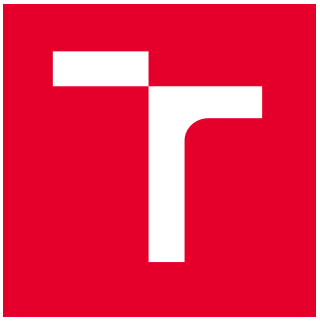


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

FIRE AND SMOKE DETECTION IN VIDEO

DETEKCE OHNĚ A KOUŘE VE VIDEOZÁZNAMU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **Viktor Buzovský**
AUTOR PRÁCE

SUPERVISOR — **Ing. Jiří Přinosil, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Viktor Buzovský ***ID:*** 230536

Ročník: 3 ***Akademický rok:*** 2022/23

NÁZEV TÉMATU:

Detekce ohně a kouře ve videozáznamu

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je navrhnout a implementovat algoritmus pro detekci ohně a kouře ve videozáznamu z reálného prostředí. Tento algoritmus by měl využívat moderní techniky hlubokého učení pracující se statickými snímky i s dynamickou složkou obsaženou ve videozáznamu. Součástí práce je rovněž výběr a případná úprava databáze vhodné pro trénování navržených modelů.

DOPORUČENÁ LITERATURA:

Podle pokynů vedoucího práce

Termín zadání: 6.2.2023 ***Termín odevzdání:*** 26.5.2023

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Práce pojednává o možnostech detekce ohně a kouře ve videozáznamu z reálného prostředí. Cílem práce je vybrat vhodný model, tento model natrénovat a jeho detekční schopnosti následně vylepšit přídavnou implementací. První část práce shrnuje potřebné teoretické znalosti, které jsou v kontextu práce využívány. Druhá, praktická část, pak představuje naučený model a jeho následné pokusy o vylepšení, jednak pomocí optického toku a dále pak pomocí přídavných klasifikačních sítí. Práce je zakončena finální implementací detektoru ohně a kouře a je představen návrh na jeho potenciální zlepšení. Součástí práce jsou mimo jiné i použité a vytvořené datasety.

KLÍČOVÁ SLOVA

detekce objektů, detekce ohně, detekce kouře, optický tok, konvoluční neuronové sítě, klasifikační neuronové sítě, počítačové vidění, neuronové sítě, trénování neuronových sítí, YOLOv7

ABSTRACT

Thesis deals with possibilities regarding detection of fire and smoke in real enviroment video. The main task is to choose suitable model, train this model, and improve detection capabilities of the model afterwards. First part of thesis is summary of theoritical knowledge needed to have understanding of discussed technical necessities. The second, practical part presents the learned model and its subsequent attempts at improvement, firstly using optical flow and then using additional classification networks. The work is concluded with a final implementation for the detector of fire and smoke, and a proposal for its potential improvement is presented. The work also includes the datasets used and created, among other things.

KEYWORDS

object detection, fire detection, smoke detection, optical flow, convolutional neural networks, computer vision, neural networks, neural network training, YOLOv7

BUZOVSKÝ, Viktor. *Detekce ohně a kouře ve videozáznamu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2023, 73 s. Bakalářská práce. Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Viktor Buzovský
VUT ID autora:	230536
Typ práce:	Bakalářská práce
Akademický rok:	2022/23
Téma závěrečné práce:	Detekce ohně a kouře ve videozáznamu

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Jiřímu Přinosilovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Detekce objektů, ať už na statickém obraze, nebo ve videozáznamu a videu v reálném čase, je nedílná součást počítačového vidění, které se poslední roky vyvíjí značnou rychlostí. Tato oblast má široké spektrum aplikací i včetně včasného varování před požárem a monitorování požáru, ať už v uzavřeném prostoru, nebo na velkých územích. Umělá inteligence se postupně stává větší součástí našich životů, za účelem zlepšit jeho kvalitu. Technika hlubokého učení, označovaná jako „Deep Learning“, je podmnožinou technik strojového učení, která je k počítačovému vidění často využívána.

V této práci se zaměříme na výběr vhodné architektury v oblasti hlubokých neuronových a konvolučních sítí jako páteřní sítě, kterou následně natrénujeme pro detekci ohně a kouře. Součástí práce je také přídavná implementace optického toku, jako způsob využití časové informace obsažené ve videozáznamu, za pomocí které, se budeme snažit o zlepšení dosavadních výsledků dosažených v oblasti detekce ohně a kouře při práci pouze se statickou složkou – tedy zpracování jednotlivých snímků ve videozáznamu samostatně, bez kontextu času. Práce kromě učení neuronových sítí zahrnuje také sběr a úpravu dat vhodných pro jejich natrénování. Pro učení neuronových sítí je nutné zajistit dostatečné množství kvalitních dat. K tomu využijeme několik veřejně dostupných zdrojů snímků a videí, zároveň však i vytvoříme vlastní specifický dataset pro trénování klasifikátorů optického toku – tedy data v podobě vícerozměrné matice.

Cílem práce je navrhnout efektivní metodu pro detekci ohně a kouře ve videozáznamu, která využívá kombinaci naučených hlubokých neuronových a konvolučních sítí s implementací klasifikačních neuronových sítí naučených na práci s časovou složkou. Na základě výsledků práce, je snaha přinést případné nové poznatky v oblasti detekce ohně a kouře – např. požáru na videozáznamu, popřípadě na videu v reálném čase.

1 Teoretická část studentské práce

1.1 Neuronové sítě

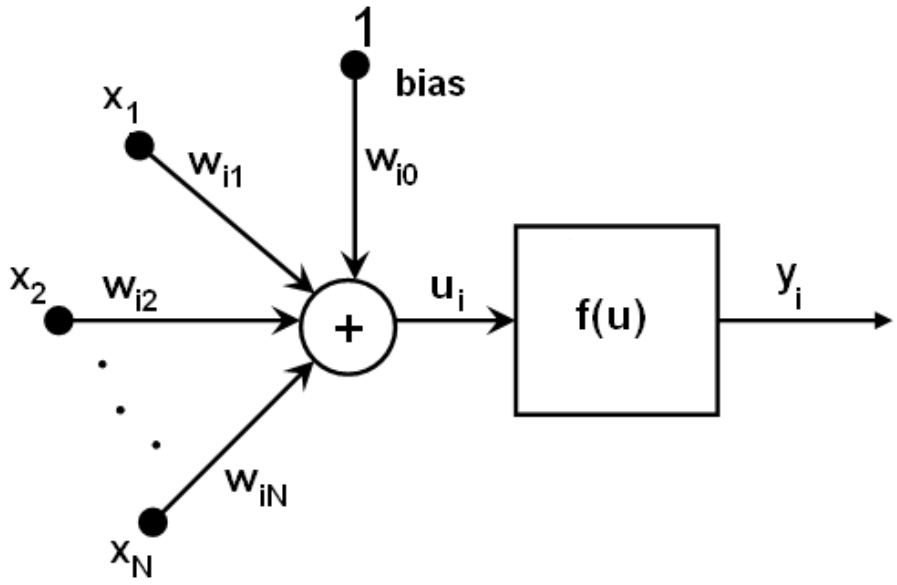
Neuronové sítě jsou výpočetní systémy, které jsou inspirovány fungováním biologických neuronových sítí lidského mozku. Umělé neurony, označované taky jako *perceptrony* viz obr. 1.1, tvoří základní strukturu těchto sítí. Představují nositele informace, který je následně spojen s několika dalšími neurony na dalších vrstvách a tvoří tak neuronovou síť. Základy pro neuronové sítě položili Warren McCulloch a Walter Pitts [25] už v roce 1943.

Celá struktura sítě viz obr. 1.2 je rozdělena do jednotlivých vrstev neuronů, přičemž první vrstva je nazývána jako vstupní vrstva, poslední vrstva jako výstupní. Mezi první a poslední vrstvou se nachází libovolné množství vrstev, kterým říkáme *skryté vrstvy*, kde každá má své opodstatnění a plní určitou funkci. [14] Sítě, které mají více než jednu skrytou vrstvu nazýváme *hluboké neuronové sítě*. Rozměr vstupní vrstvy je nastaven tak, aby odpovídal rozměru tenzoru[1] vstupních dat k trénování, např. pokud by byl vstupem černobílý obraz s rozlišením 28×28, potřebovali bychom 784 vstupních neuronů (jeden neuron připadá na jeden obrazový bod), označených x_1 až x_n a k nim korespondující váhy w_{i1} až w_{in}. Výstupní vrstva má za úkol reprezentovat výsledek vyhodnocen naší sítí, který je následně porovnáván s výsledkem očekávaným. Rozdíl mezi nimi je pak vypočten *ztrátovou funkcí* (viz kap. 1.1.2), která je nedílnou součástí zpětného učení takových sítí.

Perceptron se dělí na několik důležitých částí:

1. **Vstupní signály a jejich váhy**, které představují datový vstup do perceptronu. Vstupní signál je vynásoben příslušnou vahou a následně započten do sčítací funkce. Tyto váhy jednotlivých vstupních signálů, pak představují parametr sítě, který následně při procesu *zpětné propagace* algoritmus ladí tak, aby síť dosahovala co nejlepších výsledků.
2. **Sčítací funkci**, která z výsledného vynásobení všech vstupních signálů svými vahami udělá celkový součet těchto hodnot a přidá tzv. *odsazení* b_k (bias), sloužící k umělé upravě finální hodnoty před vstupem do aktivační funkce.
3. **Aktivační funkci** (viz kap. 1.1.1), sloužící k usměrnění oboru hodnot na výstupu perceptronu do určitého intervalu.

[1] Tenzor je datová struktura, která svým rozměrem definuje určitá data. Např. RGB obraz definujeme jako trojrozměrný tenzor (třetí rozměr odpovídá barevné hloubce). Pro šarži takových obrazů pak definujeme 4 rozměry (počet vzorků × šířka × výška × barvná hloubka).



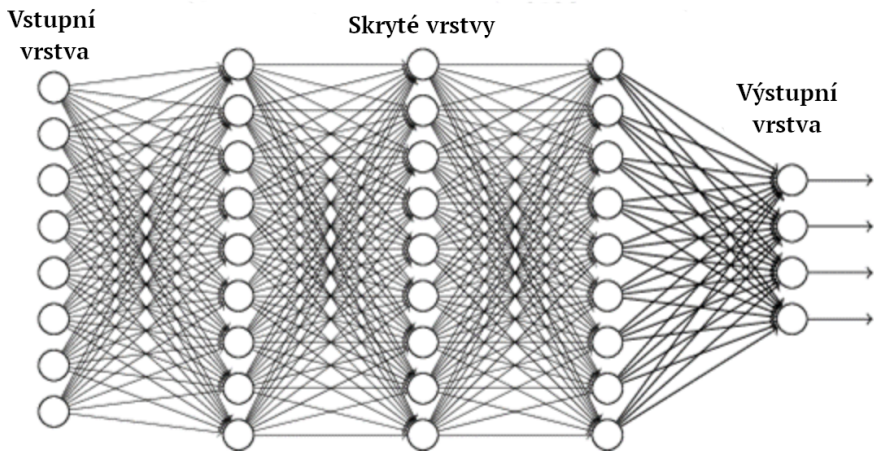
Obr. 1.1: Obrázek perceptronu, převzato z [29]

Pro u_i (výstup sčítací funkce) platí vztah:

$$u_i = \sum_{j=1}^{N}(x_j \cdot w_{ij}) + 1 \cdot w_{i0}. \tag{1.1}$$

Výstup perceptronu je hodnota y_i, která je funkcí výstupu sčítací funkce. Této funkci říkáme *aktivační funkce*.

$$y_i = f(u_i) \tag{1.2}$$



Obr. 1.2: Příklad neuronové sítě, převzato z [9]

1.1.1 Aktivační funkce

Na obr. 1.1 označená jako *f(u)*, přidává neuronové síti možnost mít vlastnosti nelinearity, protože pouze samotné výstupy z jednotlivých vrstev jsou lineární operace prováděné na tenzorech [35]. Zároveň však usměrňuje obor hodnot výstupu perceptronu (všechna reálná čísla) na daný rozsah hodnot (obvykle $\langle 0,1 \rangle$, nebo $\langle 0,\infty \rangle$), který je pak dále zpracováván v případě, že na výstup navazuje další vrstva. Aktivační funkce existují lineární i nelineární.

V případě použití lineární funkce se však síť dokáže adaptovat na vstup pouze lineárně z důvodu konstantní derivace, to je však málo využitelné, protože problém, který se snažíme učením vyřešit, je velmi často vlastnostmi nelineární, proto jsou nelineární aktivační funkce mnohem hojněji využívány. [35] Každý typ aktivačních funkcí se využívá k jiným účelům, nelze říci, že existuje pouze jedna správná aktivační funkce pro daný problém.

Lineární aktivační funkce

Znázorněna na obr. 1.3.Lineární aktivační funkce je nejjednodušší možnou hned po jednotkovém skoku, ovšem není vhodnou pro složitější praktické aplikace, neboť v procesu *zpětné propagace* se aktivační funkce derivuje a derivace je v tomto případě konstanta. To znamená, že váhy a odsazení budou měněny pokaždé o stejnou konstantní hodnotu, to vede k neschopnosti sítě minimalizovat *ztrátovou funkci*. [35]

Je dána vztahem (1.3), kde a představuje *koeficient úměrnosti* (libovolné reálné číslo mimo hodnoty 0).

$$f(x) = ax \tag{1.3}$$

Aktivační funkce sigmoid

Znázorněna na obr. 1.4. Hlavním důvodem použití funkce sigmoid je omezení rozsahu na $\langle 0,1 \rangle$ a použití jejího výstupu jako pravděpodobnostní výstup, velmi často tedy na poslední vrstvě pro klasifikační modely.

Při použití funkce sigmoid v hlubokých neuronových sítích, může nastat tzv. *problém mizejícího gradientu* – gradient ztrátové funkce se blíží k nule a síť není schopná se efektivně učit (čím více vrstev využívá aktivační funkci podléhající tomuto problému, tím rychleji se síť dostává do problému s jejím učením v procesu zpětné propagace). Její alternativou je často aktivační funkce $ReLU$, u které se tento problém nevyskytuje. [37]

Funkce sigmoid je dána vztahem (1.4).

$$f(x) = \frac{1}{1 + \mathrm{e}^{-x}} \tag{1.4}$$

Aktivační funkce hyperbolický tangens (tanh)

Znázorněna na obr. 1.5. Je svým průběhem velmi podobná aktivační funkci sigmoid, avšak je symetrická kolem nuly, to znamená, že umožňuje, aby byla výstupem neuronu i záporná hodnota.

Stejně jako funkce sigmoid, může být náchylná k mizejícímu gradientu v procesu *zpětné propagace*, proto se v hlubokých modelech sítí převážně nepoužívá. Jako alternativu lze použít vylepšenou funkci *ReLTanh*. [39]

Funkce hyperbolický tangens je dána vztahem (1.5).

$$f(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1 \tag{1.5}$$

Rectified Linear Unit (ReLU)

Velmi často používaná aktivační funkce v hlubokých neuronových a konvolučních sítích. Její hlavní výhodou je překonání problému již zmíněného *mizejícího gradientu* – z toho důvodu se stala velice populární volbou pro aktivační funkci. [37]

Funkce se dělí na dva intervaly, pro interval $(-\infty,0\rangle$ funkce vrací hodnotu 0 a takové neurony jsou deaktivovány, nepodílí se na chodu sítě a již nejsou nikdy reaktivovány – neuron je považován za mrtvý. V intervalu $(0,\infty)$ jsou výstupní hodnoty lineární funkcí hodnoty vstupní. Permanentní deaktivace neuronů však není vždy žádoucí, proto existuje také aktivační funkce *LReLU – Leaky ReLU* (znázorněna na obr. 1.6), která je v záporném intervalu nenulová. *LReLU* může mít však v některých případech nekonzistentní výsledky při *zpětné propagaci*. Tyto problémy pak řeší funkce *PReLU – Parametrická ReLU*, která má sklon funkce v záporném intervalu jako parametr sítě, který se učením mění. [37]

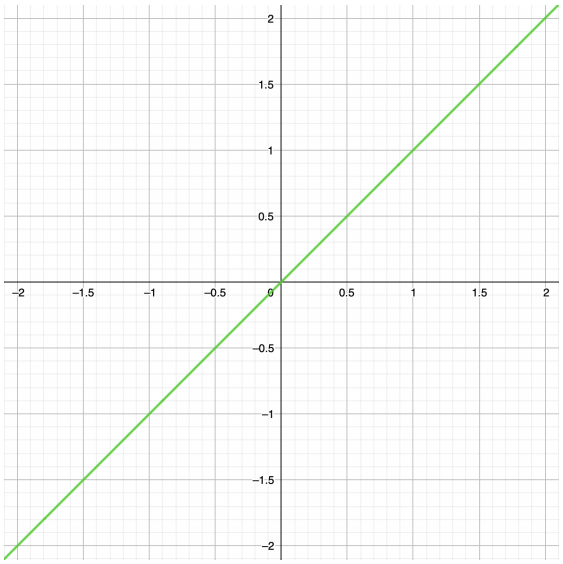
Funkce ReLU je dána vztahem (1.6)

$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{1.6}$$

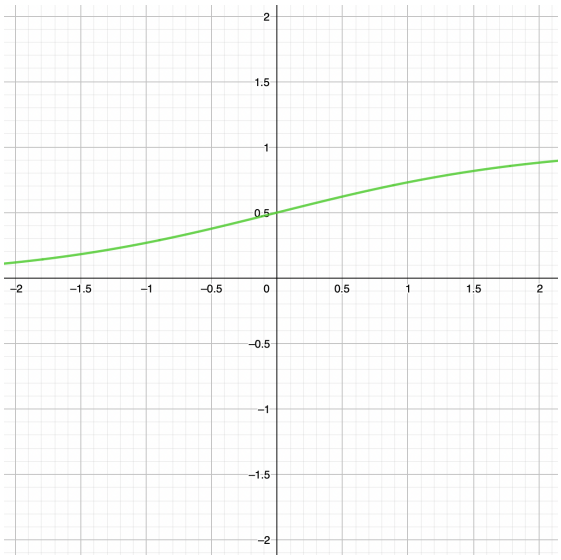
1.1.2 Ztrátová funkce

Ztrátová funkce vyhodnocuje, s jakou přesností se hodnota na výstupu sítě liší, s hodnotou očekávanou, vyjádřeno číselnou hodnotou na výstupu této funkce. [26] Těchto funkcí existuje celá řada, např. Střední kvadratická chyba (MSE):

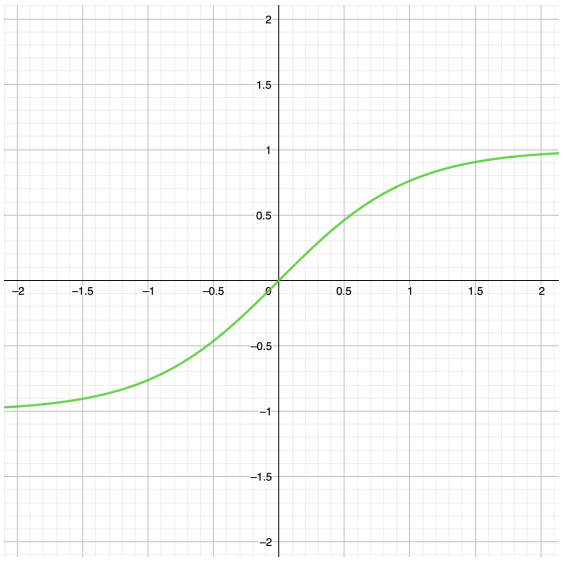
$$MSE = \frac{1}{n} \sum_{x}^{n} (y_i - \hat{y}_i)^2. \tag{1.7}$$



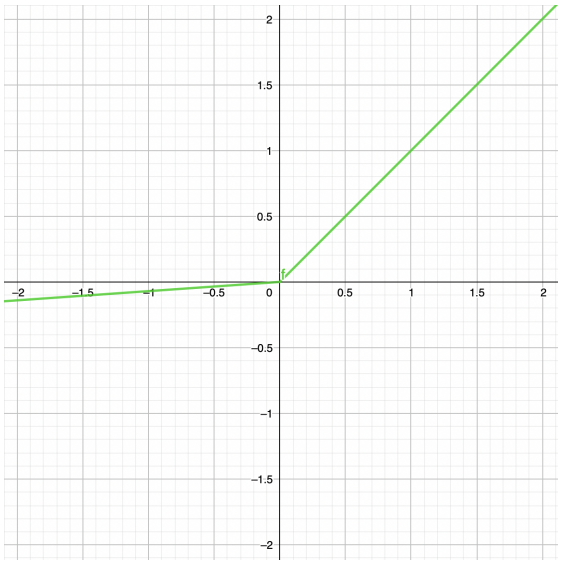
Obr. 1.3: Průběh lineární aktivační funkce



Obr. 1.4: Průběh sigmoid aktivační funkce



Obr. 1.5: Průběh aktivační funkce hyperbolický tangens



Obr. 1.6: Průběh aktivační funkce LReLU

1.2 Proces trénování neuronových sítí

Trénováním myslíme proces, při kterém se síť snaží naučit, jaké hodnoty přiřadit jednotlivým parametrům (váhy pro jednotlivá spojení a odchylky vstupující do neuronů), tak, aby se chybovost sítě co nejvíce minimalizovala. Etalonem správnosti jsou předem anotovaná data (v případě učení s učitelem), které jsou porovnávány s výstupem ze sítě, jejich rozdíl je pak počítán ztrátovou funkcí (viz kap. 1.1.2).

Proces učení začíná nastavením parametrů sítě na náhodné hodnoty jako výchozí stav sítě, bez jakékoliv znalosti o našich trénovacích datech. Poté jsou síti představena data, které projdou všemi vrstvami a na poslední vrstvě jsou rozřazeny do tříd (v případě klasifikace), z množiny možných tříd, které jsme definovali. Následuje výpočet odchylky výstupu sítě od očekávaného výstupu za pomocí ztrátové funkce. Vzhledem k náhodné inicializaci parametrů bude první výstup sítě naprosto náhodný a tudíž i hodnota ztrátové funkce bude „velká“. Cílem učení je nastavení parametrů sítě (váhy a odchylky) tak, aby tato hodnota ztrátové funkce byla co nejnižší. To provedeme pomocí tzv „gradientního sestupu“ neboli *gradient descent*. [14, 32]

Tento algoritmus (gradient descent) postupně hledá (iterační výpočet) nejmenší možnou hodnotu pro naši ztrátovou funkci, která může mít obrovský počet parametrů (miliony vah a odchylek pro větší hluboké sítě), pro názornost si lze představit alespoň 2 parametry viz obr. 1.7. Ve zkratce tento algoritmus spočítá gradient funkce (ztrátové) v daném bodě (hodnota ztrátové funkce), vzhledem k tomu, že hledáme minimum této funkce (tam, kde bude ztráta sítě nejmenší), vezmeme zápornou hodnotu tohoto gradientu, jeho velikost nám zároveň určuje o kolik se posuneme směrem k minimu (velikost změny jednotlivých vah). Každý komponent tohoto gradientu nám říká, kterým směrem (ubrat – záporná hodnota, přidat – kladná hodnota) a o jak velkou hodnotu[2], viz (1.8). [32]

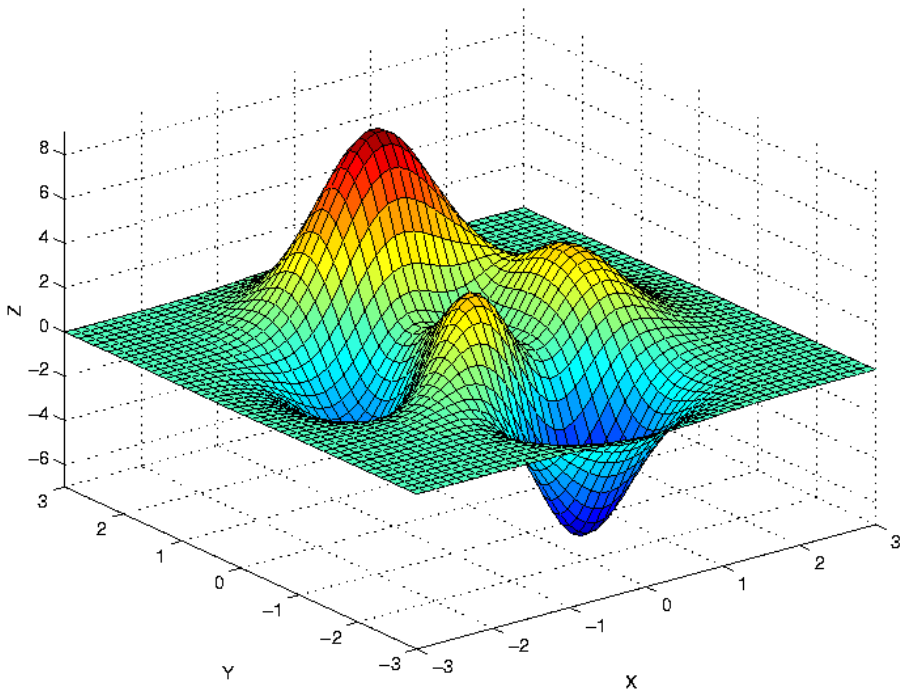
$$\overrightarrow{W} = \begin{bmatrix} 1,12 \\ 0,38 \\ -0,97 \\ \vdots \\ -1,43 \\ -1,08 \\ 2,23 \end{bmatrix} \qquad -\nabla C(\overrightarrow{W}) = \begin{bmatrix} 0,24 \\ 1,68 \\ -0,41 \\ \vdots \\ -0,29 \\ -2,37 \\ 0,72 \end{bmatrix} \tag{1.8}$$

[2]Vlevo vektor vah, odchylek s jejich hodnotami, vpravo pak vektor gradientu. Převzato z [32]

Základní typy algoritmu gradientního sestupu (GD), podle toho, kdy během učení chceme gradient počítat: [22]

1. **Batch GD** – síti představíme postupně všechna trénovací data, spočítáme průměr ztrátové funkce, poté počítáme gradient a upravujeme váhy. Tedy jeden krok směrem k minimu za epochu.
2. **Stochastic GD** – v tomto případě počítáme gradient a upravujeme váhy pro každý vzorek trénovacích dat, konverguje rychleji než Batch GD, pro velké datasety je nevhodný, kvůli své náročnosti.
3. **Mini-Batch GD** – kompromis mezi Batch a Stochastic GD, trénovací dataset rozdělíme do šarží o dané velikosti, pro každou tuto šarži vyhodnotíme ztrátu funkce, spočítáme gradient a upravujeme váhy.

Existují i další typy, které řeší problémy konvergence a lokálních minim funkce, například implementací „zrychlení“, podobnému fyzikálnímu (lze si představit jako kuličku kutálející se z kopce do příkopu viz obr. 1.7, příkladem může být například Adagrad, RMSprop. [5]



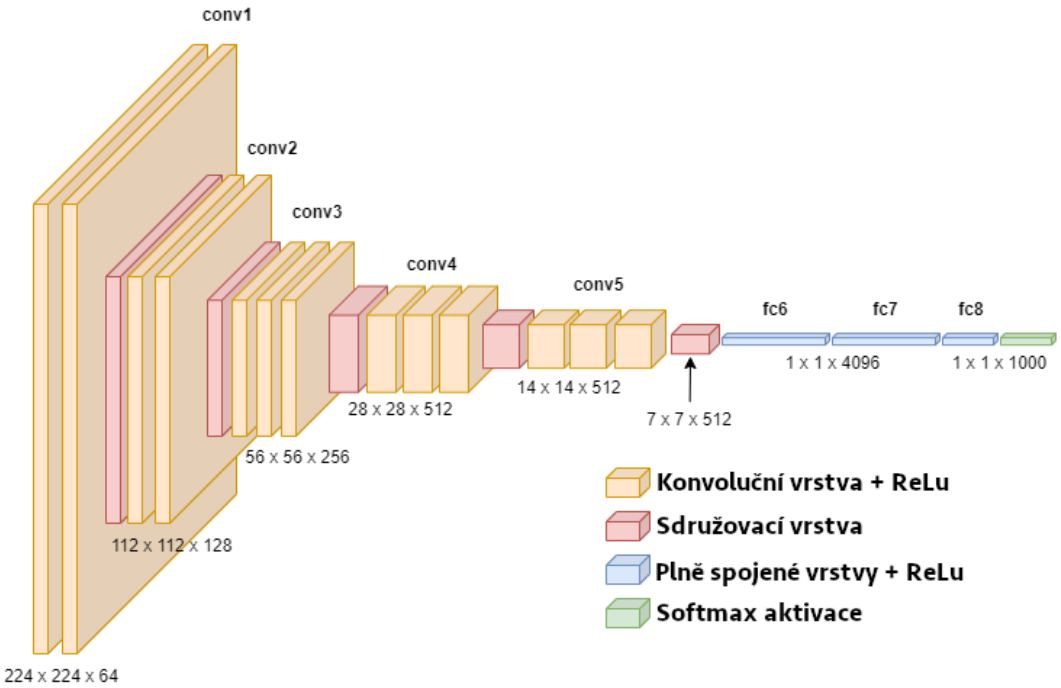
Obr. 1.7: Příklad ztrátové funkce pro 2 parametry, převzato z [10]

1.3 Konvoluční neuronové sítě

Jsou podmnožinou tzv. „hlubokých neuronových sítí“ (sítě, které mají více než 1 skrytou vrstvu). Konvoluční sítě (CNN) vykazují dobré výsledky při aplikacích, kde zpracováváme obrazový vstup, proto tvoří základ naší práce pro detekci ohně a kouře. Základní vlastností konvolučních sítí je, že z obrazu na vstupu postupně extrahují jednotlivé příznaky, od nejmenších, které pak se spojují ve větší a postupně se skládají, až nakonec tvoří jediný celek, který lze klasifikovat / detekovat na výstupní vrstvě jako jeden z prvků, který v obraze hledáme (za předpokladu, že se na daném snímku vyskytuje).

Konvoluční sítě obvykle obsahují několik základních typů vrstvev: [1]

1. **Parametrické vrstvy** – učením lze modifikovat jejich parametry
   (a) Konvoluční – extrakce příznaků z obrazu, provádí operaci konvoluce mezi vstupem a filtrem (někdy také jako „kernel“) (viz kap. 1.3.1).
   (b) Plně spojené – obvykle na výstupu sítě, umožňují přidat další vrstvu komplexity interpretace pro síť a zároveň umožňuje nastavit přesný počet výstupních neuronů podle počtu tříd.
2. **Bezparametrické vrstvy** – tyto vrstvy nelze učit
   (a) Sdružovací (Pooling) – vrstva která snižuje rozměr reprezentace obrazu, obvykle se střídá s konvolučními vrstvami. Příklady různých typů pooling vrstev: LP Pooling, Mixed Pooling, Stochastic Pooling,. . . . [19]



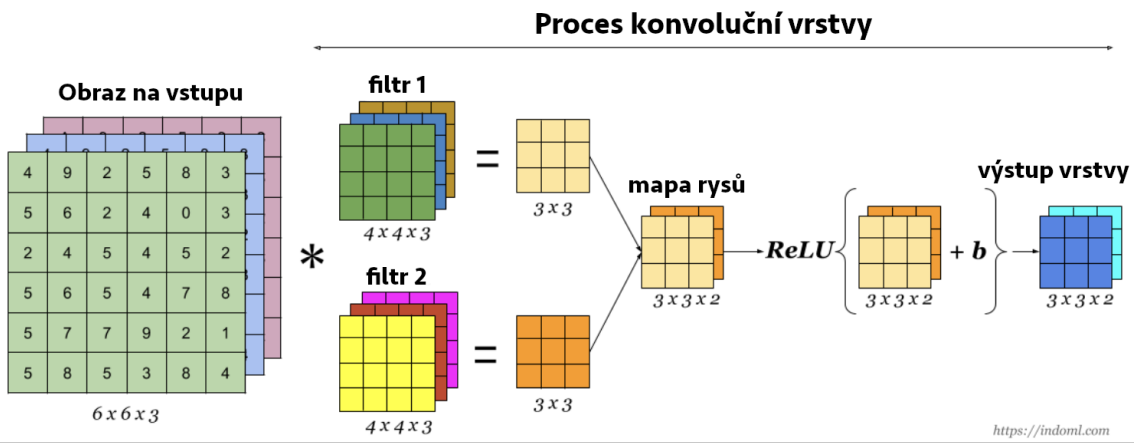
Obr. 1.8: Příklad konvoluční sítě VGG16, převzato z [23]

1.3.1 Konvoluční vrstva

Každá vrstva za pomocí několika filtrů hledá jednotlivé příznaky (jako například různé křivky, okraje objektů,..) v obraze na vstupu dané vrstvy. Obrazový vstup si lze představit jako vícerozměrnou matici a filtr pak obvykle jako matici 3×3, nebo 5×5, třetí rozměr odpovídá hloubce obrazu. Mezi maticí obrazu a filtru se provádí operace konvoluce, výsledkem je pak mapa příznaků, s hloubkou odpovídající počtu filtrů v dané vrstvě. Celý proces konvoluce lze vidět na obr. 1.9. Velikost mapy příznaků lze ovlivnit několika parametry: [20, 1]

1. **Hloubka** – definuje počet filtrů použitých při konvoluci, počet filtrů pak následně odpovídá hloubce mapě příznaků, tedy její třetímu rozměru, první dva rozměry odpovídají příznakům samotným, které vrstva z obrazu extrahovala
2. **Posun** – je skalár, který určuje o jakou hodnotu se při konvoluci posouváme přes matici vstupů.
3. **Odsazení** – provádí se za pomocí nulových hodnot okolo vstupní matice, tak abychom při konvoluci přes vstupní matici byly schopni použít matici filtru i na okrajích, přičemž nulové hodnoty odsazení nemají na výpočet vliv.

Po výpočtu jednotlivých konvolucí se k výsledné mapě příznaků přidává odchylka a následně se používá aktivační funkce, velmi často funkce ReLU viz kap. 1.1.1 – ReLU. Tato funkce prochází každý výsledný pixel a negativní hodnotu nahrazuje nulou v případě základní ReLU, tento krok je důležitý především díky tomu, že umožňuje síti se adaptovat a učit nelineárně [20], bez takové možnosti by síť byla jenom sled po sobě jdoucích lineárních operací nad maticemi, která by se logicky dala nahradit jedinou vrstvou a síť po sobě jdoucích vrstev by postrádala smysl.



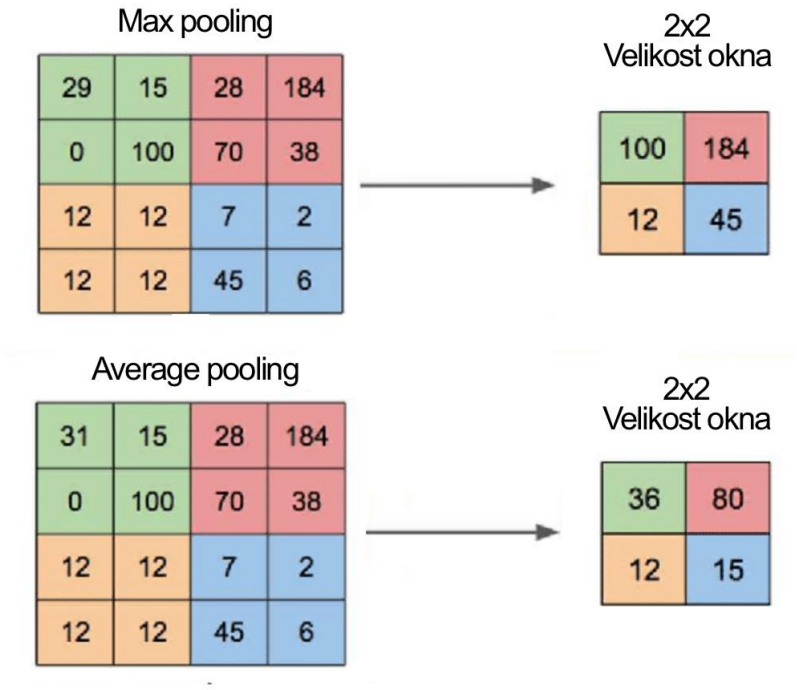
Obr. 1.9: Architektura a proces konvoluční vrstvy, převzato z [7]

1.3.2 Sdružovací vrstva

Sdružovací vrstva se obvykle nachází přímo za konvolučními vrstvami, jejím cílem je zredukování komplexity map příznaků, které jsou výstupem vrstev konvolučních, zároveň však zachovává důležité informace. [20, 1] Sdružovacích vrstev existuje nespočet typů, liší se způsobem procesu výpočtu, nejznámější typy jsou: [17]

- Average Pooling
- Max Pooling
- Mixed Pooling
- Stochastic Pooling
- Super-Pixel Pooling

Každé sdružovací vrstvě je potřeba nastavit velikost okna a jeho posun, např. okno o velikost 2×2 a posun o 2. Pro případ „Max Pooling“ je brána v potaz pouze největší hodnota v daném okně, která se následně objeví na výstupu (viz obr. 1.10).



Obr. 1.10: Příklad výpočtu sdružovací vrstvy, převzato z [30]

Sdružování nám pomáhá zredukovat množství parametrů a tím i množství výpočtů v síti, zároveň umožňuje detekovat objekty na obrazovém vstupu v jakékoliv pozici i s mírným rozostřením (např. u „Max Pooling“ bereme v potaz pouze maximální hodnotu, tudíž lehké rozostření se na výstupu téměř neprojeví). [20]

1.3.3 Plně spojené vrstvy

Jsou v podstatě klasické neuronové sítě, určené v případě detekce objektů k finální klasifikaci, obvykle s aktivační funkcí pro pravděpodobnostní výstup – tedy například funkce sigmoid nebo softmax. Pravidlem je, že všechny neurony předchozí vrstvy jsou spojeny se všemi neurony z vrstvy následující. [20, 1] Příklad takové sítě lze vidět na obr. 1.2. Tato architektura plně spojených vrstev se vyskytuje na konci CNN, kde se provádí klasifikace vstupu na jednotlivé třídy objektů.

Důvod využití konvoluce

Důvodem proč zavádíme nové typy vrstev a nepoužíváme pouze plně spojené vrstvy jako jednotnou celou síť pro práci s obrazovým vstupem je převážně problém předimenzování. Hlavní limitací tradičních hlubších neuronových sítí je jejich náročnost na výpočetní kapacitu z důvodu velkého počtu parametrů, které se musí v procesu trénování sítě (viz kap. 1.2) brát v potaz. I pro malé obrazové vstupy o rozměrech 28×28 bychom pro síť s 2 skrytými vrstvami měli nespočet parametrů a pro reálné použití při rozlišeních 640×640 a větších, by takových parametrů bylo obrovské množství, které by síti znemožňovalo efektivně tyto parametry nastavit tak, aby síť vykazovala rozumnou přesnost. [1] Proto při práci s obrazem je vhodné použít CNN, kde konvoluční vrstvy mají podstatně méně parametrů, jelikož neurony nejsou spojeny plně a každý neuron odpovídá určité části obrazu, nikoliv jedinému obrazovému bodu. [14]

1.4 You Only Look Once – YOLO

Je CNN (Convolutional Neural Network) založená na principu regrese – v jedné iteraci algoritmu detekuje požadované objekty a vyznačí ohraničující boxy tzv. „bounding boxes“ v celém obrazu, na rozdíl od ostatních jiných známých algoritmů, které vyhledávají pouze ve vybraných oblastech obrazu, kde by se hledaný objekt mohl nacházet. YOLO bylo poprvé představeno v roce 2015 [31], kdy v tu dobu výrazně překonalo dosavadní techniky počítačového vidění (DPM, R-CNN,..) a to především v rychlosti.

1.4.1 YOLO algoritmus

Algoritmus je velmi podobný pro všechny verze modelů YOLO, podstatně se liší pouze v architektuře sítě a jednotlivých procesech v různých částech sítě, obecný algoritmus však zůstává podobný. Na začátku YOLO systém změní velikost vstupního obrazu na 448×448px (toto rozlišení lze měnit, je jiné pro různé verze), který pak rozdělí na mříž o rozměrech S×S, pokud je střed hledaného objektu v dané buňce, je tato buňka zodpovědná za detekování daného objektu. [31]

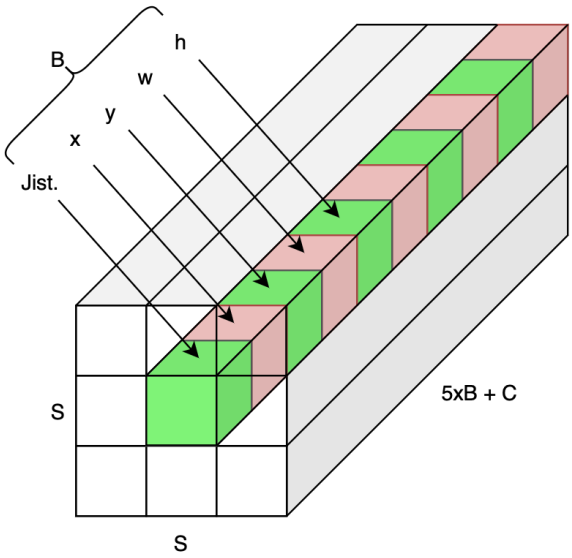
Každá buňka této mřížky tedy predikuje ve svém obrazovém prostoru počet ohraničujících boxů – B a zároveň k nim i jejich pravděpodobnosti, které určují, jak moc si je model jist svou predikcí ohraničujícího boxu. [31]

Každý takový box definujeme pomocí 5 parametrů (které model predikuje): x, y, w, h a jistota, kde (x,y) představují souřadnice středu boxu vzhledem k samotné buňce. Ovšem w a h definují výšku a šířku vzhledem k celému obrazu. Jistotu definujeme jako

$$P_{\mathrm{r}}(\mathrm{Objekt}) \cdot IoU^{\mathrm{skut}}_{\mathrm{pred}},$$

tedy jistota nám říká, jak si je model jistý, že predikovaný box obsahuje hledaný objekt a jak je tento box přesně usazen v obraze, vzhledem ke skutečnému boxu, který byl takto anotován. IoU (Intersection over Union) představuje hodnotu poměru průniku k sjednocení boxů. Pokud tedy žádný objekt v buňce neexistuje, hodnota jistoty by měla být 0, pokud se objekt v buňce nachází, je hodnota jistoty přímo hodnotou IoU. [31] Poslední hodnotou, která je součástí výstupního tenzoru jsou pravděpodobnosti pro jednotlivé třídy objektů, které hledáme, aby bylo možné rozpoznat, o kterou třídu objektu se jedná v případě jeho detekce – tyto hodnoty značíme C. Výstupní odhady jsou zapsány jako tenzor (viz obr. 1.11) o rozměrech: [31]

$$S \times S \times (B \cdot 5 + C).$$



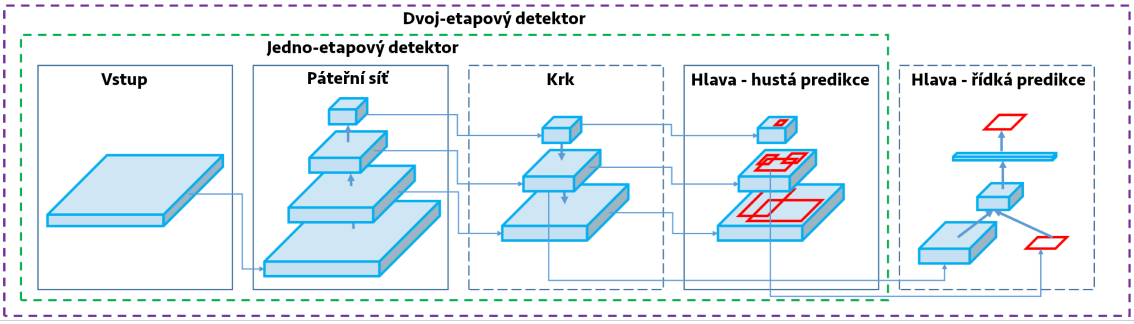
Obr. 1.11: Ilustrace výstupního tensoru YOLOv1

1.4.2 YOLOv4

YOLOv4 bylo představeno v [2] jako pokračování YOLOv3, tentokrát již bez Josepha Redmona – jako hlavního autora YOLOv1–3. Architektura sítě se dělí na 3 základní obecné celky (viz obr. 1.12):

- **Páteř** (Backbone) – tvoří předem natrénované sítě jako VGG16, CSPDarknet53, ResNet. Páteř sbírá základní příznaky (někdy také jako „rysy“) z obrazového vstupu.
- **Krk** (Neck) – jeho funkce je sbírat mapy příznaků z jednotlivých částí procesů v páteři a tvořit tak pyramidu příznaků, příkladem takové sítě je např. FPN, PAN,.. Tyto sítě obsahují obvykle několik tzv. „bottom-up“ a „top-down“ cest, které umožňují detekovat objekty o různých velikostech.
- **Hlava** (Head) se obykle se dělí na 2 části – jedno-etapové (např. YOLO, SSD, RetinaNet) a dvoj-etapové (např. R-CNN) detektory objektů. Tyto detektory vytvářejí ohraničující boxy kolem detekovaných objektů, výstupem je již zmíněný vektor hodnot popisující všechny boxy pro danou buňku.

Model YOLOv4 využívá síť CSPDarknet53 jako páteřní síť. Pro krk se využívá SPP (Spatial pyramid pooling), které umožňuje generovat příznaky z obrazového vstupu s fixní velikostí nezávisle na velikosti mapy příznaků a vstupního obrazu (tím se vyhneme potřebným úpravám obrazového vstupu z důvodu potřeby fixní velikosti při vstupu do plně spojené části sítě) se sítí PANet [24]. [16] Model obsahuje 2 hlavy (finální predikční vrstvy) – vrstvy s hustou a řídkou predikcí ohraničujících boxů (viz obr. 1.12). Celá struktura je pak dále vylepšena metodami, které nezvyšují



Obr. 1.12: Příklad architektury modelu YOLOv4, převzato z [2]

výpočetní náročnost detekce, avšak zvyšují jejich kvalitu – takové metodě říkáme „Bag of freebies“. Cílem je zvýšit proměnnost obrazových dat, které pak vedou k vyšší generalizaci a tím k lepším učebním a následně i detekčním výsledkům sítě. Příkladem takových augumentačních metod může být např: [2]

- **Fotometrické zkreslení** – může být docíleno např. upravováním jasu, odstínů barev, kontrastu, šum, atp.
- **Geometrické zkreslení** – techniky jako např. otočení obrazu, přetočení o 180°, škálování obrazu atp.
- **„MixUP“** – jedná se o geometrickou metodu prokládání (interpolace), tedy lineární kombinace dvou obrazových vstupu v jeden kombinovaný obraz.
- **„CutMix“** – je technika vkládání zmenšených obrazů do původních dat v datasetu, výsledkem je koláž obrazů, která zpřesňuje model proti poškozeným obrazovým vstupům.

Existují i další metody (někdy také jako moduly) zlepšení sítě, například tzv. „Bag of specials“, které sice zvýší náročnost detekce, ale výrazně dokáží zvýšit přesnost detekce objektů. Příkladem takových modulů jsou například SPP, ASPP, RFB, dále pak některé aktivační funkce jako SELU, Swish, Mish,.. [2]

1.4.3 YOLOv7

Model byl představen roku 2022 v [40], s několika změnami modelu YOLOv4. Jednou z nich je změna architektury páteřní sítě, kde se využívá architektura E-ELAN (Extended efficient layer aggregation networks), která zrychluje proces gradientního sestupu implementací „paměti vah“, (při každém sestupu nezahazuje všechny předchozí váhy) a tím zrychluje proces detekce. Další změnou architektury je implementace škálování modelu, kde hlavní myšlenkou je vyhovět požadavkům na různé rychlosti detekce. YOLOv7 implementuje tzv. „postupné škálování“ modelu, kde zachovává vlastnosti původního modelu a jeho struktury. [40]

Další představenou změnou je plánovaná reparametrizace konvoluce, jako součást

„bag of freebie“ metod. Reparametrizace je založena na rozdílné struktuře při učení a následně při detekování, tím nevzniká nárůst časové náročnosti při detekování. [18]

Dále je pak vylepšena technika přidělování značek na základě výstupu hlavní hlavy (lead head – zodpovědná za finální výstup sítě) s kombinací s „pravidvými značkami“ (data, které jsme před tréninkem označili) pro generování tzv. „soft labels“. Pro model YOLOv7 je výstup z hlavní hlavy převeden na sbírku „soft labels“, kde každý slouží jako reference pro hlavní i pomocnou hlavu (auxiliary head), nachazející se uprostřed modelu. Pro hluboké sítě je tento koncept pomocné hlavy nazýván jako „Deep supervision“. [40]

1.5 Optický Tok

Technika výpočtu optického toku vychází z časové informace obsažené ve video sekvenci snímků. Optický tok sleduje pohyb jednotlivých obrazových bodů v čase a z jejich rozdílných poloh (mezi dvěma snímky) počítá vektor posunu (je nutné aby okolní obrazové body měly stejný, nebo alespoň podobný směr pohybu) [27]. Problém výpočtu optického toku nastává například v případě, pokud je část obrazu homogenní (nelze poznat, který obrazový bod se pohnul, na které místo) – to je problematické například při sledování optického toku větší plochy kouře na obraze v čase. Obecnou rovnici pro obrazový bod můžeme napsat například jako:

$$I(x, y, t) = I(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t), \tag{1.9}$$

kde I představuje pixel v bodech x a y, v čase t, který se následně po čase $\mathrm{d}t$ posune o vzdálenosti $\mathrm{d}x$ a $\mathrm{d}y$. Optický tok následně zkoumá vektor pohybu složený ze složek u a v: [27]

$$u = \frac{\mathrm{d}x}{\mathrm{d}t}; \qquad v = \frac{\mathrm{d}y}{\mathrm{d}t}. \tag{1.10}$$

Existují 2 základní typy výpočtu optického toku:

1. **řídký optický tok** (Lucas-Kanade optical flow) – jak název napovídá, jedná se o výpočet optického toku pouze pro některé obrazové body, které by mohly být „informačně zajímavé“. Jedná se o výpočetně úspornější řešení, za cenu ztráty přesnější informace o optickém toku v obraze
2. **hustý optický tok** (Farneback optical flow) – k výpočtu optického toku využívá algoritmus představený v [11]. Počítá optický tok pro všechny obrazové body ve snímku, je tedy náročnější na výpočet, ale poskytuje přesnější informaci o pohybu v čase – proto byl vybrán jako výpočet optického toku ohně a kouře.

2 Výsledky studentské práce

2.1 Získání a zpracování dat – Dataset

Na začátku jakékoliv práce se strojovým učením je potřeba zajistit potřebné množství dat, které síti předáme. V našem případě používáme dataset DFire publikovaný v [4]. Ten obsahuje 21 527 již označených dat. Jejich rozložení lze vidět v tabulkách 2.1, 2.2 a 2.3. Samotné data lze vidět na obr. 2.3 a obr. 2.4.

Trénovací	Validační	Testovací
13776	3445	4306
63,9 %	16,0 %	20,0 %

Tab. 2.1: Ukázka rozložení datasetu DFire [4] po rozdělení na 3 části

Celkem	21527
Pouze oheň	1164
Pouze kouř	5867
Oheň a kouř	4658
Žádná třída	9838

Tab. 2.2: Počet obrázků v datasetu

Celkem	26557
Oheň	14692
Kouř	11865

Tab. 2.3: Počet ohraničujících boxů v datasetu

Anotované data jsou ve formě vhodné pro síť YOLO, anotace k danému snímku je uložena v textovém souboru, který obsahuje souřadnice jednotlivých ohraničujících boxů a jejich třídu – v našem případě máme 2 třídy (0 – kouř, 1 – oheň), ta se vyskytuje jako první číslo na řádku, poté následuje první mezera za kterou jsou 2 čísla označující souřadnice x a y, poslední 2 čísla označují výšku a šířku boxu, tedy hodnoty w a h. Všechny hodnoty jsou v měřítku 0–1, kde bod (0,0) na obrazu je levý horní roh, bod (1,1) je pak pravý dolní roh. Každý řádek koresponduje s jedním nálezem třídy v obraze, konkrétní anotovaná data lze vidět na obr. 2.1. V tomto případě se jedná o soubor „WEB07157.txt“, který je svým názvem svázán s odpovídajícím obrázkem se stejným názvem, YOLO tak pozná, které anotace patří k jakým datům. Dataset DFire [4] je však původně rozdělen pouze do dovu částí a to „train“ a „test“, pro trénování je však zapotřebí i část validační, která na konci každé epochy zhodnotí výsledky sítě a uchovává je pro každou epochu ve výstupním souboru *results.txt*. Validační část jsme vytvořili rozdělením části trénovací za pomocí jazyku python a balíčku *split-folders* [12]. Výsledné rozdělení lze vidět v tabulce 2.1.

WEB07157.txt

```
1 0.1020408163265306 0.4061224489795919 0.061224489795918366 0.10204081632653061
1 0.23843537414965985 0.4270408163265306 0.0564625850340136 0.02959183673469388
1 0.16768707482993195 0.3234693877551021 0.03605442176870748 0.0326530612244898
1 0.3105442176870748 0.2841836734693878 0.04285714285714286 0.02755102040816327
1 0.4918367346938775 0.30969387755102046 0.07074829931972788 0.03571428571428572
1 0.42482993197278907 0.4198979591836735 0.011564625850340135 0.017346938775510204
0 0.49795918367346936 0.2602040816326531 0.9931972789115645 0.5061224489795919
```

Obr. 2.1: Ukázka anotace pro YOLO

Obr. 2.2: Ukázka anotovaných dat z datasetu DFire [4]

2.2 Výběr sítě a její trénování

Důležitou rolí při výběru sítě pro tuto práci hraje samotný dataset, který je anotován pro síť YOLO. Bez tolik potřebných dat nelze natrénovat žádnou síť. Pokud bychom chtěli použít jakoukoliv jinou alternativu k síti YOLO, museli bychom celý dataset pře-anotovat, nebo zajistit data jinde. Tento rok vyšla nová verze YOLOv7 od autorů YOLOv4, která překonává všechny doposud známé detektory objektů jak v rychlosti tak v přesnosti od 5–160 FPS porovnáváno např. s SWIN-L(Transformer), R-CNN, atd.. [40]. V této práci je tedy použita síť YOLOv7 (podrobněji v kap. 1.4). Trénování probíhalo na výpočetním serveru osazeným grafickou kartou NVIDIA GeForce RTX 2080Ti (verze řadiče 470.141.03, CUDA verze 11.4). Před začátkem

trénování bylo potřeba vytvořit novou virtuální instalaci jazyku python, naklonovat repositář YOLOv7 a do něj stáhnout připravený dataset, následně nainstalovat všechny potřebné knihovny a balíčky, na kterých je YOLOv7 závislé. Dále bylo potřeba vytvořit konfigurační soubor (konkrétně firesmoke.yaml), který definuje, kde se nachází data a jejich 3 části, kolik chceme detekovat tříd a jak se tyto třídy nazývají, viz výpis 2.1. Následovalo samotné trénování sítě, které probíhalo ve 2 etapách. První etapa bylo natrénování nové sítě, tedy bez žádných předem naučených vah. V druhé etapě pak následovalo přeučení již naučené sítě na datasetu MS COCO.

```
train: ./data/D-fire-original-copy/train/images
val: ./data/D-fire-original-copy/val/images
test: ./data/D-fire-original-copy/test/images

# Počet tříd
nc: 2

# Názvy tříd
names: ["smoke", "fire"]
```

Výpis 2.1: Ukázka kódu ze souboru firesmoke.yaml

2.2.1 Trénování sítě bez počátečních vah

Takové trénování sítě vyžaduje mnohem více výpočetního času a objemný dataset. V tomto případě mělo trénování probíhat 300 epoch, z důvodu přerušení vzdáleného spojení na server přes terminál, se však učení přerušilo (následně již využit balíček *screen* vere 4.06.02, který je na serveru nainstalován), následovalo pokračování s vahami z poslední epochy, tentokrát bylo však nastaveno méně epoch, neboť bylo zřejmé, že výsledky se nijak výrazně nemění, učení bylo ukončeno po 260 epochách. Příkaz zadaný pro učení byl následující:

```
python3 train.py --workers 4 --device 0 --batch-size 8 --data data/
    firesmoke.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --
    weights '' --name yolov7 --hyp data/hyp.scratch.p5.yaml --epochs
    300
```

parametr *–workers* upravuje počet paralelně běžících vláken (tzn. pokud máme 2 CPU jádra, pak *–workers* 2 znamená, že proces učení bude běžet na dvou jádrech „vedle sebe“) umožňuje tak zrychlit proces trénování. *–device* definuje, který hardware chceme použít v našem případě 0 znamená jediná grafická karta. Dále specifikujeme velikost šarže zapomocí *–batch-size*, ten definuje kolik vzorků projde sítí, než se provede zpětná propagace a výpočet vah podrobněji popsáno v kapitole 1.2.

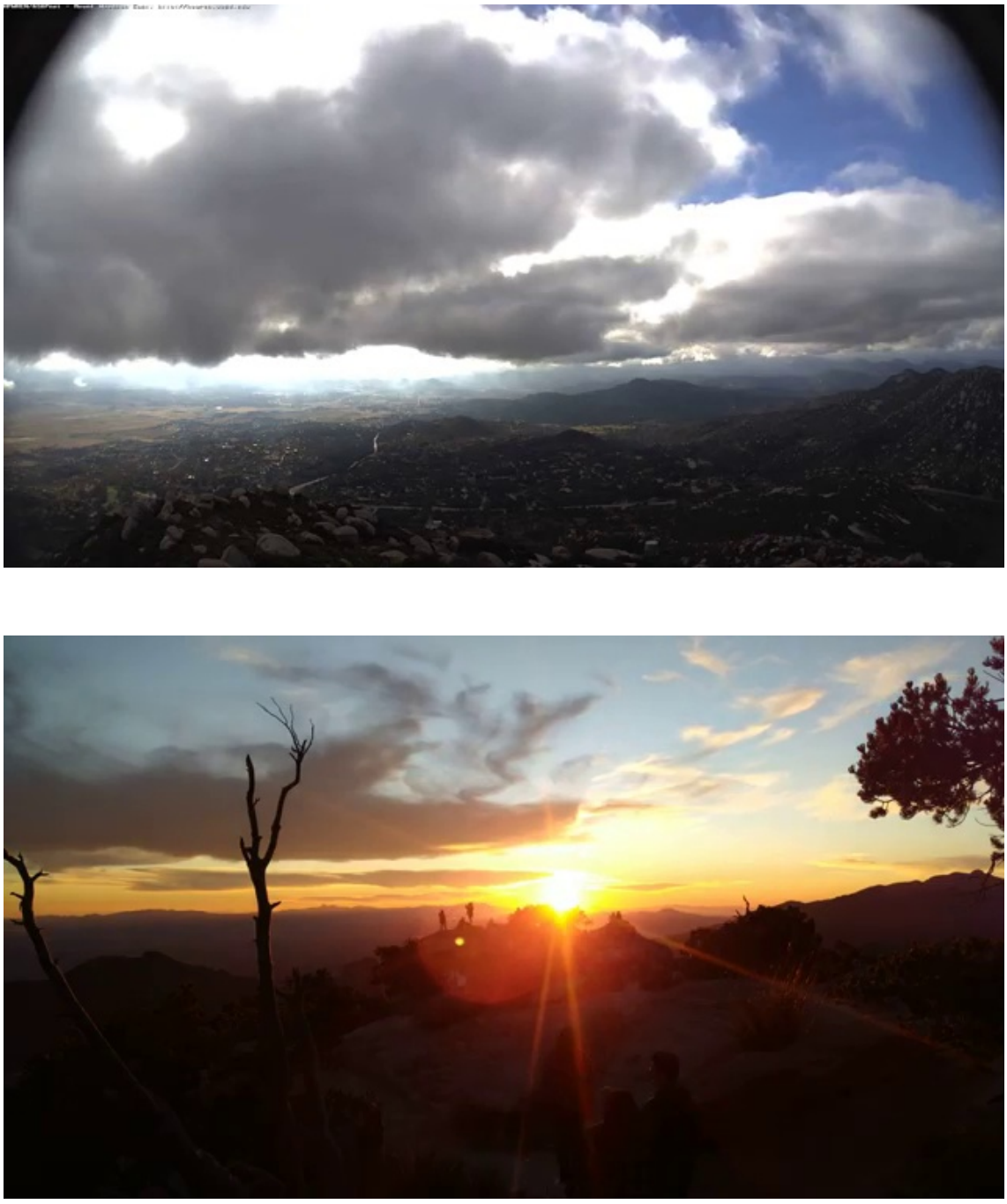
Parametr *–data* definuje, kde je uložen yaml soubor obsahující cestu k datům viz výpis 2.1. Pro nastavení rozlišení používáme *–img*, používáme rozlišení 640×640, pro menší rozlišení síť hůř detekuje menší objekty, větší rozlišení naopak je náročnější na výpočetní kapacitu, předpokládá se detekce na stejném rozlišení na které byla síť trénovaná. Parametr *–cfg* specifikuje cestu k nastavení sítě, pro nás konkrétně cestu k základnímu YOLOv7 modelu, existují i další modely, které mají odlišnou strukturu sítě a jsou například určeny pro aplikaci na méně výpočetně vybavených zařízeních, příkladem takového modelu je například YOLOv7-tiny. Prázdný parametr *–weights* používáme pro trénování sítě bez předem předučených vah, tedy takzvaně pro učení „from scratch“. Název výstupní složky s daty definujeme v parametru *–name* (ta se uloží do složky ./runs). Dále je potřeba určit cestu k tzv. „hyperparametrům“, které se dají po procesu učení měnit za účelem dosažení lepších výsledků, v této práci ponecháváme na základních hodnotách. Jako poslední definujeme jaký počet epoch se má síť trénovat.

2.2.2 Trénování sítě s před-učenými vahami

Jedná se o takzvaný „Transfer Learning“, kde používáme váhy z předem naučené sítě na detekování jiných objektů, my pouze přeučíme síť detekovat objekty, které vyžadujeme. Myšlenka takového učení spočívá v tom, že se síť nemusí znovu učit rozpoznávat různé obrazové vzory a pouze se naučí soustředit se na objekty, které definujeme jako označené v našem datasetu. Váhy které budeme používat byly natrénovány za pomocí datasetu MS COCO na modelu YOLOv7. Trénování probíhalo pro 200 epoch, vzhledem k nedostatu místa na serveru se model trénoval na prvních 100 epoch, následně pak na dalších 100 (bylo potřeba vymazat objemné soubory s vahami a zanechat pouze soubor best.pt, který obsahoval nejlepší váhy pro pokračování v učení). Byl použit příkaz:

```
python3 train.py --workers 4 --device 0 --batch-size 8 --data data/
    firesmoke.yaml --img 640 640 --cfg cfg/training/yolov7.yaml --
    weights 'yolov7_training.pt' --name yolov7 --hyp data/hyp.scratch.
    p5.yaml --epochs 200
```

parametry se liší od předchozího příkazu učení pouze v definici souborů s vahami, tentokrát definice vah není prázdná, ale ukazuje na soubor s již naučenými vahami. Takto máme natrénován model YOLOv7 dvěma způsoby, nyní je třeba vyhodnotit, který postup dosahuje lepších výsledků, tedy který bude dále použit ve snaze zlepšit jeho přesnost.



Obr. 2.3: Ukázka dat z datasetu – potencionálně falešné detekce

Obr. 2.4: Ukázka dat z datasetu očekávaných detekcí

2.3 Výsledky naučených sítí YOLOv7 a jejich testování

V této kapitole budeme porovnávat dosažené výsledky trénování 2 metod učení za pomocí různých metrik. Nejprve je však nutné jednotlivé metriky uvést a objasnit abychom mohli natrénované modely porovnat. Výčet jednotlivých metrik, které budeme používat:

- **ztráta GIoU (Box loss)** – udává, jak dobře zvládne algoritmus nalézt střed hledaného objektu a k němu predikovat odpovídající ohraničující box. [21] K tomu se využívá metrika GIoU (Generalized Intersection over Union) – ta využívá obecný výpočet IoU, neboli podíl průniku se sjednocením jednotlivých ploch ohraňičujících boxů (predikovaného a pravdivého), ovšem pro samotné IoU nastává problém, pokud se jednotlivé boxy vůbec nepřekrývají. To GIoU řeší vytvořením nejmenšího možného třetího boxu, který bude překrývat predikovaný box s boxem pravdivým a jeho plocha bude určovat velikost chyby odchylky boxů, díky tomu je možné algoritmus učit i přesto, že se nějaké predikce budou překrývat (protože bude možné definovat ztrátu). Dále je možné využít například „Distance-IoU“. [41]
- **Ztrátovost objektivity (Objectness loss)** – Objektivita jako taková, udává, jaká je pravděpodobnost P_0, že v označené oblasti existuje hledaný objekt. [21] Ztrátovost objektivity je pak $1 - P_o$.
- **Klasifikační ztrátovost (Classification loss)** – Obecně klasifikace znamená, přiřazení detekovaného objektu dané třídě. Klasifikační ztrátovost pak udává, jak moc síť zaměňuje jednotlivé třídy při detekci [21].
- **Přesnost (Precision)** – Jednoduchá metrika udávající počet správně detekovaných objektů v poměru se všemi detekcemi, které model označil jako pozitivní. Matematicky jako:

  $$P = \frac{TP}{TP + FP},$$

  kde TP – True Positive, FP – False Positive.
- **Schopnost rozpoznávání (Recall)** – Udává, kolik objektů síť zapomene označit $(1-R)$, tedy schopnost rozpoznávání objektů jako takových, čím menší Recall, tím větší šance, že objekt nebude detekován. Matematicky jako:

  $$R = \frac{TP}{TP + FN},$$

  kde TP – True Positive, FN – False Negative.
- **střední průměrná přesnost – mAP (mean average precision)** – vyjadřuje průměrnou hodnotu přesnosti pro určitý práh detekovatelnosti objektů

(např. pro práh 0,5 model nebude brát v potaz detekce, u kterých si není jist na více jak 50 %). Obvykle mAP určujeme pro práh 0,5 a 0,5–0,95. Matematicky lze střední průměrnou přesnost mAP vyjádřit jako:

$$\text{mAP} = \frac{1}{N} \sum_{i=1}^{N} \text{AP}_i,$$

kde N značí počet tříd, AP_i pak průměrná přenost pro danou třídu.

- **F1 skóre** – udává optimální hodnotu práhu detekce, kde je vztah přesnosti a schopnosti rozpoznávání nejlepší. Z grafu závislosti hodnoty F1 na jistotě modelu (viz obr. 2.5 a obr. 2.6), lze určit vhodný práh detekce pro následné použití modelu v praktickém nasazení (je ovšem také vhodné brát v potaz smysl samotného využití, pro některé případy je vhodné detekovat s co nejvyšší jistotou a naopak). Skóre F1 je definováno jako:

$$F1 = \frac{2 \times (Precision \times Recall)}{Precision + Recall}.$$

Oba natrénované modely se od sebe v těchto metrikách příliš výrazně neliší, nejprve porovnejme metriky pro posledních 10 epoch učení v každém modelu, viz tabulky 2.4 a 2.5. Hodnoty v tabulce $\mathcal{Z}(B)$, $\mathcal{Z}(O)$, $\mathcal{Z}(C)$, značí ztrátovou funkci boxu, objektivity a klasifikační ztrátovost v daném pořadí.

Epocha	$\mathcal{Z}(B)$	$\mathcal{Z}(O)$	$\mathcal{Z}(C)$	Precision	Recall	mAP@0,5
251	0,04091	0,00843	0,00518	0,7091	0,7017	0,7473
252	0,04089	0,00843	0,00518	0,7111	0,7011	0,7474
253	0,04086	0,00843	0,00519	0,7124	0,7009	0,7481
254	0,04084	0,00844	0,00518	0,7134	0,7002	0,7489
255	0,04081	0,00844	0,00520	0,7136	0,7005	0,7494
256	0,04085	0,00844	0,00520	0,7172	0,6975	0,7495
257	0,04091	0,00843	0,00519	0,7111	0,7028	0,7495
258	0,04079	0,00843	0,00519	0,7124	0,7016	0,7495
259	0,04079	0,00842	0,00519	0,7101	0,7025	0,7488
260	0,04080	0,00842	0,00518	0,7181	0,6945	0,7481

Tab. 2.4: Ukázka výsledků posledních 10 epoch – učení sítě bez počátečních vah

Z tabulek výsledků lze vidět, že výsledky se v posledních epochách mění pouze zanedbatelně a rozdíl od obou sítí je nepatrný. Lepších výsledků dosahuje síť druhá, která byla přeučena na detekování kouře a ohně z počátečních vah naučených na MS COCO datasetu – dosahuje zanedbatelných ztrát a preciznosti kolem 78 %. Samotné tabulky výsledků, které poskytuje síť YOLOv7 jako výstup učení modelu

Epocha	$\mathcal{Z}(B)$	$\mathcal{Z}(O)$	$\mathcal{Z}(C)$	Precision	Recall	mAP@0,5
191	0,03731	0,01007	0,00536	0,7857	0,7210	0,7803
192	0,03730	0,01009	0,00535	0,7795	0,7229	0,7802
193	0,03728	0,01010	0,00535	0,7822	0,7200	0,7803
194	0,03727	0,01012	0,00536	0,7689	0,7328	0,7801
195	0,03725	0,01013	0,00536	0,7680	0,7332	0,7805
196	0,03724	0,01015	0,00536	0,7678	0,7343	0,7799
197	0,03722	0,01016	0,00537	0,7685	0,7349	0,7800
198	0,03721	0,01017	0,00537	0,7693	0,7353	0,7802
199	0,03720	0,01018	0,00537	0,7699	0,7355	0,7806
200	0,03718	0,01020	0,00537	0,7694	0,7355	0,7800

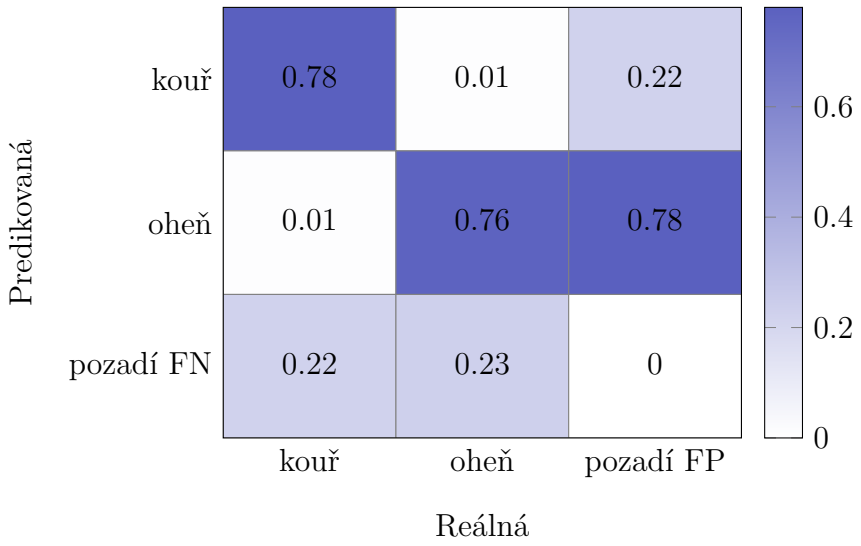
Tab. 2.5: Ukázka výsledků posledních 10 epoch – učení s předem naučenými vahami

nám ovšem o schopnostech sítě neřeknou vše, je potřeba brát v potaz i další metriky. Porovnejme tedy tzv. „Matice záměn“ pro jednotlivé modely. Matice záměn nám říká, jakou má model tendenci zaměňovat jednotlivé třídy za jiné. Zároveň nám říká, jaký je poměr falešně pozitivních detekcí jednotlivých tříd (poslední sloupec v tabulce záměn) a falešně negativních detekci (model neoznačí jednotlivé třídy – poslední řádek v tabulce záměn). Viz tabulky 2.6 a 2.7.

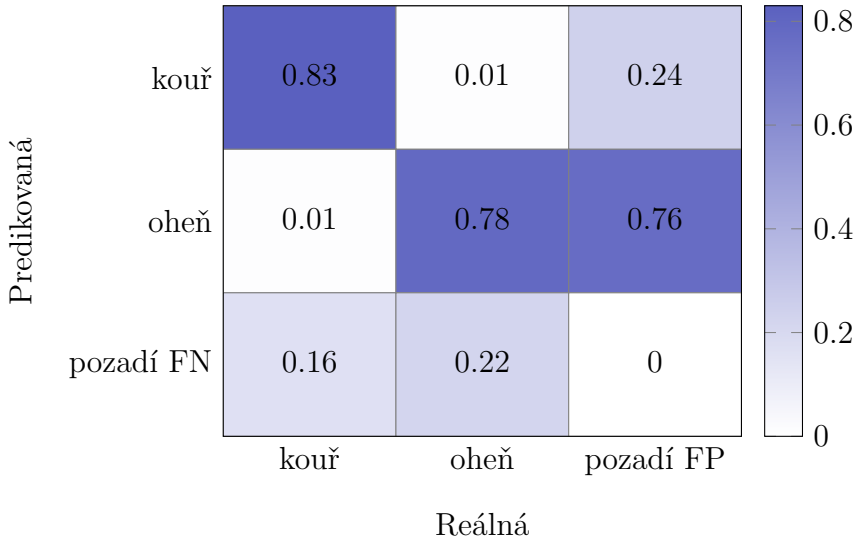
Z tabulek zámeň lze vidět, že obě sítě téměř nezaměňují oheň za kouř a naopak. Zároveň lze říci, že obě sítě jsou téměř 4× náchylnější na falešnou detekci ohně, na rozdíl od kouře, pro kouř je tato hodnota 22–24 %. Falešně negativní detekce, jsou všechny pod hodnotou 23 %, takový výsledek je pro některé reálné aplikace neuspokojivý, je však potřeba brát v potaz náročnost detekce kouře např. za nepříznivých podmínek počasí. Z tabulek záměn lze, stejně jako tabulky výsledků, usoudit, že druhý model dosahuje lepších výsledků, rozdíl však není příliš markantní.

Další metrikou používanou pro vyhodnocování vlastnostní naučených modelů je tzv. „F1 křivka“, uvedená v seznamu metrik viz začátek kapitoly 2.3. Z obou křivek (viz obr. 2.5 a obr. 2.6) lze vidět, že pro detekci ohně nepřesahuje hodnot větších jak 0,7, pro kouř pak nanejvýš 0,8. Opět lze posoudit, že model s přeučenými vahami dosahuje lepších výsledků (ideální hodnoty jsou 0,8 a více, jak pro osu x, kde je vynesena jistota, tak pro osu y na které je vynesena závislost F1 na jistotě).

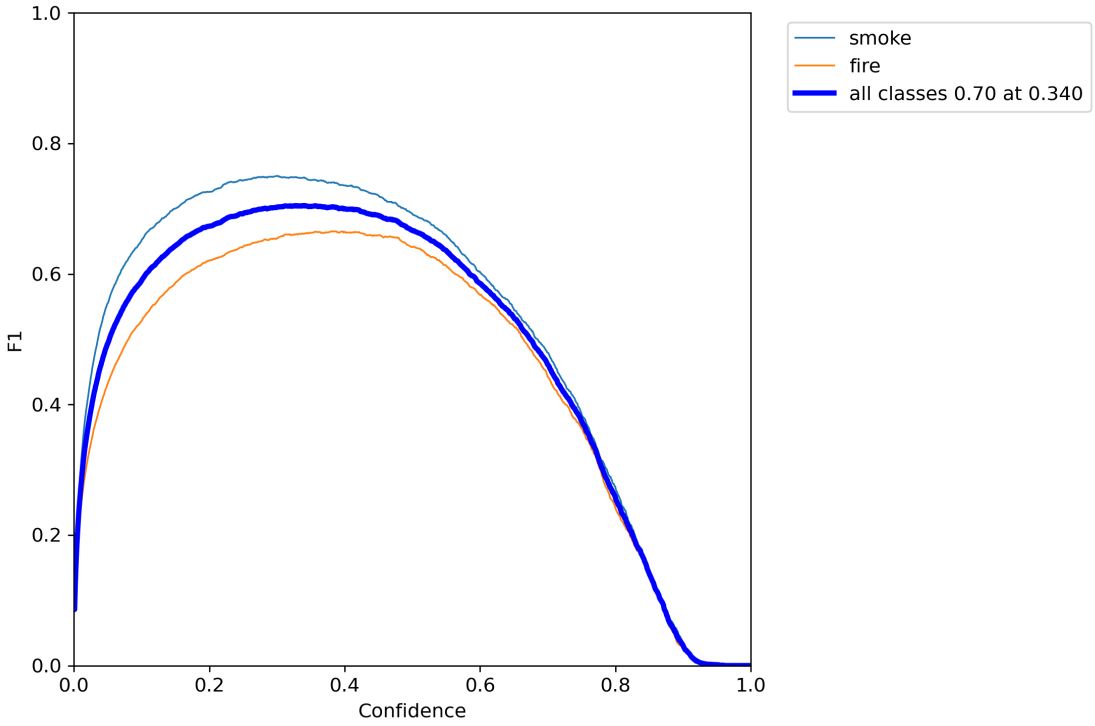
V následujících kapitolách se budeme zabývat možnostmi zlepšení modelu dosahujících vhodnějších výsledků, konkrétně jeho detekčních schopností.



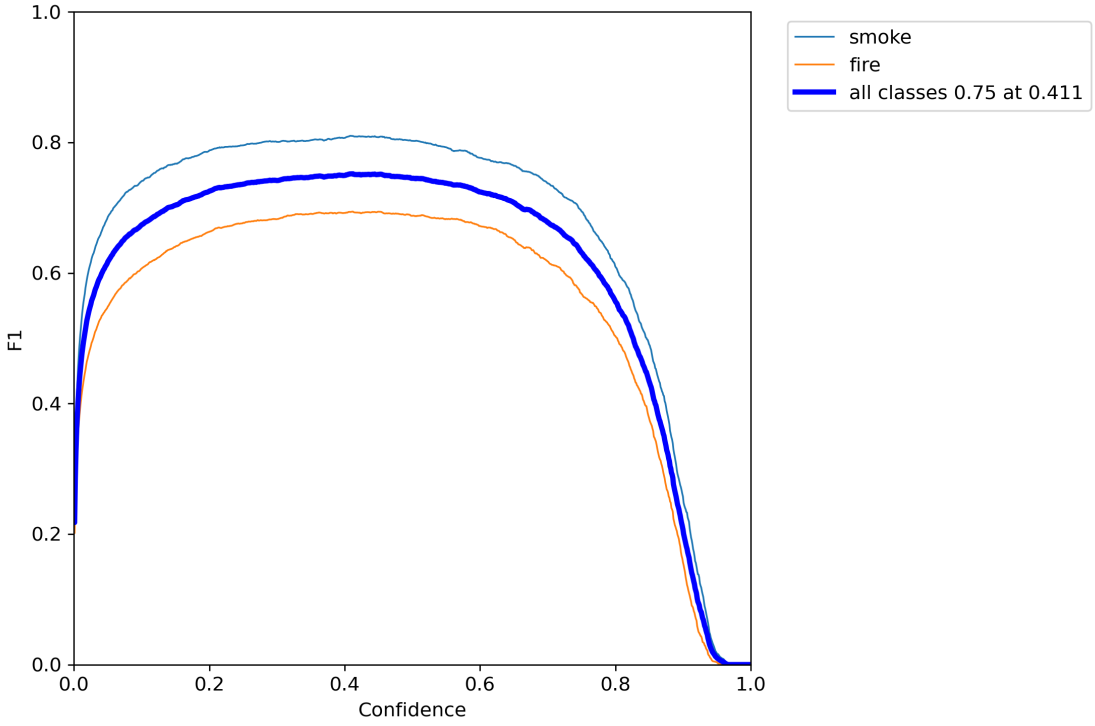
Tab. 2.6: Matice záměn pro učení bez počátečních vah



Tab. 2.7: Matice záměn pro učení s předem naučenými vahami



Obr. 2.5: křivka F1 pro učení bez počátečních vah



Obr. 2.6: křivka F1 pro učení s předem naučenými vahami

2.3.1 Testování rychlosti naučených sítí

Testování rychlosti detekce bylo provedeno na grafické kartě NVIDIA GeForce RTX 2080Ti. Test byl proveden pro oba natrénované modely postupně pro jednotlivé velikosti šarží. Pro testování byl použit již připravený kód *test.py*, který je součástí YOLOv7 repositáře. Ukázka použitých parametrů pro testování:

```
python3 test.py --data data/firesmoke.yaml --img 640 --batch 1 --conf
    0.001 --device 0 --weights ./best.pt --task speed
```

Oba modely byly testovány na 3 různých velikostech šarže – 1, 8 a 16, rozlišení obrazu odpovídá rozlišení, na kterém byla síť trénována. Výsledky testování modelu 2 lze vidět v tabulce 2.8 (hodnoty pro oba modely byly stejné). Celková rychlost detekce se skládá z detekce objektů samotných a pak také z tzv. „NMS“ – Non-Maximal Suppression. NMS je algoritmus, který používají detekční algoritmy k rozhodování a vybírání mezi jednotlivými navrženými ohraničujícími boxy, které síť generuje (síť vygeneruje více ohraničujících boxů pro každou detekci). Ten je schopen vybrat takový ohraničující box, který má největší shodu s boxem označeným jako pravý. Existují různé typy jako například Greedy-NMS, Soft-NMS,.. [3]. Výsledek procesu NMS lze vidět na obr. 2.7.

Obr. 2.7: Proces NMS

	Model 2		
Velikost šarže	1	8	16
detekce [ms]	8,3	3,7	3,6
NMS [ms]	0,4	0,4	0,4
celkem [ms]	8,6	4,1	4,0

Tab. 2.8: Výsledky testování rychlosti detekce

2.4 Implementace optického toku

S naučenou sítí YOLOv7, sloužící jako páteřní sít pro náš detekční model, je nyní možné začít implementovat klasifikátory pro optický tok kouře a ohně. Jako první je však nutné nashromáždit dostatečné množství dat – pro sběr dat je nutné upravit aktuální kód. Jako základ byl použit dataset Firesense [15], doplněný o několik pozitivních vzorků (videí) pro kouř i oheň. Upravován byl kód v souboru *detect.py* – vypočtený průměrný optický tok za 10 snímků byl jako podvzorkovaný na velikost (20×20×2) a ukládán jako .npy (numpy array) pro danou třídu. Stručný přehled dat lze vidět v tab. 2.9 a tab. 2.10.

Oheň	
celkem	6601
pozitivní	1902
negativní	4699

Tab. 2.9: data optického toku – oheň

Kouř	
celkem	8291
pozitivní	2410
negativní	5881

Tab. 2.10: data optického toku – kouř

Na nasbíraných datech budou postaveny 2 implementace: algoritmus SVM (Support Vector Machine) a neuronová klasifikační síť. Obě poté budou sloužit jako nástavba nad model YOLOv7, poskytující dodatečnou informaci z časové složky videa založenou na optickém toku. Pro implementaci algoritmu SVM použijeme knihovnu *scikit-learn* (implementaci lze najít v `/Classifiers/svm.py` – commit 07c840d7 – větev pro optický tok[1]) K navržení a naučení klasifikační neuronové sítě použijeme knihovnu *Tensorflow* a její nástavbu *Keras*. Architektura, se kterou bylo dosaženo nejlepších výsledků, je formou textového výpisu vyobrazena na obr. 2.8.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 layer1 (Dense)              (None, 20, 20, 800)       2400

 layer2 (Dense)              (None, 20, 20, 400)       320400

 layer3 (Dense)              (None, 20, 20, 200)       80200

 layer4 (Dense)              (None, 20, 20, 100)       20100

 flatten (Flatten)           (None, 40000)             0

 layer5 (Dense)              (None, 1)                 40001

=================================================================
Total params: 463,101
Trainable params: 463,101
Non-trainable params: 0
```

Obr. 2.8: Architektura klasifikační NN pro optický tok

[1]git repositář dostupný na: <`https://github.com/PrimalMight/Yolo7-Updated`>.

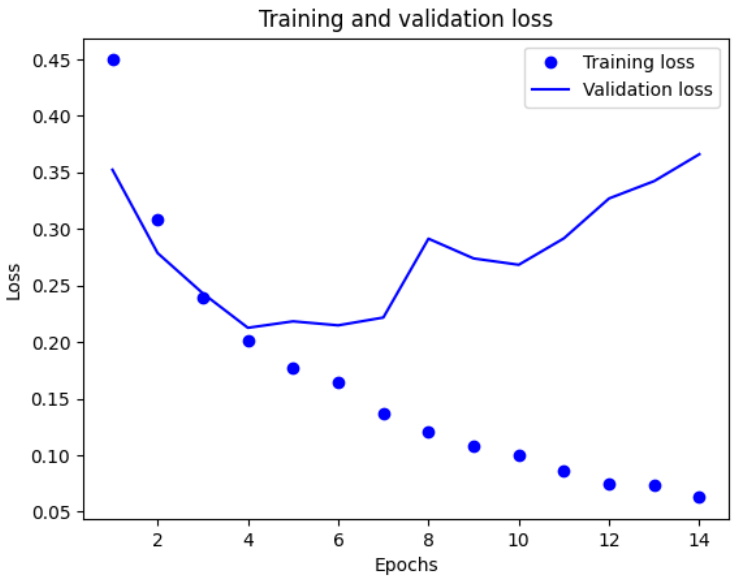
Při kompilaci modelu byla zvolena nástavba pro klasický gradientní sestup – *RMSprop* k hledání minima ztrátové funkce (viz kap. 1.2), v tomto případě byla zvolená funkce *binary_crossentropy* (pravděpodobnostní rozložení mezi hodnotami 0 - 1 pro jednu třídu). Jako nejlepší se ukázala velikost šarže 16. Počet epoch byl při spuštění trénování nastaven na 30, v implementaci však používáme tzv. „callback“, který v případě, že nastane přetrénování modelu na trénovací data a model začne „overfittovat“, trénování zastaví a vrátí se k nejlepším dosaženým metrikám modelu a uloží pro tuto epochu váhy.

Navržený a zkompilovaný model se trénoval pro každou třídu (oheň a kouř) zvlášť, výsledky učení – ztrátovost, lze vidět na obr. 2.9 a obr. 2.10, u obou modelů dochází poměrně rychle k přeučení a modely nedokáží příliš generalizovat na datech, které model nikdy neviděl. Tento fakt připisujeme k nepříliš vypovídající informaci optického toku o třídách samotných (viz obr. 2.11 pro oheň a dále obr. 2.12, 2.13 a 2.14 pro kouř), model nedokáže tak dobře rozpoznat vlastnosti optického toku pouze pro kouř a pouze pro oheň.

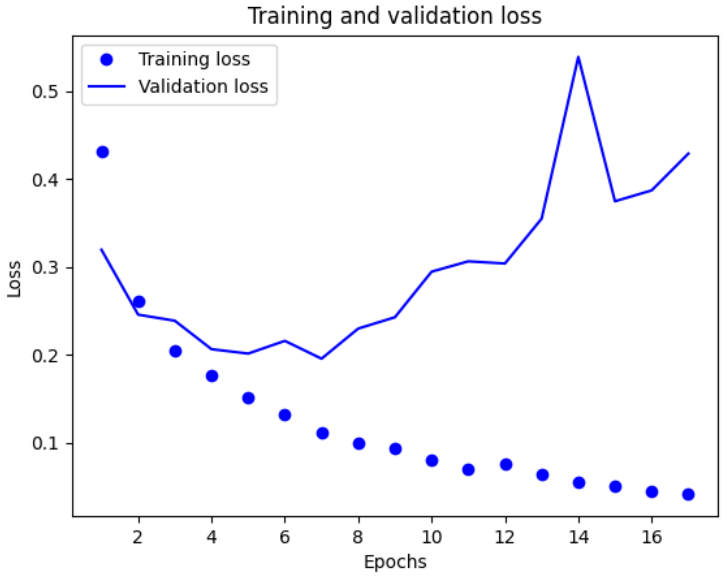
Podobně je na tom implementace SVM, která v průměru dosahuje přesnosti okolo 50 % (tedy vlastně náhodný výběr – SVM není možné využít, ani jako přídavný rozhodující faktor, protože je téměř náhodný – model se nebyl schopen naučit).

Příčinou neschopnosti sítí se kvalitně naučit rozpoznávat kouř a oheň od „pozadí“ snímků je také velká variace optického toku pro samotné třídy kouře a ohně, kdy např. záleží, zda je hledaná třída na záběru blízko, nebo ve vzdáleném pozadí – od toho se bude vyvíjet velikost optického toku (objekty v blízkém záběru vykazují při pohybu výrazně větší pohyb v obrazových bodech, než objekty vzdálené). Dále pak fakt, že optický tok ohně / kouře není vždy specifický pohyb vzhůru (viz např. obr 2.11) – pokud tomu tak není, mohou se často různé pohyby objektů ve videozáznamu podobat optickému toku kouře či ohně, který se snažíme rozpoznat. Tuto skutečnost nejsme schopni dostatečně obsáhnout v relativně malém datasetu (dodat dostatečné množství vzorků ohně i kouře, aby síť byla schopna najít alespoň některé odlišnosti), větší dataset pouze pro oheň a kouř není v tuto chvíli k dispozici, při větších datasetech lze předpokládat zlepšení hodnot ztráty sítě, tak, aby byla implementace YOLOv7 + klasifikační NN & SVM modelu dávala spolehlivější výsledky. Myšlenka spojit detekci ohně a kouře v jeden celek pro klasifikaci na základě optického toku obou tříd vykazovala podobné / horší klasifikační výsledky.

Implementaci optického toku pro detekci ohně a kouře obecně (v libovolném prostředí), lze tedy považovat za nevhodnou – v případě snímání specifického prostoru (např. prostor, kde lze vyloučit jakýkoliv pohyb) by využití optického toku našlo své uplatnění. I přes nedokonalé výsledky, je navržená implementace funkční a kód je navržen tak, aby bylo v případě potřeby možné dodat data pro naučení klasifikačních modelů a zlepšit tak přesnost celého procesu detekce ohně a kouře.



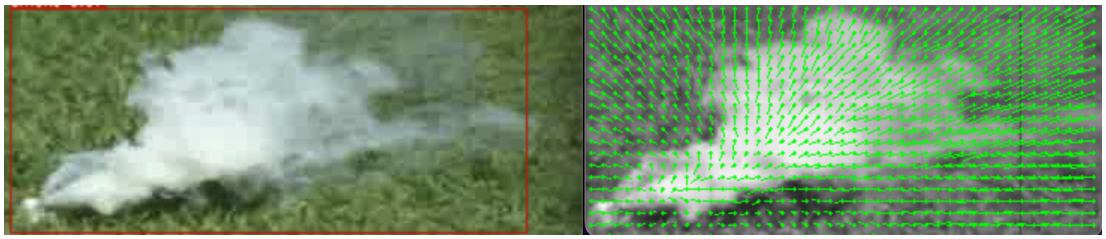
Obr. 2.9: Výsledky trénování neuronové klasifikační sítě pro třídu oheň



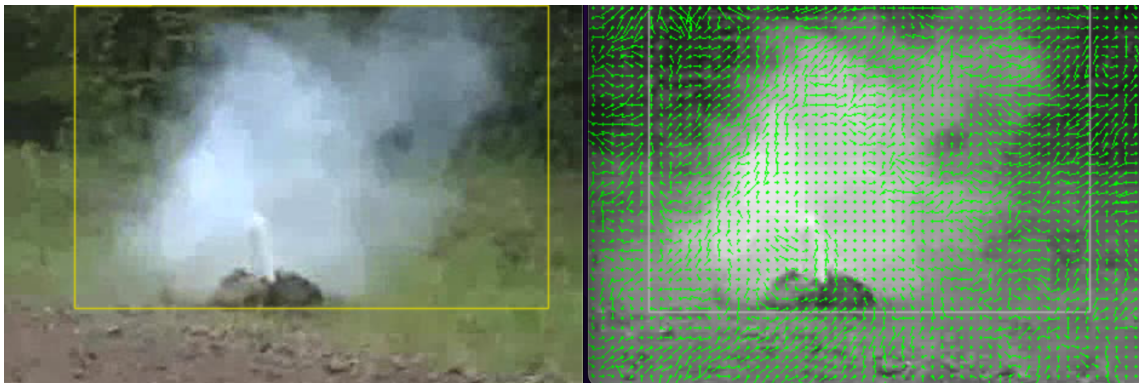
Obr. 2.10: Výsledky trénování neuronové klasifikační sítě pro třídu kouř

Obr. 2.11: Ukázka optického toku pro pozitivní a negativní detekci ohně

Obr. 2.12: Ukázka optického toku pro pozitivní detekci kouře



Obr. 2.13: Optický tok kouře při přiblížení záběru



Obr. 2.14: Optický tok kouře při pohybujícím se záběru

Finální algoritmus po implementaci optického toku a naučení klasifikátorů (*detect.py* v kořenovém adresáři yolov7 – commit 07c840d[2]) probíhá v následovně:

1. Postupné plnění vyrovnávací paměti s fixní velikostí snímky, které již prošly sítí YOLOv7 a obsahují / neobsahují detekce.
2. V případě, že je vyrovnávací paměť plná a snímky v paměti obsahují dohromady alespoň 5 detekcí jakékoliv třídy (velikost vyrovnávací paměti i počet nutných snímků pro výpočet optického toku je nastavitelná parametry při spuštění), pokračuje algoritmus na výpočet celkového ohraničujícího boxu, který svou velikostí zahrnuje všechny ohraničující boxy pro danou třídu (tento výpočet se děje pro každou třídu zvlášť). Pokud je vyrovnávací paměť plná, ale neobsahuje dostatečné množství detekcí – pokládáme detekce za falešně pozitivní a podle fronty FIFO vyhazujeme nejstarší snímek, na uvolněné místo v vyrovnávací paměti následuje snímek nový.
3. Vypočtený celkový ohraničující box je následně vyjmut z celkového záběru snímků a podle jejich časové souslednosti je spočítán optický tok (před výpočtem jsou ořezané snímky převedeny do černobílé podoby) – k výpočtu je použita funkce z knihovny openCV (cv2.calcOpticalFlowFarneback()) [27].
4. Optický tok je následně zprůměrován a podvzorkován na velikost (20×20×2).
5. Podvzorkovaný optický tok vstupuje do natrénovaných modelů SVM a NN klasifikátorů pro jednotlivé třídy.
6. Výsledkem je vypsání do terminálu průměrné jistoty sítě YOLOv7, počet detekcí pro daný vyrovnávací pamět, predikce SVM a predikce NN sítě
7. Jako volitelné možnosti výstupu jsou: ukládání jednotlivých snímků, samostatných detekcí, průměrného ohraničujícího boxu pro obě třídy, optického toku pro průměrný ohraničující box a optický tok celých snímků z vyrovnávací paměti.

[2]git repositář dostupný na: <`https://github.com/PrimalMight/Yolo7-Updated`>.

2.5 Implementace klasifikační neuronové sítě

Jako druhé, alternativní řešení zlepšení detekce se nabízí natrénovat klasifikační neuronovou síť, která bude trénována na výstupech ze sítě YOLOv7, tedy na již ořezané detekce převzorkované na 200×200 pixelů. Tyto vzorky je tedy potřeba před započetím trénování nasbírat, k tomu je potřeba opět upravit kód v souboru *detect.py* (commit – 3937dd7). Vytvořený dataset lze vidět numericky v tab.2.11.

Jednotlivé vzorky (viz obr. 2.15, 2.16, 2.17) byly „generovány“ výstupem sítě

Celkem	7206
oheň	3350
kouř	2384
pozadí	1472

Tab. 2.11: Nasbíraná data pro klasifikační síť

YOLOv7, při hodnotě jistoty 0.6 pro oheň a kouř, pro vytvoření vzorků bez detekcí byla jistota nastavena na 0.01 (simulace FP detekce ze sítě YOLOv7). Jednotlivé vzorky byly manuálně ověřeny (chybné detekce ohně / kouře). Vzorky pro třídu oheň a kouř byly získány z datasetu Firesense [15], pro vytvoření vzorků neobsahující žádnou třídu byly použity dostupná videa na [8].

Obr. 2.15: vzorky detekcí kouře z datasetu

Zvolená architektura sítě je naznačena na obr. 2.18. Jedná se o sekvenční model navržený za pomocí knihovny *Keras*, kde základ tvoří předučená síť VGG16 (viz obr. 1.8), za kterou následuje sdružovací vrstva. Výstupní vrstva je plně spojená se 3 výstupními neurony (3 třídy). Všechny vrstvy sítě VGG16 jsou „zamrzlé“ – tedy nepodléhají *zpětné propagaci*, síť má 1539 parametrů, které lze během učení měnit.

Obr. 2.16: vzorky detekcí ohně z datasetu

Obr. 2.17: vzorky bez detekcí z datasetu

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 vgg16 (Functional)           (None, 6, 6, 512)         14714688

 global_average_pooling2d (G  (None, 512)               0
 lobalAveragePooling2D)

 dense (Dense)                (None, 3)                 1539

=================================================================
Total params: 14,716,227
Trainable params: 1,539
Non-trainable params: 14,714,688
_________________________________________________________________
```

Obr. 2.18: Architektura klasifikační NN

2.5.1 Proces a výsledky učení klasifikátorů

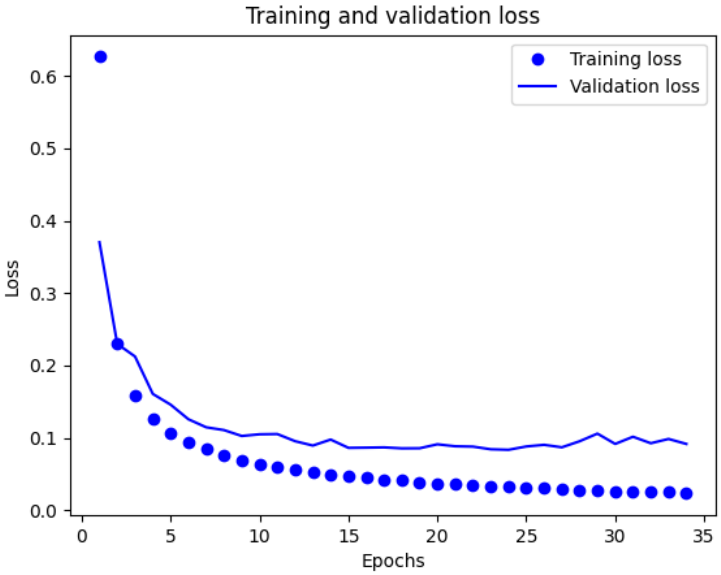
Pomocí zvolené architektury (obr. 2.18) byly naučeny 2 modely. Oba modely jsou binární klasifikátory – Oheň / Nic a Kouř / Nic. Oba modely byly trénovány po 34 epoch. Stejně jako při učení klasifikátoru optického toku i zde jsou využity „callbacky“. Výsledky v textové formě lze vidět v tab. 2.12 a 2.13.

Epocha	loss	accuracy	val_loss	val_accuracy
24	0,0323	0,9917	0,0836	0,9699
25	0,0303	0,9914	0,0882	0,9751
26	0,0303	0,9930	0,0905	0,9751
27	0,0287	0,9927	0,0870	0,9761
28	0,0276	0,9922	0,0954	0,9740
29	0,0274	0,9930	0,1060	0,9730
30	0,0263	0,9930	0,0917	0,9740
31	0,0259	0,9932	0,1016	0,9740
32	0,0253	0,9927	0,0927	0,9761
33	0,0251	0,9930	0,0986	0,9740
34	0,0232	0,9940	0,0917	0,9720

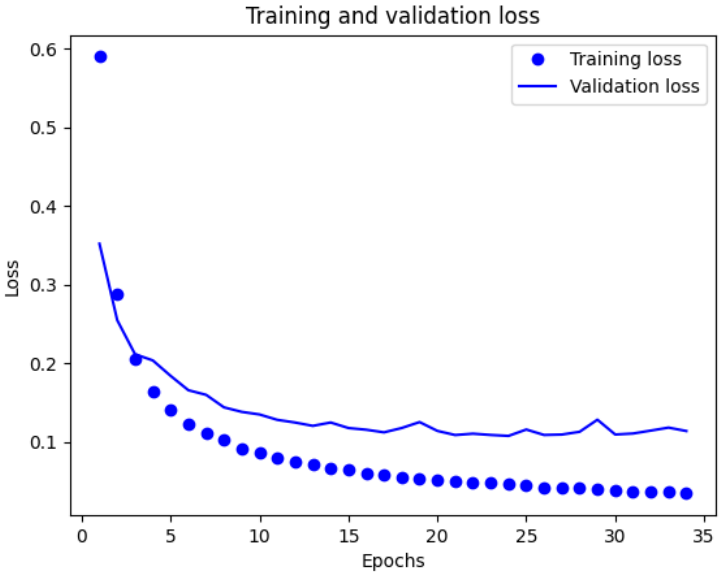
Tab. 2.12: Výsledky učení klasifikátoru ohně – posledních 10 epoch

Epocha	loss	accuracy	val_loss	val_accuracy
24	0,0458	0,9821	0,1075	0,9662
25	0,0447	0,9818	0,1156	0,9610
26	0,0418	0,9847	0,1087	0,9675
27	0,0423	0,9857	0,1092	0,9701
28	0,0413	0,9841	0,1127	0,9675
29	0,0396	0,9870	0,1281	0,9532
30	0,0384	0,9876	0,1093	0,9662
31	0,0366	0,9880	0,1105	0,9649
32	0,0360	0,9873	0,1143	0,9649
33	0,0360	0,9886	0,1181	0,9649
34	0,0342	0,9889	0,1137	0,9662

Tab. 2.13: Výsledky učení klasifikátoru kouře – posledních 10 epoch



Obr. 2.19: Výsledky učení klasifikátoru ohně v grafu



Obr. 2.20: Výsledky učení klasifikátoru kouře v grafu

Výsledkem jsou 2 natrénované binární klasifikátory, které jsou „zapojeny“ za výstup ze sítě YOLOv7. Poslední důležitou částí v rámci implementace těchto klasifikátorů je ověřit jejich schopnost celkově zlepšit výsledky detekcí z celého modelu – tedy síť YOLOv7 + 2 klasifikační neuronové sítě.

2.5.2 Měření ROC a rychlosti

Klasifikační schopnost sítě YOLOv7 bez a se zapojenými klasifikátory vyhodnotíme pomocí tzv. ROC (Receiver Operating Characteristic). Na vstupu budeme síti předávat snímky, přičemž budeme sledovat, co síť na základě snímku vyhodnotí – zdali se jedná o kouř, oheň, nebo snímek klasifikuje jako bez ohně a kouře. Výsledek ze sítě je porovnán se skutečnou značkou (labelem) snímku. Poté zaznamenáme zdali se jedná ze strany sítě o TP (True positive), FP (False positive), TN (True negative), FN (False negative) detekci. Z těchto zaznamenaných stavů následně vypočítáme

stav	značka snímku	predikce sítě
TP	na snímku je hledaná třída	síť klasifikuje snímek jako jednu ze tříd
FN	na snímku je hledaná třída	síť klasifikuje snímek jako bez třídy
TN	na snímku není hledaná třída	síť klasifikuje snímek jako bez třídy
FP	na snímku není hledaná třída	síť klasifikuje snímek jako jednu ze tříd

Tab. 2.14: Přehled všech možných stavů klasifikace

jaký je poměr FP a TP k celkovému počtu testů – tedy počítame TPR (True positive rate, někdy také jako *Recall* – již zmíněná metrika v kap. 2.3) a FPR (False positive rate):

$$TPR = \frac{TP}{TP + FN},$$

$$FPR = \frac{FP}{FP + TN}.$$

Toto měření opakujeme pro několik úrovní *confidence* (jistoty), pro oba způsoby (YOLOv7 bez a s klasifikačními modely). Výsledkem jsou dvě ROC křivky (viz obr. 2.21.

Měření proběhlo na 340 snímcích (200 – bez třídy, 85 – oheň, 55 – kouř), tyto snímky nebyly součástí trénovacího procesu žádného z modelů – jsou tedy pro sítě zcela nové. V případě měření TPR, FPR pro implementaci s přidanými klasifikátory nastavujeme hodnoty jistot pro všechny sítě stejně – např. confidence 0.25 pro YOLOv7 i pro oba natrénované modely. Numerické výsledky měření jsou vidět v tab. 2.15 a 2.16. Z obr. 2.21 lze vidět, že přidáním navazujících klasifikátorů jsme nepatrně zvýšili robustnost sítě v oblasti jistot vyšších nežli 0,1. Ideální klasifikátor by pak měl hodnoty TPR co nejblíže k 1, pro hodnoty FPR > 0, tedy křivka by se vyskytovala co nejblíže k levému hornímu rohu. Náhodný klasifikátor by pak představoval lineární funkci TPR(FPR).

jistota	TP	FP	FN	TN	TPR	FPR
0,05	139	84	1	116	0,993	0,420
0,10	134	49	6	151	0,957	0,245
0,15	129	32	11	168	0,921	0,160
0,20	118	19	22	181	0,843	0,095
0,25	106	17	34	183	0,757	0,085
0,30	93	10	47	190	0,644	0,050
0,35	79	7	61	193	0,564	0,035
0,40	60	4	80	196	0,429	0,020
0,45	47	3	93	197	0,336	0,015

Tab. 2.15: Měření TPR a FPR pro samotné YOLOv7

jistota	TP	FP	FN	TN	TPR	FPR
0,05	129	26	11	174	0,921	0,130
0,10	123	17	17	183	0,879	0,085
0,15	116	10	24	190	0,829	0,050
0,20	105	6	35	194	0,750	0,030
0,25	98	6	42	194	0,700	0,030
0,30	86	4	54	196	0,614	0,020
0,35	74	2	66	198	0,529	0,010
0,40	58	0	82	200	0,414	0,000

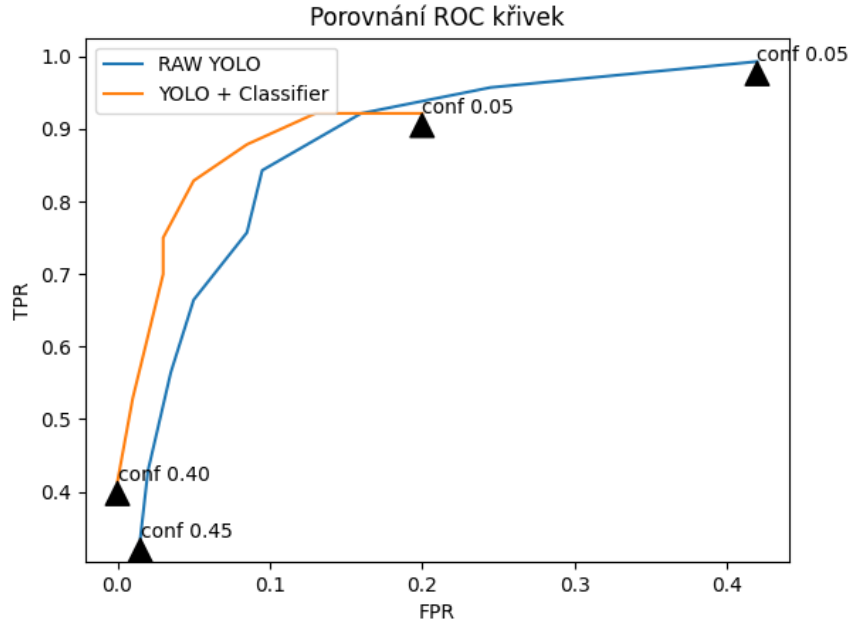
Tab. 2.16: Měření TPR a FPR pro YOLOv7 s klasifikačními modely

Testování rychlosti proběhlo na zařízení MacBook Air(2020) s CPU M1 a 16 GB RAM a to ve třech fázích (výsledky lze vidět v tab. 2.17):

1. Měření rychlosti zpracování 1 snímku (proces plnění vyrovnávací pamětu)
2. Měření rychlosti přidané implementace (zpracování vyrovnávací pamětu a vyhodnocení klasifikátory)
3. Měření naplnění vyrovnávací pamětu pro ideální scénář (dostatek detekcí na 10 snímcích, vyrovnávací pamět se naplní a splní podmínku pro zpracování ihned – nezahazuje žádné snímky)

2.5.3 Finalizace implementace

Finální implementace („pipeline“) pro detekci ohně a kouře, za použití přídavných klasifikačních modelů, lze vidět na obr. 2.22. V repositáři se nachází na hlavní větvi



Obr. 2.21: Výsledné ROC křivky

Měření	1	2	3	4	5	Průměr
1 snímek [ms]	391,5	317,4	328,1	369,9	372,2	355,8
Zpracování vyrovnávací paměti [ms]	269,2	135,1	113,0	104,3	134,5	151,2
Plnění vyrovnávací paměti [s]	3,5	3,6	3,6	3,4	3,7	3,56

Tab. 2.17: Měření rychlostí zpracování

main (commit 70eb3245). Stručný slovní popis funkcionality (graficky pak na obr. 2.22):

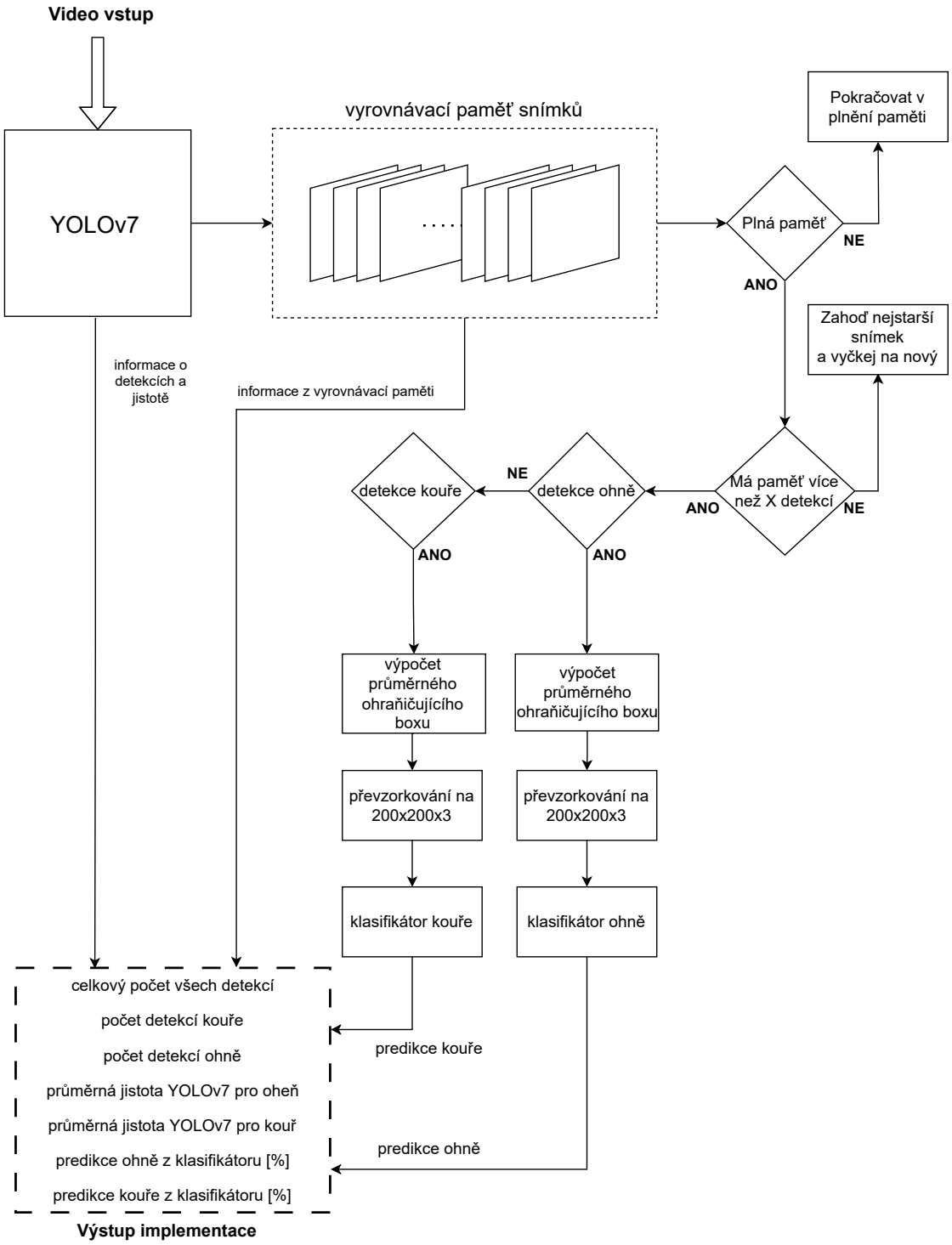
1. Vstupní snímky prochází sítí YOLOv7 (páteřní síť pipeline), která detekuje a klasifikuje jednotlivé třídy. Výstupem jsou označené i neoznačené snímky.
2. Snímky vycházející z procesu YOLOv7 postupně plní vyrovnávací paměť, pokud je vyrovnávací paměť plná a snímky v neobsahují dostatek detekcí (tento počet detekcí je možné manuálně nastavit pomocí parametru *–threshold*) vyrovnávací paměť zahodí nejstarší snímek a čeká na další.
3. Pokud je vyrovnávací paměť plná a obsahuje dostatek snímků s detekcemi, snímky se přenesou do datové struktury listu (vyrovnávací paměť je datové struktury *queue*), zároveň se počítá, kolik detekcí náleží třídě kouř a kolik oheň. To je možné z uložených dat v vyrovnávací paměti (se snímky se zároveň ukládají informace o detekcích na daném snímku)
4. Pokud existuje alespoň 1 nebo více detekcí a k nim i odpovídající ohraničující

boxy, spočítají se „celkové“ ohraničující boxy, které zahrnují celkovou plochu obrazu na kterém se detekce vyskytly. Toto se děje pro obě třídy postupně.

5. Celkové ohraničující boxy pro každou třídu je nutné následně převzorkovat na velikost 200×200×3 – takové je rozhraní (vstupní vrstva) klasifikátorů.
6. Převzorkované ohraničující boxy vstupují do klasifikátorů, jejich výstup je pak pravděpodobnostní klasifikace (kouř / oheň / nic).
7. Spolu s pravděpodobnostmi a doplňujícími informacemi ze sítě YOLOv7 (průměrná jistota detekcí z celkového počtu snímků pro danou třídu), a z vyrovnávací paměti (celkový počet všech detekcí, počet detekcí pro danou třídu) se pro uživatele vypíše do terminálu finální výstup.

2.5.4 Referenční test

Srovnání s dostupnými detektory provedeme pouze na základě několik málo statických snímků (obrázků), nikoliv na videozáznamu z důvodu nedostupnosti natrénovaných komerčních modelů na detekci ohně a kouře (placený přístup / neexistující implementace na video / komerční modely se prodávají přímo již s dedikovaným zařízením (kamera) [38]). Komerční model společnosti ScyllaAI [36] dosahuje až 98,5% přesnosti při detekci ohně a kouře. Natrénované klasifikační modely v této práci dosahují během měření na validačních datech 97,0 % pro kouř a 97,5 % pro oheň – lze tedy konstatovat, že vyhotovená implementace dosahuje výsledků téměř shodnými s komerčními modely. Existuje také několik volně dostupných modelů (jejich zdrojový kód), většina neuvádějící ovšem žádné metriky dosažených výsledků – výjimkou je například dostupný kód od Robina Cole dostupný na [6] dosahující 95 % přesnosti. Schopnost detekce porovnáme např. s volně dostupným detektorem ohně na statických snímcích na webu „Modelplace“ [13] (dostupné API pouze na 25 detekcí / den, pro větší počet detekcí je model placený). Tento model je sice určený pouze k detekci ohně na samotném snímku, ale i toto porovnání nám dá představu, zdali je navržený detektor schopen konkurovat ostatním dostupným modelům. Oba modely byly testovány proti falešně pozitivním detekcím na 25 snímcích. Placený model selhal na 3 snímcích z 25. Model představený v této práci selhal na 0 snímků z 25. Na snímcích s ohněm komerční model selhal na 6 z 25 snímků, model představený v práci selhal na 3 z 25 snímků – některé snímky byly záměrně vybrány jako složité pro detekci (nižší rozlišení, oheň vzdálený v obraze,..). Dále je také potřeba brát v potaz, že implementace představená v práci je vytvořena především na detekci na videozáznamu, kde má možnost pracovat s několika snímky (plnění vyrovnávací paměti, atd.).



Obr. 2.22: Schéma finální implementace

Závěr

V práci byla vybrána detekční síť YOLOv7 jako páteřní síť pro navržený detektor, která byla natrénovaná 2 způsoby. Na základě metody učení vykazující lepší detekční výsledky, byla vytvořena implementace optického toku v kapitole 2.4. Tato implementace nedosahovala očekávaných výsledků, tudíž byla nahrazena alternativním přístupem představeným v kapitole 2.5. Tato alternativa přispěla ke zlepšení detekčních výsledků, jak lze vidět z měření ROC (kapitola 2.5.2).

Další navazující možná implementace z hlediska zpracování dynamické složky ve videozáznamu je sbírat jednotlivé příznaky z již naučených klasifikátorů a sledovat jejich časovou souslednost, následně hledat specifické vlastnosti příznaků. Tato informace by pak následně mohla být obsáhnuta (naučena) klasifikátorem, který by nahradil, případně doplnil možnost nynější celkové implementace pracovat s časovou složkou ve formě jiné, než je optický tok.

Finální implementace by tak obsahovala nejen samotné dodatečné klasifikátory pro jednotlivé třídy, ale zároveň pro každou třídu by byl součástí i klasifikátor, který by na základě dynamické složky poskytoval svoji predikci – složením těchto predikcí (YOLOv7 + „statické“ klasifikátory + „dynamické“ klasifikátory) by vznikla ještě robustnější implementace pro detekci ohně a kouře. Takové učení na základě dynamické složky za použití časově sousledných příznaků vyžaduje další velký sběr dat a vytvoření kvalitního datasetu – tato idea je tedy pouze návrhem pro potenciální zlepšení finální implementace pro detekci ohně a kouře ve videozáznamu.

V kapitole 2.5.4 byla představená detekční síť porovnána s různými dostupnými komerčními modely. Síť je schopna konkurovat v oblasti detekcí ohně a kouře s několika komerčními modely, neboť dosahuje srovnatelných výsledků. Představená detekční síť není vázána na specifický HW, lze ji tedy použít již v existující infrastruktuře, za předpokladu zdroje záznamu (např. IP kamera) a výpočetních zdrojů (hostitelský systém pro detektor).

Literatura

[1] ALBAWI, S.; MOHAMMED, T. A.; AL-ZAWI, S. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)* [online]. IEEE, 2017, 1-6 [cit. 2022-10-19]. ISBN 978-1-5386-1949-0. Dostupné z URL: <https://doi.org/10.1109/ICEngTechnol.2017.8308186>

[2] BOCHKOVSKIY, A.; Wang, C. Y.; Liao, H. Y. M. Yolov4: Optimal speed and accuracy of object detection [online]. arXiv preprint arXiv:2004.10934. 2020 [cit. 2022-10-28] Dostupné z URL: <https://doi.org/10.48550/arXiv.2004.10934>

[3] BODLA, N.; SINGH, B.; CHELLAPPA, R.; DAVIS, L. S. Soft-NMS – Improving Object Detection With One Line of Code. *2017 IEEE International Conference on Computer Vision (ICCV)* [online]. 2017, 5562-5570 [cit. 2022-11-06]. Dostupné z URL: <https://doi.org/10.48550/arXiv.1704.04503>

[4] BORGES DE VENÂNCIO, P. V. A.; LISBOA, A. CH.; BARBOSA, A. V. An automatic fire detection system based on deep convolutional neural networks for low-power, resource-constrained devices. In: Neural Computing and Applications, 2022. Dostupné z URL: <https://github.com/gaiasd/DFireDataset>.

[5] CHOLLET, F. Deep Learning with Python. Second Edition, Shelter Island, NY 11964: *Manning*, 2021. ISBN 9781617296864.

[6] COLE, R. Fire detection from images. GitHub [online]. [cit. 2023-04-27]. Dostupné z: <https://github.com/robmarkcole/fire-detection-from-images>.

[7] Convolutional Neural Networks (CNN) Introduction. In: IndoML [online]. [cit. 2022-10-23]. Dostupné z URL: <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>.

[8] Daily Dose of Internet. In: Youtube [online]. [cit. 2023-04-21]. Dostupné z: <https://www.youtube.com/playlist?list=PLlUZ3i-FUgHqk9-C-Fw_C6YsvTyx2c8nc>.

[9] Deep Neural Network. In: RSIP Vision [online]. c2021 [cit. 2022-12-02]. Dostupné z URL: <https://www.rsipvision.com/exploring-deep-learning>.

[10] DEORE, M. 3D View Of Non-Linear Equation. In: Medium: DevGenius [online]. [cit. 2022-10-16]. Dostupné z URL: <https://blog.devgenius.io/deep-learning-in-gradient-decent-style-part-1-ed747e7cc2a3>.

[11] FARNEBACK, G. Two-Frame Motion Estimation Based on Polynomial Expansion. Image Analysis. 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden: Springer, 2003, 13, 363-370. ISSN 0302-9743.

[12] FILTER, J. split-folders 0.5.1 [online] 2018 [cit. 2022-11-1]. Dostupné z URL: <https://github.com/jfilter/split-folders>.

[13] Fire Detector based on YOLOv4. Modelplace [online]. [cit. 2023-04-29]. Dostupné z: <https://modelplace.ai/models/fire-detector>.

[14] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. Deep Learning [online]. *MIT Press*, 2016 [cit. 2022-10-16]. ISBN 978-0262035613. Dostupné z URL: <http://www.deeplearningbook.org>.

[15] GRAMMALIDIS, N.; DIMITROPOULOS, K.; CETIN, E. FIRESENSE database of videos for flame and smoke detection. In: Zenodo [online]. 31.7.2017 [cit. 2023-03-14]. Dostupné z URL <https://doi.org/10.5281/zenodo.836749>.

[16] HE, K.; ZHANG, X.; REN, S.; SUN, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 2015, 37(9), 1904-1916 [cit. 2022-10-29]. ISSN 0162-8828. Dostupné z URL: <https://doi.org/10.1109/TPAMI.2015.2389824>

[17] KHOSRAVI, H.; HOSSEIN, G. Pooling Methods in Deep Neural Networks, a Review [online]. arXiv preprint arXiv:2009.07485, 2020 [cit. 2022-10-25]. Dostupné z URL: <https://doi.org/10.48550/arXiv.2009.07485>

[18] HU, M.; FENG, J.; HUA, J.; LAI, B.; HUANG, J.; GONG, X.; HUA, X. S. Online Convolutional Re-Parameterization. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. 2022, 568-577 [cit. 2022-10-29]. Dostupné z URL: <https://doi.org/10.48550/arXiv.2204.00826>

[19] JIUXIANG, G.; WANG, Z.; KUEN, J,; a další. Recent advances in convolutional neural networks. *Pattern Recognition* [online]. 2018, 77, 354-377 [cit. 2022-10-19]. ISSN 00313203. Dostupné z URL: <https://doi.org/10.1016/j.patcog.2017.10.013>

[20] KARN, U. An Intuitive Explanation of Convolutional Neural Networks. *Ujjwal-Karn* [online]. 2016 [cit. 2022-10-23]. Dostupné z URL: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

[21] KASPER-EULAERS, M.; HAHN, N.; BERGER, S.; SEBULONSEN, T.; MYRLAND, Ø.; KUMMERVOLD, P. E. Short Communication: Detecting Heavy Goods Vehicles in Rest Areas in Winter Conditions Using YOLOv5. Algorithms [online]. 2021, 14(4) [cit. 2023-05-11]. ISSN 1999-4893. Dostupné z: <https://doi.org/10.3390/a14040114>

[22] KHASANOV, D.; TOJIYEV, M.; PRIMQULOV, O. Gradient descent in machine learning. *2021 International Conference on Information Science and Communications Technologies (ICISCT)* [online]. IEEE, 2021, 1-3 [cit. 2022-10-16]. ISBN 978-1-6654-3258-0. Dostupné z URL: <https://doi.org/10.1109/ICISCT52966.2021.9670169>

[23] LEUNG, K. VGG16 model. In: Towards Data Science [online]. [cit. 2022-10-26]. Dostupné z URL: <https://towardsdatascience.com/how-to-easily-draw-neural-network-architecture-diagrams-a6b6138ed875>.

[24] LIU, S.; QI, L.; QIN, H.; SHI, J.; JIA, J. Path Aggregation Network for Instance Segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* [online]. IEEE, 2018, 2018, 8759-8768 [cit. 2022-10-29]. ISBN 978-1-5386-6420-9. Dostupné z URL: <https://doi.org/10.1109/CVPR.2018.00913>

[25] MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5, 115–133 (1943).* Dostupné z URL: <https://doi.org/10.1007/BF02478259>.

[26] NIELSEN, M. A. Neural Networks and Deep Learning [online]. *Determination Press*, 2015 [cit. 2022-10-16]. Dostupné z URL: <http://neuralnetworksanddeeplearning.com>.

[27] Optical Flow. OpenCV: Open Source Computer Vision [online]. [cit. 2023-03-13]. Dostupné z URL: <https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html>.

[28] O'SHEA, K.; NASH, R. An Introduction to Convolutional Neural Networks [online]. arXiv preprint arXiv:1511.08458v2, 2015, 10 [cit. 2022-10-27]. Dostupné z URL: <https://doi.org/10.48550/arXiv.1511.08458>

[29] Perceptron. In: OpenCV [online]. 2010 [cit. 2022-10-08]. Dostupné z URL: <http://opencv.jp/opencv-2.1_org/cpp/neural_networks.html>.

[30] RAJPUT, V. Pooling diagram. In: Medium: AIGuys [online]. 10.1.2022 [cit. 2022-10-27]. Dostupné z URL: <https://medium.com/aiguys/pooling-layers-in-neural-nets-and-their-variants-f6129fc4628b>.

[31] REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* [online]. IEEE, 2016, 779-788 [cit. 2022-10-27]. ISBN 978-1-4673-8851-1. Dostupné z URL: <https://doi.org/10.1109/CVPR.2016.91>

[32] SANDERSON, G. Neural Networks: Gradient descent, how neural networks learn. *3Blue1Brown* [online]. 2017, 5 [cit. 2022-10-16]. Dostupné z URL: <https://www.3blue1brown.com/lessons/gradient-descent>

[33] SANDERSON, G. Neural Networks: What is backpropagation really doing?. *3Blue1Brown* [online]. 2017, 5 [cit. 2022-10-17]. Dostupné z URL: <https://www.3blue1brown.com/lessons/backpropagation>

[34] SANDERSON, G. Neural Networks: Backpropagation calculus. *3Blue1Brown* [online]. 2017, 5 [cit. 2022-10-21]. Dostupné z URL: <https://www.3blue1brown.com/lessons/backpropagation-calculus>

[35] SIDDHARTH, S.; SHARMA, S.; ANIDHYA A. Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology* [online]. 2020, 10.5.2020, 2020(12), 310-316 [cit. 2022-10-08]. Dostupné z URL: <https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf>.

[36] Smoke & Fire Detection. Scylla AI [online]. [cit. 2023-04-29]. Dostupné z: <https://www.scylla.ai/smoke-fire-detection>.

[37] TAN, H. H.; LIM, K. H. Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization. *International Conference on Smart Computing & Communications (ICSCC)* [online]. IEEE, 2019, 2019, 7, 1-4 [cit. 2022-10-09]. ISBN 978-1-7281-1557-3. Dostupné z URL: <https://doi.org/10.1109/ICSCC.2019.8843652>.

[38] Video smoke detection: Fire Sensors Detection - VISD. ORR Protection [online]. [cit. 2023-04-29]. Dostupné z: <https://www.orrprotection.com/detection/video-smoke-detection>.

[39] WANG, X.; QIN, Y.; WANG, Y.; XIANG, S.; CHEN, H. ReLTanh: An activation function with vanishing gradient resistance for SAE-based DNNs and its application to rotating machinery fault diagnosis. *Neurocomputing* [online]. 2019, 363, 88-98 [cit. 2022-10-09]. ISSN 09252312. Dostupné z URL: <https://doi.org/10.1016/j.neucom.2019.07.017>.

[40] WANG, C. Y.; BOCHKOVSKIY, A.; LIAO, H. Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors [online]. arXiv preprint arXiv:2207.02696. 2022 [cit. 2022-10-28] Dostupné z URL: <https://doi.org/10.48550/arXiv.2207.02696>

[41] ZHENG, Z.; WANG, P.; LIU, W.; LI, J.; YE, R.; REN, D. Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression. *Proceedings of the AAAI Conference on Artificial Intelligence* [online]. 2020, 34(07), 12993-13000 [cit. 2022-11-02]. ISSN 2374-3468. Dostupné z URL: <https://doi.org/10.1609/aaai.v34i07.6999>

Seznam symbolů a zkratek

GD	Gradient Descent
ReLU	Rectified Linear Unit
mAP	mean Average Precision
CNN	Convolutional Neural Network
YOLO	You Only Look Once
DPM	Deformable Parts Model
R-CNN	Region-based Convolutional Neural Network
IoU	Intersection over Union
FPN	Feature Pyramid Network
RFB	Receptive Field Block
PAN	Path Aggregation Network
SSD	Single Shot Detector
SPP	Spatial Pyramid Pooling
E-ELAN	Extended Efficient Layer Aggregation Network
FPS	Frames Per Second
CUDA	Compute Unified Device Architecture
FP	False positive
TP	True positive
FN	False negative
NMS	Non-maximal Suppression

A Obsah elektronické přílohy

GitHub repositář:

<https://github.com/PrimalMight/Yolo7-Updated>

Elektronická příloha:

<https://drive.google.com/drive/folders/10fE3ess1fwAso3T3bpbyl2NOF1YlS3gR?usp=share_link>

- `/` ..... kořenový adresář elektronické přílohy
  - `YOLOv7_Training/` ..... Výsledky trénování sítě YOLOv7
    - `vysledky_trenovani_metoda_1/`
    - `vysledky_trenovani_metoda_2/`
  - `Datasets/` ..... výsledky pro metodu 1
    - `Optical_flow_dataset/` ..... použitý pro klasifikátory optického toku
    - `ROC-dataset/` ..... použitý pro měření ROC
    - `Classificators_dataset/` .... použitý pro trénování přídavných klasifikátorů
    - `benchmark-test-dataset/` ..... použitý pro porovnání s komerčními modely
    - `optical_flow_resource_dataset/` ... zdroj videí pro vytvoření datasetu opt. toku
    - `D-Fire (YOLOv7 Dataset)/` ..... použitý pro trénování sítě YOLOv7
  - `yolov7_customized/` ..... zdrojové kódy
    - `cfg/`
    - `Classifiers/` ..... implementace klasifikátorů + soubor pro načítání dat
    - `data/`
    - `detect.py` ..... soubor určený pro spouštění detekce
    - `Dockerfile`
    - `export.py`
    - `hubconf.py`
    - `models`
    - `my_utils` ..... pomocné zdrojové kódy, např. pro zobrazení optického toku
    - `output/` ..... místo pro aktuální snímky z vyrovnávací paměti
    - `README.md`
    - `requirements_aarch64.txt` ..... výpis knihoven potřebných pro python na architektuře ARM
    - `requirements_amd64.txt` ..... výpis knihoven potřebných pro python na architektuře x86_64
    - `runs`
    - `some_testing_videos/`
    - `test.npy`
    - `test.py`
    - `traced_model.pt`
    - `train_aux.py`
    - `train.py`
    - `utils/`
    - `vgg16/` ..... výsledky a váhy natrénovaných přídavných klasifikátorů
    - `yolov7_fire_smoke_weights.pt` ..... nejlepší váhy pro model YOLOv7