, že má data předat na výstupu dál. To znamená, že výstup jednoho neuronu se stává vstupem alespoň jednoho dalšího neuronu nebo celkovým výstupem sítě.

Váhy jsou předmětem učení umělé neuronové sítě. Během trénování jsou váhy upravovány tak, aby vedly k co nejpřesnějším výsledkům. Aktivační funkce je vybrána z řady nelineárních funkcí, které pomáhají neuronovým sítím se naučit komplexnější data.

Neurony jsou poté organizovány do vrstev v rámci umělé neuronové sítě, přičemž každá vrstva se typicky skládá z několika neuronů. [3] [4] [5] Jednotlivé složky neuronu jsou:

- Vstup – Jak vychází z názvu, tak se jedná o vstup do neuronu. Po většinou jich je víc než jeden. Každý vstup je násoben vlastní vahou.
- Váha – Je to hodnota, která násobí vstup do neuronu. Čím vyšší hodnota váhy, tím je hodnota daného vstupu důležitější. Váha je získána tréninkem neuronové sítě.
- Bias – Jedná se o systematickou chybu. V umělé neuronové síti ji obsahuje každý neuron ve všech vrstvách kromě první vrstvy.
- Práh – Jde o hodnotu, kterou je potřeba „překročit“, aby byl neuron aktivován a předal hodnoty na výstupu – nebo také jak moc „aktivní“ má na výstupu být.
- Aktivační funkce – Upravuje hodnoty vstupu/vstupů do neuronu. Existuje několik aktivačních funkcí. Mezi nejznámější patří:

    - binární,
    - lineární,
    - sigmoid,
    - tanh,
    - ReLU – tato aktivační funkce má několik podob, kde má každá své výhody a nevýhody,
    - ELU,
    - GELU,
    - SELU,
    - Softmax a další. [6] [7] [8]
- Výstup – Z neuronu vede vždy jen jeden výstup. Výstupní hodnota z neuronu je pak výstupem předávána dál do sítě. [3] [4] [5]

2.1.1. Aktivační funkce

Do aktivační funkce, taktéž nazývané jako přenosová funkce, vstupuje suma vstupů vynásobených o jejich váhy. Aktivační funkce především rozhoduje na základě vnějších podnětů – vstupů do neuronu, zda má být neuron aktivován. Tyto funkce přidávají další krok a komplexnost do neuronových sítí.

Pokud by neurony nepoužívaly aktivační funkce, tak by každý neuron prováděl lineární transformace za použití vah a jejich biasů. Poté by už nezáleželo na tom, zda do umělé neuronové sítě budou přidány další skryté vrstvy a neurony, protože by se všechny vrstvy chovaly stejně. Navíc složení dvou lineárních funkcí dává dohromady lineární funkci. Pro takovou síť by poté bylo nemožné vyřešit jakýkoliv komplexnější úkol.

Aktivační funkce se dají rozdělit do třech typů:

- Binární – Suma vstupů vynásobených vahami vstupujících do aktivační funkce je poté porovnávána s hodnotou prahu. Pokud je suma větší něž práh, tak je neuron aktivován. Pokud je suma menší, neuron aktivován není.

- Lineární – Také známé jako neaktivační funkce. Pracují tak, že jen předají sumu vážených vstupů dál do neuronové sítě. Jak bylo zmíněno výše, pokud všechny vrstvy umělé neuronové sítě budou používat lineární aktivační funkci, tak se vlastně transformují do jedné.
- Nelineární – Tyto funkce dovolují vytváření latentních prostorů, které slouží ke komplexnímu mapování vstupů a výstupů umělé neuronové sítě. Hlavní výhodou je, že tento typ funkcí umožňuje skládání více vrstev do umělé neuronové sítě. Vzhledem k tomu, že funkce jsou nelineární, tak se vrstvy mohou ovlivňovat.

Nejčastěji využívaným typem jsou nelineární aktivační funkce, kde mezi nepoužívanější patří ReLU. Nejlepší aktivační funkce se ale nedá jednoznačně určit, některé se hodí pro řešení některých problémů více než jiné. Zároveň má každá aktivační funkce své klady a zápory. Proto se při vytváření umělé neuronové sítě experimentuje a snaží se najít taková aktivační funkce, která pro řešení daného problému přináší nejlepší výsledky. [6] [7] [8]

2.1.2. Princip fungování neuronu

Základní princip fungování umělého neuronu se dá snadno demonstrovat na jednoduchém příkladu. Představme si, že se nějaká osoba rozhoduje, zda jít na hřiště hrát fotbal. Má tři kritéria na základě kterých se rozhoduje:

- Je hřiště, kam chci jít hrát prázdné? (X1)
- Mám dostatek kamarádů se kterými můžu jít hrát? (X2)
- Je venku hezké počasí? (X3)

Tyto tři kritéria jsou vstupy do neuronu, které pro daný příklad budou nabývat binárních hodnot, kde 1 znamená ano a 0 znamená ne. To znamená, že jsou použity binární aktivační funkce. Zároveň každé z těchto kritérií bude mít pro danou osobu jinou váhu, tedy jedno kritérium bude pro ni důležitější než druhé. Váhy budou rozděleny například takto:

- Jestli je hřiště prázdné, má váhu 3. – (W1)
- Jestli máme dostatek kamarádů, se kterými můžeme jít hrát, má váhu 5. – (W2)
- Jestli je venku hezké počasí, má váhu 1. – (W3)

Aby se osoba nakonec rozhodla jít hrát ven fotbal, součet těchto kritérií vynásobených jejich vahami musí překročit prahovou hodnotu 4 a víc. V tu chvíli nastává případ, že si osoba řekne, že pro ni má cenu jít ven a fotbal si zahrát. Výpočet bude vypadat nějak takto:

$$O = (X1 * W1) + (X2 * W2) + (X3 * W3) - 4 > 0$$

Pokud tedy kritéria budou taková:

- Hřiště je prázdné. X1 = 1
- Mám dostatek kamarádů, se kterými můžu jít hrát. X2 = 1
- Je špatné počasí. X3 = 0

To znamená, že pokud tyto kritéria i s jejich vahami dosadíme do vzorce, tak vyjde $O = 4 > 0$. Vzhledem k tomu, že tato podmínka bude splněna, výstup z tohoto umělého neuronu bude 1, protože osoba půjde ven na fotbalové hřiště zahrát si fotbal. Pokud by ale váhy vstupů nebo práh byly nastaveny jinak, výsledek je možné změnit. Jak můžeme pozorovat z daného příkladu, zapojení více neuronů do sebe, tedy do neuronové sítě, by vedlo k možnosti řešit mnohem složitější problémy.

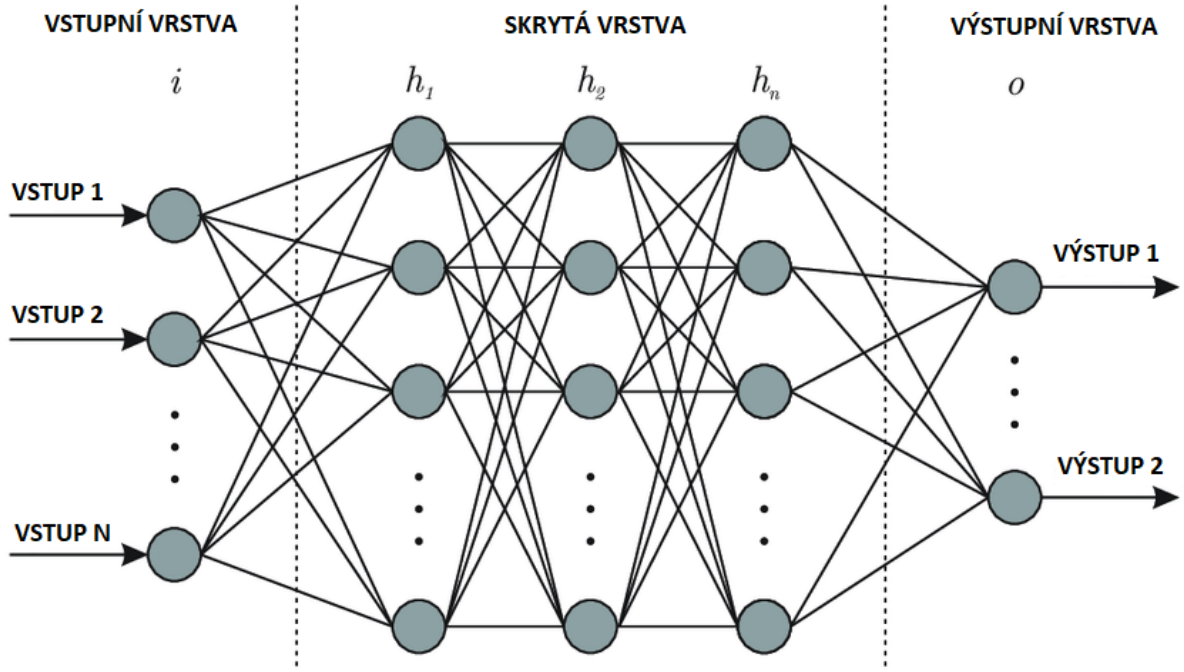
V klasických umělých neurnových sítích se jsou ale využívány sigmoidní neurony, které na svých výstupech vrací hodnoty mezi 0 a 1.

2.2. Umělá neuronová síť a její struktura

Umělá neuronová síť je klasicky tvořena minimálně třemi vrstvami. Tyto vrstvy jsou nadále děleny do třech kategorií:

- vstupní vrstva,
- skrytá vrstva,
- výstupní vrstva.

Nejjednodušší umělá neuronová síť obsahuje tři vrstvy, kde je každý typ vrstvy zastoupen právě jednou. Tedy jedna vstupní vrstva, jedna skrytá vrstva a jedna výstupní vrstva.



Obrázek 2 - Umělá neuronová síť

Vstupní vrstva je typicky první vrstva umělé neuronové sítě, kde vstupují data do neuronové sítě. Tento typ vrstvy je v síti zastoupen pouze jednou. Počet neuronů obsažených v této vrstvě se rovná počtu vstupních dat. Například pokud je vstup obrázek o velikosti 24x24 pixelů, tak by měl být počet vstupních neuronů 24x24, tedy 576 neuronů – každý pixel obrázku reprezentován jedním neuronem.

Další vrstvy umělé neuronové sítě se nazývají jako skryté. Tato vrstva by měla být vždy minimálně alespoň jedna, ale může jich být více. Počet těchto vrstev a počet neuronů v těchto vrstvách není nijak daný. Uživatel spravující neuronovou síť může skryté vrstvy skládat a vytvářet na základě minulých zkušeností nebo u již známých problémů vycházet z osvědčených postupů. Většinou je ale postupováno podle intuice a neuronová síť je postupně přizpůsobována a upravována tak, aby vracela co nejlepší výsledky. Skryté vrstvy mohou obsahovat tisíce nebo i statisíce neuronů. Zde tedy nastává problém černé skříňky, kde nejsme schopni s jistotou říct co, jak a proč se mezi neurony a jejich vrstvami děje, proč jsou jejich váhy nastaveny na určité hodnoty a podobně.

Poslední vrstvou je tedy vrstva výstupní, která je v neuronové síti zastoupená také pouze jednou. Z této vrstvy a jejich neuronů vystupují výsledná data. Pokud je například

očekáváno, že má umělá neuronová síť na výstupu vrátit souřadnice X a Y, bude poté obsahovat pouze dva neurony, kde každý jeden neuron reprezentuje jednu souřadnici – souřadnici X a souřadnici Y. [9] [10]

2.2.1. Typy umělých neuronových sítí

Existuje několik typů umělých neuronových sítí. Zpravidla se odlišují složitostí a směrem, jakým jsou data v umělé neuronové síti distribuovány a strukturou. Poté se také mohou lišit propojením jednotlivých neuronů. [11]

Mezi pět typů umělých neuronových spadají:

- dopředné umělé neuronové sítě,
- perceptron a vícevrstvé perceptonové umělé neuronové sítě,
- umělé neuronové sítě s radiální bázovou funkcí,
- rekurentní neuronové sítě,
- modulární neuronové sítě. [11]

2.2.1.1. Dopředné umělé neuronové sítě

Jak lze vyvodit již z názvu, v dopředné umělé neuronové síti jsou data distribuována směrem dopředu – tedy ze vstupní vrstvy až do výstupní vrstvy. Poté se také mohou lišit propojením jednotlivých neuronů – každý jeden neuron v jedné skryté vrstvě nemusí být spojen s každým umělý v další skryté vrstvě. I přesto, že se může skládat z více skrytých vrstev a tyto vrstvy mohou obsahovat spousty neuronů, tak je tento typ umělé neuronové sítě používán pro hlavně pro jednoduché klasifikační problémy. [11] [12]

2.2.1.2. Perceptron a vícevrstvé perceptronové umělé neuronové sítě

Perceptron je nejstarší a nejjednodušší model umělé neuronové sítě. Je to vlastně jen jeden neuron, který funguje jako binární klasifikátor. Rozděluje tedy vstupní data do neuronové sítě do dvou různých skupin – klasifikací.

Zato typ vícevrstvé umělé neuronové sítě zvyšuje komplexitu, a tím pádem i možnost pro řešení složitějších problémů, jako je složitá klasifikace nebo rozpoznání hlasu. Komplexitu nabírá tím, že je možné mezi vstupní a výstupní vrstvu přidat libovolný počet skrytých vrstev s libovolným počtem umělých neuronů. Neurony ve všech vrstvách jsou vzájemně

propojeny. To znamená, že jeden neuron je propojen se všemi neurony další vrstvy. Tyto sítě jsou tedy plně propojené a dají se využít pro hluboké učení.

Jedná se o nejpoužívanější typ umělé neuronové sítě. [11] [13]

2.2.1.3. Umělé neuronové sítě s radiální bázovou funkcí

Tento typ umělé neuronové sítě se skládá ze tří vrstev – vstupní vrstva, vrstva s neurony, které používají funkce radiální báze a výstupní vrstva. Funkce radiální báze vypočítává absolutní hodnotu mezi střední hodnotou a danou hodnotou. Tento typ umělé neuronové sítě je možné využívat pro klasifikaci, časové řady nebo kontrolní systémy. Nejčastěji je používán pro kontrolní systémy, jako je například systém pro obnovení proudu, kde umělá neuronová síť dokáže rozeznat, kde je po obnově proudu proud nejdůležitější. [11] [14]

2.2.1.4. Rekurentní neuronové sítě

Rekurentní neuronové sítě zpracovávají data postupně, podobně jako dopředné umělé neuronové sítě, ale po jejich zpracování v dané vrstvě se vrátí zpracovaná data zpět za účelem zlepšení predikcí. Paměť výstupů z vrstvy je cyklována zpět na vstup, kde je držena, aby se zlepšil proces predikce pro další vstup. Díky tomuto principu je tento typ umělé neuronové sítě schopen lépe porozumět kontextu vstupu a je poté schopen zpřesnit predikci výstupu. Tento model je nejčastěji používán pro práci s jazykem, kde je například možné použít předchozí slovo k predikci dalšího slova v kontextu řetězce nějakých slov. Dále se také používá pro chatboty nebo shrnutí dokumentů. [11] [15] [16]

2.2.1.5. Modulární neuronové sítě

Jedná se o velmi zajímavý a výkonný typ umělé neuronové sítě. Modulární neuronové sítě se skládají z několika částí – modulů. Tvoří ho několik umělých neuronových sítí, které si rozdělí složitější úkol na menší části, kde na něm pak jednotlivé moduly pracují paralelně. Tento proces může značně urychlit práci na daném úkolu. [11] [17]

2.2.2. Parametry umělé neuronové sítě

Umělá neuronová síť obsahuje spoustu základních parametrů, které je potřeba k jejímu fungování nastavit. Mezi základní parametry, které je potřeba nastavit, spadá: [18] [19]

- Počet skrytých vrstev – je potřeba se rozhodnout, kolik skrytých vrstev bude použito.

- Počet neuronů – počet neuronů ve skryté vrstvě. Každá skrytá vrstva může mít jiný počet neuronů.
- Aktivační funkce neuronů – jak již bylo zmíněno v kapitole 2.1.1., každý neuron používá svou vlastní aktivační funkci. Aktivační funkce je tedy jedním z parametrů umělé neuronové sítě. Ve většině případů se pro jednu vrstvu umělé neuronové sítě používá jedna aktivační funkce pro všechny její neurony. Jednak z důvodu jednodušší implementace, ale také z důvodu jednodušší interpretace. Není ale vyloučeno použít více aktivačních funkcí v jedné vrstvě.
- Rychlost učení (learning rate) – jde o rychlost, s jakou umělá neuronová síť upravuje váhy neuronů během tréninku. Čím větší je rychlost učení, tím rychleji se váhy mění a naopak. Pokud je ale použita příliš velká rychlost, tak je možné, že během tréninku je přeskočen optimální bod řešení. Pokud je rychlost učení moc malá, je možné, že model dojde k suboptimálnímu řešení. Proto také existují optimalizační algoritmy, které upravují rychlost učení během tréninku. Mezi takové algoritmy patří Adam, Adagrad nebo RMSprop.
- Počet epoch – kolikrát bude síť iterovat přes celý tréninkový set dat během tréninku.
- Velikost dávky (batch size) – jedná se o parametr, kde se zvolí, kolik trénovacích dat bude zprocesováno, než se upraví váhy. Je to z toho důvodu, že celý set dat je moc veliký na to, aby byl zpracován celý najednou. Například pokud bude velikost dávky nastavená na 10, tak umělá neuronová síť zpracuje 10 testovacích dat a poté aktualizuje váhy neuronů. Tento proces je opakován do té doby, dokud není zpracován celý data set.
- Optimalizační algoritmy (optimizers) – jsou to algoritmy, které mají rozhodující roli při úpravě vah v umělé neuronové síti. Tyto algoritmy vlastně vnáší inteligenci do umělé neuronové sítě. Jsou blízce spojeny s rychlostí učení, díky níž na základě její hodnoty upravují hodnoty vah. Většina těchto algoritmů pracuje s fixní hodnotou rychlosti učení. Mezi ně patří algoritmy jako Stochastic Gradient Descent (SGD), Gradient Descent (GD) a další. Jak již bylo zmíněno u parametru rychlosti učení, existují také algoritmy, které pracují s rychlostí učení a jsou schopny si tuto

hodnotu upravovat, aby byla co nejideálnější. Sem spadají optimalizační algoritmy jako Adam, RMSprop a podobně.

- Ztrátová funkce (loss function) – ztrátová funkce v umělé neuronové síti určuje rozdíl mezi predikovaným výstupem a požadovaným (pravým) výstupem. Cílem tréninku umělé neuronové sítě je minimalizovat hodnotu této funkce na minimum, protože čím menší hodnota, tím přesnější je predikovaný výstup. Ztrátová funkce je konfigurovatelná z toho pohledu, že je potřeba vybrat správný typ funkce na základě toho, jaký problém model sítě řeší. Pro regresní problémy se může použít průměrná kvadratická chyba (MSE – mean squared error) nebo průměrná absolutní chyba (MAE – mean absolute error). Pro klasifikaci zase křížová entropie (CE – cross entropy). Existují ale i další funkce, které se dají používat. [18] [19]

Nakonec je potřeba dodat, že nikde není dáno, jaké je ideální nastavení parametrů umělé neuronové sítě. Každý jeden z těchto parametrů má dopad na výkonnost konečného modelu umělé neuronové sítě. Dále se také tyto parametry navzájem ovlivňují. Stačí upravit jeden z parametrů a výkonnost sítě se může drasticky změnit. A to, že se ovlivňují je vlastně logické, protože některé parametry jsou na sobě přímo závislé.

Například rychlost učení a optimalizační algoritmus spolu přímo spolupracují. Optimalizační algoritmus upravuje váhy neuronů, které vlastně ovlivňují neuron. Neuron je poté ovlivňován sumou vstupů jiných neuronů, dále také svou aktivační funkcí. Takto by se dalo pokračovat dál. Pokud je tedy poté brán v potaz problém interpretace, tak je z toho jasně zřetelné, že když všechno ovlivňuje všechno, tak je už jen na úrovni parametrů složité umělou neuronovou síť interpretovat. Složitost interpretace se poté ještě zvětšuje, pokud se snažíme vysvětlit co a hlavně proč jeden jednotlivý neuron dělá to, co dělá.

Při vytváření umělé neuronové sítě a vybírání jejích parametrů je často nejlepším postupem experimentovat s různými nastaveními a zjistit, co pro daný problém, který konkrétní umělá neuronová síť řeší, funguje nejlépe. Navíc je důležité mít na paměti, že optimální nastavení pro jeden problém nemusí být nejlepší řešení pro jiný problém. [18][20][42][43]

3. PROBLEMATIKA ČERNÉ SKŘÍŇKY

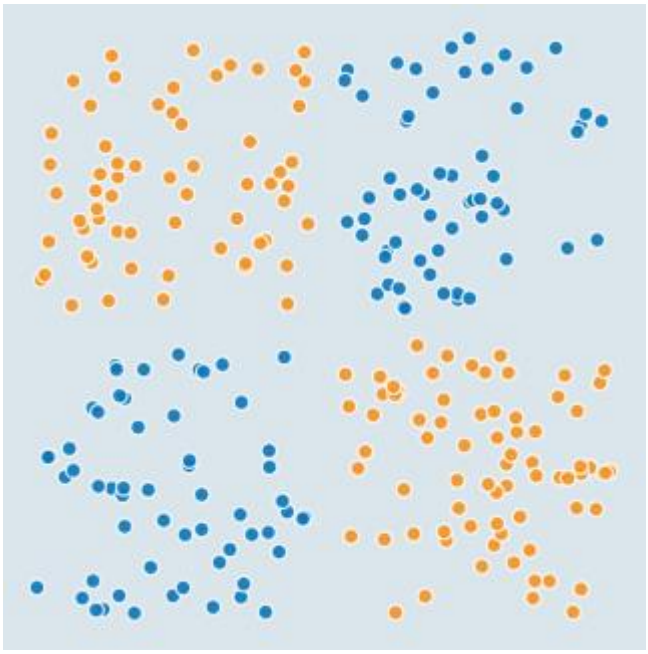
Jak již bylo naznačeno v předchozí kapitole, složité umělé neuronové sítě jsou velice komplexní na to, aby se v nich dalo jednoznačně určit, co se v nich děje. Problematiky, kde nedokážeme učit, jaký děj uvnitř probíhá, nazýváme černá skříňka nebo také black box. Jak ale může být něco černá skříňka, pokud jsme schopni do toho nahlédnout a máme k dispozici kód? Proč jsou tedy vlastně hluboké umělé neuronové sítě označovány jako černé skříňky?

Na základní úrovni hlubokých umělých neuronových sítích jde o to, že nejde říct, jak jednotlivé neurony spolupracují a jak se dopracují k danému výstupu. Velice často pak není ani jasné, co každý jeden neuron sám o sobě dělá. Jak již bylo řečeno a vysvětleno v předchozí kapitole, každý neuron má nějaké vstupy, které vynásobí váhou, kterou se naučil v tréninku, vstupy se po vynásobení sečtou a vypočítá se výstup podle aktivační funkce. Tento proces je možné zmapovat. Problém černé skříňky nastává tehdy, pokud se bude pokoušet porozumět tomu, jak mezi sebou více neuronů vzájemně spolupracuje. [21]

3.1. Černá skříňka v neuronových sítích

Černá skříňka a problém interpretace spolupráce mezi neurony se dá demonstrovat na jednoduchém příkladu. Mějme tréninkový set dat obsahující dva typy bodů – oranžové a modré.

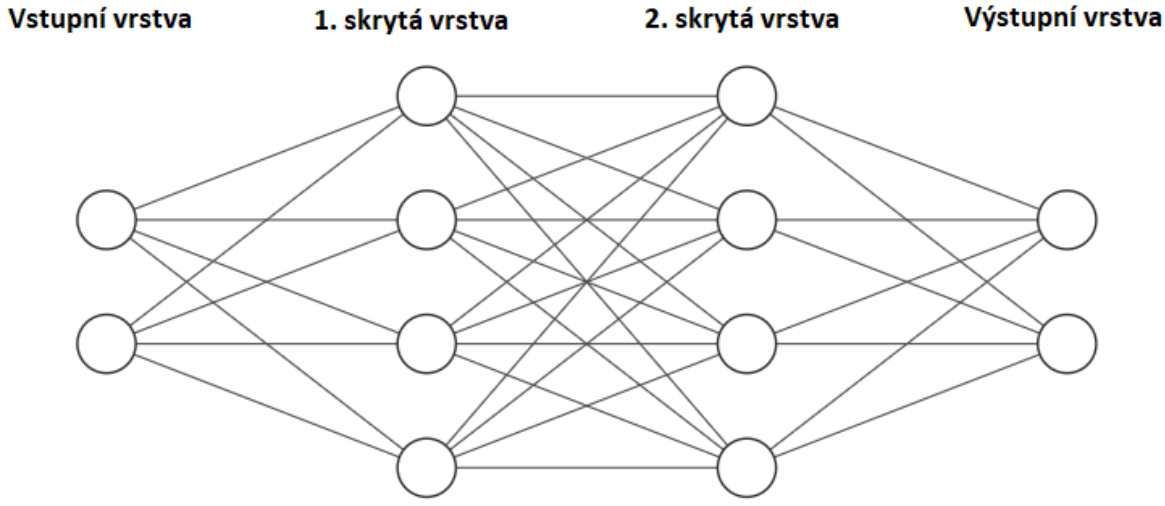
K simulování problému černé skříňky bylo využito TensorFlow playground. Jedná se o program umožňující vytvoření jednoduché neuronové sítě v prohlížeči, kde se dají nastavit základní vlastnosti neuronové sítě a vybrat set dat, na kterých bude umělá neuronová síť vytrénována a poté i testována. TensorFlow playground využívá knihovnu speciálně vyvinutou a upravenou, která vychází z knihovny TensorFlow. TensorFlow je knihovna pro strojové učení, která byla navržena společností Google. Všechny obrázky, kromě obrázku čtyři, ve zbytku této kapitoly byly pořízeny za pomoci TensorFlow playground.



Obrázek 3 – Tréninkový data set [22]

Jak již bylo řečeno, k dispozici je tréninkový set dat obsahující dva typy bodů – oranžové a modré. Každý bod je reprezentován hodnotou souřadnice X a Y, kde je bod umístěn. Umělá neuronová síť, na kterou byl použit trénovací data set, byla vytvořena následovně:

- vstupní vrstva – dva neurony,
- skrytá vrstva – čtyři neurony,
- skrytá vrstva – čtyři neurony,
- výstupní vrstva – dva neurony.

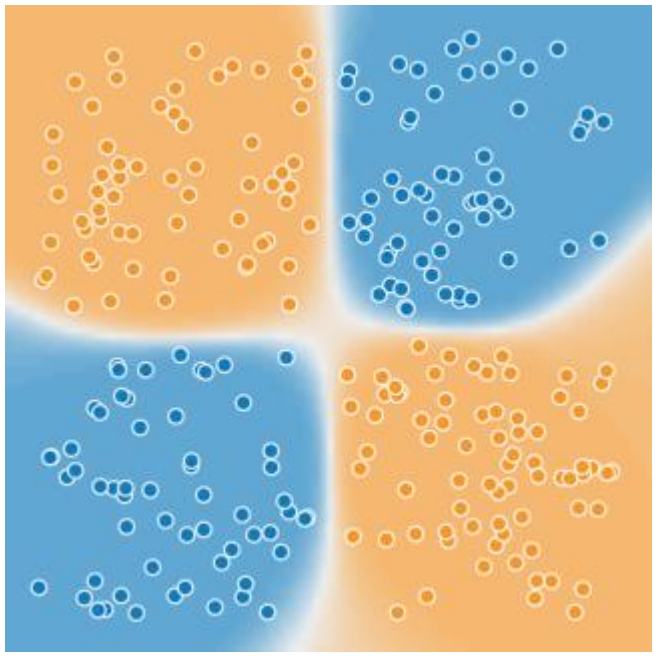


Obrázek 4 – Neuronová síť [23]

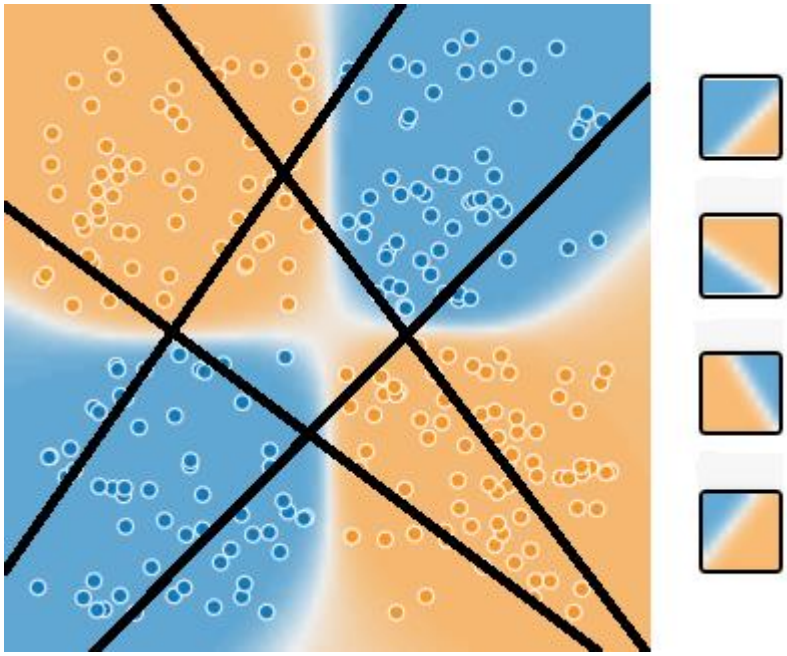
Atributy umělé neuronové sítě byly nastaveny takto:

- rychlost učení – 0.03,
- aktivační funkce neuronů – Tanh,
- počet epoch – 300.

Po dokončení tréninku neuronové sítě se dá konstatovat, že neuronová síť dokáže velice dobře predikovat, kam jaký bod bude patřit, viz obrázek pět.

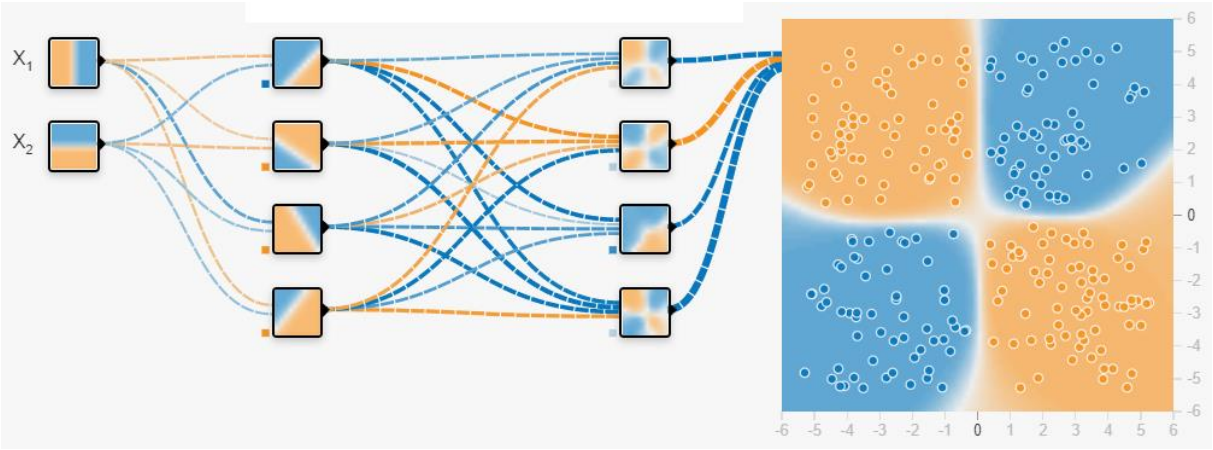


Obrázek 5 – Vytrénovaná neuronová síť [22]



Obrázek 6 – Rozhodovací hranice první skryté vrstvy [22]

Pokud se ale podíváme na obrázek šest, kde jsou černou čarou vyznačeny rozhodovací hranice, které si určily první čtyři neurony, tedy první skrytá vrstva, je možno pozorovat, že nedávají tak zcela smysl. Tyto rozhodovací hranice ne zcela odrážejí realitu toho, jak je daný data set rozdělen. To je to, co je označováno jako černá skříňka. Černou skříňkou se dají označovat tato zvláštní rozhodnutí neuronů, které se mohou zdát jako nepotřebné nebo možná dokonce zmatečné v cestě ke konečnému výstupu neuronové sítě. Není to tedy velice komplexní a složitá nelineární matematika, ale tyto rozhodnutí. Pokud by se nahlédlo na celý model této umělé neuronové sítě, tak je v druhé skryté vrstvě již jasně vidět, že neurony obsažené v ní už mají rozhodovací hranice mnohem přesněji a více logicky nastavené.



Obrázek 7 – Celý model vytrénované neuronové sítě [22]

Tento příklad se týkal pouze problému jednoduché klasifikace a rozhodovací hranice nebyly v první skryté vrstvě neuronové sítě až tak extrémní, jak by mohly být. Když by proces trénování byl nastaven na menší počet epoch, menší počet neuronů ve skrytých vrstvách nebo menší počet trénovacích dat, mohly být tyto hranice extrémnější. Pokud by se řešila nějaká mnohem složitější klasifikace – například rozpoznávání obrazu a zjišťování, jaké na něm je zvíře nebo pokud by umělá neuronová síť řešila problémy regrese, tak by první vrstvy neuronové sítě fungovaly pro pozorovatele ještě mnohem víc nelogicky, než tomu bylo v tomto příkladu.

3.2. Jak černá skříňka vzniká

V hlubokých umělých neuronových sítích černá skříňka vzniká už při trénování. Ve výše uvedeném případě se neurony v první skryté vrstvě trénují, jak převést hodnoty vstupní vrstvy do nějakého latentního prostoru. Neurony v další skryté vrstvě jsou trénovány tak, aby se data z latentního prostoru daly převést na výstupní data.

Vzhledem k tomu, že ale první neurony nevědí, co neurony po nich budou dělat a k čemu je celá neuronová síť jako celek určená, tak se pokouší vytvořit takový latentní prostor, který je užitečný pro cokoliv, co by mohly hlubší neurony dělat. Ty neurony to nemohou tušit z toho důvodu, že se všechny trénují najednou a na začátku začínají s náhodně nastavenými vahami.

Problém pokračuje tím, že hlubší neurony dostanou nějaký univerzální latentní prostor, se kterým se musí naučit pracovat, aby mohly poskytovat srozumitelné výstupy. Tyto výstupy ale musí být dost univerzální na to, aby s nimi mohla další vrstva jakkoli pracovat.

Každý neuron se tedy vlastně pokouší dělat svoji práci, ale je ovlivněn výkonem ostatních neuronů. Jinak řečeno, každý neuron dělá trošku všeho, aby byl schopen vyrovnat se s výstupem, který dostává z předchozí vrstvy. Proto žádný neuron nemá jasně danou funkci, i když se to tak může na první pohled zdát.

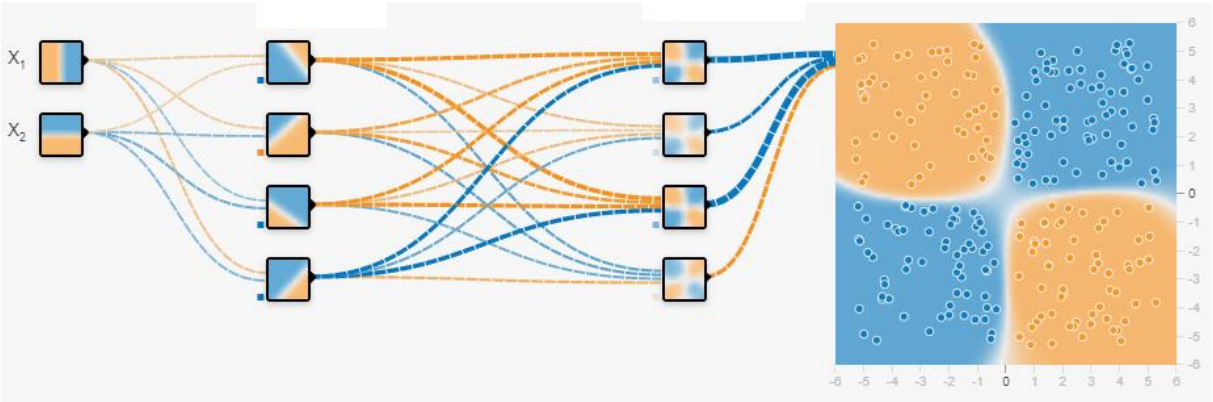
Jak můžeme vidět na obrázku sedm, druhá skrytá vrstva se vlastně pokouší kompenzovat nedostatky neuronů z první skryté vrstvy, tím pádem vznikne velice složitá síť závislostí, která se nedá vysvětlit.

Nevysvětlitelnost funkce a závislostí skrytých neuronů, která je produktem zvláštních rozhodovacích hranic je také zodpovědná za útoky na tyto umělé neuronové sítě, které se dají poté jednoduše obelstít. [21]

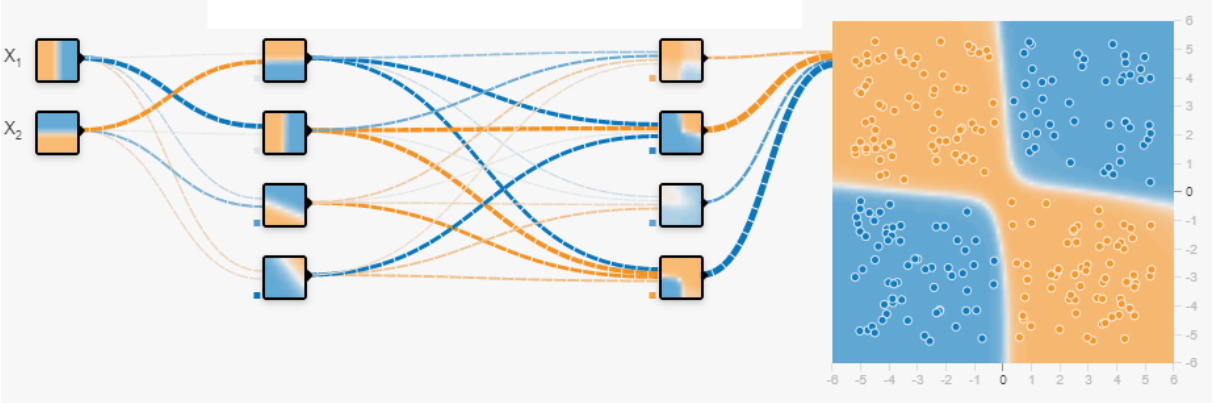
3.3. Vztah tréninkových dat a černá skříňky

Tréninková data mají obrovský vliv na celkovou funkci umělé neuronové sítě. Díky nim jsou upravovány váhy, které upravují, jak každý neuron pracuje. Je to něco, co vnáší inteligenci do umělé neuronové sítě.

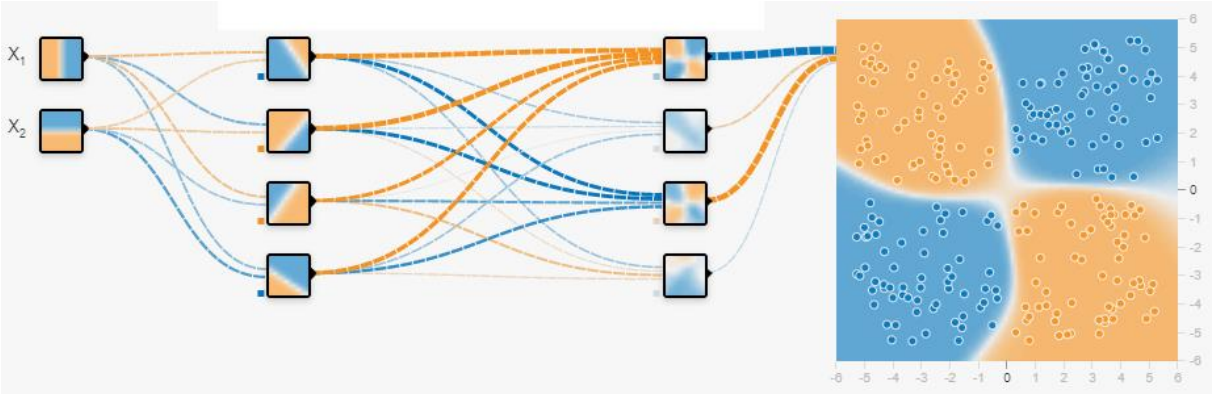
Ve výše uvedeném příkladu jsou tréninková data rozprostřena v celém poskytnutém prostoru a v učícím procesu jsou konzumovány náhodně. Nefunguje to tak, že je definováno, že teď půjdou v tréninkových datech všechny body, co mají oranžovou barvu a jsou z levého horního rohu. V tréninkových datech nelze tedy najít žádnou strukturu. Na obrázcích osm, devět a deset je možné pozorovat, jak jiná tréninková data ovlivňují fungování celé umělé neuronové sítě.



Obrázek 8 – První model ovlivněný jinými trénovacími daty [22]



Obrázek 9 – Druhý model ovlivněný jinými trénovacími daty [22]



Obrázek 10 – Třetí model ovlivněný jinými trénovacími daty [22]

Ve všech třech modelech byla použita jiná tréninková data a všechny ostatní vlastnosti a atributy neuronové sítě byly ponechány stejně. Je možné si všimnout, že všechny rozhodovací hranice v prvních skrytých vrstvách jsou jiné. V druhých skrytých vrstvách se již začínají podobat, ale stále nejsou stejné. Toto vše ovlivnila tréninková data i přesto, že byla velice podobná a principiálně úplně stejná.

Nesmí být opomenuto, že umělé neuronové sítě a proces jejich trénování je složitý proces, který je ovlivnitelný nejenom trénovacími daty, ale i dalšími faktory. Proces trénování může být ovlivněn také:

- aktivačními funkcemi v neuronech,
- rychlostí učení – learning rate,
- počtem iterací učení – počtem epoch,
- počtem skrytých vrstev,

- počet neuronů atd.

Dalo by se říct, že umělá neuronová síť je jeden velký organismus, kde všechno navzájem ovlivňuje ostatní věci. Proto je i velice důležité, jak k tvorbě této sítě přistupuje její tvůrce. Ten vlastně nakonec definuje, co umělá neuronová síť bude dělat, jak se bude chovat, a tím pádem může výrazně přispět k tomu, aby výsledky odpovídaly tomu, co budeme očekávat. [21]

4. INTERPRETACE UMĚLÉ NEURONOVÉ SÍTĚ

Umělé neuronové sítě slaví v posledních letech velké úspěchy například v rozpoznávání jak už mluveného slova, obličeje nebo hlasu, tak i v medicíně a dalších oblastech. S tímto velkým úspěchem ale také roste zájem o jejich nepříjemnou vlastnost černé skříňky. Jak bylo v předchozí kapitole popsáno, problematika černé skříňky rozhodně není žádoucí vlastnost. Nemožnost vysvětlit a jasně definovat co, jak a proč se uvnitř umělé neuronové sítě děje, ovlivňuje důvěru lidí v ně. Zde přichází na řadu snahy o interpretaci umělé neuronové sítě, obor, který má za úkol popsat, jak umělá neuronová síť funguje uvnitř. Interpretace umělých neuronových sítí se týká pochopení a vysvětlení chování či rozhodovacích procesů. To může také například zahrnovat identifikaci, která vstupní data mají největší dopad na výstupy modelu nebo pochopení toho, jak model vytváří své predikce. Dále může pomoci vysvětlit výkon modelu umělé neuronové sítě.
Mimo to, možnost interpretace je chtěná vlastnost, která může umělým neuronovým sítím pomoci k tomu, aby se staly ještě silnějšími nástroji, než jsou doposud. Pokud bude možné interpretovat, umělé neuronové sítě bude možné používat i v dalších odvětvích, jako je výzkum léků nebo genomika.

4.1. Proč interpretace

Momentálně největší vytvořenou a fungují umělou neuronovou sítí je GPT-3, která pracuje se 175 miliardami parametrů a potřebuje přes 800 gigabitů úložiště [24][25]. I přesto, že tyto sítě dokážou svoji práci provádět neuvěřitelně dobře, je možné je nějakým způsobem ošálit, jak ukázal A. Nguyen při rozpoznávání obrázků [26]. I přesto, že jsou umělé neuronové sítě schopny dosahovat extrémní výkonnosti a přesnosti, jejich vnitřní mechanismy se mohou velice lišit od těch lidských a je velice složité jim porozumět. O větší pochopení těchto procesů usiluje interpretace, které má do černé skříňky vnést jisté světlo a pomoci nahlédnout a vysvětlit chování umělých neuronových sítí.

Interpretovatelnost umělé neuronové sítě je obzvláště důležitá v oblastech jako medicína nebo samořídící auta, kde musí být zaručena spolehlivost modelu a správné vlastnosti. [27]

4.2. Potřeba a důležitost interpretace

Potřeba a důležitost interpretace je jasná a její nedostatek může mít v některých případech škodlivé, až katastrofální účinky nebo následky. I přesto, že není interpretace jako taková nikde jasně popsána a strukturována, tak Y. Zhang ve svém průzkumu [27] rozdělil podle důležitosti a potřeby interpretace do tří skupin:

1. vysoké požadavky na spolehlivost,
2. etické a právní požadavky a
3. vědecké využití.

4.2.1. Vysoké požadavky na spolehlivost

Jak již bylo zmíněno, i přesto, že umělé neuronové sítě dosahují vysoké spolehlivosti, prostředí reálného světa je mnohem složitější, a proto jsou některé neočekávané chyby umělých neuronových sítí nevyhnutelné. U těchto chyb je potřeba, aby byly k dispozici nějaké prostředky, skrze které by umělá neuronová síť byla i přes tyto chyby pod kontrolou. Umělé neuronové sítě ale žádné takové prostředky nemají a jejich uživatel nad nimi tedy nemá žádnou kontrolu. Jak Y. Zhang [27] ve své práci píše: „Hluboké neuronové sítě takovou možnost neposkytují. V praxi bylo často pozorováno, že v určitých situacích mají neočekávaný pokles výkonu, nemluvě o potenciálních útocích“. Dále také zmiňuje, že interpretace jako taková není potřeba pokaždé, ale je velice důležitá pro predikční systémy, ve kterých chyba může způsobit katastrofální následky, jakou jsou ztráty na životech nebo velké finanční ztráty. Tedy u modelů umělých neuronových sítí, kde jsou opravdu vysoké požadavky na spolehlivost.

Jinými slovy interpretace může tyto potenciální chyby jednodušeji detekovat a model umělé neuronové sítě může být upraven tak, aby se těmto katastrofálním následkům předešlo. Důležité je si také uvědomit, že interpretace neudělá umělou neuronovou síť spolehlivější nebo snad výkonnější. Její hlavní výhoda tkví hlavně v definici umělé neuronové sítě a určení potenciálních míst pro zlepšení nebo opravu. [27]

4.2.2. Etické a právní požadavky

Tato část, jak z názvu vyplývá, se rozděluje na dvě části. První, etická část, se zabývá algoritmickou diskriminací (alghoritmic discrimination). Algoritmická diskriminace může nastat u automatizovaných systémů, kde tyto systémy mohou bezdůvodně rozdílně

pracovat v podstatě identických situacích nebo mohou mít diskriminující dopady vůči lidem na základě pohlaví, barvy pleti, náboženského vyznání a podobně [28] [29] [30]. Toto se úzce váže na kapitolu 3.3., kde jsou na vině převážně tréninková data, která vytrénují umělou neuronovou síť tak, aby diskriminovala určité typy lidí na základě nějakých jejich rysů. Model umělé neuronové sítě potom nerozlišuje, jestli je to tak správně, nebo ne. Model byl vytrénován tak, aby určitý typ lidí na základě nějakého jejich atributu diskriminoval, a je to tak podle něj správně. Jak bylo řečeno, tento *bias* byl do sítě zaveden už v tréninkových datech. Toto se může například projevit při vyhodnocování pojištění nebo hypotéky, kde jsou umělé neuronové sítě běžně používány. [31] Díky interpretaci by poté bylo možné tyto problémy odhalit.

Druhá část pojednává o právních požadavcích, zejména GDPR (ochrana osobních údajů) a o jeho článku 22 [32], který se zabývá automatizovaným individuálním rozhodováním včetně profilování. Tento článek říká: „Subjekt má právo nebýt předmětem žádného rozhodnutí založeného výhradně na automatizovaném zpracování, včetně profilování, které má pro něj právní účinky nebo se jej obdobně významně dotýká.“ Jak Y. Zhang ve své práci uvedl, pokud nemáme ponětí, jak se umělá neuronová síť rozhoduje, tak není možné tato práva zaručit. [27]

4.2.3. Vědecké využití

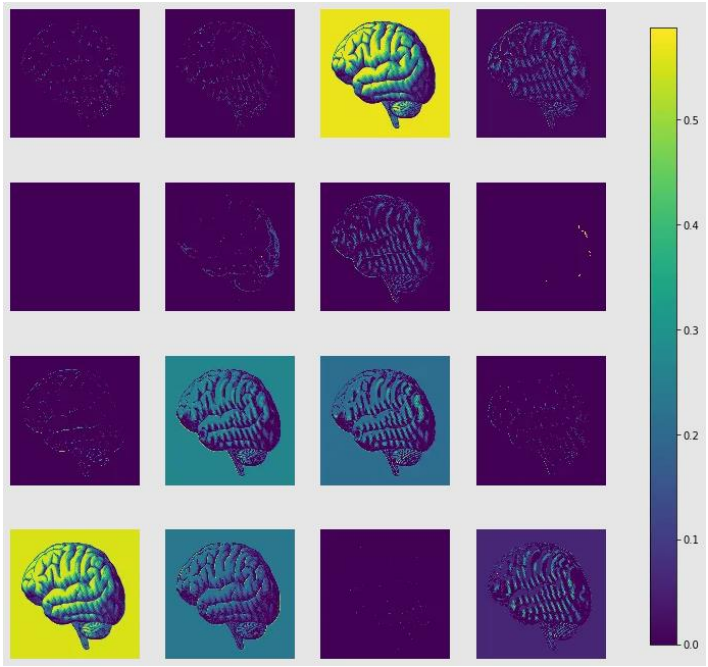
Umělé neuronové sítě se stále víc prosazují v oblastech vědeckých výzkumů, kde pracují s daty, která obsahují složité vnitřní vzorce. Jedná se například o oblast fyziky, sociálních věd nebo genomiky. Pokud nové modely umělých neuronových sítí dosahují lepších výsledků než ty starší modely, musely najít nějaké nové znalosti v daném oboru. Interpretace může tedy v tomto případě hrát velikou roli a může fungovat jako způsob, jak tyto nové znalosti odhalit. [27]

4.3. Způsoby interpretace

Existuje spousta způsobů, jak interpretovat umělou neuronovou síť. Mezi tyto způsoby se například řadí:

- Mapy aktivací (Activation maps) – jak z názvu vyplývá, tak mapy aktivací se používají k vizuální reprezentaci aktivací neuronů v umělých neuronových sítích. Tyto mapy ukazují, jak moc jsou které neurony „aktivovány“ v dané vrstvě. Mapy aktivací mohou pomoci porozumět tomu, jak umělá neuronová síť pracuje se

vstupy. Můžeme díky nim vidět, které neurony jsou aktivovány při predikcích. Pokud by se například rozpoznávalo, co model vidí na obrázku, mohou nám zvýraznit oblasti obrázku, které jsou pro predikci důležité. [33]

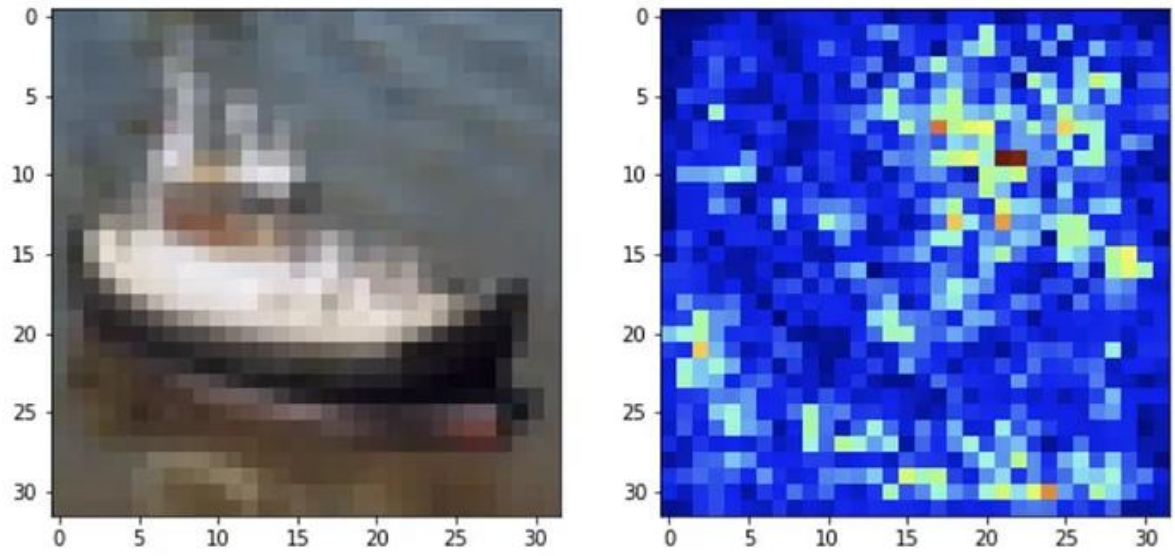


Obrázek 11 – Aktivační mapy jedné vrstvy umělé neuronové sítě, která se snaží rozpoznat obrázek mozku [33]

- Pozornostní mapy (Attention maps) – pozornostní mapy se používají k vizualizaci míst vstupních dat, která jsou zajímavá pro vytváření predikcí. Umožňuje to tedy pochopit, které části dat jsou pro model a vytvoření nějaké predikce důležité. Svoje uplatnění mají hlavně v rámci komplexních modelů, kde jsou vstupní data více dimenzionální, a tím pádem se těžko interpretují. Jedná se například o modely počítačového vidění (computer vision) nebo zpracování přirozeného jazyka (natural language processing). [39]

- Mapy teploty (Heat maps) - mapa teplot je vizuální technika, která umožňuje zobrazit důležitost různých částí vstupního obrazu nebo datového vzorku pro rozhodnutí neuronové sítě. Každému pixelu nebo rysu vstupního obrazu se přiřadí barva na základě jejich úrovně důležitosti. Například při klasifikaci obrazů může mapa teplot ukázat, které pixely nebo regiony obrazu byly nejvíce vlivné pro klasifikační rozhodnutí neuronové sítě. [33] [41]

- Mapy síly vlivu (Saliency maps) – podobně jako pozornostní mapy mají za úkol identifikovat, která část vstupních dat nejvíce ovlivňují predikce. S tím rozdílem, že cílem těchto map je vizualizovat oblasti vstupu, které jsou pro danou předpověď nejdůležitější. Například při rozpoznávání obrázku se snaží ukázat, které části nejvíce přispívají k předpovědi modelu. [34] [38]



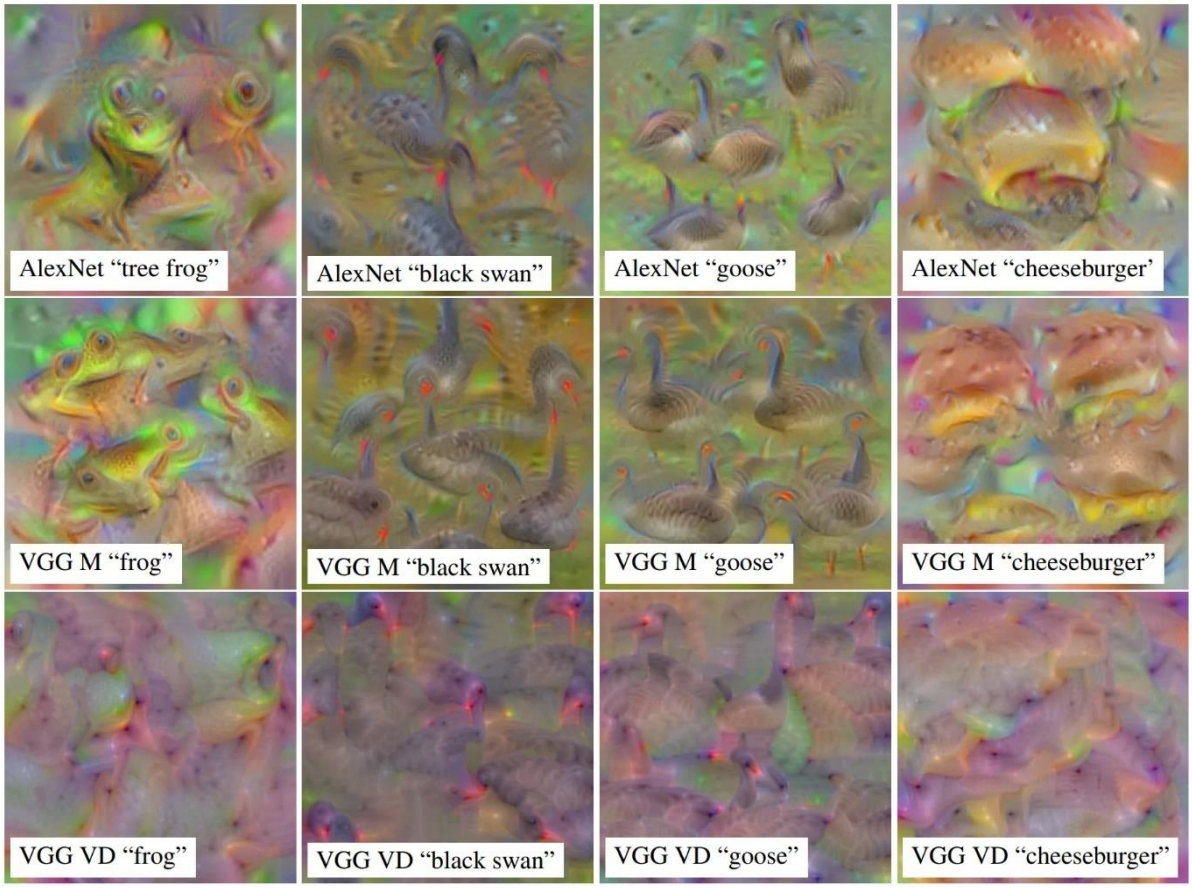
Obrázek 12 – Mapa síly vlivu při rozpoznávání lodi [34]

- Vrstvení šíření významnosti (Layer-wise Relevance Propagation – LRP) – LRP vstupním hodnotám přiřazuje určité skóre, které má za úkol reprezentovat to, jak moc daný vstup ovlivnil predikci. Ten propaguje skóre zpět skrze celou síť. Funguje tak, že šíří skóre od výstupu ke vstupu skrze všechny vrstvy. Tento způsob interpretace se používá nejenom k pochopení rozhodovacího procesu, ale také pro detekci zkreslení (biasu) nebo chyb v předpovědích. [40]

- Destilace modelu (Model distillation) – technika, kde je menší a jednodušší model umělé neuronové sítě vytvořen z většího a složitějšího modelu. Ten menší model je poté trénován tak, aby co nejvíce napodobil predikce toho většího modelu. Menší model je poté jednodušší interpretovat.

- Extrakce modelu (Model extraction) – extrakce modelu je převzetí vytrénovaného modelu, který je transformován do menšího modelu na subtilnější a kompaktnější model, který bude jednodušší interpretovat a analyzovat.

- Maximalizace aktivace (Activation maximization) – cílem maximalizace aktivace je porozumět vstupu, který maximalizuje aktivaci daného neuronu nebo skupině neuronů, díky čemu je poté možné porozumět, čemu dává umělá neuronová síť pozornost. Toho se dosáhne upravením vstupu tak, aby byla aktivační funkce toho daného neuronu nebo té skupiny co největší. [37]



Obrázek 13 – Příklad obrázků po použití maximalizace aktivace na modely umělých neuronových sítí AlexNet, VGG M a VGG VD [37]

- Citlivost na překrytí (Occulsion sensitivity) – jedná se o zakrývání části vstupních dat a pozorování predikcí modelu. Díky této metodě se dá zjistit, jak moc je daná zakrytá část důležitá v procesu rozhodování a jak moc ovlivní výslednou predikci modelu. [35]

Obrázek 14 – Příklad obrázků pro použití metody interpretace citlivost na překrytí [35]

- CAM (Class Activation Mapping) - je interpretační metoda v neuronových sítích, která umožňuje vizualizovat oblasti vstupního obrazu, které byly klíčové pro rozhodnutí sítě. Tato metoda je obzvláště užitečná pro vizualizaci, které oblasti vstupu byly nejvíce aktivní pro rozhodnutí o konkrétní třídě. [41]
- Grad-CAM (Gradient-weighted Class Activation Mapping) – jedná se o pokročilejší verzi metody CA. Podobně jako CAM, Grad-CAM umožňuje vizualizaci klíčových oblastí vstupního obrazu pro rozhodnutí neuronové sítě, ale využívá jiný způsob výpočtu důležitosti oblastí. Grad-CAM používá gradienty (směrové derivace) výstupu neuronové sítě vzhledem k aktivacím v konkrétní vrstvě sítě. [46]

Existuje mnoho dalších metod, které jsou různě upravovány tak, aby odpovídaly specifickým potřebám intepretace.

4.3.1. Post-hoc a ad-hoc

Výše zmíněné způsoby interpretace je možno podle práce Feng-Lei F. a jeho spolupracovníků [36] rozdělit na dvě skupiny post-hoc a ad-hoc. Post-hoc metody intepretace jsou používány na již vytrénovaný model umělé neuronové sítě. Účelem

těchto metod je porozumění vztahu mezi vstupem a výstupem z modelu. Ale nejenom to. Pokouší se také identifikovat oblasti vstupu, které měly na finální predikci (výstup z modelu) největší vliv. Jako hlavní výhodu této skupiny metod ve výzkumu zmiňuje, že není potřeba dělat kompromisy mezi intepretací a výkonem sítě. „Hlavní výhoda post-hoc metod je, že není třeba dělat kompromisy v interpretovatelnosti s prediktivním výkonem, protože predikce a interpretace jsou dva samostatné procesy bez vzájemného ovlivňování.“ Dále také zmiňuje, že výsledky post-hoc metod interpretace nejsou zcela věrné svému modelu – a pokud by intepretace byla 100% přesná, stane se sama modelem. Mezi tyto modely patří například mapy síly vlivu, maximalizace aktivace nebo extrakce modelu. [36]

Ad-hoc metody jsou ve většině případů upraveny pro specifický úkol, dataset nebo architekturu umělé neuronové sítě. Vzhledem k tomu, že jsou upraveny pro specifický účel, nejsou tedy tak generalizované a mohou poskytovat subjektivní výsledky. Jejich hlavní rozdíl je, kdy jsou takové metody používány. Na rozdíl od post-hoc metod, které jsou používány až po dokončení tréninku umělé neuronové sítě, tyto metody jsou integrovány a použity během tréninku a mohou ovlivňovat chování modelu a jeho výsledky. Ovlivnit v tom smyslu, že mohou dokonce upravovat strukturu umělé neuronové sítě nebo nějakým způsobem ovlivňovat vstupní data. Mezi takové ad-hoc metody se řadí citlivost na překrytí, mapy aktivace tříd a další.

Některé metody jako mapy síly vlivu, maximalizace aktivace nebo LRP se dají použít jako ad-hoc nebo post-hoc metody.

Ve zkratce, ad-hoc metody mohou ovlivnit tréninkový proces změnou architektury umělé neuronové sítě, ztrátové funkce, tréninkových dat nebo rychlosti učení. Myšlenka za tím je taková, že jde o snahu začlenit interpretační cíl do návrhu sítě nebo tréninku tak, aby byla síť trénována k lepší interpretovatelnosti. Výsledkem může být síť, která je transparentnější a snáze srozumitelná, ale také může být síť složitější a obtížněji trénovatelná. Na druhou stranu, post-hoc metody se aplikují po natrénování sítě a nemění strukturu sítě ani tréninkový proces. Tyto metody poskytují vysvětlení chování sítě, ale nezbytně nečiní síť lépe interpretovatelnou. Jinak řečeno, pokud jsou post-hoc metody aplikovány na složitou umělou neuronovou síť, tak to nutně nemusí vést k jednoduché interpretaci. [36]

5. PROSTŘEDÍ

Pro účely praktické části této práce bylo využito několik základních, ale i pokročilejších interpretačních metod, které budou představeny později. Pro potřeby interpretace byl použit již navržený a vytrénovaný model umělé neuronové sítě MobileNetV2. Pro implementaci těchto metod bylo využito prostředí Python a knihovny TensorFlow, Keras, NumPy, Matplotlib a OpenCV-Python.

K testování byl použit stroj s těmito parametry:

- Intel Core i7-7700HQ,
- NVIDIA GeForce GTX 1050 Ti,
- 16 GB RAM.

5.1. Neuronová síť MobileNetV2

Model neuronové sítě MobileNetV2 je založen na architektuře konvoluční neuronové sítě, kterou v roce 2018 vyvinuli výzkumníci z Google. Je navržena tak, aby poskytovala efektivní a přesnou klasifikaci obrázků na mobilních zařízeních a v dalších vestavěných systémech.

Vychází z MobileNetV1, což je také model umělé neuronové sítě pro počítačové vidění navržený s ohledem na mobilní zařízení. Podporuje klasifikaci, detekci a další funkce.

Model MobileNetV2 používá oddělitelné konvoluce, které jsou účinnější než tradiční konvoluce, protože oddělují prostorové a kanálové filtry do dvou samostatných operací. To snižuje počet parametrů a výpočtů potřebných pro model. MobileNetV2 také zahrnuje funkci nazvanou "lineární bottleneck", což umožňuje rychlejší trénování a lepší výkon generalizace. Model také používá techniku nazvanou "inverted residuals", která přidává nelinearitu do sítě, přičemž stále udržuje nízké výpočetní požadavky.

Celkově je MobileNetV2 výkonným modelem neuronové sítě, který jak již bylo řečeno, je optimalizován pro mobilní a vestavěné zařízení, což ho činí ideálním pro aplikace sloužící k rozpoznávání obrazů, detekci objektů na smartphonu, dronu a dalších mobilních zařízeních. [44] [45]

Na tomto modelu se nebude nic měnit nebo upravovat. Bude tedy použit tak, jak byl vytvořen a vytrénován. Model je vytvořený pro účely klasifikace obrázků. Obrázky

klasifikuje až do 1000 tříd, které se zabývají jednoznačnější specifikací. Neprovádí tedy běžnou klasifikaci, jako je rozdělení na pes nebo kočka. Je schopen detekovat, jestli se například jedná o plemeno čivava nebo anglický kokršpaněl. Je také schopen klasifikovat běžné předměty, jako je kartáček na zuby nebo notebook.

5.2. Python

Pro vytvoření prostředí pro implementaci interpretačních metod bude použit jazyk Python ve verzi 3.10.2. Jde o moderní vysokoúrovňový programovací jazyk, který se používá v různých oblastech, jako jsou webové aplikace, databáze, vědecké výpočty, umělá inteligence a mnoho dalších.

V Pythonu existuje mnoho knihoven, které rozšiřují jeho funkce a usnadňují vývoj aplikací. Některé z nejpopulárnějších knihoven jsou například NumPy pro vědecké výpočty, Django pro webové aplikace a TensorFlow pro umělou inteligenci.

5.2.1. Použité knihovny

Jak již bylo zmíněno, tak Python podporuje spoustu knihoven, které práci výrazně ulehčují. Pro implementaci interpretačních metod byly využity tyto knihovny:

- TensorFlow – verze 2.8.0rc1,
    - Keras – verze 2.8.0,
- NumPy – verze 1.22.1,
- Matplotlib – verze 3.5.1,
- OpenCV-Python – verze 4.7.0.72.

5.2.1.1. TensorFlow

TensorFlow je open-source knihovna pro strojové učení a vytváření umělých neuronových sítí. Byla vyvinuta společností Google Brain Team. TensorFlow je široce používán pro rozpoznávání řeči, překlad jazyka, zpracování obrazu, detekci a klasifikaci objektů v reálném čase a mnoho dalších aplikací. Knihovna je navržena tak, aby byla snadno použitelná a kompatibilní s různými programovacími jazyky včetně Pythonu, C++ a Java.

5.2.1.2. Keras

Keras je open-source knihovna pro strojové učení napsaná v jazyce Python. Tato knihovna poskytuje jednoduché rozhraní pro tvorbu, trénování a testování neuronových sítí. Jedná se o jakousi nadstavbu pro knihovnu TensorFlow. Také obsahuje mnoho předpřipravených modelů neuronových sítí, které lze snadno použít pro různé úlohy jako klasifikace obrazů, rozpoznávání řeči a další.

5.2.1.3. NumPy

NumPy, také znám jako Numerical Python, je open-source knihovna pro programování v jazyce Python, která se zaměřuje na výpočty s vícerozměrnými poli a maticemi. NumPy poskytuje mnoho funkcí a nástrojů pro efektivní práci s daty v Pythonu, zejména pro matematické, vědecké a statistické výpočty. Hlavní funkcí NumPy je vytváření a manipulace s vícerozměrnými poli, také obsahuje mnoho funkcí pro rychlé a efektivní matematické operace s maticemi, jako jsou maticové násobení, inverzní matice, vlastní čísla a další. Knihovna NumPy je široce používána v oblastech, jako je strojové učení, věda o datech, fyzika, matematika a další.

5.2.1.4. Matplotlib

Matplotlib je open-source knihovna pro vizualizaci dat v jazyce Python. Umožňuje vytváření různých druhů grafů a vizualizací, jako jsou čárové grafy, sloupcové grafy, histogramy, scatter ploty, konturové grafy, 3D grafy a mnoho dalších. Matplotlib umožňuje vytvářet grafy s různými typy dat. Knihovna také poskytuje mnoho funkcí a nástrojů pro přizpůsobení grafů a vizualizací, jako jsou popisky os, titulky, legenda, barvy a tak dále.

5.2.1.5. OpenCV-Python

OpenCV-Python je rozšíření knihovny OpenCV pro jazyk Python, které umožňuje používat funkce a nástroje knihovny OpenCV v jazyce Python. OpenCV-Python poskytuje snadný a intuitivní způsob pro práci s obrazovými daty v jazyce Python a je často používán v oblastech, jako jsou počítačové vidění, robotika, biometrika a další.

5.2.2. Využití knihoven

Knihovna TensorFlow a její nadstavba Keras bude používána naprosto pokaždé u jakékoliv metody interpretace umělé neuronové sítě. Skrze tuto knihovnu se budou nejenom předzpracovávat obrázky před prací s nimi, ale bude se z ní načítat umělá

neuronová síť MobileNetV2. Budou se ale využívat i další metody této knihovny a její nadstavby. Knihovna NumPy se bude používat za účelem převedení obrázku na pole nebo rozšíření dimenzí daného pole. Matplotlib se bude používat k prezentaci výsledků. Jako poslední bude tedy využita knihovna OpenCV-Python. Tato knihovna se použije jen při implementaci dvou metod, kde budou plnit víceméně vizuální činnost.

6. IMPLEMENTANCE ZPŮSOBŮ INTERPRETACE

Pro implementaci byly vybrány tyto způsoby interpretace:

- mapy aktivací (Activation maps),
- mapy síly vlivu (Saliency maps),
- citlivost na překrytí (Occulsion sensitivity),
- maximalizace aktivace (Activation maximization).

6.1. Mapy aktivací (Activation maps)

Mapa aktivací je vizuální reprezentace aktivací skupiny neuronů v určité vrstvě umělé neuronové sítě. Lze ji představit jako 2D mřížku, kde každá buňka odpovídá aktivaci konkrétního neuronu nebo kanálu ve vrstvě.

Aktivační mapy v prvních vrstvách umělé neuronové sítě typicky zachycují nízko-úrovňové rysy a vzory. Tyto vrstvy jsou blíže vstupní vrstvě a mají tendenci učit se jednoduché věci, jako jsou hrany, rohy nebo textury. Aktivační mapy v prvních vrstvách jsou obvykle lokalizovanější a konkrétnější. Zaměřují se na zachycení informací o menších oblastech vstupu.

Jak se generují mapy aktivací pro hlubší vrstvy umělé neuronové sítě, aktivační mapy v mezivrstvách začnou zachycovat složitější a abstraktnější rysy. Tyto vrstvy vycházejí z nízko-úrovňových prvků naučených v prvních vrstvách a začínají představovat smysluplnější struktury nebo kombinace prvků. Aktivační mapy v mezivrstvách mohou odpovídat sofistikovanějším vzorům, jako jsou části objektů nebo specifické textury v kontextu úloh analýzy obrazu.

V nejhlubších vrstvách neuronové sítě aktivační mapy často představují sémantické koncepty na vysoké úrovni nebo celé objekty. Tyto vrstvy integrují informace z více mezivrstev a učí se rozpoznávat složité objekty nebo scény. Aktivační mapy v hlubokých vrstvách mívají větší receptivní pole a zachycují kontext celého obrázku.

Pro generování mapy aktivací musíme vstup předat do sítě a pak extrahovat jeho aktivace v dané vrstvě. Jakmile máme aktivace, můžeme je vizualizovat jako mapu teploty (heat map), kde barva každého pixelu odpovídá hodnotě aktivace.

Mapy aktivací mohou pomoci identifikovat, které části vstupu jsou pro rozhodnutí sítě nejrelevantnější. Například pokud se konkrétní neuron v nějaké vrstvě umělé neuronové sítě pravidelně aktivuje pro určitou třídu obrázků, můžeme usoudit, že tato vlastnost je důležitá pro rozlišení této třídy od ostatních.

6.1.1. Kód

Jako první budou naimportovány tyto balíčky:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

Dále v kódu bude nejprve načten obrázek psa, změní jeho velikost na 224x224 pixelů a převede ho na pole NumPy. Následně se aplikuje předzpracování pomocí funkce preprocess_input() v rámci vestavěné funkce modelu MobileNetV2. Poté kód načte již vytrénovaný model MobileNetV2 pomocí funkce tf.keras.applications.MobileNetV2(). Vrstva, pro kterou se bude snažit vizualizovat mapy aktivací, je nastavena na 'block_1_expand_relu'.

```
# Load an image
image = tf.keras.preprocessing.image.load_img('images/dog.jpg', target_size=(224, 224))
image = tf.keras.preprocessing.image.img_to_array(image)

# Preprocess the image
image = tf.keras.applications.mobilenet_v2.preprocess_input(image)

# Load the MobileNetV2 model
model = tf.keras.applications.MobileNetV2()

# Define the layer to visualize
layer_name = 'block_1_expand_relu'
```

Funkce nazvaná get_activation_maps() je definována k získání aktivací pro zadanou vrstvu a obrázek. Funkce get_activation_maps() přijímá jako vstupy model MobileNetV2, název vrstvy a obrázek. Poté je vytvořen sub-model, který vytvoří výstupy aktivací pro danou vrstvu, na základě vstupního obrázku. Poté jsou aktivace vráceny touto funkcí. Nakonec kód zavolá funkci get_activation_maps() pro získání aktivací pro specifikovanou vrstvu a obrázek.

```
# Define a function to get the activation maps for a given layer and image
```

```
def get_activation_maps(model, layer_name, image):
  # Create a sub-model that outputs the activations for the given layer
  outputs = [model.get_layer(layer_name).output]
  sub_model = tf.keras.models.Model(inputs=model.inputs, outputs=outputs)

  # Get the activations for the image
  activations = sub_model.predict(np.array([image]))

  # Return the activations
  return activations[0]

# Get the activation maps for the defined layer
activations = get_activation_maps(model, layer_name, image)
```

Poté se zobrazí aktivační mapy pomocí knihovny matplotlib, která ukazuje mřížku o velikosti 4x4, kde každá jedna buňka zobrazuje jednu aktivační mapu pro jeden neuron dané vrstvy. Výsledný graf je uložen do souboru a zobrazen na obrazovce.

```
# Plot the activation maps
fig, axes = plt.subplots(4, 4, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(activations[..., i], cmap='jet')
    ax.set_axis_off()
plt.savefig('interpretation_methods/activation_maps/output/dog/activation_maps_' + layer_name + '.png')
plt.show()
```

6.1.2. Testování a výsledky

Pro testování generování map aktivací, byly vybrány dva obrázky:

- pes – blenheimský španěl,
- kočka – nespecifikovaný druh.

Mapy aktivací poté budou generovány pro tyto vrstvy:

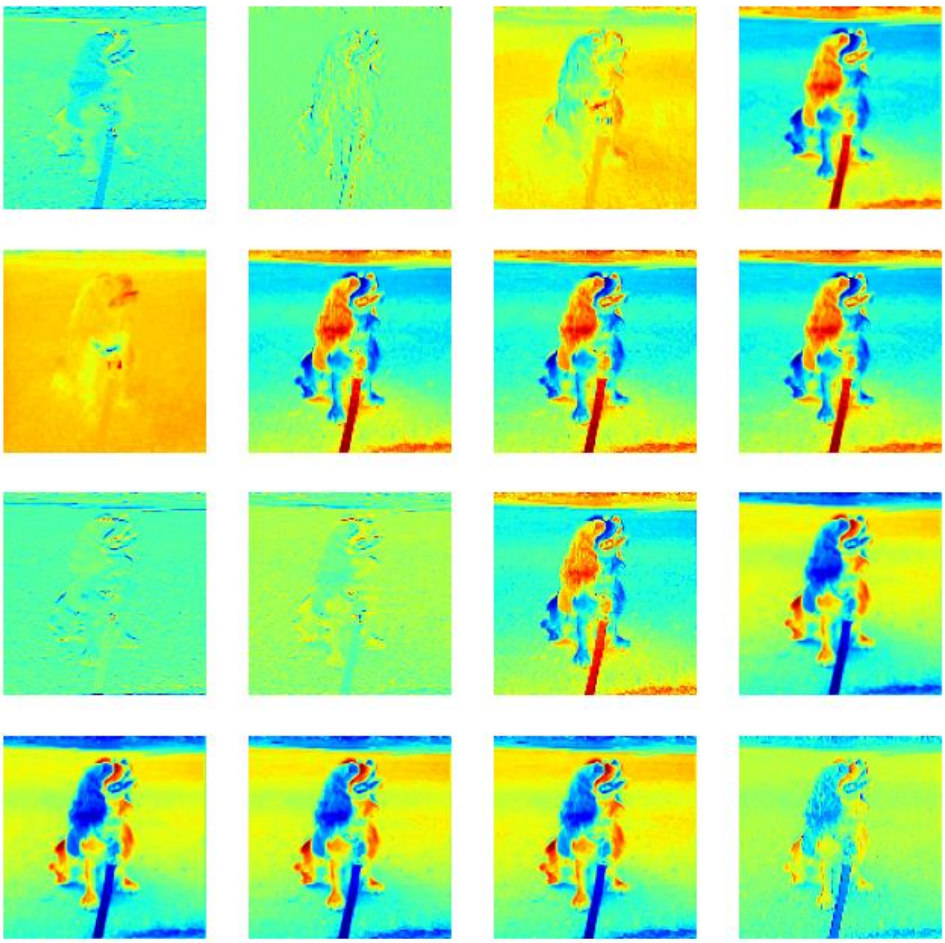
- conv1,
- block_1_expand_relu,
- block_4_expand_relu,
- block_8_expand_relu,

- block_12_expand_relu,
- block_16_expand_relu,
- out_relu.

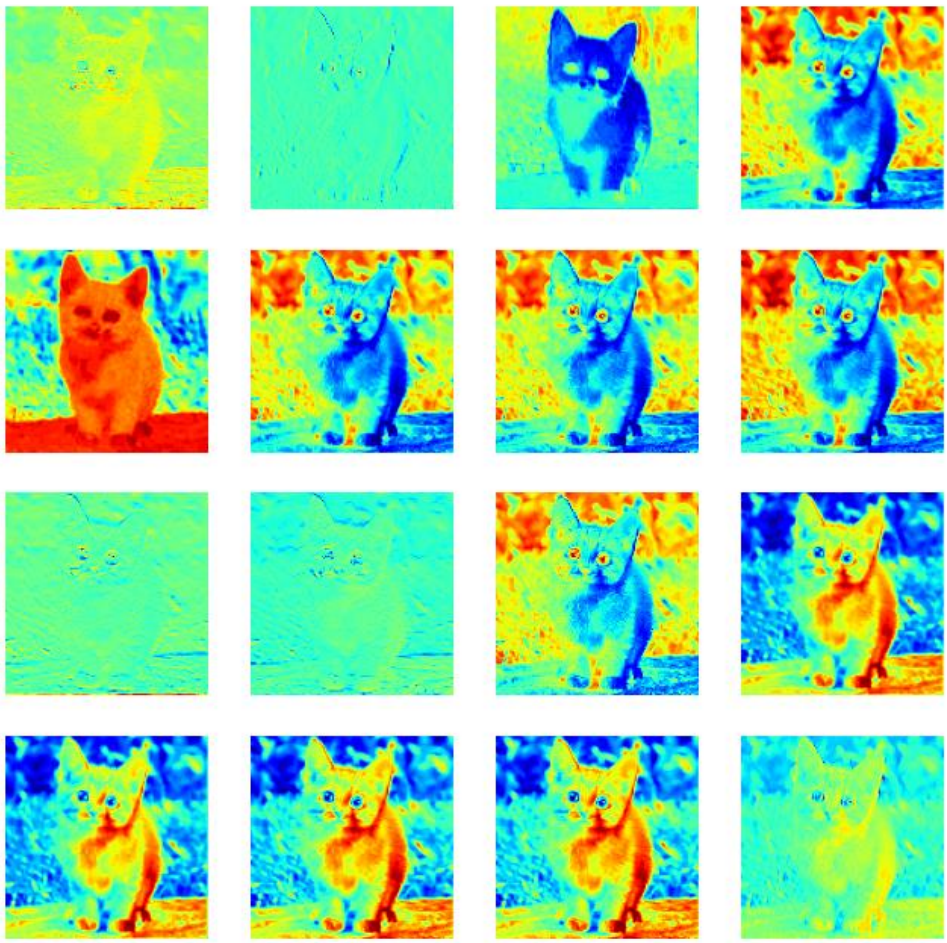
Očekávání jsou taková, že aktivační mapy pro první vrstvy umělé neuronové sítě budou velice připomínat původní obrázek. V těchto mapách se budou aktivace zaobírat spíše hranami a menšími detaily. Čím hlouběji se bude posunovat, tím abstraktnější budou obrázky a neuronová síť se bude snažit v nich najít nějaký vzor. Původní obrázek zcela vymizí a postupně budou aktivovány jen určité části obrázku, kde zbytek nebude aktivován vůbec.

6.1.2.1. Vrstva Conv1

Vrstva Conv1 je první vrstva nacházející se hned za vstupní vrstvou v modelu umělé neuronové sítě MobileNetV2. Mapy aktivací pro oba obrázky je možné si prohlédnou níž:



Obrázek 15 – Aktivační mapy pro vrstvu Conv1 – obrázek psa

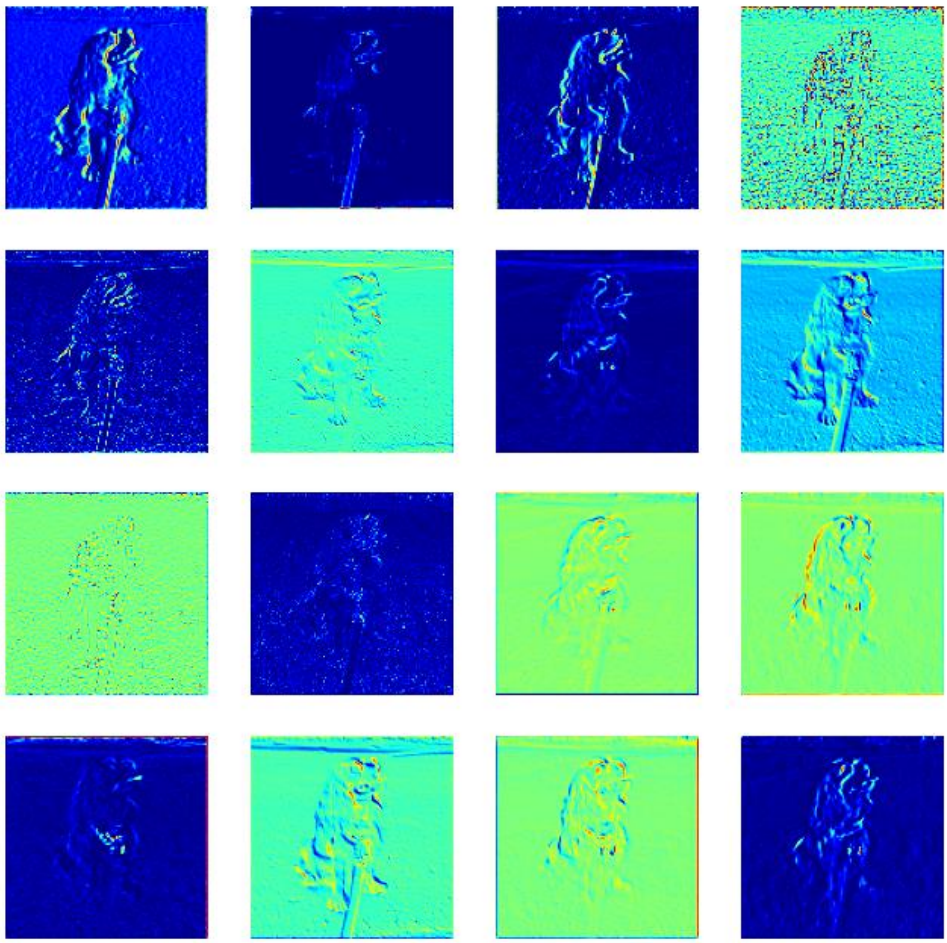


Obrázek 16 – Aktivační mapy pro vrstvu Conv1 – obrázek psa

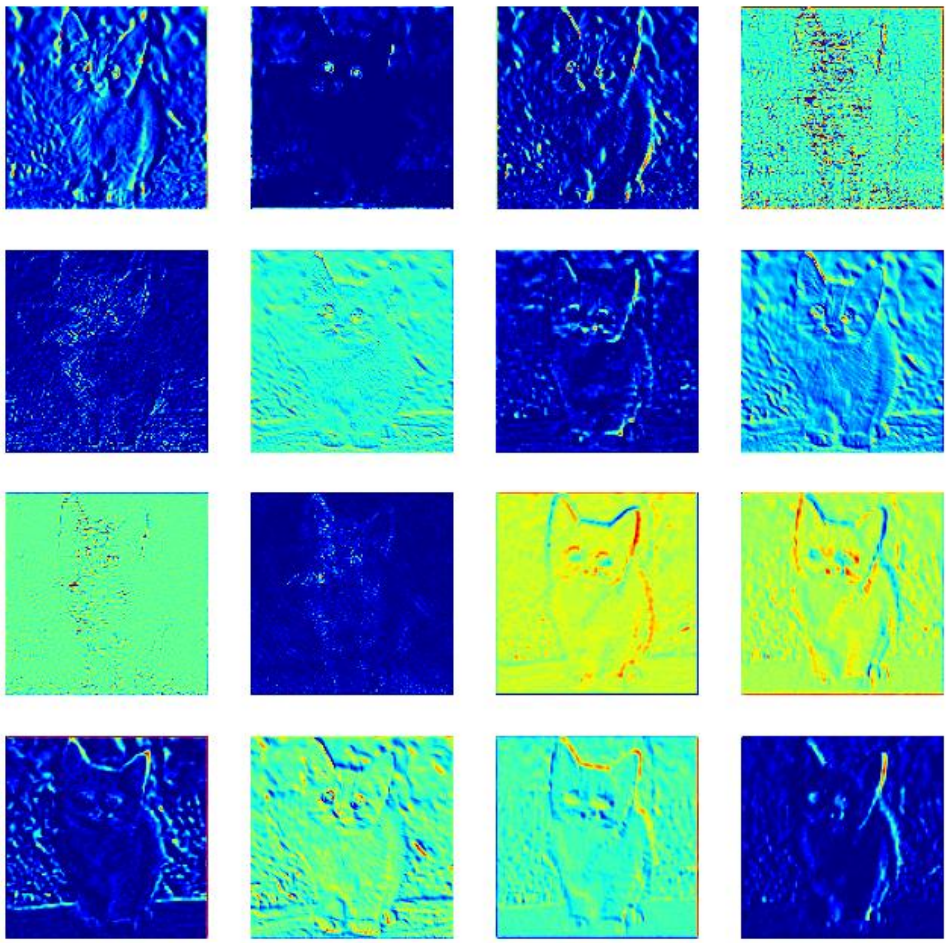
Oba obrázky 15 a 16 mají společné rysy u všech aktivačních map. Každý obrázek reprezentuje jeden neuron v dané vrstvě umělé neuronové sítě. Je možné vidět, že neurony reagují dosti podobně, a tím pádem generují v principu i podobnou aktivační mapu. Podle obrázků to vypadá, že se tato vrstva zaměřuje na tmavší oblasti a pozadí.

6.1.2.2. Vrstva block_1_expand_relu

V tomto testu byla použita první z 16 rozšířených relu aktivačních vrstev. Tato vrstva je stále jednou z prvních vrstev v modelu použité umělé neuronové sítě.



Obrázek 17 – Aktivační mapy pro vrstvu block_1_expand_relu – obrázek psa

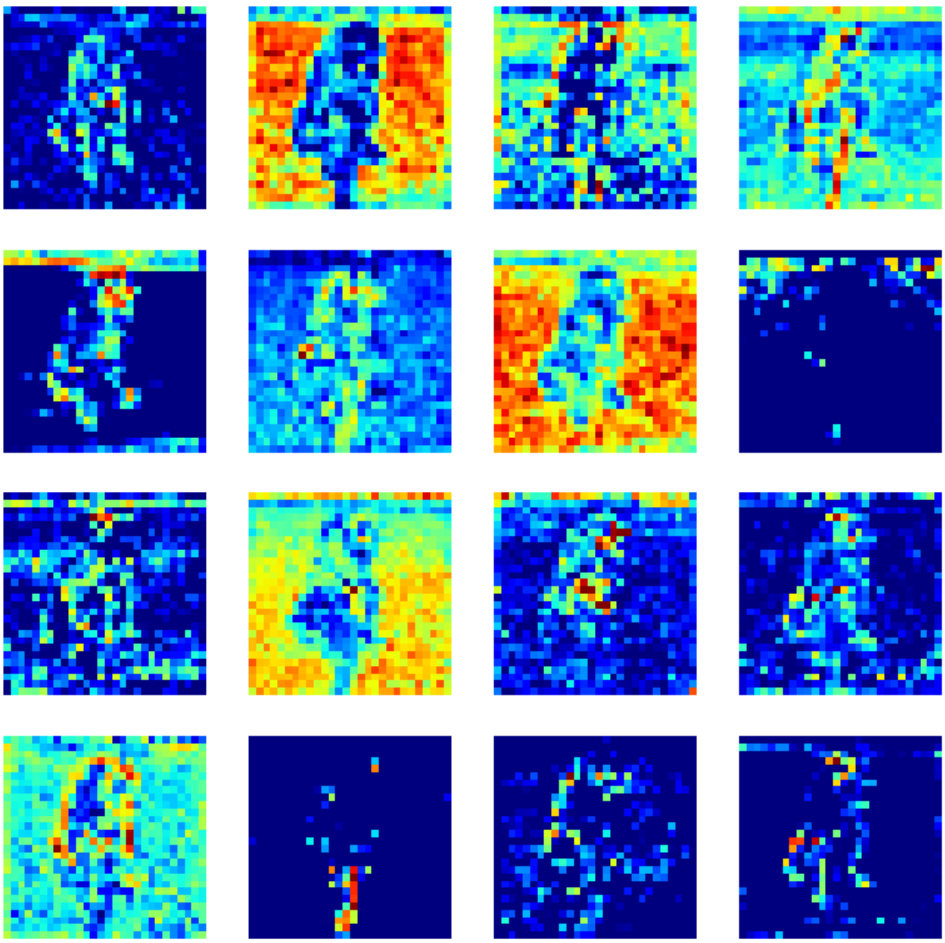


Obrázek 18 – Aktivační mapy pro vrstvu block_1_expand_relu – obrázek kočky

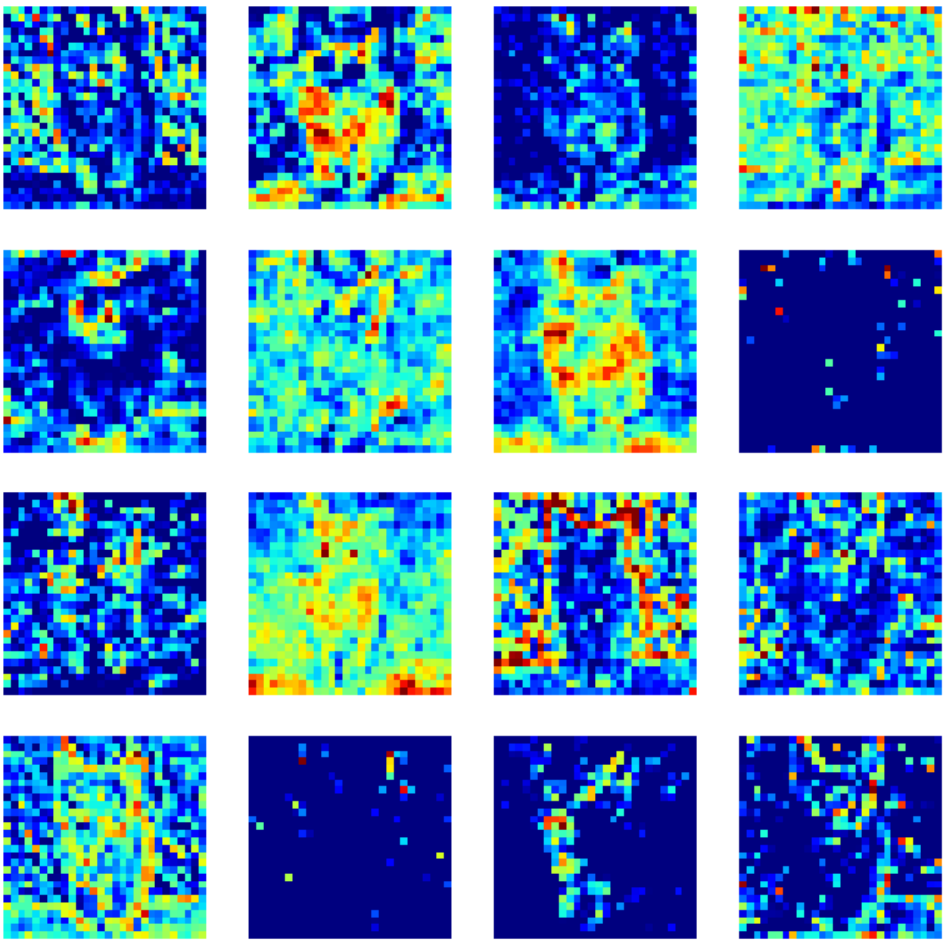
Podle obrázků aktivačních map pro vrstvu block_1_expand_relu to vypadá, že se tato vrstva zaměřuje na jisté hrany a ostré rysy obrázku. Stále se totiž jedná o jednu z prvních vrstev, proto je stále jasně vidět původní obrázek.

6.1.2.3. Vrstva block_4_expand_relu

Zde se již posouvá hlouběji do nitra umělé neuronové sítě MobileNetV2. To tedy znamená, že už by původní obrázek neměl být tak jasně poznat, ale stále by ho měl pozorovatel rozlišit. Aktivační mapy pro tuto vrstvu vypadaly takto:



Obrázek 19 – Aktivační mapy pro vrstvu block_4_expand_relu – obrázek psa

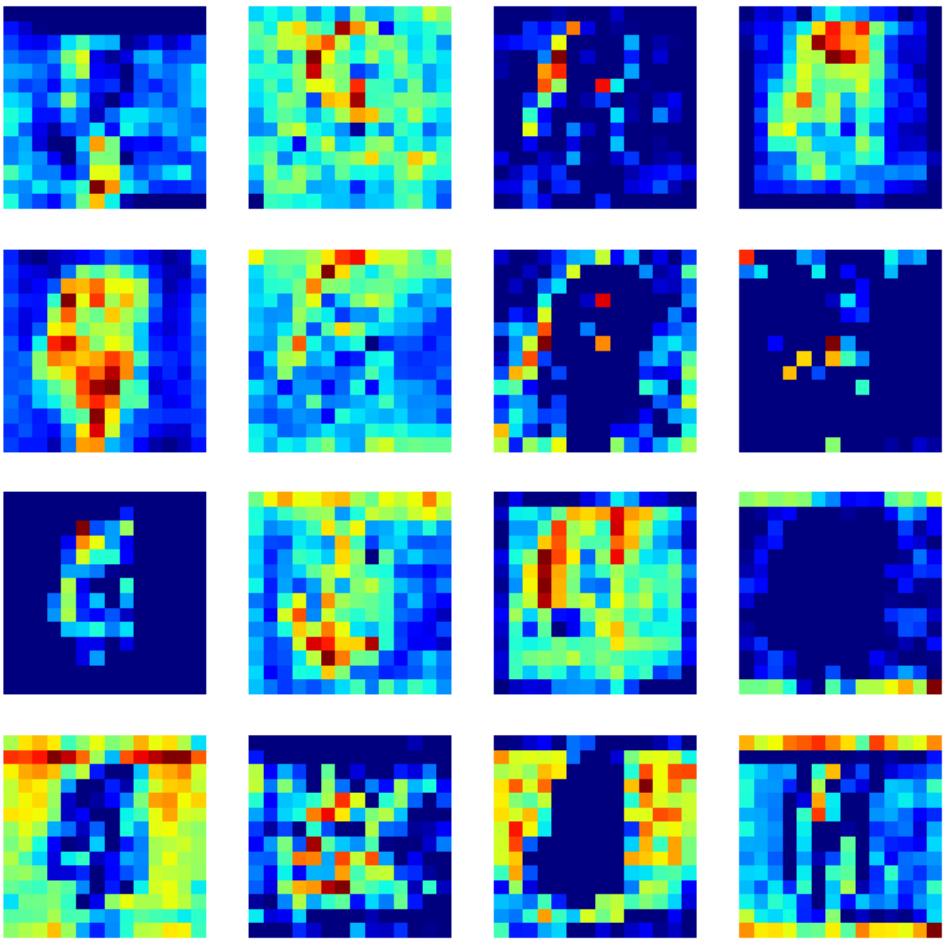


Obrázek 20 – Aktivační mapy pro vrstvu block_4_expand_relu – obrázek kočky

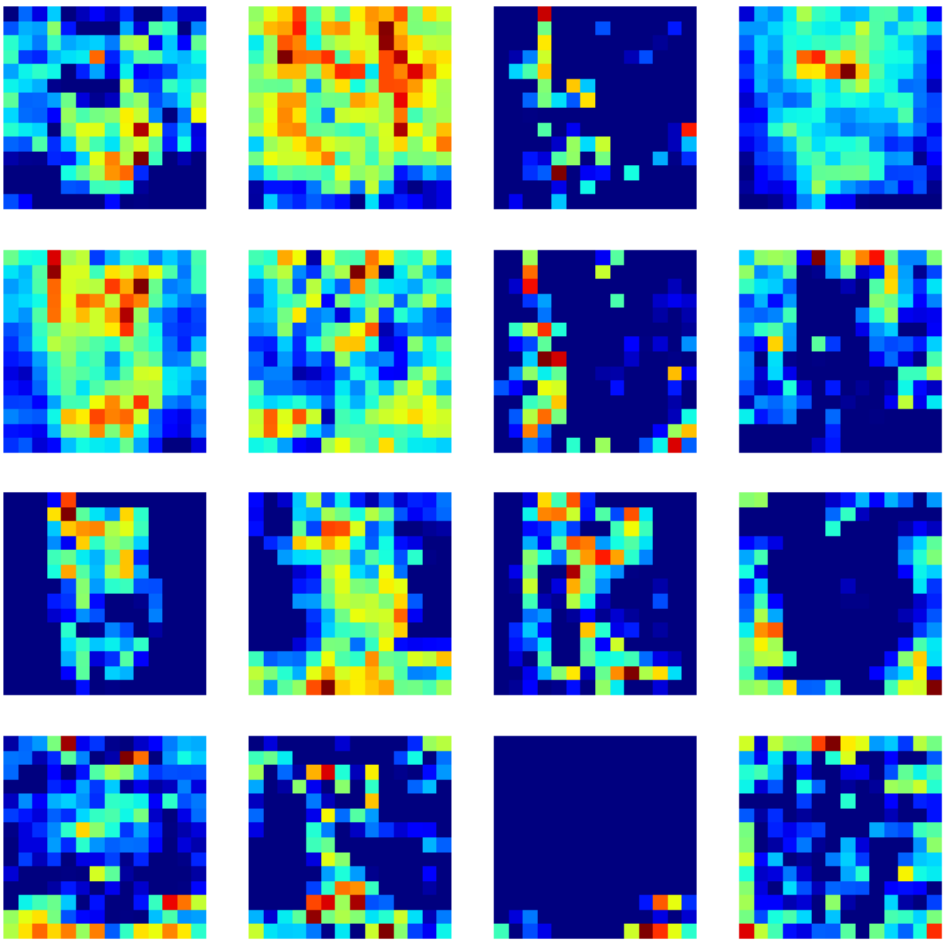
Jak bylo očekáváno, původní obrázek již není tak jednoznačný, ale stále je rozpoznatelný. Zde už nevypadá každá mapa aktivace pro stejný neuron stejně. To může být tím, že zvířata jsou na obrázku jinak natočena. Každopádně je vidět, že tuto vrstvu už neaktivují jen jemné hrany, ale o trochu větší oblasti v okolí těchto hran. Některé neurony nejsou aktivovány všude, například druhý neuron v posledním řádku je pro oba obrázky aktivován jen v jistých místech. Což se v předchozích vrstvách nedělo.

6.1.2.4. Vrstva block_8_expand_relu

Zde se pro generování aktivačních map využila vrstva block_8_expand_relu. Její vygenerované aktivační mapy pro oba obrázky je možné prohlédnou na obrázku 21, respektive 22.



Obrázek 21 – Aktivační mapy pro vrstvu block_8_expand_relu – obrázek psa

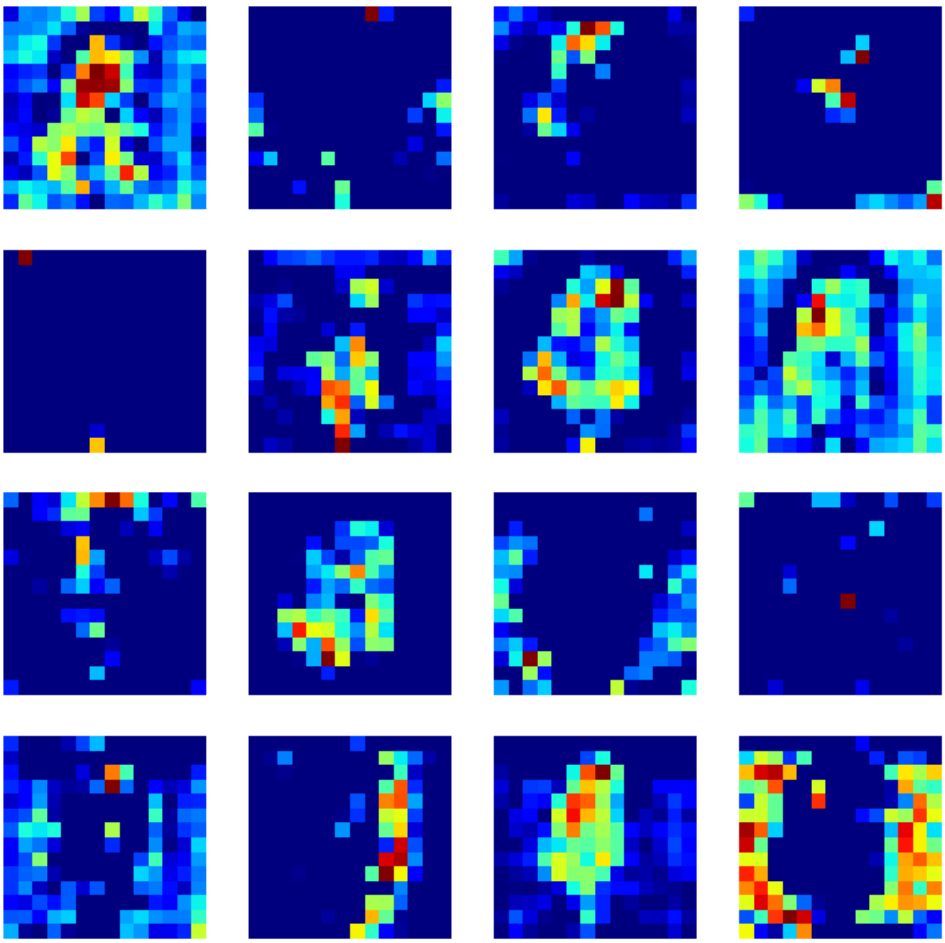


Obrázek 22 – Aktivační mapy pro vrstvu block_8_expand_relu – obrázek kočky

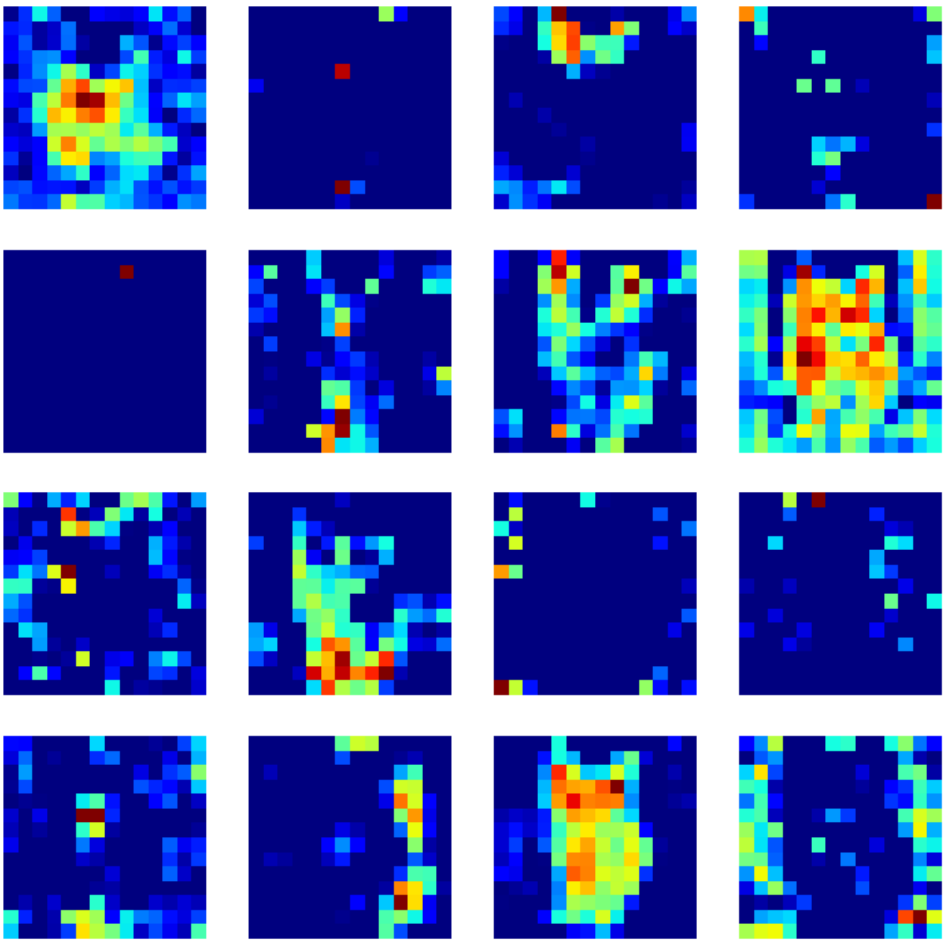
Jak je možné prohlédnout si na obrázcích výš, původní obrázek už skoro nejde rozeznat, ale jisté rysy zde stále vidět jsou. To by se asi dalo přisuzovat tomu, že se jedná o prostřední vrstvu z 16 rozšířených relu aktivačních vrstev. Jsme tedy zřejmě na pomezí toho, kde se začínají hledat jisté vzorce a opakující se prvky v obrázku. Jak je možné vidět, některé neurony aktivuje samotné zvíře a pozadí vůbec. U jiných je tomu přesně naopak.

6.1.2.5. Vrstva block_12_expand_relu

Zde byla použita 12. rozšířená vrstva s aktivační funkcí relu – vrstva block_12_expand_relu. Aktivační mapy pro tuto vrstvu je možné si prohlédnout na obrázcích 23 a 24.



Obrázek 23 – Aktivační mapy pro vrstvu block_12_expand_relu – obrázek psa

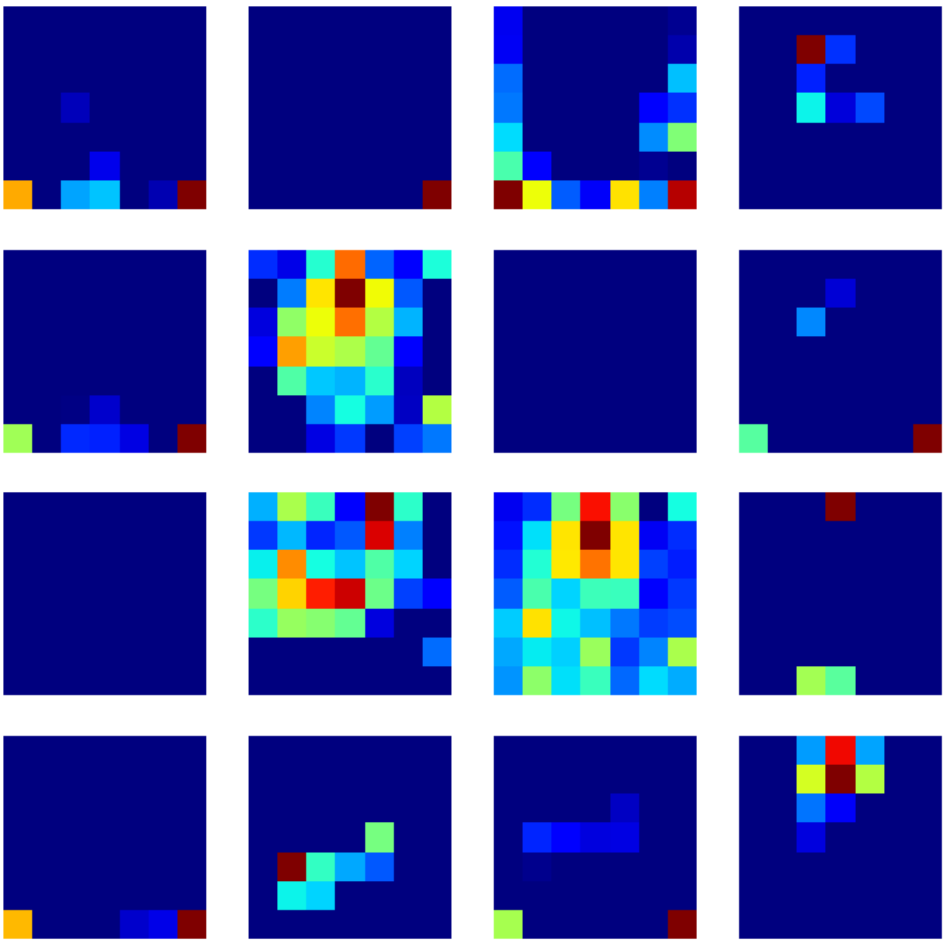


Obrázek 24 – Aktivační mapy pro vrstvu block_12_expand_relu – obrázek kočky

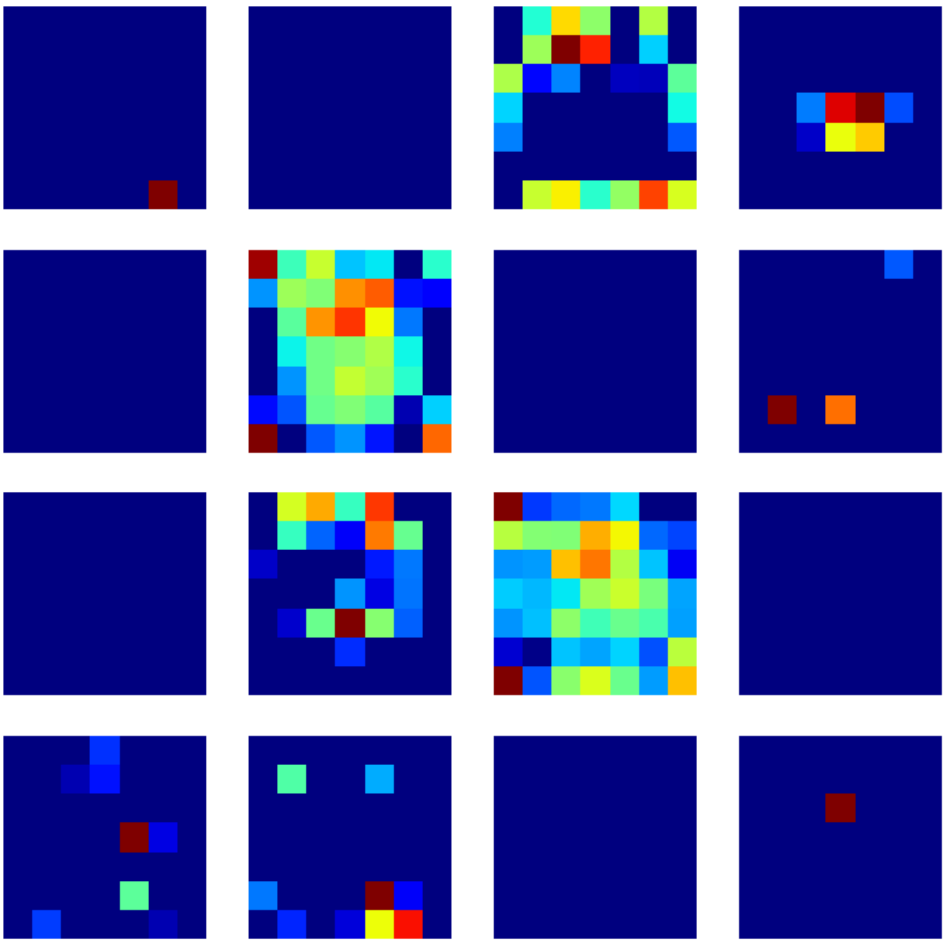
Zde už se dá původní obrázek rozeznat jen v několika málo případech. Obrázku už jsou mnohem více rozpixelované, což značí, že se všechny neurony v umělé neuronové sítí snaží na obrázek nahlížet abstraktněji a soustředí se spíše na primární oblasti obrázku.

6.1.2.6. Vrstva block_16_expand_relu

Zde je použita jedna z posledních vrstev v modelu umělé neuronové sítě MobileNetV2. Jedná se o vrstvu block_16_expand_relu. Vygenerované mapy aktivací pro oba obrázky jsou zobrazeny níž.



Obrázek 25 – Aktivační mapy pro vrstvu block_16_expand_relu – obrázek psa

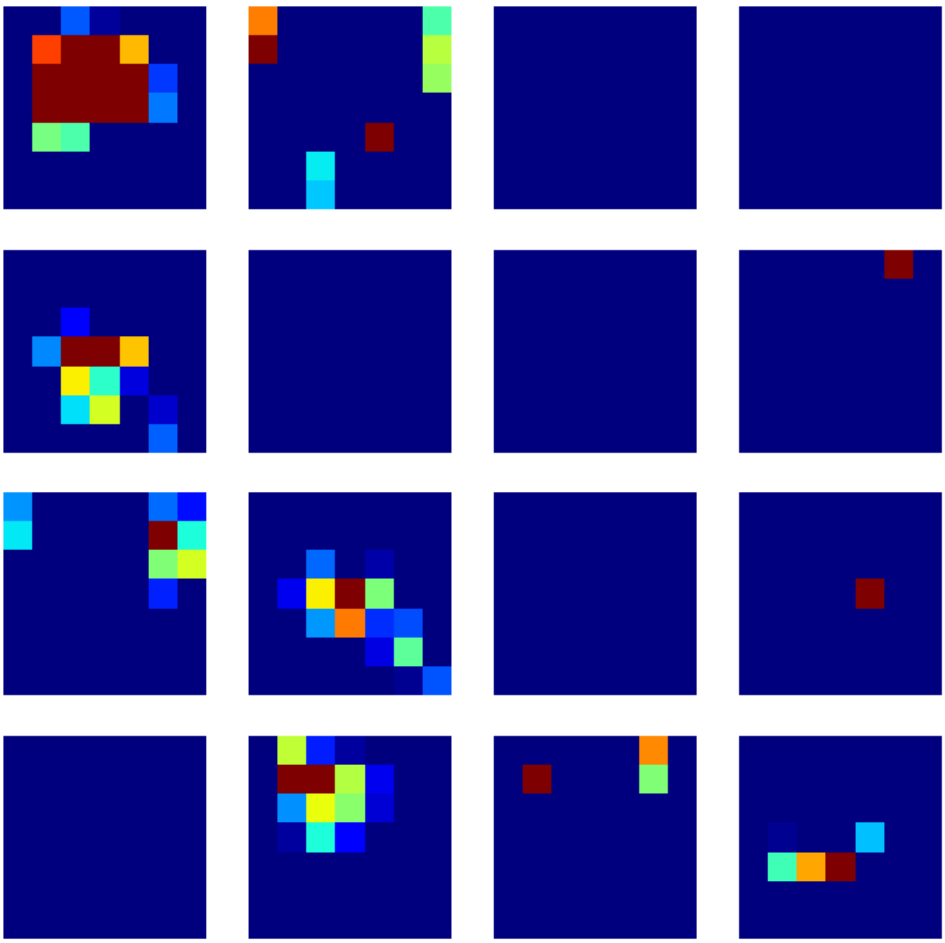


Obrázek 26 – Aktivační mapy pro vrstvu block_16_expand_relu – obrázek kočky

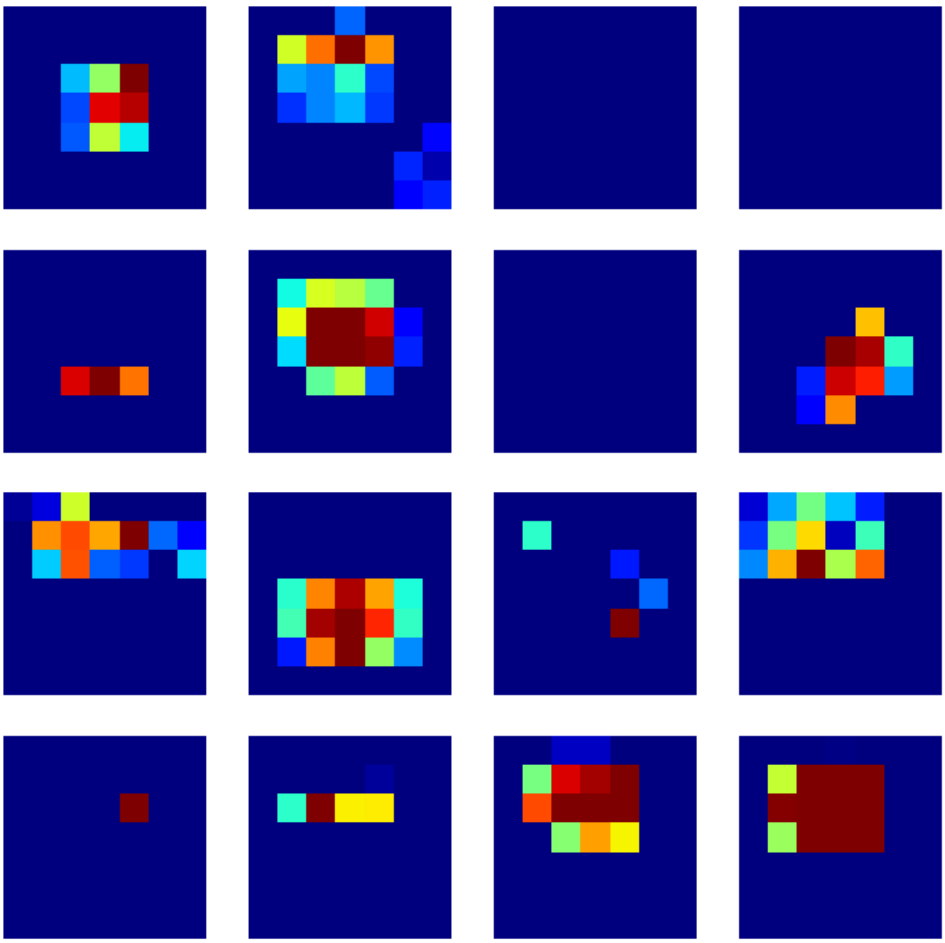
Jak je vidět na obrázcích 25 a 26, v mapách aktivací pro danou vrstvu už není původní obrázek vůbec rozeznatelný. Aktivované už jsou opravdu jen oblasti obrázku – což značí, že tato vrstva se zcela zaměřuje jen na kusy obrázků. Některé neurony nejsou aktivované vůbec. To znamená, že zde není žádná oblast, která by se danému obrázku zdála důležitá.

6.1.2.7. Vrstva out_relu

U posledního testu generování aktivačních map byla použita vrstva out_relu. Jedná se o předposlední vrstvu před výstupní vrstvou.



Obrázek 27 – Aktivační mapy pro vrstvu out_relu – obrázek psa



Obrázek 28 – Aktivační mapy pro vrstvu out_relu – obrázek kočky

Jak je možné pozorovat na obou obrázcích 27 a 28, mapy aktivací jsou už jen v podstatě obrázky 7x7 pixelů. Jedná se tedy o maximální abstrakci. Původní obrázek už dávno není možné rozeznat. Zde už se opravdu zobrazují jen jisté rysy a vzory. Na prvním obrázku psa a jeho prvním neuronu je možné pozorovat podobnou oblast jako na druhém obrázku kočky a jeho posledním neuronu. Vzhledem k tomu, že se ale jedná o jiné neurony, tak jde bezpochyby o shodu náhod. Dá se z toho ale vyčíst, že se hledají jen celé oblasti obrázku, které napomáhají přesné klasifikaci.

6.2. Mapy síly vlivu (Saliency maps)

Mapy síly vlivu jsou vizuální reprezentace nejdůležitějších oblastí vstupu pro konkrétní výstupní třídu neuronové sítě. Lze si ji představit jako teplotní mapu, kde intenzita každého pixelu odpovídá jeho saliančnímu skóre.

Tyto mapy jsou generovány pomocí techniky zvané gradient-based attribution, kde se vypočte gradient výstupu vzhledem k oblasti vstupu, který nejvíc přispívái pro danou predikci vůči vybrané třídě. Konkrétně se salienční skóre každého pixelu vypočítá jako absolutní hodnota parciální derivace výstupu vzhledem k pixelu.

Mapy síly vlivu mohou pomoci identifikovat, které části vstupu jsou pro rozhodnutí sítě nejdůležitější a mohou poskytnout náhled do toho, jak se síť rozhoduje. Například při klasifikaci obrázků může pomoci identifikovat, na které části obrazu se síť dívá jako na určitou třídu.

6.2.1. Kód

Stejně jako v prvním případě se načtou importované knihovny. Kód pokračuje tím, že se načítají obrázky psa, kočky, psa s kočkou a slona. Poté jsou obrázky předzpracovány pomocí funkce preprocess_input(). Následně se načte model MobileNetV2.

```
# Load an image
image = tf.keras.preprocessing.image.load_img('images/dog.jpg', target_size=(224, 224))
image = tf.keras.preprocessing.image.img_to_array(image)


image1 = tf.keras.preprocessing.image.load_img('images/cat.jpg', target_size=(224, 224))
image1 = tf.keras.preprocessing.image.img_to_array(image1)


image2 = tf.keras.preprocessing.image.load_img('images/catAndDog.jpg', target_size=(224, 224))
image2 = tf.keras.preprocessing.image.img_to_array(image2)


image3 = tf.keras.preprocessing.image.load_img('images/elephant.jpg', target_size=(224, 224))
image3 = tf.keras.preprocessing.image.img_to_array(image3)


# Preprocess the image
image = tf.keras.applications.mobilenet_v2.preprocess_input(image)
image1 = tf.keras.applications.mobilenet_v2.preprocess_input(image1)
image2 = tf.keras.applications.mobilenet_v2.preprocess_input(image2)
```

```
image3 = tf.keras.applications.mobilenet_v2.preprocess_input(image3)

# Load the MobileNetV2 model
model = tf.keras.applications.MobileNetV2()
```

Poté se vytvoří funkce pro získání map sil vlivu. Funkce get_saliency_maps() potřebuje na vstupu model, pole obrázků a index třídy, kterou chceme predikovat jako vstup. Nejprve se získá gradient výstupu třídy vzhledem k vstupnímu obrázku pomocí GradientTape. Poté se vypočítají mapy síly vlivu. Ty se normalizují a převedou se na pole NumPy. Funkce nakonec vrátí mapy síly vlivu.

```
# Define a function to get the saliency maps for a given batch of images and class index
def get_saliency_maps(model, images, class_index):
  # Get the gradient of the class output with respect to the input image
  input_images = tf.cast(images, tf.float32)
  with tf.GradientTape() as tape:
    tape.watch(input_images)
    output = model(input_images)
    class_output = output[:, class_index]
  gradient = tape.gradient(class_output, input_images)

  # Compute the saliency maps
  saliency_maps = tf.math.reduce_max(tf.math.abs(gradient), axis=-1)

  # Normalize the saliency maps
  saliency_maps /= tf.math.reduce_max(saliency_maps, axis=(1, 2), keepdims=True)

  # Convert the saliency maps to numpy arrays
  saliency_maps = saliency_maps.numpy()

  # Return the saliency maps
  return saliency_maps
```

Následně se vytvoří dávka obrázků a zavolá se funkce get_saliency_maps() na pole obrázků s indexem třídy požadované třídy. Poté se mapy síly vlivu zobrazí pomocí funkce imshow() a uloží se do souboru pomocí funkce savefig().

```
# Create a batch of images
images = np.array([image, image1, image2, image3])

# Get the saliency maps for defined class
```

```
class_index = 156
saliency_maps = get_saliency_maps(model, images, class_index)

#Plot preprocessed original images
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(images[i])
    ax.set_axis_off()
plt.savefig('interpretation_methods/saliency_maps/output/original_images_'
+ str(class_index) + '.png')
plt.show()

# Plot the saliency maps
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
for i, ax in enumerate(axes.flat):
    ax.imshow(saliency_maps[i], cmap='jet')
    ax.set_axis_off()
plt.savefig('interpretation_methods/saliency_maps/output/saliency_maps_'   +
str(class_index) + '.png')
plt.show()
```

6.2.2. Testování a výsledky

Jak již bylo řečeno v kapitole 6.2., tak pro vygenerování map sil vlivu (salienční mapy) je nejpodstatnější, pro kterou třídu tyto mapy generujeme. Je možnost ale testovat i na dalším atributu. Pokud je vybrán model umělé neuronové sítě, pro které se mapy síly vlivu mají generovat, tak lze testovat na těchto dvou kritériích, která se dají v kódu upravit:

- index třídy,
- obrázky.

Nejdůležitějším, nejzajímavějším a nejvlivnějším kritériem, které se dá změnit, je index třídy, podle kterého se má vybraný obrázek či více obrázků klasifikovat. Pokud pro set obrázků vygenerujeme mapu síly vlivů a poté se změní třída a vygeneruje se mapa znovu, měla by se mapa síly vlivu lišit. To proto, že pro každou třídu jsou během rozhodování důležité jiné detaily obrázku.

6.2.2.1. Testování skupiny různých obrázků pro danou třídu

Pro toto testování generování map sil vlivů byla vybrána skupina čtyř obrázků, kde jsou zobrazeni:

- pes – blenheimský španěl,
- kočka – nespecifikovaný druh,
- kočka a pes – nespecifikované druhy,
- slon – slon africký.

Pro tuto skupinu obrázků budou generovány mapy sil vlivů oproti těmto třídám:

- 156 – blenheimský španěl,
- 281 – kočka,
- 386 – slon africký.

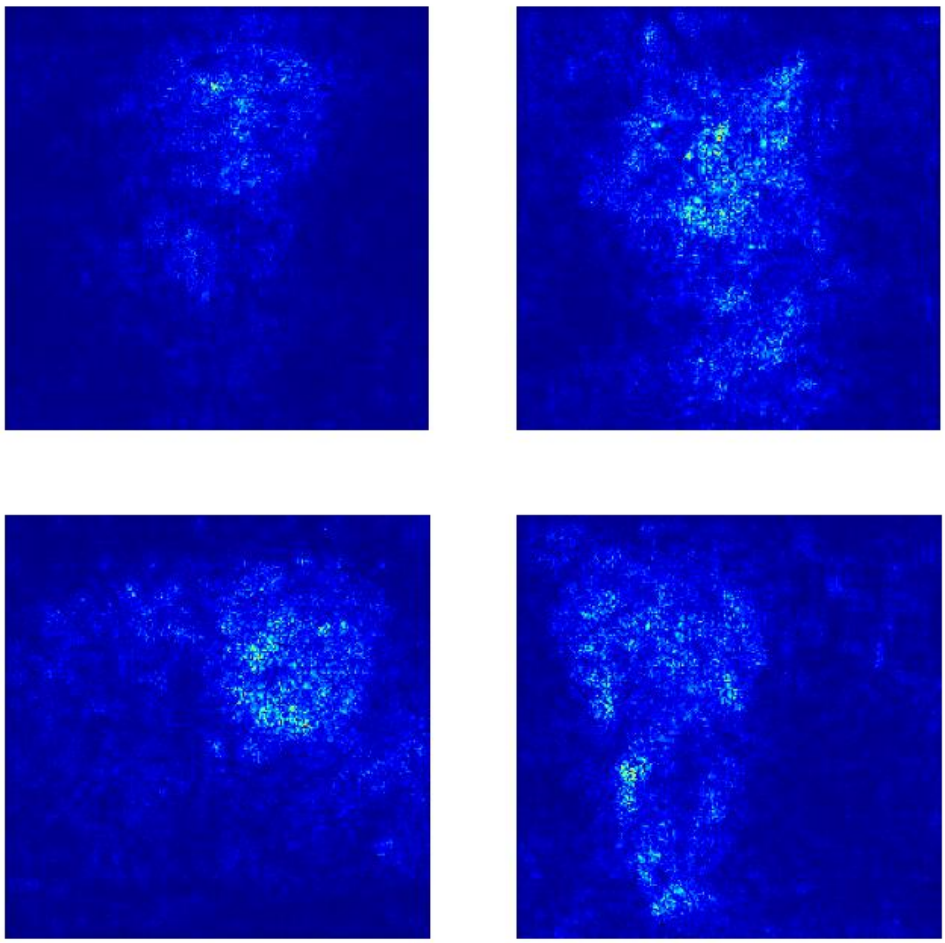
Jak bylo již v kapitole 6.2.1. vysvětleno, obrázky jsou načteny a pak předzpracovány. Obrázky po předzpracováním vypadají takto:

Obrázek 29 – Skupina různých obrázků po zpracování

Je potřeba si prvně uvědomit, že obrázky byly předzpracovány pomocí knihovny TensorFlow a její nadstavby Keras, a proto mají tmavší barvu. Úplně původní obrázky nejsou takto tmavé. Poté jsou tyto předzpracované obrázky uloženy do pole a jsou pro ně vygenerovány mapy sil vlivů oproti požadované třídě.

6.2.2.1.1. Třída 156 - blenheimský španěl

Jako první byla použita třída s indexem 156, která je používána pro klasifikování psa, konkrétně plemena blenheimský španěl.



Obrázek 30 – Mapy síly vlivu pro třídu 156 – blenheimský španěl

Na mapách sil vlivu z obrázku 30 je vidět, že na obrázku psa (první obrázek) byla nejdůležitější oblast rozhodování jeho hlava. Konkrétněji pak oblast mezi čumákem a očima. Pokud je oblast důležitá, tak má světlejší barvu. Čím světlejší, tím důležitější oblast je. Pokud je oblast nedůležitá, tak má tmavou barvu.

U druhého obrázku, kočky byla nejdůležitější oblast hlavy, především čumáku. Je dobré si také povšimnout toho, jak jsou z vygenerované mapy krásně vidět uši a vlastně celkový tvar hlavy. Další důležitou částí byl i hrudník.

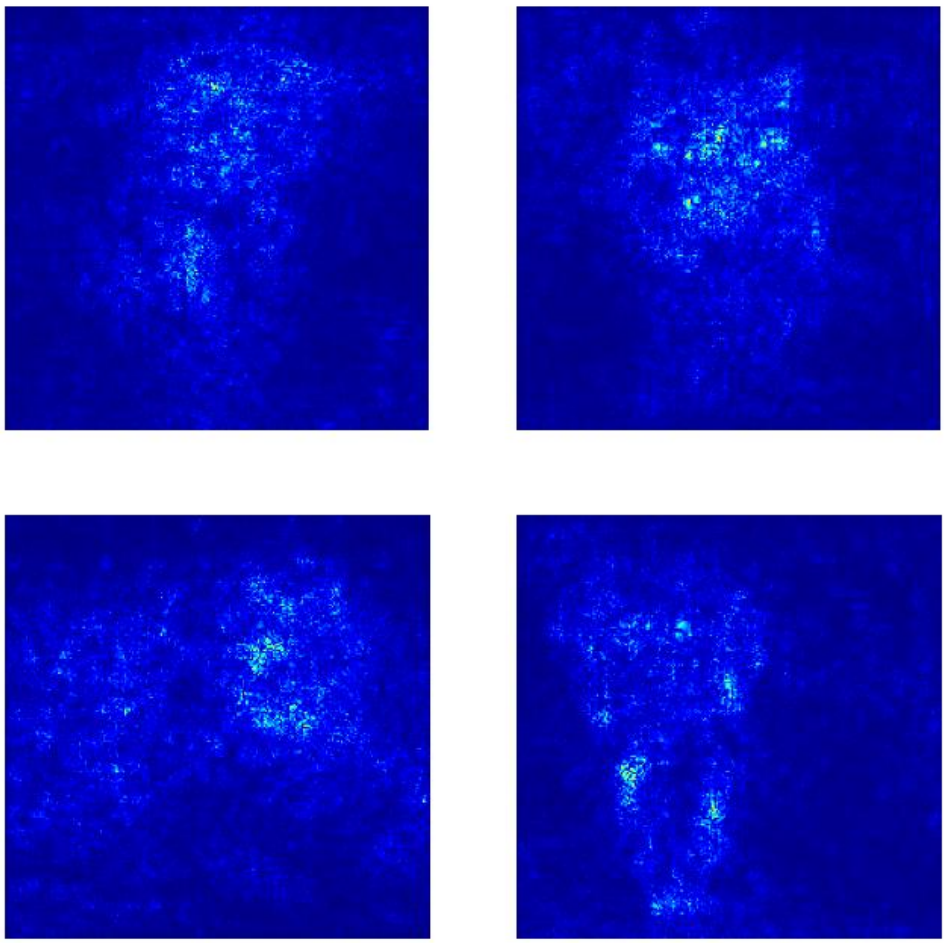
Třetí obrázek, který obsahuje jak psa, tak ale i kočku, se docela správně zaměřil na tu část, kde se vyskytuje pes. U něj byla zase důležitá část kolem čumáku, která se zdá být pro tuto

třídu celkem důležitým prvkem. Pokud nahlédneme na druhou stranu, kde je kočka, nějaké vlivné oblasti jsou zde také, ale nemají takovou váhu jako strana, kde se vyskytuje pes.

U posledního obrázku slona afrického je vidět, že při rozhodování byla důležitá celá postava slona. Cože je společný rys s obrázkem kočky. Dále byla také podstatná oblast obou klů a levé nohy. Je zde také možné, podobně jako u obrázku kočky, vidět zvýrazněnou oblast uší.

6.2.2.1.2. Třída 281 - kočka

Další byla použita třída s indexem 281, která je používána pro klasifikování kočky. V anglickém znění jde o třídu „tabby“, českém překladu mourek. Tato třída tedy byla využita pro klasifikaci kočky všeobecně. Pro tuto třídu byly vygenerovány tyto mapy sil vlivů:



Obrázek 31 – Mapy síly vlivu pro třídu 156 – kočka

První mapa síly vlivů ukazuje, že pro obrázek psa byla důležitá oblast hlavy a čumáku. Což je stejné jako u předchozího testování s třídou blenheimského španěla. Co je zde ale navíc, je hlavně oblast levé přední nohy a poté také tělo.

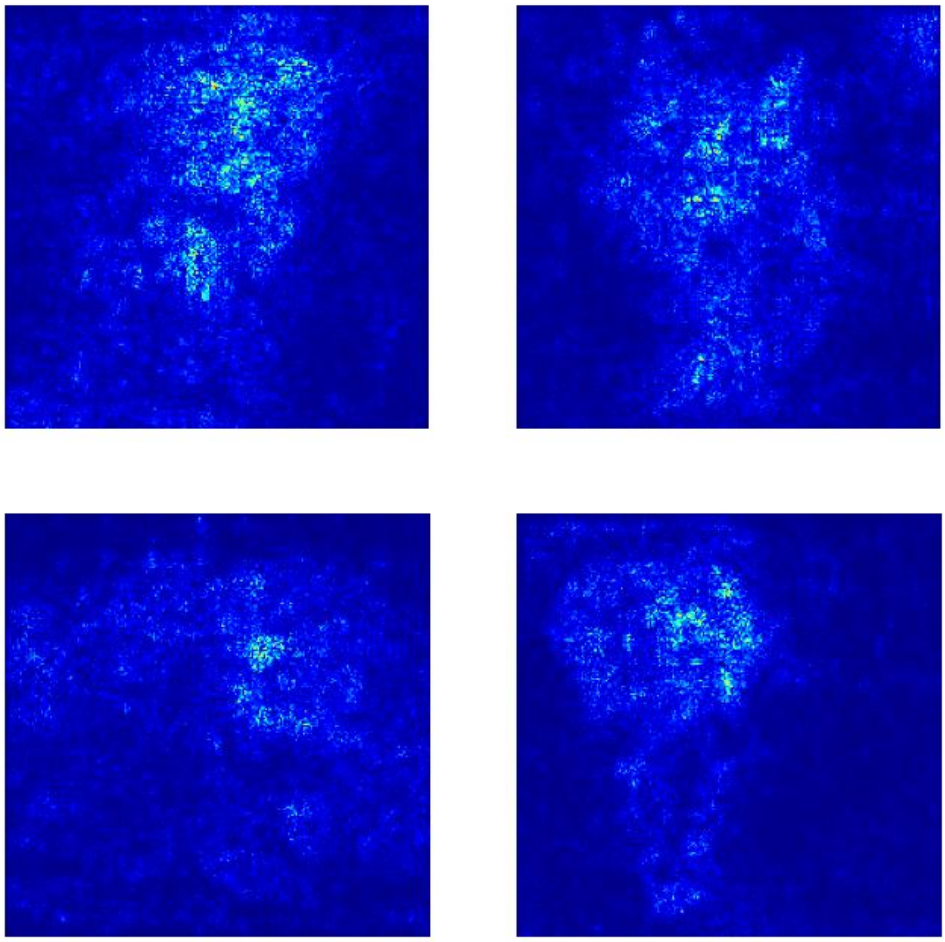
Druhý obrázek s kočkou je více méně podobný mapě síly vlivu s třídou blenheimského španěla. Jen oblast čumáku je celkově méně aktivní, jinak je mapa velice podobná.

U třetího obrázku psa a kočky je strana se psem zase velice podobná jako u předchozí třídy. Strana kočky je ale u této třídy mnohem důležitější. I přesto je při rozhodování stále důležitější strana psa.

Co se týče poslední mapy síly vlivů pro slona, tak je skoro totožná jako pro třídu blenheimského španěla. Podstatná oblast obou klů a levé nohy i pravé nohy. A je zde vidět, podobně jako u předchozí třídy, oblast uší.

6.2.2.1.2. Třída 386 – slon africký

Pro třetí, poslední test v rámci různých obrázků, byla vybrána třída slona afrického. Vzniklou mapu síly vlivů je možné prohlédnout si níže na obrázku 32.



Obrázek 32 – Mapy síly vlivu pro třídu 386 – Slon africký

Celkově se mapy síly vlivu pro třídu slona afrického velice liší od předchozích dvou analyzovaných objektů (blenheimský španěl, mourek). První obrázek psa je celý mnohem „důležitější“, to hlavně v části hlavy. Podobně je na tom i obrázek kočky.

Třetí obrázek kočky a psa je velice podobný předchozí třídě, ale strana kočky je tentokrát méně důležitá.

Oproti předchozím testovaným třídám je u posledního obrázku slona afrického mnohem důležitější jeho hlava a poté nohy. Kdežto paradoxně již nejsou tak důležité kly. Zde by se asi dalo očekávat, že pro třídu slona afrického budou kly důležité, ale jak můžeme vidět, tak v tomto rozhodovacím procesu nehrály tak důležitou roli. Může to být například tím, že neuronová síť pro třídu slona afrického kly očekává. Jsou pro ni tedy důležitější tvary a rysy v oblasti hlavy, které poté pomůžou slona afrického lépe identifikovat od jiných druhů slona.

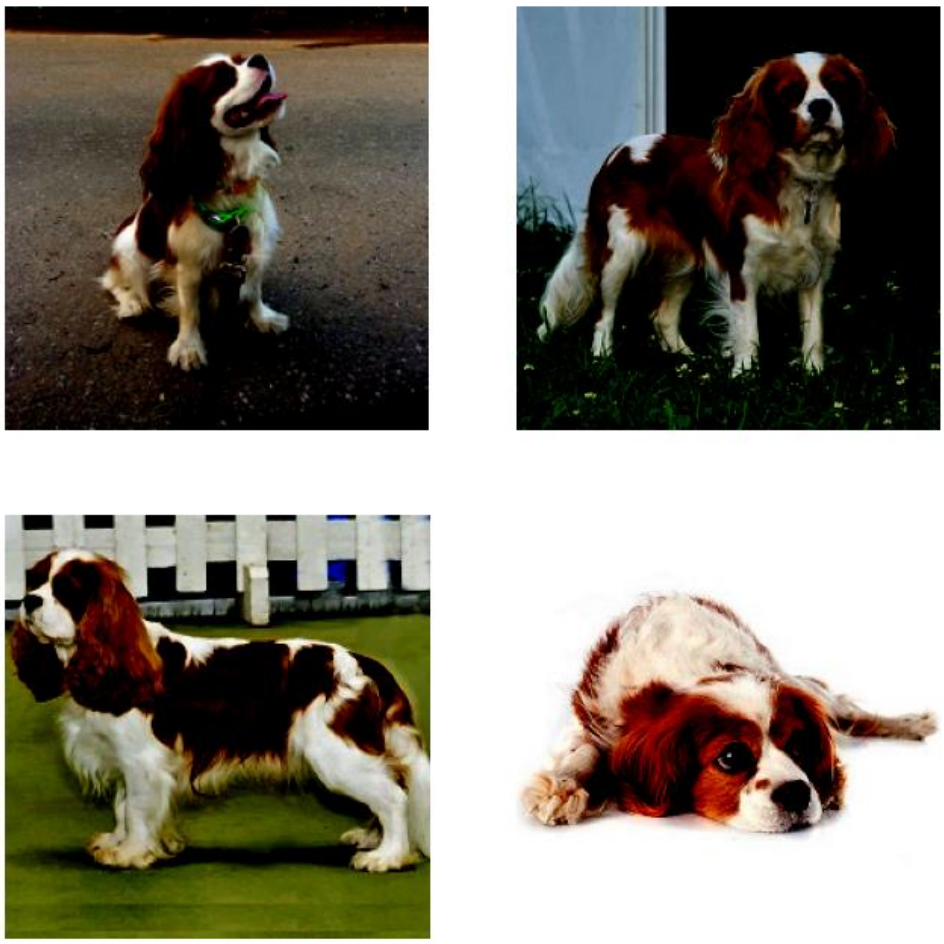
6.2.2.2. Testování skupiny stejných obrázků pro danou třídu

Pro toto testování generování map sil vlivů byla vybrána skupina čtyř obrázků, kde na všech je zobrazen pes – blenheimský španěl. Pro tuto skupinu obrázků budou generovány mapy sil vlivů oproti těmto třídám:

- 156 – blenheimský španěl,
- 281 – kočka,
- 386 – slon africký.

Očekávání jsou taková, že různé třídy pro tuto sadů obrázků budou vykazovat jiné vzory v mapách sil vlivů nebo alespoň nějakým způsobem rozdílné. Třídy 156 a 281 budou ale pravděpodobně velice podobné.

Předzpracované obrázky pro tento test vypadaly takto:

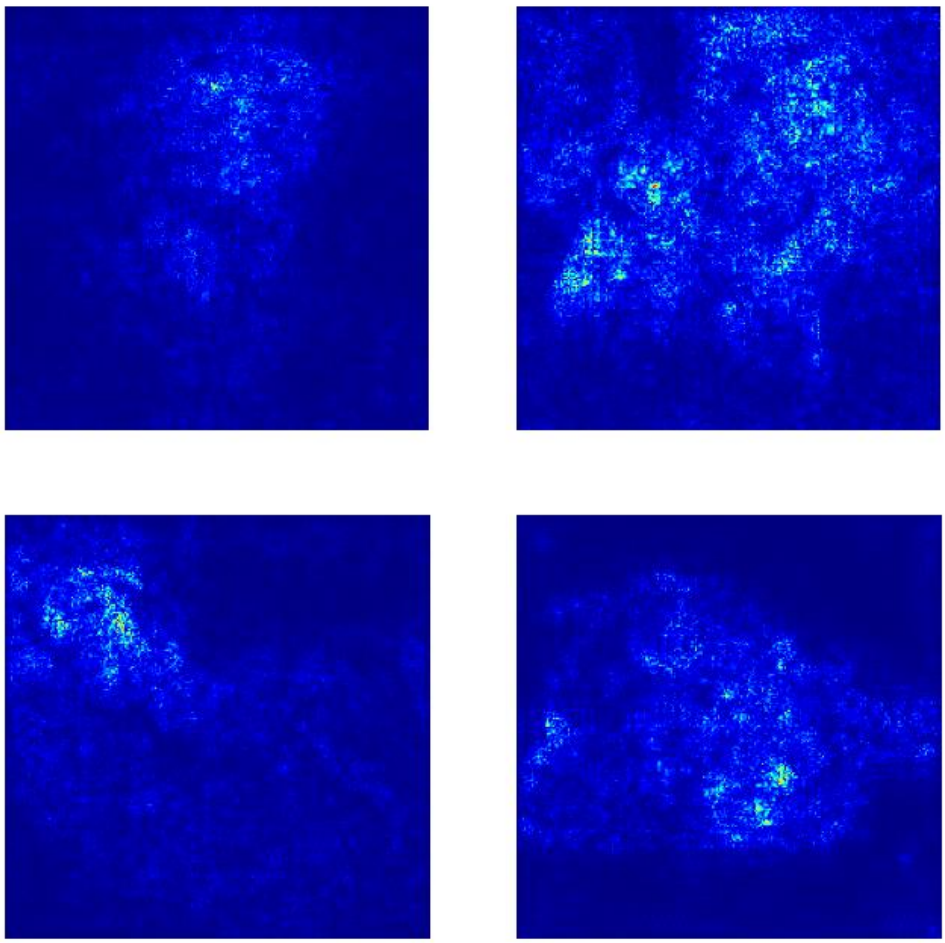


Obrázek 33 – Skupina stejných obrázků po zpracování

Znovu je potřeba nezapomenout, že obrázky na obrázku 33 jsou tmavší než původní obrázky, protože na ně byla použita metoda pro předzpracování obrázků.

6.2.2.2.1. Třída 156 - blenheimský španěl

Zde byla použitá třída s indexem 156, která má za úkol klasifikaci psího plemene blanheimského španěla. Obrázky s mapami sil vlivů vypadaly po vygenerování takto:

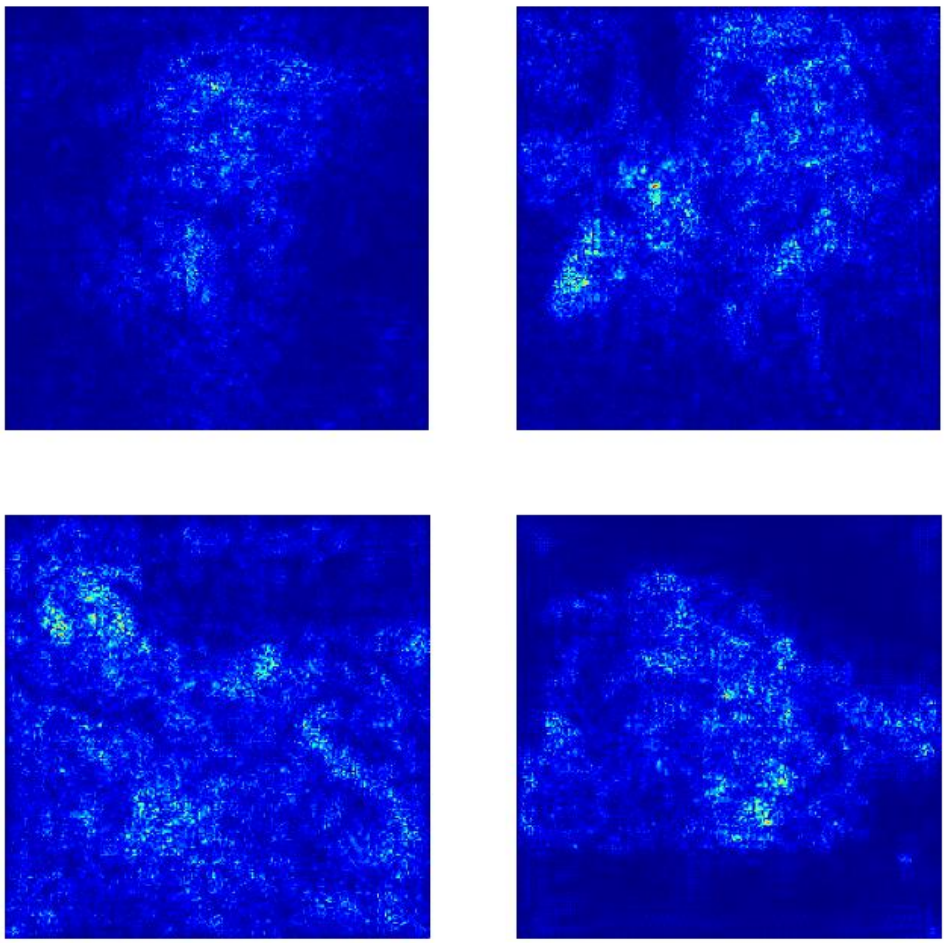


Obrázek 34 – Mapy síly vlivu pro třídu 156 – blenheimský španěl

Jak je možné si všimnout na obrázku výš, na všech obrázcích byla nejvlivnější část obličeje. U druhého obrázku hrálo velice důležitou roli také tělo. Tento jev je možné pozorovat i u posledního obrázku.

6.2.2.2.2. Třída 281 - kočka

U druhého testu pro stejnou skupinu obrázků byla použitá třída s indexem 281 – třída pro kočku. Mapy sil vlivů byly pro tuto třídu vygenerovány takto:

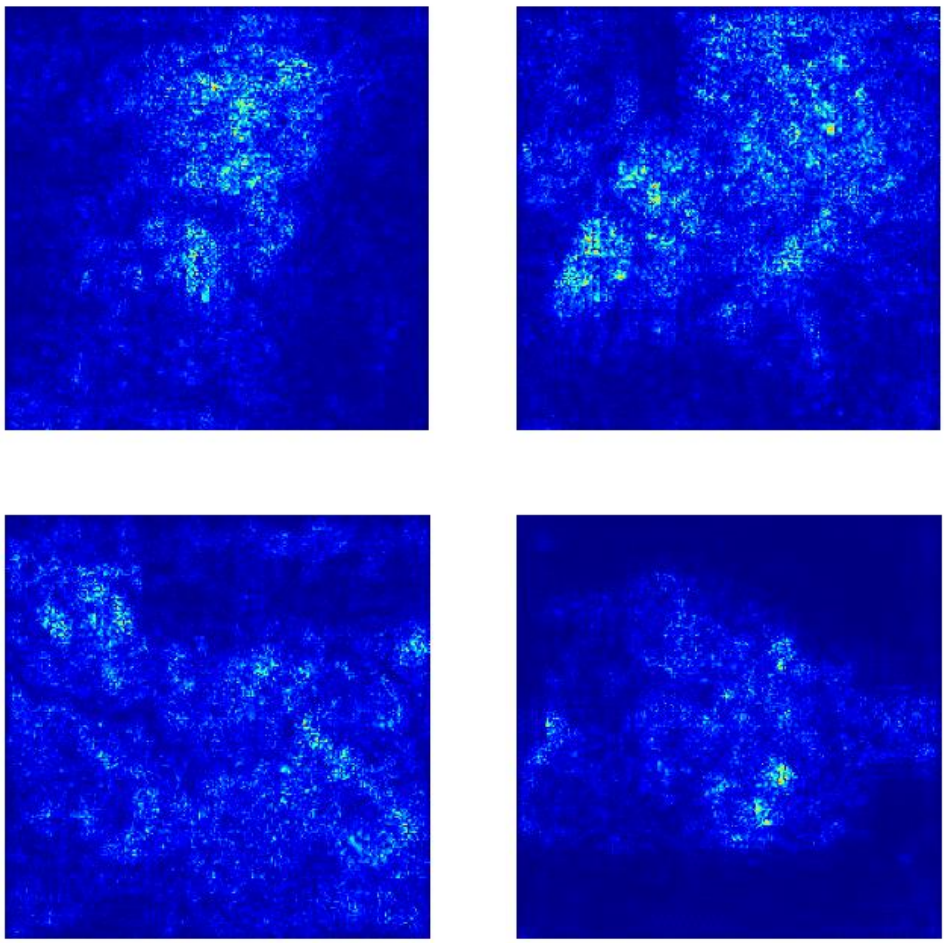


Obrázek 35 – Mapy síly vlivu pro třídu 156 – kočka

Na obrázku 35 je vidět, že vlivné oblasti zůstaly podobné pro obrázky dva a čtyři. U prvního a třetího obrázku se rozšířily na tělo psa. Hlavně u třetího obrázku došlo k rozšíření vlivných oblastí. Dalo by se říct, že se rozhodovalo na základě celého obrázku.

6.2.2.2.1. Třída 386 – slon africký

U posledního testu skupiny obrázku blenheimského španěla byla použitá třída slona afrického, tedy třídy s indexem 386. Mapy síly vlivů vypadaly takto:



Obrázek 36 – Mapy síly vlivu pro třídu 386 – Slon africký

U výsledných map sil vlivů pro třídu s indexem 386 je vidět, že kromě posledního obrázku měl na výběr vliv skoro celý obrázek. Opět jako v přechozích pokusech byla jednou z nejdůležitějších oblastí hlava.

6.3. Citlivost na překrytí (Occulsion sensitivity)

Citlivost na překrytí je technika používaná k identifikaci důležitých oblastí vstupního obrázku pro konkrétní výstupní třídu neuronové sítě prostřednictvím postupného překrytí nebo zakrytí částí vstupního obrázku a pozorování účinku na výstup sítě.

Základní myšlenka tohoto způsobu interpretace spočívá v iterativním zakrývání částí vstupního obrázku prázdnou částí nebo náhodným šumem. Po zakrytí dané části je

vypočítáno skóre dané třídy pro daný obrázek. Porovnáním skóre třídy pro původní obrázek s částečně zakrytým obrázkem lze identifikovat oblasti obrázku, které jsou pro rozhodnutí sítě nejdůležitější.

Tento postup se dá využít různými způsoby. V první řadě k identifikaci nejdůležitějších částí obrázku pro jeho úspěšnou klasifikaci. Poté například pro vyhodnocení robustnosti sítě vůči poruše na vstupu (například zakrytí kusu objektivu při pořízení fotografie) nebo ke generování nových obrázků se stejnými rysy jako obsahuje originální obrázek.

Je to velice jednoduchá, ale účinná technika, která nevyžaduje žádné úpravy architektury sítě. Jednou nevýhodou citlivosti na překrytí je, že může být výpočetně náročná, zejména pro velké obrázky nebo složité sítě.

6.3.1. Kód

Stejně jako v předchozích případech se načtou potřebné knihovny a obrázek se zpracuje tak, aby mohl vstoupit do modelu umělé neuronové sítě MobileNetV2.

Poté se definuje funkce citlivosti na překrytí (occlusion_sensitivity()), která bere vstupem model MobileNetV2, obrázek, index cílové třídy a velikost náplasti.

```
# Define occlusion sensitivity function
def occlusion_sensitivity(model, image, target_class, patch_size):
    # Get the predicted probabilities for the target class
    probs = model.predict(np.array([image]))
    target_prob = probs[0][target_class]

    # Initialize an empty heatmap
    heatmap = np.zeros(image.shape[:2])

    # Loop over patches of the image
    for row in range(0, image.shape[0], patch_size):
        for col in range(0, image.shape[1], patch_size):
            print('Row is ' + str(row))
            # Copy the original image
            occluded_image = np.copy(image)

            # Occlude the current patch
            occluded_image[row:row+patch_size, col:col+patch_size, :] = 0

            # Get the predicted probabilities for the occluded image
```

```
            probs = model.predict(np.array([occluded_image]))
            occluded_prob = probs[0][target_class]

            # Compute the difference in probabilities
            prob_diff = target_prob - occluded_prob

            # Update the heatmap with the probability difference
            heatmap[row:row+patch_size, col:col+patch_size] = prob_diff

    # Normalize the heatmap
    heatmap = (heatmap - np.min(heatmap)) / (np.max(heatmap) -
np.min(heatmap) + 1e-10)

    return heatmap
```

Jak je možné výše vidět, funkce occlusion_sensitivity() prochází postupně po obrázku a zakrývá ho „záplatou“. Pro každé takové zakrytí se vypočítává s jakou pravděpodobností je model schopen určit, že obrázek spadá pod danou třídu. Poté se vypočítá rozdíl pravděpodobností pro cílovou třídu mezi původním obrázkem a obrázkem se „záplatou“. Následně je postupně vytvořena heatmapa (mapa teploty), která je poté normalizována a vrácena na výstupu funkce.

Dále se definuje, jaká třída má být predikována a velikost „záplaty“, která má čtvercový tvar. Potom je zavoláno generování heatmapy (mapa teploty) pro citlivosti překrytí - metoda occlusion_sensitivity(). Nakonec se zobrazí graf s původním obrázkem, heatmapa na původní obrázku a samotná heatmapa. Celý tento graf se uloží do souboru a zobrazí se.

```
# Get the occlusion sensitivity heatmap
target_class = 156 # class index for "Blenheim spaniel"
patch_size = 16 # size of a patch
heatmap = occlusion_sensitivity(model, image, target_class, patch_size)

# Plot the original image and the heatmap
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(image)
ax[0].axis('off')
ax[0].set_title('Original Image')
ax[1].imshow(heatmap, cmap='jet', alpha=0.5)
ax[1].imshow(image, alpha=0.5)
ax[1].axis('off')
ax[1].set_title('Occlusion Sensitivity Heatmap')
ax[2].imshow(heatmap, cmap='jet')
```

```
ax[2].axis('off')
ax[2].set_title('Heatmap')
plt.savefig('interpretation_methods/occlusion_sensitivity/output/occlusion_
sensitivity_'+ str(patch_size)+'.png')
plt.show()
```

6.3.2. Testování a výsledky

Citlivost na překrytí je metoda interpretace, která si zakládá na iterativním překrývání vybraného obrázku jiným prázdným obrázkem nebo šumem. Po každé iteraci se vypočítá skóre dané třídy pro daný obrázek. Poté je překrývající část posunuta. Takto se pokračuje až na konec obrázku, který je překrýván.

Z toho vyplývají tři kritéria, podle kterých se dá testovat:

- obrázek,
- velikost překrývajícího obrázku,
- index třídy.

Změna obrázku jde ruku v ruce se změnou indexu, kdy změna jednoho nebo druhého se vzájemně ovlivňuje. Co se týče velikosti překrývajícího obrázku, tak se jedná o parametr, jehož změna by měla nejzásadněji ovlivnit generování mapy.

6.3.2.1. Velikost překrývajícího obrázku

V kódu, který je vysvětlen v kapitole 6.3.1., je proměnná „patch_size“, která definuje, jak velký bude obrázek, který bude překrývat původní obrázek. Pokud bude daná proměnná mít hodnotu například 16, bude to znamenat, že obrázek použitý k překrytí bude mít tvar čtverce o obsahu 256 pixelů. Předmětem tohoto testování tedy bude testovat různé velikosti překrývajícího obrázku. Překrývající obrázek bude testován na těchto velikostech:

- 16 – čtverec s obsahem 256 pixelů,
- 8 – čtverec s obsahem 64 pixelů,
- 4 – čtverec s obsahem 16 pixelů,
- 2 – čtverec s obsahem 4 pixely,

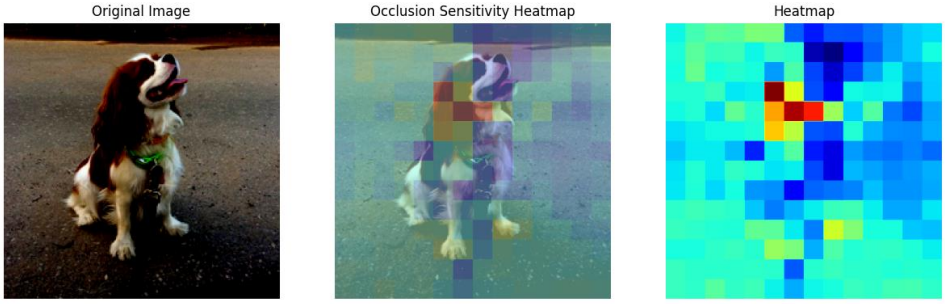
- 1 – čtverec s obsahem 1 pixelu.

Tyto velikosti budou testovány pro dvě klasifikační třídy:

- 156 – blenheimský španěl,
- 281 – kočka.

6.3.2.1.1. Velikost překrývajícího obrázku – 16

Generování heat map citlivosti na překrytí při velikosti překrývajícího obrázku 16 byla velice rychlá. Vzhledem k tomu, že obrázek, který překrýváme má vždy velikost 224*224 pixelů, tak se jednalo o velice rychlý proces. Vygenerované heat mapy pro třídu blenheimského španěla a kočky vypadaly takto:



Obrázek 37 – Mapa citlivosti překrytí s velikostí 16 – pro třídu blenheimský španěl

Obrázek 38 – Mapa citlivosti překrytí s velikostí 16 – pro třídu kočka

Jak je možné vidět na obrázku 37 a 38, tak obě heat mapy citlivosti překrytí nejsou ostré. Jsou více rozpixelované a původní obrázek na nich není jasně vidět. To je z důvodu toho,

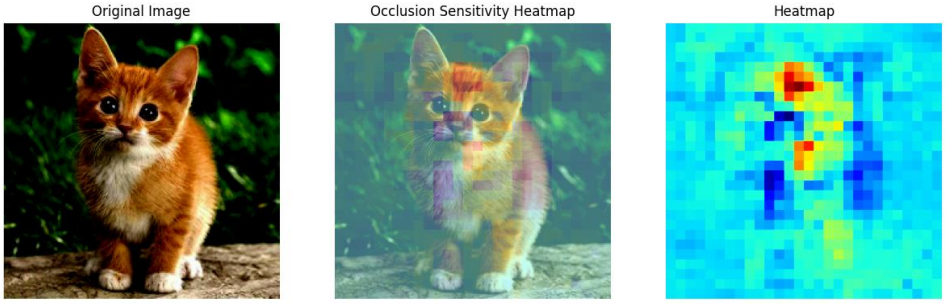
že překrývající obrázek je oproti překrývanému obrázku veliký. To vede k tomu, že se na obrázku nezachytí jemnější detaily, ale spíš jen obecné tvary, které poté vedou k rozpoznání obrázku. Pokud bude ale překrývající obrázek moc veliký, může mít algoritmus potíže s rozpoznáním objektu, protože zakrytá oblast může obsahovat důležité rysy nebo kontext, který je nezbytný pro přesné rozpoznání.

Na obou případech je ale vidět, že byly důležité hlavně oblasti hlavy. V případě kočky byla zajímavá také oblast obou levých nohou.

6.3.2.1.2. Velikost překrývajícího obrázku - 8

V případě velikosti překrývajícího obrázku o velikosti 8 bylo generování heat map citlivosti na překrytí také rychlé. Jak vypadaly vygenerované heat mapy pro dvě vybrané třídy je možné vidět níž na obrázku 39 a 40.

Obrázek 39 – Mapa citlivosti překrytí s velikostí 8 – pro třídu blenheimský španěl



Obrázek 40 – Mapa citlivosti překrytí s velikostí 8 – pro třídu kočka

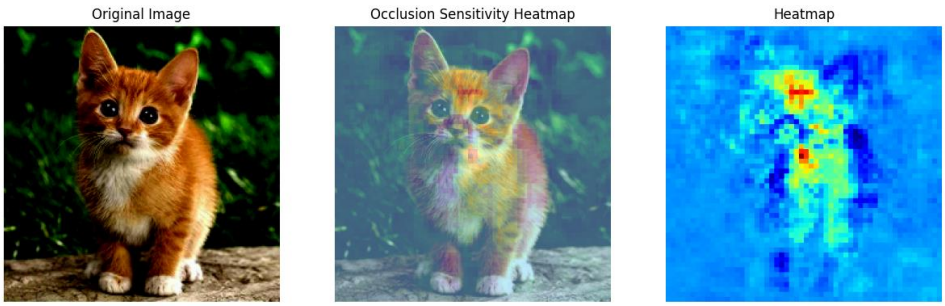
Obě heat mapy nabraly na ostrosti. U obrázku kočky zůstala zajímavou stále oblast hlavy a levých nohou. Kdežto u obrázku psa se stal zajímavým celý obrázek. Může to být

z důvodu barvy pozadí nebo něčeho jiného. Nejdůležitější je, že stále byla nejdůležitější oblast hlavy a teď i předních pacek.

6.3.2.1.3. Velikost překrývajícího obrázku - 4

Velikost 4 překrývajícího obrázku už zapříčinila to, že se heat mapa citlivosti překrytí generovala docela dlouho. Samotné heatmapy už začaly být ostřejší a dá se v nich vypozorovat původní tvar zvířete z originálního obrázku – alespoň v případu kočky.

Obrázek 41 – Mapa citlivosti překrytí s velikostí 4 – pro třídu blenheimský španěl



Obrázek 42 – Mapa citlivosti překrytí s velikostí 4 – pro třídu kočka

Na obrázku 41, kde je vygenerována heat mapa blenheimského španěla, pozadí obrázku už není tak důležité, což je logické, protože čím menší překrývající obrázek, tím více heat mapa citlivosti překrytí reaguje na detaily. Stále je důležitá hlava, hlavně levé ucho a čumák. Mapa citlivosti kočky viz obrázek 42, je víceméně stále stejná, jen je heat mapa detailnější jako v případě psa.

6.3.2.1.4. Velikost překrývajícího obrázku – 1

Při velikosti překrývajícího obrázku 1 se heat mapy generovaly velice dlouho. Přibližně v rámci desítek minut. Jak tyto mapy vypadly je možné si prohlédnout na obrázcích níž.

Obrázek 43 – Mapa citlivosti překrytí s velikostí 1 – pro třídu blenheimský španěl

Obrázek 44 – Mapa citlivosti překrytí s velikostí 1 – pro třídu kočka

Na obrázku 43 se psem je z nějakého důvodu důležité pozadí. Očekávání pro malý obrázek překrývání byla taková, že pozadí nebude vůbec důležité a budou důležité jen detaily. Pro psa i kočku jsou důležitá jistá místa na hlavě a jejich detaily. U psa je navíc stále důležité i pozadí. Jak už bylo zmíněno dříve, může to být z důvodu barvy pozadí nebo jeho vzoru.

Obrázek 44 s kočkou krásně zobrazuje podobiznu kočky na její heat mapě. Důležitý pro ni byl hlavně čumák. V případě psa platí to samé.

Z pozorování vyplývá, že menší velikosti zakrývajících obrázků mohou zvýšit citlivost tím, že algoritmu umožní zachytit jemnější detaily objektu. Je tomu tak proto, že menší

„záplaty“ schovají méně kontextových informací. Větší zakrývající obrázky pomohou odhalit, jaké abstraktnější informace jsou na obrázku důležité pro predikce.

Pokud je velikost zakrývajícího obrázku příliš malá, výsledná mapa citlivosti může být zašuměná a obtížně interpretovatelná. Na druhou stranu, pokud je velikost tohoto obrázku příliš velká, může být obtížné identifikovat specifické vlastnosti nebo oblasti vstupu, které jsou pro výstup nejdůležitější.

6.4. Maximalizace aktivace (Activation maximization)

Maximalizace aktivace je způsob interpretace, který se používaná k vytváření uměle vytvořených obrazů, které maximálně aktivují konkrétní neuron, celou vrstvu neuronů nebo celou umělou neuronovou sít. Základní myšlenkou této interpretační metody je optimalizovat vstupní obraz tak, aby byla aktivace cílových neuronů maximalizována, přičemž se zároveň udržovala vizuální pravděpodobnost obrazu.

Maximalizace aktivace může být použita k různým účelům, jako je vizualizace vlastností nebo vzorů, na které je určitá vrstva neuronů citlivá nebo vytváření nových obrazů, které jsou podobné těm, které dané neurony nejvíc aktivují. Maximalizace aktivace byla použita například pro generování obrazů halucinací, které maximálně aktivují neurony spojené se sněním v mozku. V otázce interpretace umělé neuronové sítě byla použita k vizualizaci interních reprezentací a k lepšímu porozumění tomu, jak síť zpracovává vstupní data.

Jedním z omezení maximalizace aktivace je, že může generovat obrazy, které nejsou vždy realistické nebo smysluplné. Nicméně stále může poskytnout náhledy do toho, jak síť rozhoduje a jaké vlastnosti nebo vzory hledá ve vstupu.

6.4.1. Kód

Jako u všech metod interpretace jsou opět nejprve naimportovány potřebné knihovny a načten model MobileNetV2. Poté se definuje vrstva, pro kterou se má maximalizace aktivace provést a také velikost obrázku, který má být generován. Zároveň jsou definovány parametry pro počet iterací a ‚step_size‘. Parametr ‚step_size‘ je v podstatě rychlost učení. Jinak řečeno tento parametr určuje, jak rychle se bude obrázek upravovat, aby co nejvíc aktivoval danou vrstvu.

```
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# Load the MobileNetV2 model
model = tf.keras.applications.MobileNetV2()

# Define the layer to visualize
layer_name = 'Conv1'
layer = model.get_layer(name=layer_name)

# Define the size of the generated image
img_size = 224

# Define the number of iterations and the step size for optimization
iterations = 100
step_size = 1.0
```

Dále je definována funkce activation_loss() pro výpočet aktivací zvolené vrstvy pro daný obrázek. Je také definována funkce generate_image(), která generuje obrázek tím, že maximalizuje aktivaci zvolené vrstvy pomocí optimalizéru (učícího algoritmu). Tato funkce inicializuje obrázek s náhodným šumem a iterativně ho aktualizuje tím, že vypočítá gradient ztráty vzhledem k obrázku a aplikuje ho na obrázek pomocí optimalizéru. Postupně je skrze iterace obrázek upravován tak, aby co nejvíce aktivoval danou vrstvu. Vygenerovaný obrázek je nakonec oříznut tak, aby hodnoty pixelů zůstaly v rozmezí [0, 1].

```
# Define the function to generate the image by maximizing the activation of the chosen layer
def generate_image(layer, img_size, iterations, step_size):
    # Initialize the image with random noise
    img = tf.Variable(tf.random.uniform((1, img_size, img_size, 3), 0.0, 1.0))

    # Define the optimizer
    optimizer = tf.keras.optimizers.Adam(learning_rate=step_size)

    # Run the optimization for the specified number of iterations
    for i in range(iterations):
        with tf.GradientTape() as tape:
            # Compute the loss
            loss = activation_loss(layer, img)

        # Compute the gradients of the loss with respect to the image
        gradients = tape.gradient(loss, img)
        gradients /= tf.sqrt(tf.reduce_mean(tf.square(gradients))) + 1e-5
```

```
        # Update the image
        optimizer.apply_gradients([(gradients, img)])

        # Clip the image to keep the pixel values within the [0, 1] range
        img.assign(tf.clip_by_value(img, 0.0, 1.0))

    # Return the generated image
    return img.numpy()[0]
```

Nakonec je funkce generate_image() volána s vybranou vrstvou, velikostí obrázku, počtem iterací a velikostí kroku. Výsledný obrázek je zobrazen pomocí Matplotlib a je uložen do souboru.

```
# Generate the image by maximizing the activation of the chosen layer
img = generate_image(layer, img_size, iterations, step_size)

# Display the generated image
plt.imshow(img, cmap='jet')
plt.savefig('interpretation_methods/activation_maximization/output/activation_maximization_' + layer_name + '.png')
plt.show()
```

6.4.2. Testování a výsledky

Z kapitol 6.4. a 6.4.1. plyne, že využití této metody se dá testovat na několika různých kritériích, která se dají měnit. Je důležité si uvědomit, že se tato kritéria mohou také navzájem ovlivňovat, takže jejich správný výběr může být pro maximalizaci aktivace velice důležitý. Mezi nejdůležitější kritéria patří:

- step_size,
- iterations,
- layer_name.

Dala by se testovat i změna těchto dalších parametrů:

- optimizer,
- image.

Pro účely tohoto testování budou testovány hlavně první tři parametry. Parametr ‚step_size' je vlastně rychlost učení pro vybraný učící algoritmus neboli představuje velikost aktualizací aplikovaných na obrázek v každé iteraci optimalizace. Vyšší hodnota parametru ‚step_size' vede k větším aktualizacím, což může mít za následek rychlejší konvergenci k požadovanému obrázku, ale s rizikem přestřelení optimálního obrázku. Naopak nižší hodnota ‚step_size' zpomaluje konvergenci, ale může zlepšit stabilitu.

Dalším je parametr ‚iterations', který jak napovídá z jeho názvu, určuje počet iterací provedených k vygenerování co nejoptimálnějšího obrazu. V každé iteraci je obraz aktualizován na základě ztráty vzhledem ke generovanému obrazu. Zvýšení počtu iterací zvedá počet optimalizačních kroků, který poté může vést ke zpřesnění obrazu, aby se co nejvíce maximalizovala aktivace zvolené vrstvy. Vyšší počet iterací poté logicky prodlužuje výpočetní čas a náročnost operace.

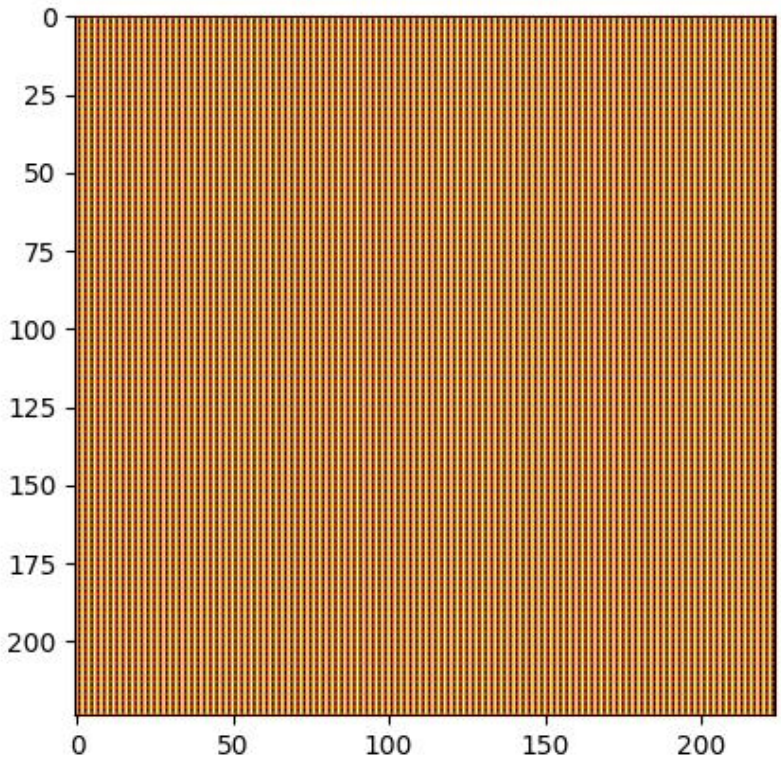
Posledním z prvních třech parametrů je parametr ‚layer_name', který definuje, pro jakou vrstvu z modelu umělé neuronové sítě MobileNetV2 se bude maximalizovat aktivace a generovat obraz.

6.4.2.1. Maximalizace aktivace pro vrstvu Conv1

Pro první test byla vybrána skrytá vrstva modelu MobileNetV2 s názvem Conv1. Parametry pro funkci generate_image() byly vybrány takto:

- layer_name = 'Conv1',
- iterations = 100,
- step_size = 1.0.

Po spuštění generování obrázku pro metodu maximalizace aktivace vypadal obrázek takto:



Obrázek 45 – Maximalizace aktivace – vrstva Conv1

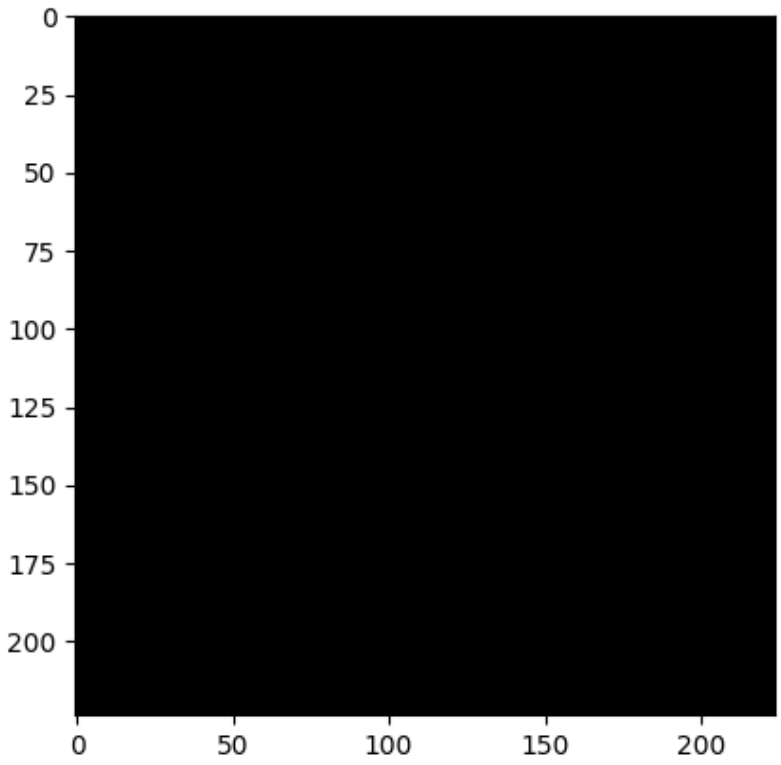
Jak je možné vidět na obrázku 45, maximalizace aktivace pro tuto vrstvu je plocha vytvářející hrany. Z toho tedy plyne, že daná vrstva modelu umělé neuronové sítě MobileNetV2 se zaměřuje převážně na hrany a ty ji nejvíce aktivují.

6.4.2.2. Maximalizace aktivace pro vrstvu Conv1_relu

Pro další test byla využita třetí skrytá vrstvá Conv1_relu. Parametry funkce pro generování obrázku maximalizace aktivace byly nastaveny:

- layer_name = 'Conv1_relu',
- iterations = 100,
- step_size = 1.0.

Vygenerovaný obrázek pro tuto vrstvu byla ale jenom černá plocha. Obrázek je možné si prohlédnout níž.



Obrázek 46 – Maximalizace aktivace – vrstva Conv1_relu

Zde nastává problém. Generovaný obrázek by neměl být celý černý. To může být z několika důvodů:

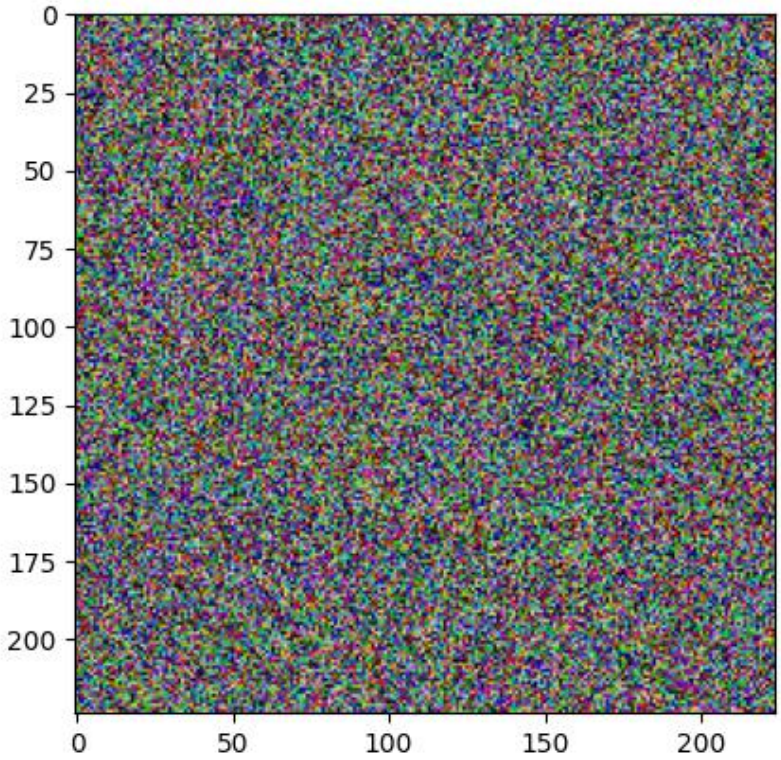
- Nedostatečný počet iterací – pokud je počet iterací (parametr ‚iterations') příliš nízký, proces optimalizace nemusel mít dostatek iterací ke konvergování a vytvoření obrazu.
- Příliš vysoká hodnota učení – vysoká hodnota ‚step_size' může způsobit, že proces optimalizace rychle přestřelí optimální obrázek, což má za následek, že hodnoty pixelů překračují zobrazitelný rozsah (0-1). V důsledku toho se obrázek může jevit jako černý, protože hodnoty pixelů jsou během procesu optimalizace oříznuty na rozsah [0, 1].

6.4.2.2.1. Úprava parametrů Conv1_relu

Vzhledem k důvodům vysvětleným na konci předchozí kapitoly byla jako první snížena rychlost učení, tedy parametr ‚step_size' na 0,1, a parametr ‚iterations' zůstal stejný, tedy

hodnota 100. Toto ale nevedlo k lepšímu výsledku a stále byl vygenerován černý obrázek. Stejný výsledek byl i při změně ‚step_size' na parametr 0,01.

Po změně ‚step_size' na ještě menší hodnotu 0,001, byl vygenerován obrázek, který vypadal takto:



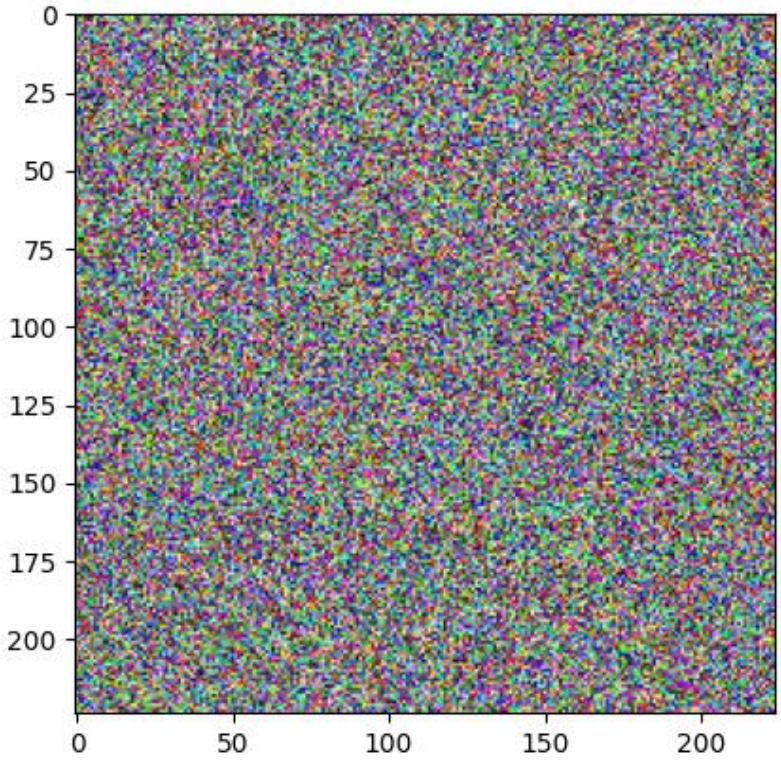
Obrázek 47 – Maximalizace aktivace – vrstva Conv1_relu

Tento obrázek pro maximalizaci aktivace dané vrstvy vypadá oproti obrázku 47 naprosto náhodně. Očekávání bylo takové, že by bylo možné v něm spatřit alespoň nějaký vzor nebo něco podobného. Nic takového se na obrázku pozorovat nedá. Proto byl porovnán vůči náhodně vygenerovanému obrázku, na kterém se maximalizace aktivace prováděla. Do kódu byl přidán následující kód, který obrázek vykreslil a uložil:

```
# Display random noise image
plt.imshow(img.numpy()[0], cmap='jet')

plt.savefig('interpretation_methods/activation_maximization/output/activati
on_maximization_' + layer_name + '_generated' +'.png')
plt.show()
```

Původní obrázek vypadal takto:



Obrázek 48 – Náhodně generovaný obrázek pro maximalizaci aktivace

Jak je vidět na obrázku 48, tak je naprosto stejný jako obrázek 47, jen je trochu světlejší. To znamená, že se vůbec nic neděje, obrázek se nijak nemění. Jen je z nějakého důvodu tmavší. Další možností je zkusit více iterací. Vzhledem k tomu, že pro rychlost učení s hodnotou 0,001 zůstal alespoň nějaký obrázek, zkusíme více iterací pro tuto hodnotu.

Pro ‚iterations' s hodnotou 500 zčernal obrázek ještě víc. Pro hodnotu 1000 byl zase naprosto černý. Obrázek pro maximalizaci aktivace této vrstvy vypadal stejně jako na obrázku 46.

Po dalším zkoumání bylo zjištěno, že vrstva Conv1_relu používá, jak již z jejího názvu vyplývá, aktivační funkci ReLU. Tato aktivační funkce pouze odstraňuje záporné hodnoty a nahrazuje je nulou, což vede k nezáporným aktivacím. Tedy cokoliv je menší než nula se rovná nule. Při provádění maximalizace aktivace je cílem maximalizovat aktivaci zvolené

vrstvy. Pokud však vrstva již používá funkci aktivace ReLU, aktivace jsou ze své podstaty nezáporné a jejich další maximalizace nemusí vést k viditelným změnám v obraze. [47]

6.4.2.3. Maximalizace aktivace pro další vrstvy

Kvůli nemožnosti generování mapy maximalizovaných aktivací pro jakoukoliv vrstvu, která obsahuje aktivační funkci ReLU, nastává problém kódu uvedeného v kapitole 6.4.1. Tento kód totiž generuje obrázek v jisté velikosti a rozměru – což měly všechny tyto vrstvy společné. Pokud by byla použita jiná vrstva, tak bude mít jiný tvar, který se poté nedá jednoduše interpretovat jako obrázek, a tedy se ani nedá jednoduše generovat její mapa. „Obrázek“ by musel být generován v jiných velikostech a dimenzích. Proto implementace pro další vrstvy není možná a byla by velice časově náročná. Toto ukazuje problém implementace této interpretační metody.

6.4.2.4. Maximalizace aktivace pro celý model

Další a hlavně mnohem více využívána možnost maximalizace aktivace je její použití na celý model. Pro to je ale potřeba upravit kód do následující podoby. To funguje tak, že na začátku existuje nějaký obrázek, který je postupně upravován na obrázek, který je požadován. Zároveň ten obrázek musí co nejvíc aktivovat daný model pro jeho predikci.

6.4.2.4.1. Úprava kódu

Začátek kódu bude stejný jako v kapitole 6.4.1. Dále bude definována proměnná target_class, která bude obsahovat index třídy, pro kterou bude maximalizace aktivace prováděna.

```
# Define the target classification class
target_class = 156
```

Metoda activation_loss() bude mít teď navíc další vstupní parametr target_class, který bude značit to, pro jaký obrázek se bude ztráta počítat a co má generovaný obrázek napodobovat. Dále je nahrazen parametr layer za parametr model, protože se teď maximalizace aktivace počítá pro celý model umělé neuronové sítě MobileNetV2 a ne pro jednu vrstvu, jako tomu bylo v předchozích testech.

```
# Define the loss function to maximize the output of the target classification
class
def activation_loss(model, img, target_class):
    output = model(img)
    target_output = output[:, target_class]
```

```
    return tf.reduce_mean(target_output)
```

Poté bude změna také funkce generate_image(), která bude mít navíc znovu vstupní parametr target_class. Dále je podobně, jako u funkce activation_loss(), nahrazen parametr layer za model ze stejných důvodů.

```
# Define the function to generate the image by maximizing the activation of the entire network based on the target classification class
def generate_image(model, image, target_class, iterations, step_size):
    # Add a batch dimension to the image tensor
    image = np.expand_dims(image, axis=0)
    # Initialize the image
    image = tf.Variable(image)

    plt.imshow(image.numpy()[0])

plt.savefig('interpretation_methods/activation_maximization/output/VGG16/activation_maximization_whole_network_image_start_' + str(target_class) + '.png')
    plt.show()

    # Define the optimizer
    optimizer = tf.keras.optimizers.Adam(learning_rate=step_size)

    # Run the optimization for the specified number of iterations
    for i in range(iterations):
        with tf.GradientTape() as tape:
            # Compute the loss
            loss = activation_loss(model, image, target_class)
            print(str(loss))
            print(str(i))

        # Compute the gradients of the loss with respect to the image
        gradients = tape.gradient(loss, image)
        gradients /= tf.sqrt(tf.reduce_mean(tf.square(gradients))) + 1e-5

        # Update the image
        optimizer.apply_gradients([(gradients, image)])

        # Clip the image to keep the pixel values within the [0, 1] range
        image.assign(tf.clip_by_value(image, 0.0, 1.0))

    # Return the generated image
    return image.numpy()[0]
```

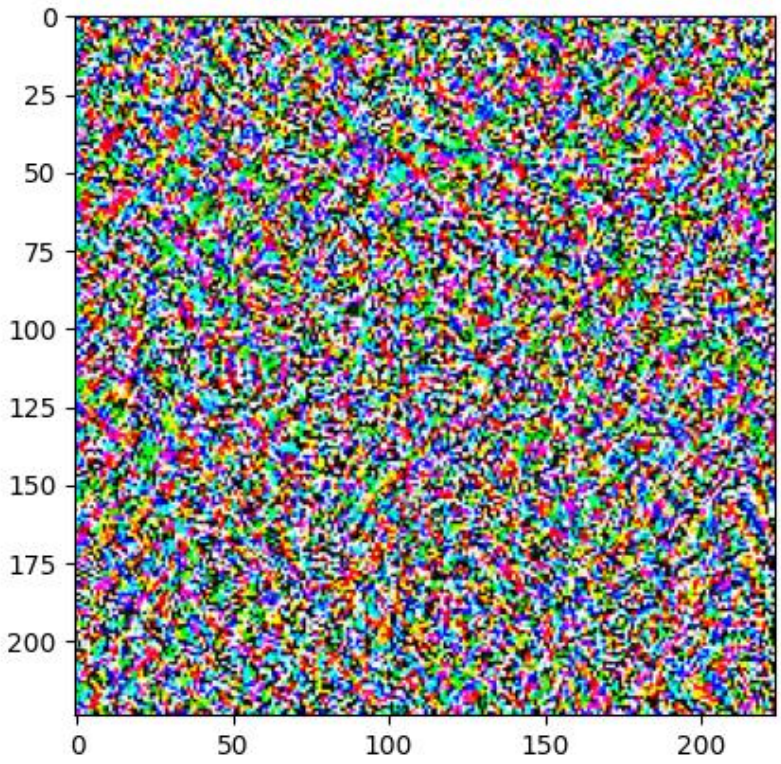
Nakonec je vygenerovaný obrázek vykreslen a uložen stejně, jako je tomu v kapitole 6.4.1.

6.4.2.4.2. Náhodně generovaný obrázek

Pro testování maximalizace aktivace byl použit náhodně vygenerovaný obrázek šumu. Obrázek byl velice podobný obrázku X34. Dále pak byly nastaveny tyto parametry:

- target_class - 156 - blenheimský španěl,
- iterations – 100,
- step_size - 0.1.

Třída s indexem 156 je třída blenheimského španěla, takže vygenerovaný obrázek by měl nějakým způsobem připomínat alespoň částečně tento druh psa. Obrázek maximalizace aktivace vypadal takto:



Obrázek 49 – Maximalizace aktivace – celý model

Na obrázku 49 není vůbec nic, co by připomínalo obrázek psa. Je zde ale možné spatřit jisté vzorce, které obsahují jakési pruhy, které se různě stáčí.

Jedním z důvodu by mohlo být málo iterací, ale i po jejich zvednutí na 1000 obrázek vůbec nenapodoboval psa. Dalším důvodem by tedy mohlo být, že obrázek maximalizace aktivace pro danou třídu není vůbec nijak podobný své třídě – je to totiž jen náhodně generovaný šum.

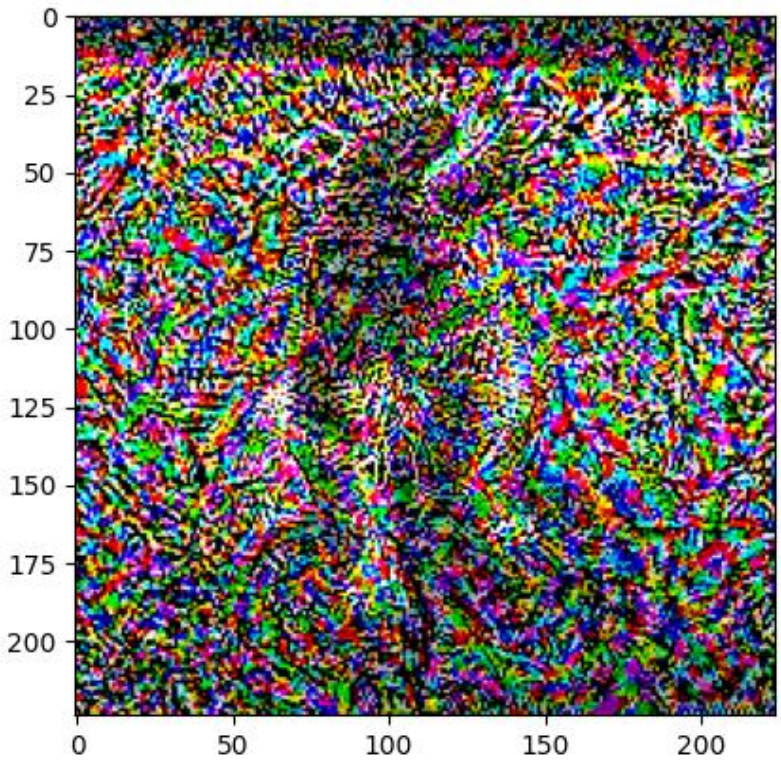
6.4.2.4.3. Obrázek odpovídající vybrané třídě

Ze závěru kapitoly 6.4.2.4.3. byl pro tento test vybírán nějaký obrázek, který není jen náhodně generovaný šum, ale jedná se o reálný obrázek, který odpovídá vybrané třídě s indexem 156 - blenheimský španěl. Vzhledem k upravenému kódu v kapitole 6.4.2.4.1. bude opět provedena další menší změna. Na začátku bude načten a předzpracován obrázek, na kterém bude maximalizace aktivace prováděna.

```
# Load an image
image = tf.keras.preprocessing.image.load_img('images/dog.jpg', target_size=(224, 224))
image = tf.keras.preprocessing.image.img_to_array(image)

# Preprocess the image
image = tf.keras.applications.mobilenet_v2.preprocess_input(image)
```

Jak načtený obrázek vypadal je možné vidět na obrázku 29, jedná se o první obrázek. Po generování obrázku maximalizace aktivace vypadal obrázek takto:



Obrázek 50 – Maximalizace aktivace – celý model

V pozadí je stále vidět původní obrázek psa. I přesto byla ztrátová funkce velká a obrázek na se na první pohled od původního nijak neliší. Vzorce točících se pruhů zmíněných v předchozím testu jsou teď mnohem víc viditelné v okolí předních tlapek a celkově ve spodní části obrázku. Dále to ale vypadá tak, že původní obrázek byl náhodně generován šum. Z toho výsledku by si pozorovatel neměl co vzít.

6.4.2.4.3. Změna modelu

Vzhledem k tomu, že maximalizace aktivace s vybranými parametry pro daný model umělé neuronové sítě MobileNetV2 skoro vůbec nefungovala, tak se další test pokusí využít implementaci dané interpretační metody na jiný model umělé neuronové sítě. Pokud vezmeme v potaz, jak velká a složitá je architektura umělé neuronové sítě MobileNetV2, tak pro tento test byla vybrána menší, ale za to silnější umělá neuronová síť s názvem VGG16. Tento model vyvinula skupina Visual Geometry Group na univerzitě v Oxfordu. Číslo 16 označuje, že obsahuje pouze 16 vrstev. Což je oproti 54 vrstvám modelu MobileNetV2 mnohem méně. [48] [49]

Je však třeba poznamenat, že samotná hloubka a konfigurace vrstev neurčují pouze výkon nebo kvalitu modelu neuronové sítě. Další faktory, jako je typ použitých vrstev, volby architektonického návrhu a datová sada použitá pro učení také hrají důležitou roli v celkové účinnosti modelu.

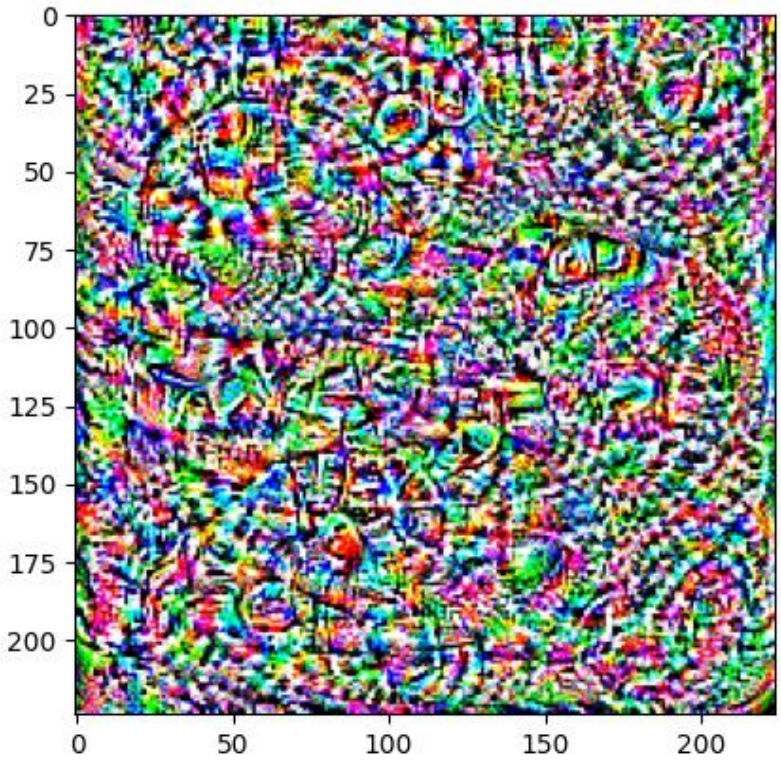
Hlavně tedy s ohledem na architekturu sítě by mohla metoda maximalizace aktivace pro model VGG16 fungovat lépe. Dalším výrazným plusem je, že klasifikuje obrázky do stejných tříd jako model MobileNetV2.

Jediné, co je tedy potřeba v kódu změnit je načítání modelu. Změna kódu bude vypadat takto:

```
# Load the neural network model
model = tf.keras.applications.VGG16()
```

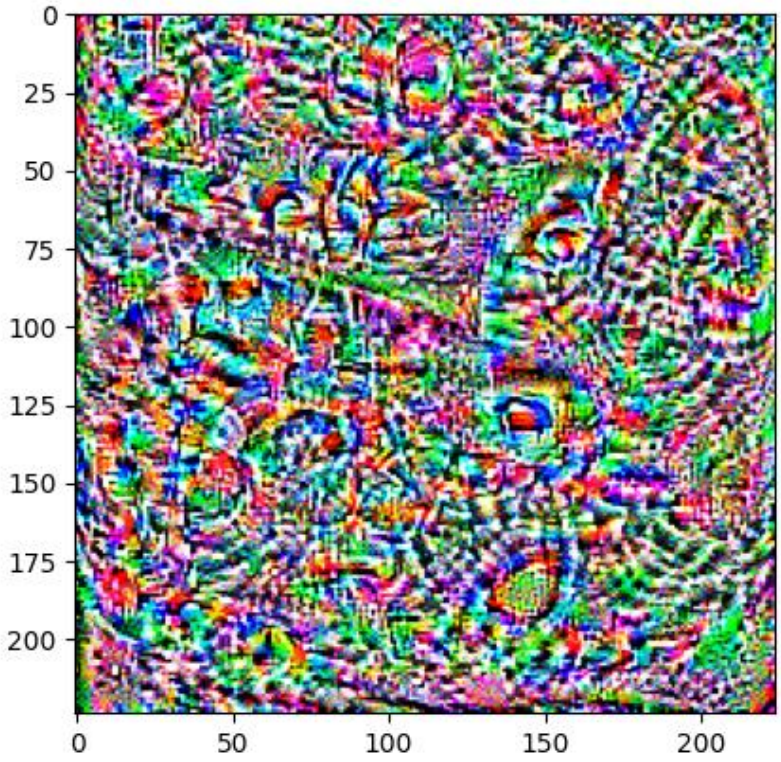
6.4.2.4.3.1. Náhodně generovaný obrázek

Zde se bude jednat o úplně stejný test se stejnými parametry sítě, jako byl v kapitole 6.4.2.4.2. Pro náhodně generovaný obrázek vypadala maximalizace aktivace takto:



Obrázek 51 – Maximalizace aktivace – celý model VGG16

Na obrázku 51 je vidět, jak po100 iteracích vypadala maximalizace aktivace pro třídu 156. Pokud se pozorovatel zaměří lehce doprava od prostřední části obrázku, tak je možné pozorovat něco jako oko a tvář. Uprostřed je možné pozorovat něco jako nos. Je zde také možné pozorovat jisté tvary a vzorce. Zároveň to ale ani zdaleka nepřipomíná psa a jeho plemeno blenheimského španěla. Ani zvýšení počtu parametru iterací na 1000 nic nezměnilo. Na obrázku jsou vidět nějaké tvary a vzory. Maximalizace aktivace pro 1000 iterací je vidět na obrázku 52 níž.

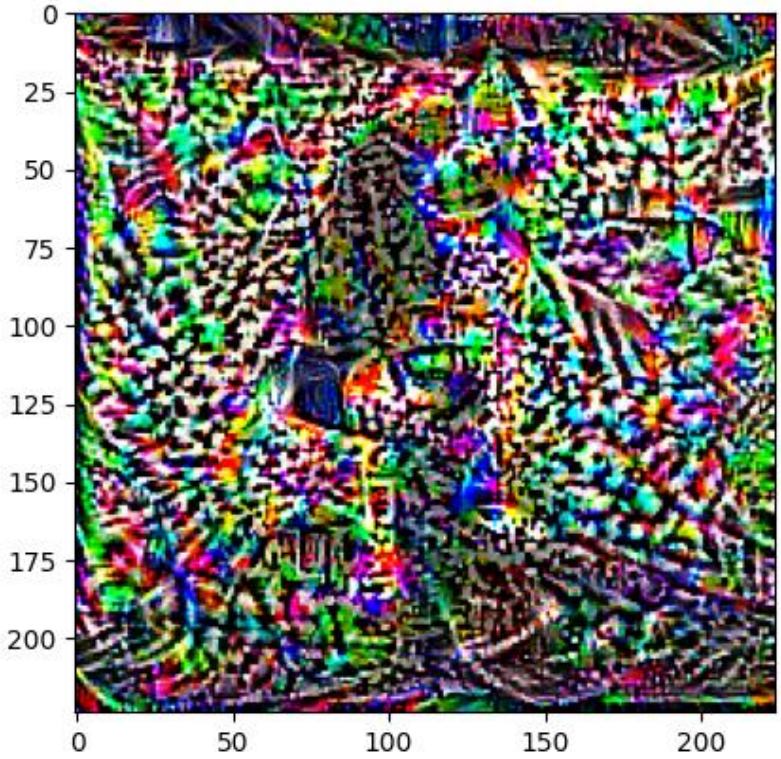


Obrázek 52 – Maximalizace aktivace – celý model VGG16

Pokud jde o výsledky, tak ani jeden obrázek maximalizace aktivace nepřipomíná nic, co by se podobalo dané třídě. Pokud se ale výsledky porovnají s aktivací maximalizace pro model MobileNetV2 na obrázku 49, tak se generuje alespoň něco. Tím něčím jsou myšlené nějaké tvary a vzory, ze kterých by se při větší představivosti dalo vyvodit něco, co připomíná část psa.

6.4.2.4.3.1. Obrázek odpovídají vybrané třídě

Tento test odpovídá testu z kapitoly 6.4.2.4.3., kde je použitý obrázek psa blenheimského španěla a na něm bude použita maximalizace aktivace. Obrázek po 1000 iteracích vypadal takto:



Obrázek 53 – Maximalizace aktivace – celý model VGG16

V pozadí je na obrázku 53, stejně jako na obrázku 50, kde se pracuje s modelem MobileNetV2, vidět pes z původního obrázku. Při pohledu na prostřední část je jasně vidět obličej s okem, uchem a čumákem. Celkově obličej spíš připomíná tvář kočičí. Každopádně je zde maximalizace aktivace mnohem výraznější než ve všech předchozích případech. Délka trvání této metody a generování obrázku byla okolo 50 minut.

7. SHRNUTÍ VÝSLEDKŮ

Pro účely testovaná interpretace umělých neuronových sítí byly použity tyto metody mapy aktivací (Activation maps), mapy síly vlivu (Saliency maps), citlivost na překrytí (Occulsion sensitivity) a maximalizace aktivace (Activation maximization).

Všechny tyto metody jsou použity pro interpretaci umělých neuronových sítí, které pracují s obrázky převážně za účelem klasifikace obrázků. Tyto metody interpretace byly použity pro interpretaci modelu umělé neuronové sítě MobileNetV2, která se zabývá právě klasifikací obrázků a dokáže obrázky klasifikovat až do tisíců různých tříd.

7.1. Výsledky metody mapy aktivací (Activation maps)

Použití map aktivací mělo podle původních předpokladů zobrazovat aktivace skupiny neuronů pro nějakou vrstvu umělé neuronové sítě. Výsledkem tedy měl být obrázek, který zobrazoval skupinu obrázků, kde každý obrázek reprezentoval jeden neuron a jeho aktivace.

Dále se čekalo, že čím mělčí vrstva umělé neuronové sítě, tím se budou mapy aktivací zajímat o nízko-úrovňové rysy, jako jsou hrany a rohy nebo textury. Naopak u hlubších vrstev už by měly se začít identifikovat vzory nebo dokonce objekty. Poslední vrstvy by se pak měly snažit zachytit kontext celého obrázku. Obrázek map aktivací by se tedy měl stávat postupně více rozpixelovaným, čím hlubší vrstva je vybrána.

Pokud tedy shrneme výsledky z kapitoly 6.1.2., tak se dá považovat implementace této metody za úspěšnou. Od obrázku 15 až po obrázek 28 je vidět, že obrázky jsou čím dál rozdělenější na jednotlivé pixely a postupně se zaměřují na specifické oblasti, kdežto na začátku se opravdu mapy aktivací zaměřují na hrany, textury nebo třeba barvy. U prvních vrstev umělé neuronové sítě jsme schopni na vygenerovaném obrázku map aktivací pozorovat původní obrázek. Zde znovu platí, že čím hlouběji jdeme, tím je původní obrázek méně čitelný. Protože jak již bylo zmíněno, čím hlubší vrstva, tím spíše se zaměřuje daná vrstva na objekty a kusy původního obrázku, čímž se stává více rozpixelovaným. Implementace a test této metody tedy proběhl úspěšně.

7.2. Výsledky metody mapy síly vlivu (Saliency maps)

Mapy síly vlivu se používají pro učení nejdůležitějších oblastí pro predikci oproti vybrané klasifikační třídě. Mapy síly vlivu by měly pomoci identifkovat, které části obrázku jsou

na vstupu do umělé neuronové síti nejdůležitější. Díky tomu je možné sledovat, na které části obrázku umělá neuronová síť nahlíží při klasifikaci. Takto vygenerovaná mapa síly vlivu by tedy měla zobrazovat obrázek, který by z části měl připomínat obrázek původní a v něm by měly být vyznačené pixely, které při predikci byly důležité.

Při testování implementace této metody bylo provedeno několik pokusů. Prvně byl vytvořen set čtyř různých obrázků, kde každý obrázek zastupoval jednu klasifikační třídu – celkem to byly tři třídy. Jeden z těchto obrázku obsahoval třídy dvě – psa a kočku. Tento set byl poté klasifikován oproti třem třídám, který byly na vytvořeném setu obrázků. Předpokládalo se, že mapa sil vlivů bude pro každou třídu jiná, ale přesto v některých aspektech podobná, protože se klasifikovala zvířata. Což se také stalo a pro každý test oproti třem vybraným klasifikačním třídám byl patrný rozdíl v mapách sil vlivů. Bylo tedy vidět, že pro predikce vybrané třídy bylo důležité pokaždé něco jiného. Výsledky je možné si prohlédnout na obrázcích 30 až 32.

Další test proběhl tak, že byl vytvořen set čtyř stejných obrázků. Slovem „stejných“ je myšleno, že se jednalo o obrázky ze stejné třídy. Konkrétně se jednalo o obrázky psa plemena blenheimského španěla v různých polohách. Pro tento set byly mapy síly vlivů, stejně jako v předchozím případě, generovány oproti třem klasifikačním třídám. Podobně jako v prvním testu se měly mapy síly vlivů lišit. Na obrázcích 34, 35 a 36 je to zřetelně vidět, protože pro každou třídu je důležité něco jiného, což se dá z těchto obrázků jasně poznat.

Oba popsané testy dopadly přesně podle očekávání. Implementace této metody a její implementace se, stejně jako metoda mapy aktivací, dá považovat za úspěšnou.

7.3. Výsledky metody citlivost na překrytí (Occulsion sensitivity)

Metoda citlivosti na překrytí je technika, která je používaná k identifikaci důležitých oblastí vstupního obrázku. Funguje na základě postupného zakrývání části obrázku a pozorování výstupu, tedy predikce sítě. Dá se říct, že tento způsob interpretace kombinuje mapy aktivací a mapy síly vlivu. Očekávání jsou taková, že čím větší část bude používána pro zakrytí obrázku, tím abstraktnější bude vygenerovaný obrázek citlivosti na překrytí. Vygenerovaný obrázek by měl být rozpixelovanější a měl by vyobrazovat větší oblasti, které jsou pro predikci důležité. Čím menší bude překrývající část, tím více by měl být na

vygenerované heat mapě citlivosti překrytí vidět původní obrázek, a bude tedy vyobrazovat spíše detaily, které jsou pro predikci důležité.

Výsledky testů je možné pozorovat na obrázcích 37 až 44, kde je vidět, že předpokládané chování bylo správné a výsledky mu odpovídají. Test a implementace citlivosti na překrytí byla tedy úspěšná.

7.4. Výsledky metody maximalizace aktivace (Activation maximization)

Tato metoda se používá k vytváření uměle vytvořených obrázků. Tyto obrázky se vytvářejí za účelem co nejvíc aktivovat daný neuron, vrstvu neuronů nebo celou umělou neuronovou síť. Jedná se o složitou metodu interpretace, která má spoustu atributů, které se dají nastavit, a tím ovlivní generování těchto obrázků. Nastavují se parametry podobně jako při tréninku sítě, takže je zde mnoho možností, jak tuto metodu otestovat nebo dokonce zprovoznit tak, aby fungovala.

Byly provedeny vlastně dva testy v různých variacích. První test bylo otestovat generování obrázků s maximální aktivací pro různé vrstvy vybraného modelu. Očekávání pro tento test byla taková, že pro každou vrstvu bude vygenerován obrázek, který danou vrstvu maximálně aktivuje. Pro první konvoluční vrstvu byl obrázek vygenerován velice rychle a bylo vidět, že tato vrstva vyhledává hlavně hrany – viz obrázek 45. Pro další vrstvy už ale nebylo možné nic vygenerovat. Buď byl obrázek černý nebo aplikace zcela selhala. To bylo hned z několika důvodů. Obrázky byly černé z toho důvodu, že vrstvy používaly aktivační funkci ReLU. Vysvětlení, proč byly obrázky černé je na konci kapitoly 6.4.2.2.1. Další z důvodu byl, že vrstvy měly jiný výstupní tvar, a tím pádem mohly nastat dva problémy. Prvním problémem mohlo být to, že vybrané vrstvy měly takový tvar, že z nich nebylo možné vytvořit obrázek. V druhém případě měly vrstvy jiný tvar, než byl v kódu v kapitole 6.4.1. nastaven. Zde nastává problém univerzálního vytvoření metody maximalizace aktivace pro všechny vrstvy. Je tedy potřeba je upravit pro speciální použití na danou vrstvu. Ale i přesto není možné tuto metodu použít pro všechny vrstvy ve složitějších sítích.

Druhý test měl fungovat tak, že maximalizace aktivace byla použita na celou umělou neuronovou síť. Na začátku byl vygenerován obrázek náhodného šumu a podle vybrané třídy byl postupně upravován tak, aby připomínal danou třídu – tedy aby byla umělá

neuronová síť na výstupu maximálně aktivována pro danou třídu. Bylo předpokládáno, že pokud byla vybrána například třída slona afrického, tak obrázek po doběhnutí specifikovaných iterací bude co nejvíce připomínat právě obrázek slona afrického.

Při prvním pokusu s modelem sítě MobileNetV2 a náhodně generovaným obrázkem šumu výsledný obrázek ani zdaleka nepřipomínal něco, co by se podobalo vybrané třídě. Ani po úpravě různých atributů jako je počet iterací nebo rychlost učení se nic nezměnilo. Poté byl místo obrázku šumu použit obrázek reálný, ale ani tak se generovaný obrázek maximalizace aktivace vůbec nepodobal vybrané třídě.

Proto byla v dalším pokusu v kapitole 6.4.2.4.3. a jejích podkapitolách použita umělá neuronová síť VGG16. Pro tento model se již generovaly obrázky, které alespoň z části připomínaly vybranou třídu. Nejprve byl test pro náhodný obrázek šumu, který sice nepřipomínal úplně danou třídu, ale už alespoň obsahoval nějaké tvary a vzory. Výsledek je možné si prohlédnou na obrázku 51 a 52. Pokud by se například nastavilo více iterací, upravil učící algoritmus nebo rychlost učení, tak by obrázek nakonec mohl připomínat vybranou třídu ještě víc.

V posledním pokusu druhého testu byl použit reálný obrázek, který dokonce vygeneroval část obličeje vybrané třídy. I přesto, že připomíná spíše nějakou šelmu než blenheimského španěla, tak to byl výrazný pokrok. Obrázek 53 je důkazem toho, že alespoň z části byla implementace této metody úspěšná.

Celkově bych implementaci a výsledky této metody interpretace hodnotil průměrně. Částečně je to z důvodu toho, že se nedá implementovat neutrálně, aby fungovala pro všechny vrstvy. Dále z důvodu velké výpočetní náročnosti, pokud generujeme celé obrázky.

8. ZÁVĚR

V první části diplomové práce byla nejdřív popsána základní teorie toho, co je neuron a umělá neuronová síť. U neuronu bylo vysvětleno, jak v principu funguje a jaké má aktivační funkce. V části věnující se umělým neuronovým sítím pak byly popsány jejich typy a jaké jsou jejich parametry. Dále byla probrána problematika černé skříňky umělých neuronových sítí a jak černá skříňka vzniká. Zároveň byl také vysvětlen vztah tréninkových dat vzhledem k černé skříňce. Poslední kapitola teoretické části se zabývá popisem interpretace umělé neuronové sítě a k čemu je interpretace jako taková důležitá. Zmíněny jsou i důvody, proč je interpretovatelnost umělých neuronových sítí žádoucí, nebo v některých případech dokonce bezvýhradně vyžadována. Nakonec jsou popsány některé základní způsoby interpretace a na jaké typy se dělí.

Praktická část, která začíná od páté kapitoly, se na začátku zaobírá popisem prostředí, ve kterém budou interpretační metody implementovány. Konkrétně s jakým modelem umělé neuronové sítě bude pracováno a také jaký programovací jazyk a potřebné knihovny budou při implementaci použity. Poté se v kapitole šest implementuje několik interpretačních metod, kde je popsáno využití dané interpretační metody, její kód a jaké výsledky přinesla.

Nakonec jsou všechny výsledky shrnuty v poslední kapitole. Z výsledků se dá konstatovat, že implementace vybraných implementačních metod byla vcelku zdařilá. Metody by se daly určitě vylepšit a po jistých úpravách použít na jiné modely umělých neuronových sítí. Z výsledků a zkušeností nabytých při interpretaci daných metod ale plyne, že interpretace umělých neuronových sítí je velice složitá a zatím ne zcela probádaná oblast.

Je také důležité si uvědomit, že interpretační metody se točily okolo klasifikace obrázků. Pokud by byla třeba použita umělá neuronová síť, která generuje text jako ChatGPT, musely by být použité jiné interpretační metody. Zároveň také platí, že všechny umělé neuronové sítě nejsou stejné. Tím pádem je složitější tyto interpretační metody vytvořit neutrálně tak, aby fungovaly pro všechny umělé neuronové sítě stejně. I jejich aplikace je tedy náročnější, pokud jsou použity na nějaké komplikovanější umělé neuronové sítě.

Dále je pak náročné s výsledky těchto metod nějak smysluplně naložit a něco si z nich vzít. Zároveň by bezpochyby byla vhodná nějaká hlubší analýza těchto výstupů, k jejichž celistvější a komplexnější interpretaci by jich byly potřeba minimálně stovky.

Na závěr je potřeba říct, že u interpretace umělých neuronových sítí je naprosto klíčové čerpat z předchozích zkušeností. A stejně jako u tvorby a tréninku umělé neuronové sítě je vždy potřeba brát v potaz, že její chování a výsledky určuje svým nastavením pozorovatel a tvůrce této sítě. To stejné platí pro implementaci a použití interpretačních metod.

SEZNAM POUŽITÉ LITERATURY

[1] CHEN, James. What Is a Neural Network? [online]. [cit. 16.2.2023] Dostupné z: https://www.investopedia.com/terms/n/neuralnetwork.asp

[2] IBM. What is a Neural network? [online]. [cit. 16.2.2023] Dostupné z: https://www.ibm.com/topics/neural-networks

[3] NAGYFI, Richard. The differences between Artificial and Biological Neural Networks [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[4] BUETTGENBACH, Maurice Henry. Explain like I'm five: Artificial neurons [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/explain-like-im-five-artificial-neurons-b7c475b56189

[5] PRAMODITHA, Rukshan. The Concept of Artificial Neurons (Perceptrons) in Neural Networks [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc

[6] BAHETI, Pragati. Activation Functions in Neural Networks [12 Types & Use Cases] [online]. [cit. 16.2.2023] Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions#h1

[7] CHEN, B. 7 popular activation functions you should know in Deep Learning and how to use them with Keras and TensorFlow 2 [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/7-popular-activation-functions-you-should-know-in-deep-learning-and-how-to-use-them-with-keras-and-27b4d838dfe6

[8] SHARMA, Sagar. Activation Functions in Neural Networks [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[9] BAHETI, Pragati. The Essential Guide to Neural Network Architectures [online]. [cit. 16.2.2023] Dostupné z: https://www.v7labs.com/blog/neural-network-architectures-guide

[10] GOOGLE DEVELOPERS. Neural Networks: Structure [online]. [cit. 16.2.2023] Dostupné z: https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/anatomy

[11] SELDON. Neural Network Models Explained [online]. [cit. 16.2.2023] Dostupné z: https://www.seldon.io/neural-network-models-explained

[12] DEEPAI. What is a Feed Forward Neural Network? [online]. [cit. 16.2.2023] Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

[13] BENTO, Carolina. Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141

[14] SIMPLILEARN. What are Radial Basis Functions Neural Networks? [online]. [cit. 16.2.2023] Dostupné z: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-are-radial-basis-functions-neural-networks

[15] IBM. What are recurrent neural networks? [online]. [cit. 16.2.2023] Dostupné z: https://www.ibm.com/topics/recurrent-neural-networks

[16] SAEED, Mehreen. An Introduction to Recurrent Neural Networks and the Math That Powers Them [online]. [cit. 16.2.2023] Dostupné z: https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/

[17] ANAND, Ashesh. The Basics of Modular Neural Networks [online]. [cit. 16.2.2023] Dostupné z: https://www.analyticssteps.com/blogs/basics-modular-neural-networks

[18] ALTO, Valentina. Neural Networks: parameters, hyperparameters and optimization strategies [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/neural-networks-parameters-hyperparameters-and-optimization-strategies-3f0842fac0a5

[19] NYUYTIYMBIY, Kizito. Parameters and Hyperparameters in Machine Learning and Deep Learning [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

[20] DEEPLEARNING.AI. Parameter optimization in neural networks [online]. [cit. 16.2.2023] Dostupné z: https://www.deeplearning.ai/ai-notes/optimization/index.html

[21] BLAZEK, Paul J. Why We Will Never Open Deep Learning's Black Box [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/why-we-will-never-open-deep-learnings-black-box-4c27cd335118

[22] TENSORFLOW. A Neural Network Playground [online]. [cit. 16.2.2023] Dostupné z: https://playground.tensorflow.org

[23] LENAIL, Alex. NN-SVG [online]. [cit. 16.2.2023] Dostupné z: http://alexlenail.me/NN-SVG/index.html

[24] MARR, Bernard. What Is GPT-3 And Why Is It Revolutionizing Artificial Intelligence? [online]. [cit. 25.1.2023] Dostupné z: https://bernardmarr.com/what-is-gpt-3-and-why-is-it-revolutionizing-artificial-intelligence/

[25] ROMERO, Alberto. A Complete Overview of GPT-3 — The Largest Neural Network Ever Created [online]. [cit. 28.2.2023] Dostupné z: https://towardsdatascience.com/gpt-3-a-complete-overview-190232eb25fd

[26] NGUYEN, Anh, YOSINSKI, Jason, CLUNE, Jeff. Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [online]. [cit. 16.2.2023] Dostupné z: https://arxiv.org/pdf/1412.1897.pdf

[27] ZHANG, Yu, TIŇO, Petr, LEONARDIS, Aleš, TANG, Ke. A Survey on Neural Network Interpretability [online]. [cit. 16.2.2023] Dostupné z: https://arxiv.org/pdf/2012.14261.pdf

[28] THE WHITE HOUSE. Algorithmic Discrimination Protections [online]. [cit. 16.2.2023] Dostupné z: https://www.whitehouse.gov/ostp/ai-bill-of-rights/algorithmic-discrimination-protections-2/

[29] LEE, Nicol Turner, RESNIK, Paul, BARTON, Genie. Algorithmic bias detection and mitigation: Best practices and policies to reduce consumer harms [online]. [cit. 16.2.2023] Dostupné z: https://www.brookings.edu/research/algorithmic-bias-detection-and-mitigation-best-practices-and-policies-to-reduce-consumer-harms/

[30] EUROPEAN UNION AGENCY FOR FUNDAMENTAL RIGHTS. Bias in algorithms - Artificial intelligence and discrimination [online]. [cit. 16.2.2023] Dostupné z: https://fra.europa.eu/sites/default/files/fra_uploads/fra-2022-bias-in-algorithms_en.pdf

[31] SEETHARAMAN, Krishna. Financial Applications of Neural Networks [online]. [cit 16.2.2023] Dostupné z: https://blog.aspiresys.com/banking-and-finance/financial-applications-neural-networks/

[32] INTERSOFT CONSULTING. Automated individual decision-making, including profiling [online]. [cit. 16.2.2023] Dostupné z: https://gdpr-info.eu/art-22-gdpr/

[33] SAKAR, Tirthajyoti. Activation maps for deep learning models in a few lines of code [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/activation-maps-for-deep-learning-models-in-a-few-lines-of-code-ed9ced1e8d21

[34] KANA, Kana. Practical Guide for Visualizing CNNs Using Saliency Maps [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/practical-guide-for-visualizing-cnns-using-saliency-maps-4d1c2e13aeca

[35] EDDINS, Steve. Network Visualization Based on Occlusion Sensitivity [online]. [cit. 16.2.2023] Dostupné z: https://blogs.mathworks.com/deep-learning/2017/12/15/network-visualization-based-on-occlusion-sensitivity/

[36] FENG-LEI, Fan, JINJUN, Xiong, MENGZHOU, Li, GE, Wang. On Interpretability of Artificial Neural Networks: A Survey [online]. [cit. 16.2.2023] Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9380482

[37] YE, Andre. Neural Network Interpretability Fundamentals [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/every-ml-engineer-needs-to-know-neural-network-interpretability-afea2ac0824e

[38] GEEKS FOR GEEKS. What is Saliency Map? [online]. [cit 16.2.2023] Dostupné z: https://www.geeksforgeeks.org/what-is-saliency-map/

[39] DRAELOS, Rachel. Learn to Pay Attention! Trainable Visual Attention in CNNs [online]. [cit 16.2.2023] Dostupné z: https://towardsdatascience.com/learn-to-pay-attention-trainable-visual-attention-in-cnns-87e2869f89f1

[40] KOHLBRENNER, Maximilian, BAUER, Alexander, NAKAJIMA, Shinichi, BINDER, Alexander, SAMEK, Wojciech, LAPUSCHKIN, Sebastian. Towards Best Practice in Explaining Neural Network Decisions with LRP [online]. [cit. 16.2.2023] Dostupné z: https://arxiv.org/abs/1910.09840

[41] MATCHA, Anil Chandra Naidu. Class activation maps: Visualizing neural network decision-making [online]. [cit. 16.2.2023] Dostupné z: https://heartbeat.comet.ml/class-activation-maps-visualizing-neural-network-decision-making-92efa5af9a33

[42] DOSHI, Sanket. Various Optimization Algorithms For Training Neural Network [online]. [cit. 16.2.2023] Dostupné z: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[43] GUPTA, Ayusha. A Comprehensive Guide on Deep Learning Optimizers [online]. [cit. 16.2.2023] Dostupné z: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

[44] SANDLER, Mark. MobileNetV2: The Next Generation of On-Device Computer Vision Networks [online]. [cit. 26.4.2023] Dostupné z: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

[45] SANDLER, Mark, HOWARD, Andrew, ZHU, Menglong, ZHMOGINOV, Andrey, CHEN, Liang-Chieh. MobileNetV2: Inverted Residuals and Linear Bottlenecks [online]. [cit. 26.4.2023] Dostupné z: https://arxiv.org/pdf/1801.04381.pdf

[46] REIFF, Daniel. Understand your Algorithm with Grad-CAM [online]. [cit. 26.4.2023] Dostupné z: https://towardsdatascience.com/understand-your-algorithm-with-grad-cam-d3b62fce353

[47] BROWNLEE, Jason. A Gentle Introduction to the Rectified Linear Unit (ReLU) [online]. [cit. 11.5.2023] Dostupné z: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[48] ROHINI, G. Everything you need to know about VGG16 [online]. [cit. 11.5.2023] Dostupné z: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[49] THAKUR, Rohit. Beginner's Guide to VGG16 Implementation in Keras [online]. [cit. 11.5.2023] Dostupné z: https://builtin.com/machine-learning/vgg16

PŘÍLOHY

- dp – soubor s diplomovou prací
- obrázky – složka obsahující obrázky využité pro testování interpretačních metod
- pomocné soubory – složka obsahující soubory s klasifikačními třídami a výpisem vrstev modelu MobileNetV2
- zdrojové kódy – složka obsahující zdrojové kódy interpretačních metod a jejich výstupy

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Daniel Tesař**
Studium:	I2000811
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Interpretační analýza hlubokých ANN**
Název diplomové práce AJ:	Interpretation analysis of deep ANNs

Cíl, metody, literatura, předpoklady:

- neuron
- neuronová síť
- černá skříňka v neuronových sítích
- interpretace neuronových sítí
- Implementovat a porovnat vybrané interpretační metody ANN
- Cíl: použití interpretačních metod a jejich vyhodnocení

- Y. Zhang, P. Tiňo, A. Leonardis and K. Tang, "A Survey on Neural Network Interpretability," in IEEE Transactions on Emerging Topics in Computational Intelligence, vol. 5, no. 5, pp. 726-742, Oct. 2021, doi: 10.1109/TETCI.2021.3100641.
- F. -L. Fan, J. Xiong, M. Li and G. Wang, "On Interpretability of Artificial Neural Networks: A Survey," in IEEE Transactions on Radiation and Plasma Medical Sciences, vol. 5, no. 6, pp. 741-760, Nov. 2021, doi: 10.1109/TRPMS.2021.3066428.
- S. Das, N. Agarwal, D. Venugopal, F. T. Sheldon and S. Shiva, "Taxonomy and Survey of Interpretable Machine Learning Method," 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, ACT, Australia, 2020, pp. 670-677, doi: 10.1109/SSCI47803.2020.9308404.

Zadávající pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Karel Mls, Ph.D.
Oponent:	doc. Ing. Pavel Čech, Ph.D.
Datum zadání závěrečné práce:	15.10.2021