Prohlášení

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

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

V Plzni dne 10. srpna 2022

David Žahour

Poděkování

Chtěl bych tímto poděkovat svému vedoucímu bakalářské práce panu Ing. Miroslavu Jiříkovi za odborné vedení, konzultace a rady, které mi předal pro vypracování této práce. Dále mu děkuji za poskytnuté zdroje a nástroje v podobě anotovaných datasetů. A každému kdo mě podpořil při tvorbě této práce

Abstrakt

Práce se zabývá návrhem aplikace pro automatickou detekci špičky jehly ve videu chirurgického šití a jejího sledování. Aplikace je založena na metodách zpracování obrazu a byla vyvinuta v programovacím jazyce Python s frameworkem TensorFlow. Systém funguje na principu neuronové sítě, konkrétně je využita architektura ResNet34. Při tvorbě aplikace byl také naprogramován pomocný software pro práci s datasety ve formátu CVAT for images. V rámci práce zároveň proběhlo srovnání s již existujícími metodami pro sledování objektů. Datasety byly vytvořeny na Západočeské univerzitě v Plzni a jsou složeny z videí chirurgického šití, které bylo natočeno na mobilní telefon.

Klíčová slova

Neuronová síť, Python, Metody zpracování obrazu, Chirurgické šití, ResNet, Matice, CVAT, Dataset

Abstract

This thesis deals with the design of an application for automatic needle tip detection in surgical suture video and its tracking. The application is based on image processing methods. The application has been developed in Python programming language and TensorFlow framework. During the development of the application, an auxiliary software was also programmed to work with datasets in CVAT for images format. The work also included a comparison with existing methods for object tracking.

Keywords

Neural network, Python, Image processing methods, Surgical suturing, ResNet, Matrix, CVAT, Dataset

Obsah

Seznam obrázků

1 Úvod

Cílem této práce je vytvoření aplikace, která bude automaticky sledovat pohyb jehly ve videu chirurgického šití a to za využití metod počítačového vidění.Potřeba sledování špičky jehly vzniká kvůli teorii že množství pohybu co chirurg při práci vykoná odpovídá kvalitě zašití rány Práce vznikla na katedře kybernetiky Fakulty aplikovaných věd Západočeské univerzity v Plzni (FAV ZČU) ve spolupráci s Biomedicínským centrem Lékařské fakulty UK v Plzni. Za účelem vytvoření algoritmu byla poskytnuta videa chirurgického šití společně s anotacemi vytvořenými na FAV. V rámci práce jsem se primárně snažil o vytvoření přesného algoritmu, aby aplikace nalezla své využití v praxi.

Dále se práce zaměřuje na vytvoření aplikace, která automatizuje vyhodnocení pohybu jehly ve videu. Toto umožní kvantitativní analýzu a další výzkum na poli chirurgického šití. Pro srovnání: ruční prací může anotace jednoho pětiminutového videa zabrat až jednu hodinu, kdežto mnou navrhovaný program vyhodnotí video za zlomek času v závislosti na výkonnosti počítače a délce videa. Zde tedy můžeme pozorovat markantní úsporu času. Bakalářská práce je rozdělena do pěti kapitol a závěru.

V první části práce se věnuji základům chirurgického šití. Tato kapitola byla přidána z důvodu multioborovosti práce, jelikož zasahuje i do lékařství a čtenáři tyto informace nemusí být známy. Konkrétně rozebírám postup při zašívání rány a typy jehel na chirurgické šití.

Dále se věnuji základům sledováním objektů. Na začátku kapitoly definuji formát dat, následně provádím citace principů fungovaní dvou klasických metod, které jsem zvolil k porovnání se svojí metodou, a zároveň se snažím vysvětlit, jak fungují a důvod proč je využívám ke srovnání.

Třetí část pojednává o neuronových sítích, konkrétně o jejich historii a celkovému fungování. Vzhledem k tomu, že algoritmus spojený s touto prací je postaven na neuronových sítích, snažil jsem se vysvětlit potřebné náležitosti, konkrétně jak je síť organizovaná a jaké vrstvy se v rámci sítě vyskytují. Na závěr kapitoly vysvětluji ztrátové a aktivační funkce a problém mizejícího gradientu.

Dále popisuji implementaci klasických metod, jejich výsledky v rámci testování a jejich chyby a nedostatky pro aplikaci. Dále pojednávám o mnou navržené aplikaci, konkrétně o třech hlavních modulech: modul pro manipulaci s daty, modul pro trénování neuronové sítě a modul pro spuštění vyhodnocení videa. Na konci hodnotím výsledky provedené práce a experimentů.

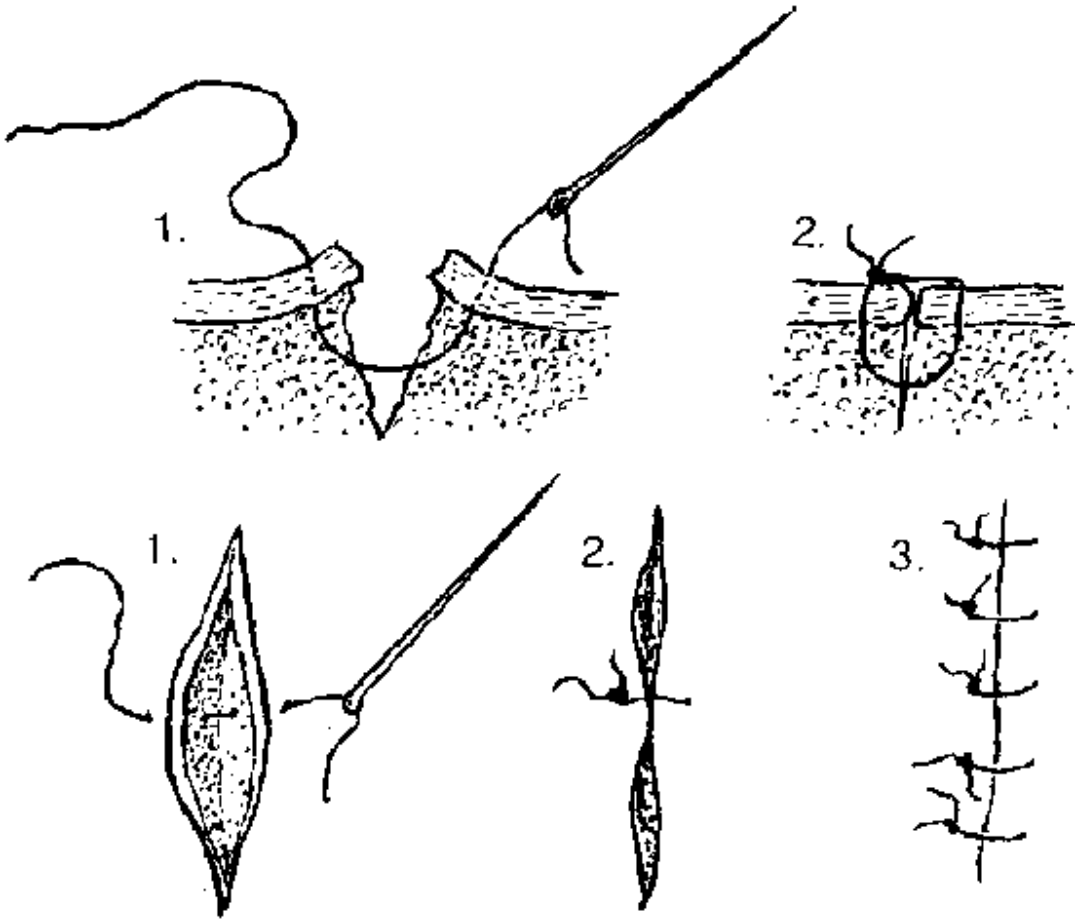
Závěrečné shrnutí práce je obsaženo v závěru.

2 Chirurgické šití

Odborně sutura, laicky steh či sešití, je termín pro opětovné spojování tkáně, které je způsobeno buď úrazem, nebo při chirurgickém zákroku. Důležitá je kvalita samotného provedení. Existuje mnoho typů stehů pro různé druhy ran např. pro spojování kůže lze zmínit

1. jednouzlové – jednoduché, matracové, intradermální, Algowerovy stehy
2. pokračující – většinou pro intradermální stehy.

Jednoduchý přerušovaný steh je technika šití používaná k uzavírání ran. Je to nejčastěji používaná technika při uzavírání kůže. Název přerušovaný steh je odvozen od vzhledu samotného stehu, jelikož jednotlivé stehy nejsou navzájem spojené, ale oddělené. Nevýhodou tohoto typu stehu je jeho samotné umístění a uvázání každého stehu zvlášť. Je tedy časově náročný, ale tato technika udrží ránu pohromadě i v případě, že jeden steh selže.



Obrázek 1: Náčrtek zašívání rány pomocí jednouzlového stehu
Zdroj: [1]

Chirurgický uzel (nebo uzly) kříží ránu kolmo, přičemž uzly by neměly zůstat nad ránou, ale měly by být umístěny na jedné straně rány (viz obr. (1) vpravo dole), aby se zabránilo jizvení a usnadnilo se odstranění stehů. To vše dokážeme pozorovat a vyhodnocovat prostřednictvím navrhovaného algoritmu.

Stehy jako takové, bez ohledu na způsobu šití, dělíme na dva druhy − vstřebatelné, které se v těle časem bez zásahu neškodně rozpadnou, a nevstřebatelné, které se musí ručně odstranit, pokud nejsou ponechány na neurčito. Typ použitého stehu se liší v závislosti na operaci, přičemž hlavním kritériem je náročnost místa a prostředí.

Dále můžeme stehy dělit dle jejich umístění. Stehy umístěné uvnitř těla jsou ty, které by vyžadovaly opětovné otevření rány, pokud by měly být odstraněny. Vnější stehy jsou ty, které leží na vnější straně těla a lze je odstranit během několika minut bez opětovného otevření rány. V důsledku toho se často používají vstřebatelné stehy uvnitř, nevstřebatelné zevně. Stehy, které mají být umístěny ve stresovém prostředí, například v srdci (stálý tlak a pohyb) nebo v močovém měchýři (nepříznivá přítomnost chemických látek) mohou vyžadovat specializované nebo pevnější materiály, aby mohly plnit svou úlohu. Obvykle jsou takové stehy buď speciálně upraveny, nebo vyrobeny ze speciálních materiálů a většinou jsou nevstřebatelné, aby se snížilo riziko předčasné degradace.

Dále je třeba zmínit materiály vláken. Zásadní dělení materiálů vláken je na nevstřebatelné a vstřebatelné. Jak již název napovídá, tak nevstřebatelná vlákna je potřeba po zhojení rány a časovém úseku vyndat.

Mezi hlavní zásady při zašívání kůže patří:

1. jemné zacházení s tkání,
2. dobrá adaptace okrajů rány,
3. uzávěr rány pod minimálním napětím,
4. místa vpichu stejně daleko od okrajů rány i od sebe,
5. sešívání okrajů rány rovnoměrně k sobě – podkoží na podkoží.

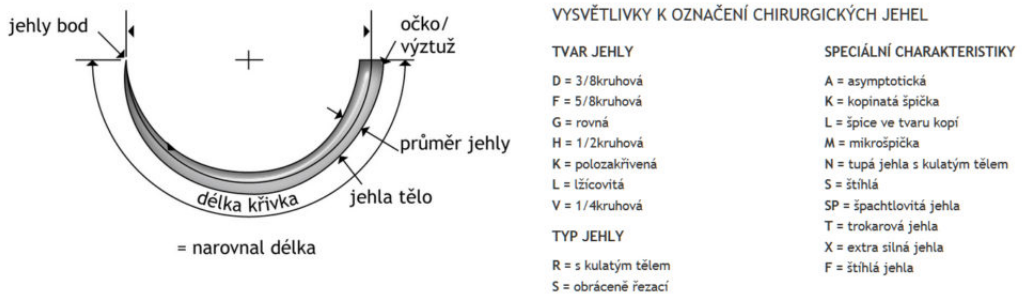
Pro tuto práci jsou zásadní body 4 a 5, protože teoreticky je možné tyto informace získat ze sledování špičky jehly ve videu.

2.1 Chirurgické jehly

Traumatické jehly jsou jehly s otvory nebo očky, které se do nemocnice dodávají odděleně od šicí nitě. Nit se musí navléknout na místě, stejně jako se to dělá při domácím šití. Atraumatické jehly se skládají z jehly bez očka, která je připevněna (nabatikována) k určité délce šicí nitě. Výrobce jehel vytvoří atraumatickou jehlu bez očka s nití již ve výrobním závodě.

Výhody atraumatických jehel jsou, že zdravotní personál nemusí ztrácet čas navlékáním nitě na jehlu. Další důležitou výhodou je to, že konec šicího vlákna na nabatikované jehle je menší než tělo jehly. To způsobuje, že při průniku jehly tkání se vpich

nezvětšuje, kdežto u traumatických jehel s očky vychází vlákno z otvoru jehly na obou stranách, tudíž při průchodu této jehly tkání se vpich zvětšuje oproti vpichu samotnou jehlou. Téměř všechny moderní šicí stroje mají nabatikované atraumatické jehly. Různé typy jehel můžeme vidět na obrázku 2.



Obrázek 2: Druhy a typy jehel
Zdroj: [2]

3 Sledování objektů

Sledování objektů je úloha v počítačovém vidění, kdy na základě počáteční informace sledujeme objekt, který se pohybuje v rámci videa. Sledování objektů je využitelné v širokém spektru aplikací. Cílem sledování objektu je lokalizace objektu v po sobě navazujících snímcích v rámci videa. Toto sledování může být obtížné například v momentě, kdy se objekt pohybuje příliš rychle. Dalším problémem může nastat, když 3D objekt mění orientaci v rámci 2D snímků, tudíž dochází ke změně tvaru objektu v rámci 2D zobrazení na snímku. Pro aplikaci sledování objektu algoritmus analyzuje sekvence snímků z videa a jako výstup předá pohyb objektu mezi jednotlivými snímky. Existuje celá řada algoritmů, každý má svoje benefity i nevýhody.

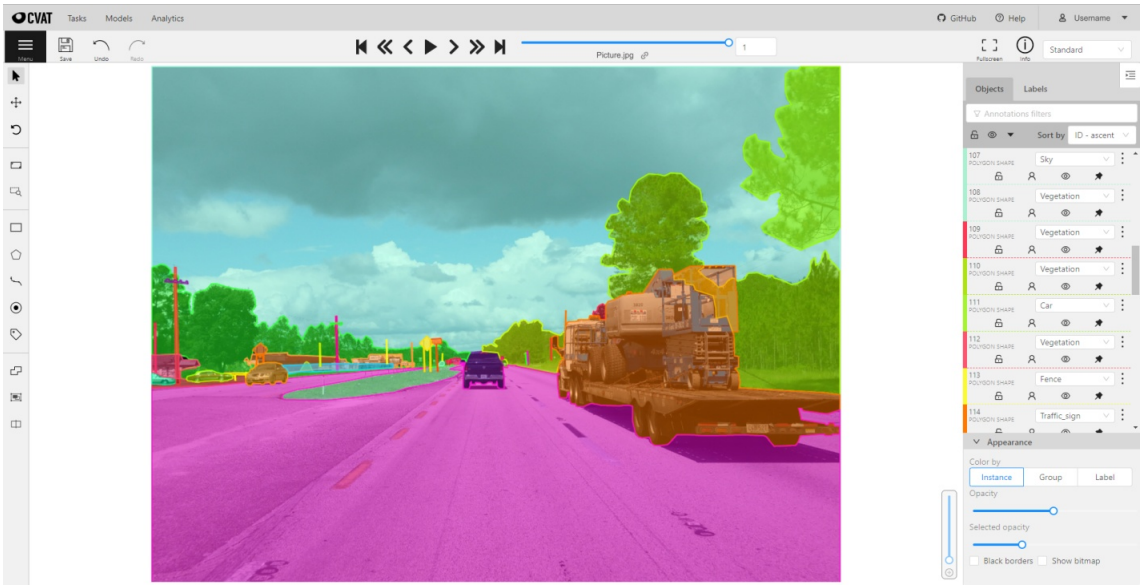
3.1 Vstupní data

Nejprve je třeba definovat, v jaké formě dostáváme jednotlivé snímky videa, ze kterých se snažíme získat data. Data jsou reprezentována přes trojdimenzionální matici, kde pro každý pixel máme hodnotu daného barevného spektra, které se skládá z červené (R), modré (B) a zelené (G) barvy. Pro jednoduchost budu ale následující metody a cíle popisovat na černobílých snímcích, tudíž i výsledné matice budou dvoudimenzionální. K implementaci jsem využil knihovnu OpenCV [4] která zajšitovala veškeré potřebné algoritmy a umožnovala zpracovávat video. Samotná knihovna je napsaná v programovacím jazyce C++ tudíž je i velice rychlá a efektivní.

Data byla anotována v nástroji Computer Vision Annotation Tool (CVAT) [17]. Následný formát datasetu má tuto podobu XML souboru. CVAT podporuje anotování pomocí polygonů, bodů a bounding boxů.

```
<image height="2160" width="3840" name="frame_000000" id="0">
<points z_order="0" points="1833,427"
source="manual"label="scissors">
</points>
</image>
```

Samotný CVAT je vhodný nástroj k anotaci, protože umožňuje přenášet nastavené masky mezi snímky. Dále umožňuje kooperaci více lidí na projektech. Díky tomu to byl vhodný nástroj pro vytvoření masek. Maska v programu může být definovaná jako polygon nebo čtverec dále je zde podpora i bodovéhé . Na obrázku (3) vidíme uživatelské rozhraní.



Obrázek 3: Ukázka softwaru CVAT
Zdroj: [17]

3.2 Metody tradičního počítačového vidění v OpenCv

3.2.1 Metoda Multiple instance learning (MIL)

Tuto metodu využívám ke stanovení baseline. Metodu ve svém článku Boris Babenko popisuje následovně: *„V tomto článku se zabýváme problémem učení se adaptivního modelu vzhledu pro sledování objektů. Konkrétně se ukázalo, že třída sledovacích technik nazývaná ‚sledování pomocí detekce' poskytuje slibné výsledky v reálném čase. Tyto metody trénují diskriminační klasifikátor způsobem, aby oddělil objekt od pozadí. Tento klasifikátor se sám zavádí pomocí aktuálního stavu sledovače objeku k získání pozitivních a negativních příkladů z aktuálního snímku. Mírné nepřesnosti ve sledovacím algoritmu proto mohou vést k nesprávně označeným trénovacím příkladům, což zhoršuje klasifikaci a může způsobit další drift. V tomto článku ukazujeme, že použití učení s více instancemi (Multiple Instance Learning, MIL) namísto tradičního učení s učitelem zabraňuje těmto problémům, a proto může vést k robustnějšímu učení sledovače s menším*

počtem parametrů. Představujeme nový algoritmus MIL pro sledování objektů, který dosahuje vynikajících výsledků s výkonem v reálném čase.“ Cit. [3]

Tento algoritus je už implementován v knihovně openCV[4].

Algorithm 1 MILTrack

Input: New video frame number k

1: Crop out a set of image patches, $X^s = \{x | s > ||l(x) - l^*_{t-1}||\}$ and compute feature vectors.
2: Use MIL classifier to estimate $p(y = 1|x)$ for $x \in X^s$.
3: Update tracker location $l^*_t = l\Big(\text{argmax}_{x \in X^s}\, p(y|x)\Big)$
4: Crop out two sets of image patches $X^r = \{x | r > ||l(x) - l^*_t||\}$ and $X^{r,\beta} = \{x | \beta > ||l(x) - l^*_t|| > r\}$.
5: Update MIL appearance model with one positive bag X^r and $|X^{r,\beta}|$ negative bags, each containing a single image patch from the set $X^{r,\beta}$

Obrázek 4: MIL sledovací algoritmus
Zdroj: [3]

3.2.2 Metoda Kernelized Correlation Filters tracker (KCF)

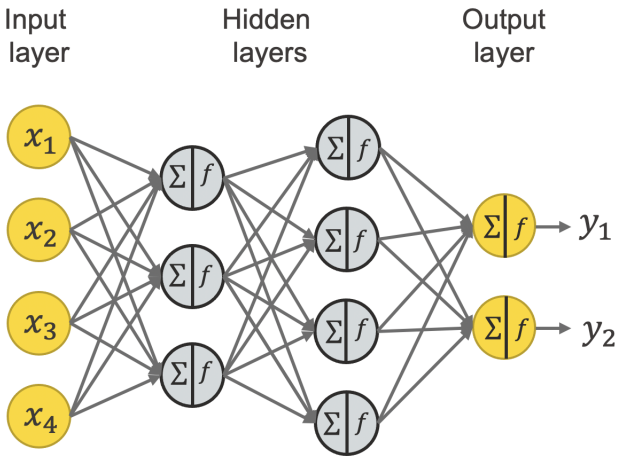
Další z metod, které využívám, je Kernelized Correlation Filters tracker neboli KCF. O této metodě João F. Henriques ve svém paperu napsal: *„Základní součástí většiny moderních sledovačů je diskriminační klasifikátor, jehož úkolem je rozlišovat mezi cílovým objektem, cílem a okolním prostředím. Aby se tento klasifikátor vyrovnal s přirozenými změnami obrazu, je obvykle trénován s přeloženými a zmenšenými obrázky. vzorkovými políčky. Takové sady vzorků jsou plné redundancí – všechny překrývající se pixely musí být stejné. Na základě tohoto jednoduchého pozorování navrhujeme analytický model pro datové sady tisíců přeložených políček. Tím, že ukážeme, že výsledný je matice dat cirkulární, můžeme ji diagonalizovat pomocí diskrétní Fourierovy transformace, čímž se sníží objem paměti i výpočet o několik procent. o několik řádů. Zajímavé je, že pro lineární regresi je naše formulace ekvivalentní korelačnímu filtru, který používají někteří z nejrychlejších konkurenčních sledovačů. Pro jádrovou regresi však odvozujeme nový korelační filtr (Kernelized Correlation Filter, KCF), který na rozdíl od jiných jádrových algoritmů má přesně stejnou složitost jako jeho lineární protějšek. V návaznosti na něj navrhujeme také rychlé vícekanálové rozšíření lineárních korelačních filtrů prostřednictvím lineárního jádra, které nazýváme duální korelační filtr (DCF). Jak KCF, tak DCF překonávají špičkový jako je Struck [20] nebo TLD [21], a to i přesto, že běží rychlostí stovek snímků za sekundu a jsou implementovány na 50 videích. v několika řádcích kódu (algoritmus 1). Abychom podpořili další vývoj, byl náš sledovací rámec zpřístupněn jako open-source.“* Cit. [5]

Jedná se o další z trackerů, které byly k dispozici v knihovně OpenCV.

4 Neuronové sítě

Umělé neuronové sítě , obvykle známé pouze jako neuronové sítě (NN), jsou výpočetní systémy, které jsou inspirovány biologickými neuronovými sítěmi, jež představují mozky zvířat.

NN jsou založeny na souboru propojených jednotek nebo uzlů, nazývaných umělé vegetativní buňky, které volně modelují neurony v biologickém mozku. Každá afilace, stejně jako synapse v biologickém mozku, může přenášet symbol na alternativní neurony. Syntetický neuron přijme signál, pak jej zpracuje a může signalizovat neuronům, které jsou k němu připojeny. V této bakalářské práci je „síla spojení“ definována jako reálné číslo. Výstup každého neuronu se vypočítá specifikovanou nelineární operací součtu jeho vstupů. Spojení se nazývají hrany. Hrany mají obvykle váhu, která se v průběhu učení upravuje. Váha zvýší nebo sníží sílu signálu na spojení. Neurony, které mají prahovou hodnotu, jsou aktivovány pouze v momentě, kdy dojde k překročení nastavené hodnoty. Obvykle jsou neurony hromadně rozděleny do vrstev. Zcela odlišné vrstvy mohou na svých vstupech provádět různé transformace. Signály putují z primární vrstvy (vstupní vrstva), do poslední vrstvy (výstupní vrstva). Signál musí projít všemi vrstvami. Toto je vyobrazeno na obrázku (5).



Obrázek 5: Struktura neuronové sítě
Zdroj: [6]

4.1 Historie neuronových sítí

V roce 1957 Frank Rosenblatt, výzkumný pracovník Cornellovy fyzikální laboratoře, integroval Hebbův model aktivity neuronů se Samuelovými myšlenkami strojového učení a vytvořil perceptron Mark 1. Rosenblatt využil neurobiologické experimenty, které byly provedeny ve 40. letech 20. století, a vytvořil model fungování mozkových neuronů.

Perceptron Mark 1 nebyl původně plánován jako program, ale jako stroj. Původní

myšlenka byla, že tato metoda poběží na single purpose machine. Navzdory tomu byl jeho kód napsán pro IBM 704 (a byl určen pro rozpoznávání obrazu). Tato kombinace se naštěstí osvědčila a poskytla pevný důkaz, že algoritmy a kód lze přenášet a efektivně používat v různých podobných počítačích. Předtím nebylo možné software přenášet mezi podobnými počítači. Je třeba poznamenat, že Rosenblattovým hlavním cílem nebylo vytvořit počítač, který by rozpoznával a klasifikoval obrázky, ale získat poznatky o fungování lidského mozku. Neuronová síť Perceptron byla původně naprogramována se dvěma vrstvami: vstupní a výstupní.

4.2 Učení neuronové sítě

Neuronové sítě se učí (nebo jsou trénovány) prostřednictvím zpracování příkladů, z nichž každý obsahuje známý „vstup“ a „výsledek“. Vytváří se mezi nimi vážené vazby, které jsou uloženy v nastavení vah samotné sítě. Učení neuronové sítě se obvykle provádí prostřednictvím zjištění rozdílu mezi zpracovaným výstupem sítě (často předpovědí) a cílovým výstupem. Tento rozdíl je chyba inforamce o chybách a způsobech jak je vyhodnocovat naleznete popsané níže v práci. Síť pak upraví své váhy dle parametru učení a hodnoty chyby. Postupnými úpravami se neuronové sítě snaží poskytovat výstup, který se stále více podobá požadovanému výstupu, který byl zadán učitelem. Po dostatečném počtu těchto modifikací může být učení ukončeno především na základě určitých kritérií. Tomuto postupu se říká učení s učitelem.

Z matematického hlediska je neuronová sít trénována pomocí metody zpětného šíření chyby [22]. Síť při učení vypočítává gradient ztrátové funkce vzhledem k vahám sítě pro jeden či více vstupů. Díky pravidlům pro práci s derivacemi je možné používat gradientní metody pro trénování vícevrstvých sítí, které aktualizují váhy tak, aby minimalizovaly ztráty; běžně se používá gradientní sestup nebo jeho varianty, jako je stochastický gradientní sestup. Algoritmus zpětného šíření funguje tak, že se vypočítá gradient ztrátové funkce vzhledem ke každé váze pomocí řetězového pravidla, přičemž se gradient počítá po jednotlivých vrstvách a iteruje se zpět od poslední vrstvy, aby se zabránilo nadbytečným výpočtům již vypočtených členů. Toto ovšem přináší své nevýhody. Problémem je, že v některých případech bude gradient mizivě malý, což efektivně zabrání tomu, aby váha změnila svou hodnotu. V nejzazším případě to může zcela zastavit další trénování neuronové sítě.

Takové sítě se „učí“ plnit úkoly prostřednictvím přemýšlení o příkladech, obvykle bez toho, aby byly naprogramovány s přesnými pravidly pro daný úkol. Například při rozpoznávání fotografií mohou objevit způsoby, jak vybrat obrázky, které obsahují kočky, pomocí čtení příkladů obrázků, které byly ručně kategorizovány jako „kočka“ nebo „bez kočky“, a pomocí důsledků vybrat kočky na jiných obrázcích. Síť se to učí bez předchozí znalosti koček, například že mají srst, ocas, vousy a obličeje podobné kočkám. Místo toho automaticky generují identifikační znaky z příkladů, které zpracovávají.

Architektura neuronové sítě je taková, že buňky jsou obvykle organizovány do více vrstev, zejména v tzv. deeplearningu. Neurony 1. vrstvy se připojují většinou k neuronům bezprostředně předcházející a bezprostředně následující vrstvy. V případě

neuronové sítě s architekturou resnet tomu ale tak není a to z důvodu odstranění problému mizejícího gradientu. Vrstva, která přijímá vnější poznatky, je vrstva vstupní. Vrstva, která vytváří konečný výsledek, je vrstva výstupní. Mezi nimi se nachází nula nebo mnoho skrytých vrstev. Používají se také jednovrstvé sítě. Mezi dvěma vrstvami je možné použít více typů propojení. Vrstvy mohou být „plně propojené“, kdy se každý neuron v jedné vrstvě připojí ke každému neuronu v další vrstvě, nebo sdružené, což jsou vrstvy, kdy se skupina somatických buněk v jedné vrstvě připojí k jednomu neuronu v další vrstvě, čímž se sníží množství neuronů v další vrstvě. Neurony s výhradně takovými spojeními vytvářejí směrovaný acyklický graf a označují se jako feedforward sítě. Alternativně se sítě, které umožňují spojení mezi neurony ve stejných nebo předchozích vrstvách, označují jako kontinuální sítě, což je případ sítě, která je použita v této práci.

4.3 Vrstvy

Neuronové sítě v dnešní době mají několik druhů vrstev, které jsou používané pro různé účely. Hlavní přínos specializovaných vrstev je ve zlepšení výsledků neuronové sítě a zároveň zjednodušení celé sítě díky specializaci dané vrstvy. To má hlavně dopad na paměťové nároky. Když vezmeme v potaz paměťové nároky neuronových sítí v momentě, kdy nechceme utratit velké množství peněz za výpočetní jednotky, jsme často omezeni samotnou velikostí paměti. U dnešních nejlepších specializovaných GPU na 24 GB (RTX3090) a v případě profesionálních počítačů se prodávají grafické karty s velikostí paměti až 80 GB, čemuž velmi zřetelně odpovídá i jejich cena.

4.3.1 Konvoluční vrstva

Konvoluční vrstva obsahuje sadu filtrů, pro které je třeba se naučit parametry. Rozměry filtru jsou menší než rozměry vstupních dat. Každý filtr je konvolutován se vstupními daty, aby se vypočítala aktivační mapa neuronu. To znamená, že filtr je převeden na celou šířku a výšku vstupu a v každé prostorové pozici je vypočten bodový součin mezi vstupem a filtrem. Výstupní objem konvoluční vrstvy se získá poskládáním aktivačních map všech filtrů podél hloubkové dimenze. Šířka a výška každého filtru je navržena tak, aby byla menší než vstup, takže každý neuron v aktivační mapě je připojen pouze k malé lokální oblasti vstupního objemu. Jinými slovy, velikost receptivního pole každého neuronu je malá, stejná jako velikost filtru. Lokální konektivita je motivována strukturou zrakové kůry zvířat s malými receptivními poli buněk. Lokální spojení v konvoluční vrstvě umožňují síti naučit se filtry, které nejlépe reagují na lokální oblast vstupu a využívají prostorovou lokální korelaci vstupu (u vstupního obrazu jsou pixely korelovány silněji s blízkými pixely než s těmi vzdálenými). Aktivační mapa se navíc získá provedením konvoluce mezi filtrem a vstupem, takže parametry filtru jsou sdílené ve všech lokálních místech. Sdílení vah snižuje počet parametrů pro expresivitu, efektivitu učení a správnou generalizaci.[7]

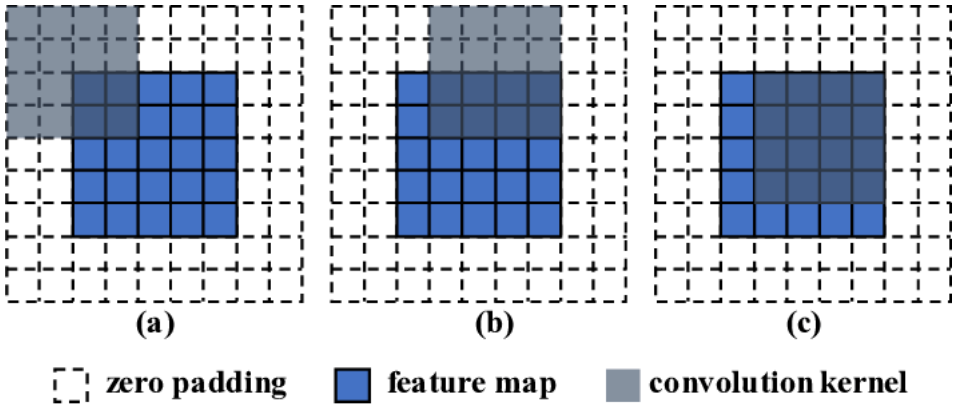
4.3.2 Zero padding

Když filtr konvoluje daný vstupní kanál, získáme výstupní kanál. Tento výstupní kanál je matice pixelů s hodnotami, které byly vypočteny během konvolucí, k nimž došlo na vstupním kanálu. Když toto nastane, dochází k redukci dimenze našeho obrázku. Zjednodušeně to znamená, že když budeme mít matici 4×4 a filtr 3×3, tak dostaneme výstup 2×2. Toto se dá zapsat tak, že když máme $n \cdot n$ rozměr a aplikujeme $f \cdot f$ filtr dostaneme rozměr výstupu.

$$(n - f + 1) \cdot (n - f + 1) \tag{1}$$

Problém nastává, když bychom chtěli konvoluci několikrát opakovat, jelikož by docházelo k opakovanému zmenšování matice, a tudíž i informace. Zároveň nelze aplikovat filtr na okrajích matice, tam bychom také ztráceli informace.

Zero padding, jak již samotný název napovídá, bere matice a vytváří na okrajích nuly (viz obr. 6), aby nedocházelo k takové redukci matice a zároveň to umožňuje získávat informace i z okrajů snímku.



Obrázek 6: Ilustrace zero padding
Zdroj: [10]

4.3.3 Fully connected

V plně propojených vrstvách je každý neuron v jedné vrstvě propojen s každým neuronem v následující vrstvě. Plně propojené vrstvy lze nalézt ve všech formách neuronových sítí, od základních neuronových sítí až po konvoluční neuronové sítě (CNN). S rostoucí velikostí vstupu se plně propojené vrstvy mohou stát výpočetně nákladnými, což vede ke kombinatorické explozi vektorových operací, které je třeba provést, a potenciálně špatné škálovatelnosti. V důsledku toho se často snažíme snížit počet těchto vrstev na minimum.

4.3.4 Batch Normalizace

Dávková normalizace (známá také jako dávková norma) je metoda, která se používá k urychlení a zvýšení stability NN pomocí normalizace vstupů vrstev jejich opě-

tovným centrováním a škálováním. Navrhli ji Sergey Ioffe a Christian Szegedy v roce 2015 [8].

Batch normalizace normalizuje hodnoty mezi mini-batchemi. Její cíl je aby aktivace měli průměr jedna a standardní odchylku 1. To způsobuje že se aktivace pohybují okolo nuly. Zatímco účinek dávkové normalizace je zřejmý, důvody její účinnosti zůstávají předmětem diskuse. Předpokládalo se, že může zmírnit problém vnitřního kovariantního posunu, kdy inicializace parametrů a změny v rozložení vstupů jednotlivých vrstev ovlivňují rychlost učení sítě [8]. V poslední době někteří vědci tvrdí, že dávková normalizace nesnižuje vnitřní kovariantní posun, ale spíše vyhlazuje goal funkci, což následně zlepšuje výkonnost. Při inicializaci však dávková normalizace ve skutečnosti vyvolává v hlubokých sítích silnou gradientní explozi, kterou zmírňují pouze přeskočené spoje ve zbytkových sítích. Jiní trvají na tom, že dávkovou normalizací se dosahuje oddělení a tím se neuronové sítě zrychlují [9]. V poslední době byla v sítích bez normalizace, tzv. sítích NF, zavedena technika ořezávání gradientu normalizace a inteligentního ladění hyperparametrů, která zmírňuje potřebu dávkové normalizace.

Každá vrstva neuronové sítě má vstupy s odpovídajícím rozdělením, které je během procesu trénování ovlivněno náhodností v inicializaci parametrů a náhodností ve vstupních datech. Vliv těchto zdrojů náhodnosti na rozdělení vstupů do vnitřních vrstev během trénování se popisuje jako vnitřní kovariantní posun. Ačkoli se zdá, že jednoznačná přesná definice chybí, jevem pozorovaným v experimentech je změna na středních hodnotách a rozptylech vstupů do vnitřních vrstev během tréninku.

Ke zmírnění vnitřního kovariantního posunu byla původně navržena dávková normalizace [1]. Během fáze trénování sítí se při změně parametrů předchozích vrstev odpovídajícím způsobem mění rozložení vstupů do aktuální vrstvy, takže aktuální vrstva se musí neustále přizpůsobovat novým rozložením. Tento problém je obzvláště závažný u hlubokých sítí, protože malé změny v mělčích skrytých vrstvách se při šíření v síti zesílí, což vede k výraznému posunu v hlubších skrytých vrstvách. Proto je navržena metoda dávkové normalizace, která má tyto nežádoucí posuny omezit, urychlit trénování a vytvořit spolehlivější modely.

Kromě snížení vnitřního kovariantního posunu se předpokládá, že dávková normalizace přináší mnoho dalších výhod. Díky této dodatečné operaci může síť používat vyšší rychlost učení, aniž by došlo k mizení nebo explozi gradientů. Kromě toho se zdá, že dávková normalizace má regularizační účinek, takže síť zlepšuje své generalizační vlastnosti, a není tedy nutné používat dropout ke zmírnění overfittingu. Bylo také pozorováno, že s dávkovou normalizací se síť stává odolnější vůči různým inicializačním schématům a rychlostem učení.

4.3.5 Max Pooling

Konvoluční vrstvy v konvoluční neuronové síti shrnují přítomnost prvků ve vstupním obraze. Problémem výstupních map rysů je jejich citlivost na umístění rysů na vstupu. Jedním z přístupů, jak tuto citlivost vyřešit, je snížit počet vzorků map příznaků. To má za následek, že výsledné mapy prvků se sníženým vzorkováním jsou odolnější vůči změnám polohy prvku v obraze, což se označuje odborným výrazem

„lokální translační invariance“.

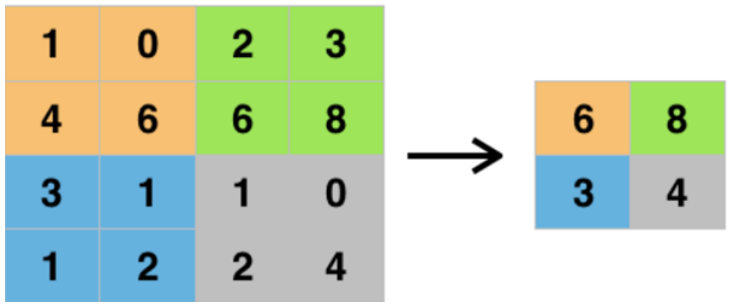
Vrstvy sdružování poskytují přístup ke snížení vzorkování map prvků shrnutím přítomnosti prvků v políčkách mapy prvků. Dvě běžné metody sdružování jsou průměrné sdružování a maximální sdružování, které shrnují průměrnou přítomnost prvku, resp. nejvíce aktivovanou přítomnost prvku.

Kromě toho, že se jedná o levnou náhradu konvoluční vrstvy, existuje ještě jeden důvod, proč může být Max Pooling velmi užitečný: translační invariance.

Pokud je model translačně invariantní, znamená to, že nezáleží na tom, kde se objekt na obrázku nachází; bude rozpoznán tak jako tak. Například pokud držím telefon u hlavy nebo u kapsy − v obou případech by měl být součástí klasifikace.

Jak si dokážete představit, dosažení translační invariance v modelu značně zvýší jeho predikční schopnost, protože již nemusíte poskytovat snímky, na kterých se objekt nachází přesně v nějaké požadované poloze. Spíše stačí poskytnout obrovskou sadu snímků, které obsahují daný objekt, a případně získat dobře fungující model.

Jak se tedy pomocí maximálního sdružování dosáhne translační invariance v neuronové síti? Řekněme, že máme jednopixelový objekt i když objekty jsou obvykle vícepixelové, ale demonstraci to prospěje. Tento objekt má nejvyšší kontrast, a proto generuje vysokou hodnotu pro pixel ve vstupním obrázku. Předpokládejme, že šestka na pozici (1, 2) v obrázku (7) v oranžové části obrázku je námi zvolený pixel. Při maximálním sdružování je stále zahrnut do výstupu, jak vidíme.



Obrázek 7: Ilustrace Max Pooling
Zdroj: [11]

Nyní si představte, že se tento objekt, a tedy i šestka, nenachází v bodě (1, 2), ale místo toho v bodě (1, 1). Zmizí z modelu? Ne. Naopak, výstupem vrstvy Max Pooling bude stále 6. Proto nezáleží na tom, kde se objekt v oranžovém bloku nachází, protože bude stejně „zachycen“.

To je důvod, proč Max Pooling znamená translační invarianci a proč je opravdu užitečný, kromě toho, že je i relativně levný.

Uvědomte si však, že pokud by se objekt nacházel v některé z neoranžových oblastí, byl by rozpoznán, ale pouze v případě, že tam není nic s větší hodnotou pixelu (což je případ všech prvků!). Proto Max Pooling nevytváří translační invarianci, pokud poskytujete pouze snímky, kde se objekt nachází stále ve velmi malé oblasti. Pokud je však vaše datová sada dostatečně pestrá a objekt se nachází v různých polohách, Max Pooling skutečně prospívá výkonu vašeho modelu.

4.4 Ztrátová funkce

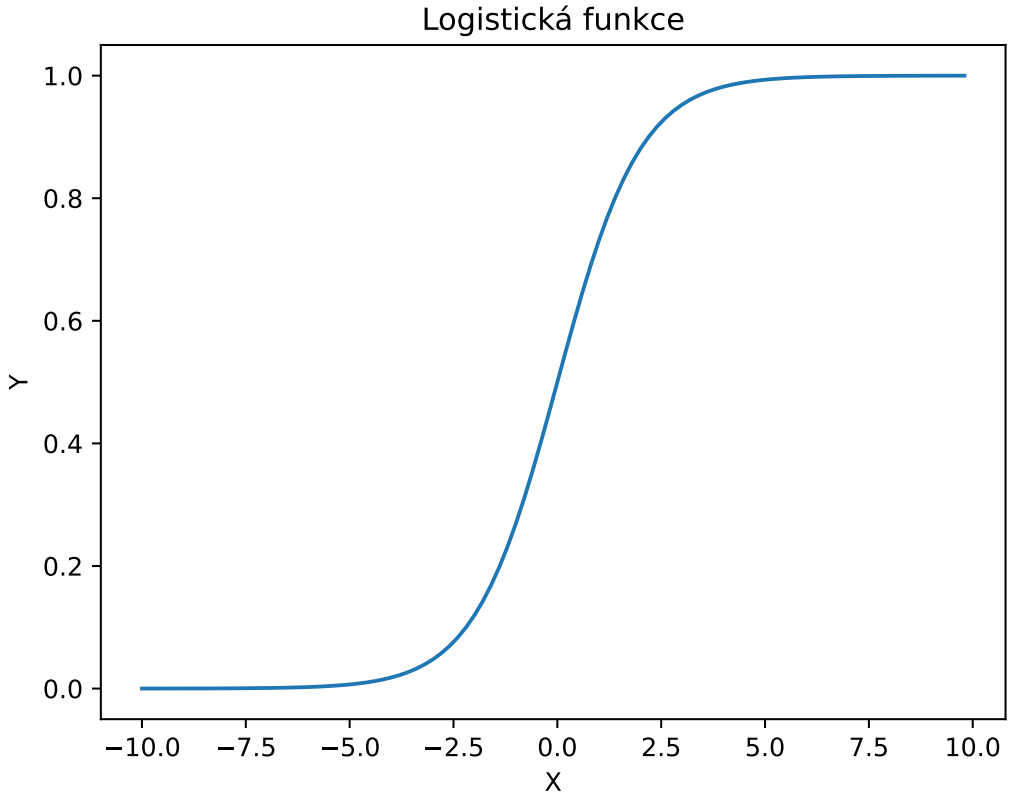
Obecně lze ztrátové funkce rozdělit do dvou hlavních kategorií v závislosti na typu učební úlohy, kterou řešíme − regresní ztráty a klasifikační ztráty. Při klasifikaci se snažíme předpovědět výstup z množiny konečných kategoriálních hodnot. Ztrátová funkce nám definuje, jak dobře se neuronová sít chová; trénování je v podstatě snaha o minimalizaci ztrátové funkce pro daný dataset. Zároveň lze ze ztrátovou funkci využít pro detekci přetrénování, protože v momentě kdy se bude přibližovat nule pouze na trénovacím datasetu, a nikoliv na testovacím, je potřeba změnit nastavení či ukončit proces učení, jelikož dochází k přetrénování.

4.5 Aktivační funkce

V umělých neuronových sítích tvoří každý neuron vážený součet svých vstupů a výslednou skalární hodnotu předává funkci označované jako aktivační funkce nebo přenosová funkce. Má-li neuron n vstupů. tak tato funkce g se označuje jako aktivační funkce. Pokud je funkce g brána jako lineární funkce, pak neuron provádí lineární regresi nebo klasifikaci. Obecně se g bere jako nelineární funkce, aby bylo možné provádět nelineární regresi a řešit klasifikační problémy, které nejsou lineárně oddělitelné. Pokud se g považuje za sigmoidální funkci neboli funkci ve tvaru písmene „s“, která se mění od 0 do 1 nebo od -1 do 1, lze výstupní hodnotu neuronu interpretovat jako odpověď ANO/NE nebo binární rozhodnutí. Sytící aktivační funkce však může v hlubokých sítích způsobit problém mizejícího gradientu. Nahrazení nasycující sigmoidální aktivační funkce aktivačními funkcemi, jako je ReLU (Rectified Linear Unit), které mají vyšší hodnoty derivace, umožnilo poprvé trénovat hlubší sítě. Od té doby byly nalezeny nemonotónní a oscilující aktivační funkce, které výrazně překonávají ReLU [12], zejména oscilující aktivační funkce zlepšují gradientní tok, urychlují trénink a umožňují jednotlivým neuronům učit se funkci XOR, podobně jako některé lidské mozkové neurony.

Sigmoidní funkce přijímá na vstupu libovolnou reálnou hodnotu a na výstupu má hodnoty v rozsahu 0 až 1. Čím větší (kladnější) je vstup, tím blíže bude výstupní hodnota k 1, zatímco čím menší (zápornější) je vstup, tím blíže bude výstup k 0, jak je znázorněno níže na grafu (8).

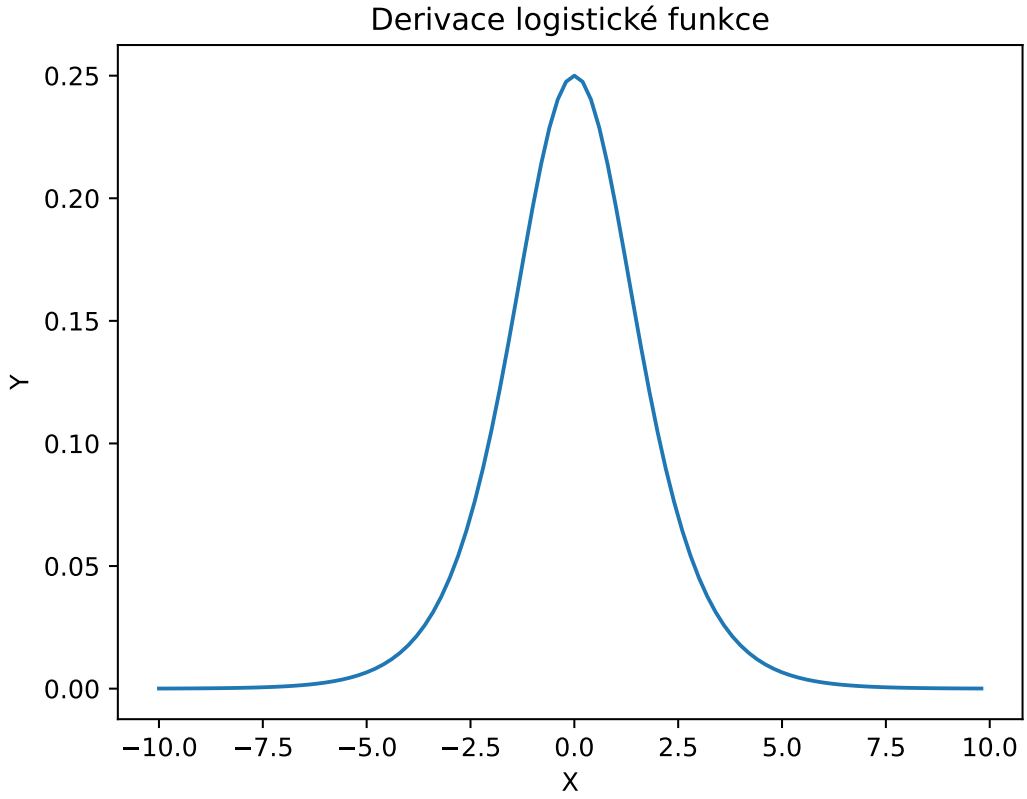
Sigmoidní neboli logistická aktivační funkce graf na grafu (8) je používaná funkce. U modelu, kde máme jako výstup předpovědět například s jakou pravděpodobností daný pixel obsahuje námi sledovaný nástroj. Jelikož interval pravděpodobnosti je $(0, 1)$ tak je sigmoidní funkce optimální. Funkce je diferencovatelná (graf na obrázku (9)) a poskytuje hladký gradient, tj. zabraňuje skokům ve výstupních hodnotách. To je znázorněno esovitým tvarem sigmoidní aktivační funkce.



Obrázek 8: Sigmoidní funkce

Bohužel, metoda má i své nevýhody, což si níže ukážeme. Nejprve je ale potřeba zavést derivaci.

$$\dot{f}(x) = sigmoid(x) \cdot (1 - sigmoid(x)) \tag{2}$$



Obrázek 9: Derivace sigmoidní funkce

Jak je vidět z uvedeného obrázku, hodnoty gradientu jsou významné pouze pro rozsah -7.5 až 7.5, jelikož v ostatních oblastech je graf mnohem plošší. Z toho vyplývá, že pro hodnoty větší než 7.5 nebo menší než -7.5 bude mít funkce velmi malé gradienty. Jakmile se hodnota gradientu blíží nule, síť se přestává učit a trpí problémem mizejícího gradientu [13].

5 Implementace metody MIL a KCF

V této části se snažím navrhnout aplikaci abych mohl svojí metodu porovnat s již existujícími algoritmy tudíž cílem je vyhodnotit video ke kterému máme anotace a následně provést vyhodnocení chyby . Experiment je navržen následovně: na začátku je definováno rozpětí okolí, které zadáváme jako informaci algoritmu. V případě reálného světa by bylo potřeba vybrat oblast ručně. Pokud chce člověk provést ruční zadání, stačí odkomentovat část kódu. Poté se měří euklidovská vzdálenost mezi body, kdy jeden bod je nalezen algoritmem a druhý dán od učitele. Důležitou součástí bylo testování při různých velikostech okolí.

Experiment byl vytvořen v jazyce Python v již zmiňované knihovně OpenCv [4]. Na začátku je načten manipulátor s datasetem, poté jsou dva vnořené cykly. Je důležité

zmínit, že v momentě, kdy se bod nenachází na snímku, není tento snímek nahrán do trackeru. Toto by ovšem v realitě nebylo možné. Domnívám se tedy, že pro tuto aplikaci to nezpůsobuje zásadní zkreslení výsledků. Při iteraci je možné sledovat video, případně si pouze nechat exportovat detekované body.

Je taktéž nutné si uvědomit, že tracker vrací ohraničující okno a z tohoto okna je propočten střed, který uvažujeme jako pozici nástroje. Implementace metody MIL a KCF je prakticky identická. Důvodem je, že obě metody vycházejí ze stejné knihovny, tudíž bylo v kódu potřeba změnit pouze jeden parametr, jinak se metody neliší. To byl také jeden z důvodů použití těchto metod.

```
tracker = cv2.TrackerMIL_create()
```

```
tracker = cv2.TrackerKCF_create()
```

5.1 Návrh vlastní metody pro sledování špičky jehly

Při návrhu metody jsem se rozhodl využít neuronové sítě a na problém přistoupit jako na segmentaci. Segmentace obrazu je proces, kdy snímek chceme rozdělit na několik segmentů známých jako obrazové oblasti nebo obrazové objekty (sady pixelů). Cílem segmentace je obvykle vyhledání objektů a hranic ve snímcích. Přesněji řečeno segmentace obrazu je proces přiřazení značky každému pixelu v obraze tak, aby pixely se stejnou značkou sdílely určité vlastnosti. V našem případě pixely značím obsahuje/-neobsahuje špičku jehly.

Segmentaci jednotlivých snímků videa jsem zvolil, protože jsem očekával dobré výsledky a zároveň je segmentace poměrně běžný úkol, který provádějí neuronové sítě, a je na toto téma řada postupů a materiálů. Rozhodl jsem se, že výstup bude reprezentován jako maska daného snímku, kde budou uvedeny pravděpodobnosti přítomnosti špičky jehly. Pro další zpracování by bylo možné využít Kalmanův filtr, ovšem jeho implementace by byla nad rámec této práce. Hlavní překážkou při vytváření algoritmu bylo vytvoření správné struktury neuronové sítě, kde jsem se nakonec rozhodl využít již vytvořenou síť ResNet34 [14] a pouze ji upravit pro naše použití.

Aplikace je vytvořena z jazyce Python. Aplikace má několik hlavních komponent − komponentu pro práci s daty, komponentu zodpovědnou za trénování a vyhodnocování a v neposlední řadě komponentu pro koncového uživatele. Kód je psán jako kombinace objektového a funkcionálního programování. Jednotlivé komponenty mají dokumentaci dle Numpy standardu.

5.2 CVAT manipulátor

CVAT manipulátor je proprietární součástí aplikace, při jejímž vývoji byl kladen důraz na použitelnost i pro jiné aplikace, jelikož CVAT for images je přenositelný datový formát. Díky tomu jsem se při práci snažil postupovat tak, aby byl kód přehledný a snadno čitelný, jelikož samotný datový manipulátor je objekt. Manipulátor byl vytvořen tak, aby bylo možné přes něj iterovat, tudíž lze snadno postupovat skrze

snímky. Manipulátor nenačítá data najednou ale postupně. Načtení dat v jeden okamžik do operační paměti by mělo výhodu menší latence ale přináší sebou nesmyslné paměťové nároky. Pro příklad dataset který mám k dispozici by obsadil přibližně 176.2 GB operační paměti. Tudíž manipulátor namísto toho uchovává cestu k souborům v jednotlivých datasetech a k načtení dochází po zadání požadavku. Nejdůležitější funkce manipulátoru jsou init, addDataset a getMask. Init funkce vytváří inicializace. Objektu lze kromě parametrů cest k datasetu nastavit požadovanou velikost snímku při práci s nimi. Dále je možné zapnout automatické zamíchání. Toto je vhodné z důvodu posloupnosti videa a rizika možného přetrénování. Funkce add dataset umožňuje spojit více CVAT datasetů pro rozšíření datasetu při trénování. Dále je možno rozdělení datasetu na validační a trénovací část pomocí funkce split.

Stěžejní je funkce get mask. Informace od učitele jsou ve formě bodu. My ale potřebujeme mít informace od učitele jako binární masku. Ta se vytváří jako nulová matice o rozměrech snímku a následně se bod nahradí jednotkovou maticí o daném rozměru. Velikost jednotkové matice nám udává, jak je velké samotné okolí bodu. Osobně vidím tuto část práce jako jednu s největší možností budoucího přínosu a to z důvodu širokého spektra aplikací a dle mého výzkumu neexistující vhodnou alternativou.

5.3 Trénování sítě

Neuronová síť je postavena na sítí ResNet34, která slouží jako backbone. K vytvoření sítě jsem se rozhodl použít framework TensorFlow Keras [19] a to z důvodu osobní zkušenosti a pro mě přehledné dokumentace. Na začátku této části kódu definuji veškeré primární knihovny použité v kódu. Za zmínku stojí NumPy, což je knihovna pro numericko-aritmetické operace, dále knihovna Matplotlib, která slouží ke generování grafů. Na začátku se nachází několik důležitých proměnných, například počet trénovacích epoch, velikost cílového obrázku a batch size.

V další části kódu leží DataLoader, který vytváří dvojici snímku a masky. Následně je pak DataLoader odesílá do neuronové sítě k trénování. TensorFlow disponuje vlastními metodami jak nahrávat fotky z adresáře, což se mi velmi osvědčilo. A to zejména z důvodu snadného odstraňování chyb při trénování. Mezi nejčastěji vyskytující se problémy patřily nesprávné dimenze nebo špatný formát dat.

5.4 Architektura neuronové sítě

Můj původní plán byl vytvořit síť s vlastní architekturou bez předtrénovaných vah. Po několika pokusech, které byly neúspěšné a to převážně z důvodu nedostatku dat, jsem dospěl k výsledku, že tudy cesta nevede. Proto jsem se začal přemýšlet o jiných řešeních. Po zvážení možností jsem se rozhodl využít předtrénovanou síť ResNet, která po prvotním experimentu předvedla takřka perfektní výsledky po mírném ladění na straně loss funkce. Následně jsem začal dosahovat výsledků, které jsou prezentované v této práci.

Velice mi pomohla knihovna Segmentation models [14], která vytváří samotný model neuronové sítě v rámci aplikace. Vzhledem k tomu, že jedné epoše trvalo vytrénování

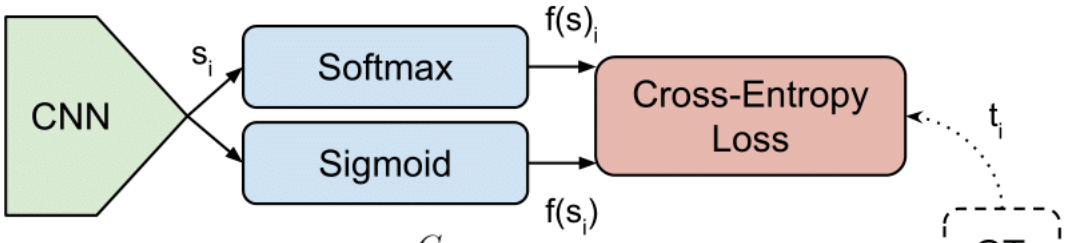
cca 45 minut na GTX 1060, vytvořil jsem automatické volání po skončení epochy. Při tomto volání bylo provedeno srovnání s přesností předchozí epochy a v momentě, kdy současná neuronová síť dosahovala lepších výsledků na validačním datasetu byla uložena. Na závěr kód uloží váhy neuronové sítě a provede vykreslení ztrátové funkce v čase.

5.4.1 Optimizér

Jako optimizér využívám tzv. Adam. Podle [15] je tato metoda *„výpočetně efektivní, má malé nároky na paměť, je invariantní vůči diagonálnímu přeškálování gradientů a je vhodná pro problémy, které jsou velké z hlediska dat/parametrů.“* Optimizér má následující parametry: $learning\_rate$ jehož hodnota je 0.001 a dále máme $beta\_1$, což je exponenciální míra rozpadu pro odhady 1. momentu. Exponenciální míra rozpadu pro odhady druhého momentu je nastavena na hodnotu 0.999.

5.4.2 Ztrátová funkce

V aplikaci sice provádíme binární dělení do tříd *jehla je přítomna/jehla není přítomna* ale přesto využívám nebinární ztrátovou funkci. Důvod využití Categorical cross-entropy je dán již zmíněnými experimenty, kdy jsem se snažil detekovat nejen špičku jehly, ale i „okraje jehly“. Od této snahy jsem následně upustil, protože mi neposkytovala žádné výhody pro řešení práce a ztrátová funkce zůstala nezměněná. Nákres ztrátové funkce můžeme vidět na snímku (10).



$$CE = -\sum_{i}^{C} t_i log(f(s)_i)$$

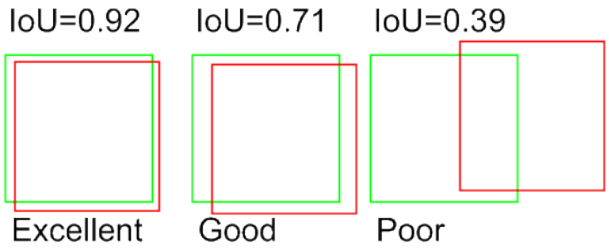
$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Obrázek 10: Ilustrace ztrátové funkce
Zdroj: [16]

5.4.3 Metrika přesnosti

Pro metriku přesnosti jsem se rozhodl využít IoU metriku, neboli Intersection over Union, která se vypočítá jako podíl průniku originální a vyhodnocené masky se

sjednocením originální a vyhodnocené masky (viz obrázek (11)). Pro jejich vyhodnocení je důležité zdůraznit, že vzhledem k relativně malé velikosti hledaného objektu se jedná řádově o několik pixelů. IoU matrika se pohybuje v malých číslech, ale je to způsobeno velkým dopadem každého nevyhodnoceného pixelu.



Obrázek 11: Ilustrace IoU metriky
Zdroj: [6]

5.4.4 Ukládání modelu

Z důvodu, že model není sekvenční, není možné využívat funkci TensorFlow pro ukládání celého modelu. Vzhledem k podstatě práce, to ale není nutné a postačilo nám ukládání vah. Tento přístup nám zároveň umožnil zmenšení potřebného místa na disku, jelikož váhy mají necelých 100 MB. Nevýhodou tohoto postupu je, že při použití neuronové sítě musím separátně přenášet strukturu sítě. Zároveň mám obavu, že vzhledem k tomu, že je architektura generována z knihovny Segmentation models, je zde možnost, že s novou verzí knihovny by aplikace mohla přestat fungovat.

5.5 Použití aplikace

Spuštění aplikace je vzhledem k podstatě práce poměrně komplikovaný proces. Již instalace je dosti složitá, i přesto jsem se snažil možnému uživateli spuštění aplikace co nejvíce ulehčit tím, že má k dispozici Python virtual environment, kde jsou veškeré potřebné knihovny. Případně je může instalovat ze souboru requirements.txt . Aplikace se spouští skrze příkazovou řádku, kde se jako argument uvádí video. V dalším kroku se otevře okno, kde se vykresluje výstup. Snímky se vyhodnocují v průběhu videa a zároveň je možné při specifikaci argumentu na dané místo uložit sekvence detekovaných bodů, kde se nacházela špička jehly. Samotné spuštění neuronové sítě není výpočetně náročné a nevyžaduje výkonný hardware. Při testování na procesoru I7 7700HQ bylo 500 snímků vyhodnoceno za 36 minut. Na grafické kartě ale dostáváme daleko lepší výsledky – přibližně 6 minut. Mezi nejsložitější části pro mě bylo vytvořit body ze získané pravděpodobnostní matice. Následně se z této masky snažím vytvořit body, k

tomu využívám funkci regionprops [18].

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3}$$

Optimální by samozřejmě bylo, aby matice vypadaly jako v případě A, kdy se v matici nachází pouze jeden ostrov. V momentě, kdy dostaneme matici B, tak máme 2 ostrovy, tudíž 2 body, které jsou reprezentovány jako centroidy daných pozic. Největší problém vzniká v případě matice C, kde je jeden ostrov, ale je velice pravděpodobné že se jedná v realitě o dva nezávislé body, které bychom následně mohli vyhodnotit.

Každá ze čtyř části vytváří celek. Při jejich vytváření jsem se snažil, aby byl kód přehledný a logický. Troufám si říci, že se mi to podařilo a aplikaci by mohl využívat další člověk, který by mohl pokračovat v práci na ní.

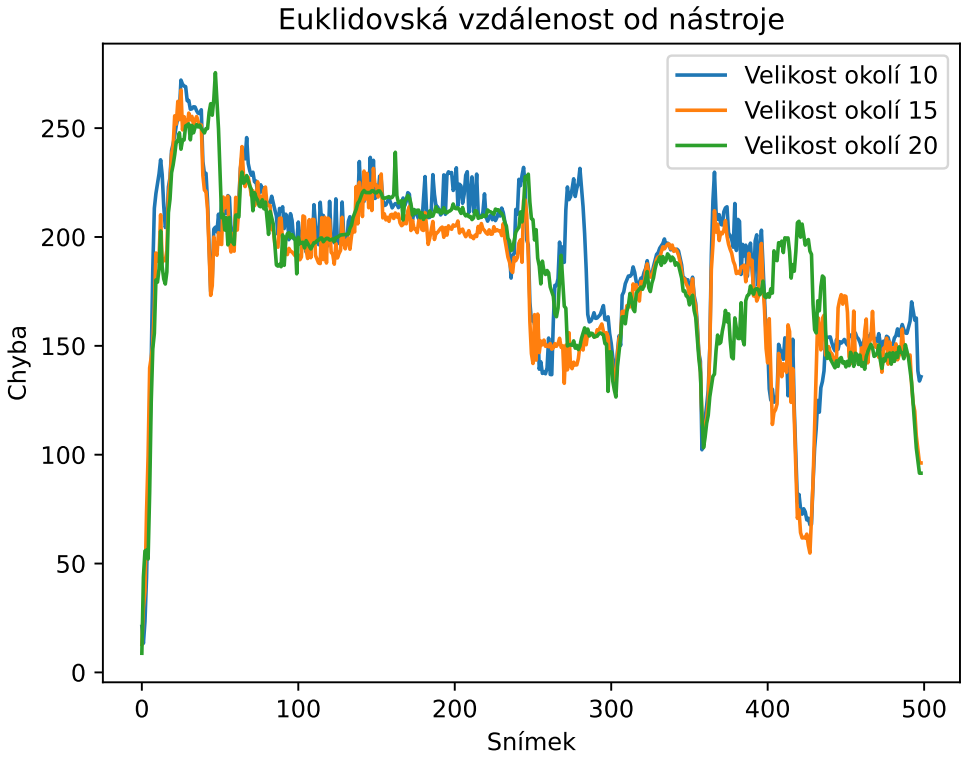
5.6 Nástroj pro vyhodnocení výsledků

Další částí je aplikace pro vyhodnocení výsledků. Na rozdíl od části pro použití aplikace, vyhodnocovací část očekává na vstupu CVAT dataset, následně se spustí klasické iterování skrze snímky, kde se propočítává vzdálenost mezi informací od učitele a vyhodnocením neuronové sítě. Zde je důležité zdůraznit, že vzdálenost bereme od bodu, který byl nejblíže vyhodnocen k učiteli. Tento postup jsem zvolil, protože chybně detekované body byly většinou stacionární, tudíž by jejich filtrace nebyla problematická a v momentě, kdy byla chybě vyhodnocena pozice, se v okolí nenacházel žádný jiný bod. Pro lepší představu se správný bod vykresluje červenou barvou a předpovídaný zelenou barvou.

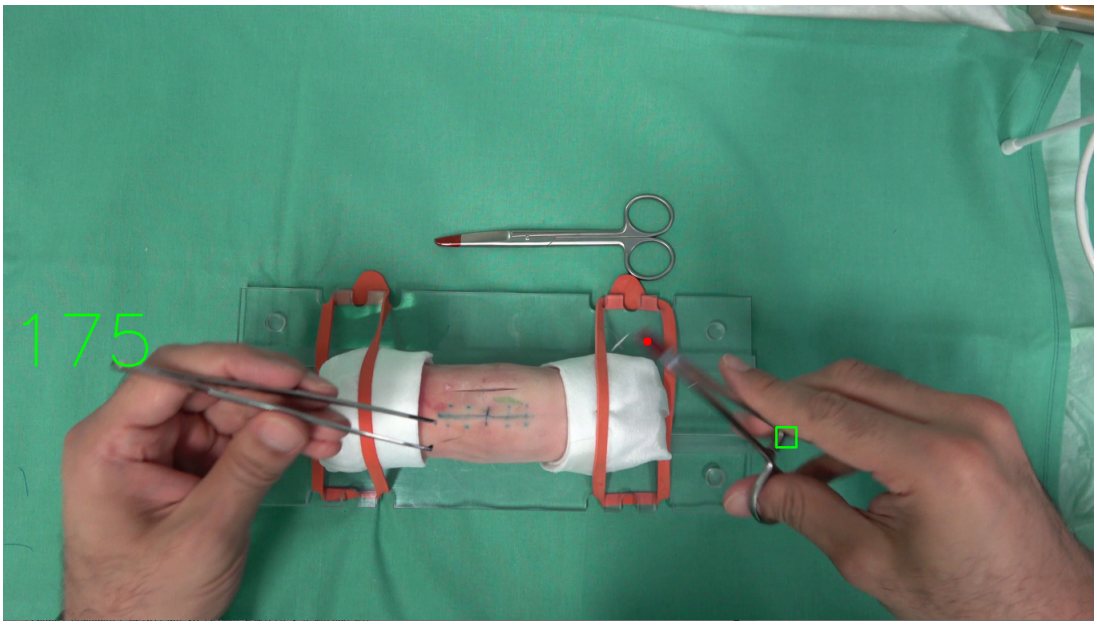
6 Experimenty a jejich vyhodnocení

6.1 Vyhodnocení výsledků metody MIL

Metoda MIL předvádí výsledky, které bych na některých videích označil za uchá-zející, zatímco na jiných se hned za začátku ztratí, viz graf na obrázku (12) a je tudíž prakticky nepoužitelná. Metoda trpí mnoha problémy: jeden z nich je možnost ztracení, kdy dojde k tomu, že algoritmus sleduje jiný objekt než byl původně definován. Toto lze vidět na obrázku (13). Dále byla poměrně častá situace, kdy algoritmus zachytil značku od fixy pro určení místa zašití. V ten moment bylo jen otázkou náhody, zda se chirurg posune na ono místo a algoritmus se znovu správně nastaví, přičemž nezáleží na velikosti sledovaného okna. Proto bych řekl, že tato metoda je pro naše účely nedo-statečná. Zároveň byla rychlost vyhodnocování přibližně 7 snímků za vteřinu, tudíž by nebylo možné využití v reálném čase.



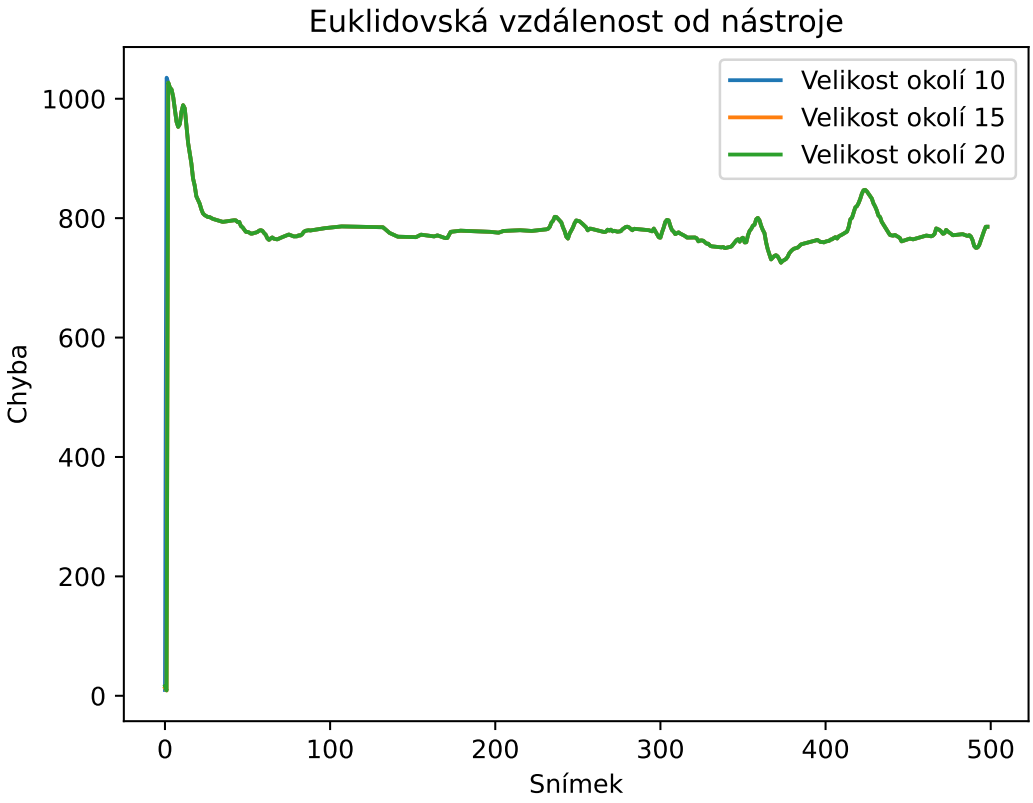
Obrázek 12: Velikost okolí je délka strany sledovaného čtverce v pixelech



Obrázek 13: Ukázka ztracené MIL metody

6.2 Vyhodnocení výsledků metody KCF

Tato metoda vždy po několika snímcích kompletně ztratí obraz a přestane poskytovat data, jak lze vidět z grafu na obrázku (14). Tento jev bych ale hodnotil kladně, vzhledem k tomu, že lze snáze detekovat moment, kdy je metoda ztracena. Ovšem i v momentech, kdy metoda fungovala správně, byla přesnost tristní a nedostatečná pro naší aplikaci. Za zmínku ovšem stojí její rychlost – za pouhou 1 vteřinu byla schopna vyhodnotit přibližně 30 snímků namísto 8 jako u metody MIL.



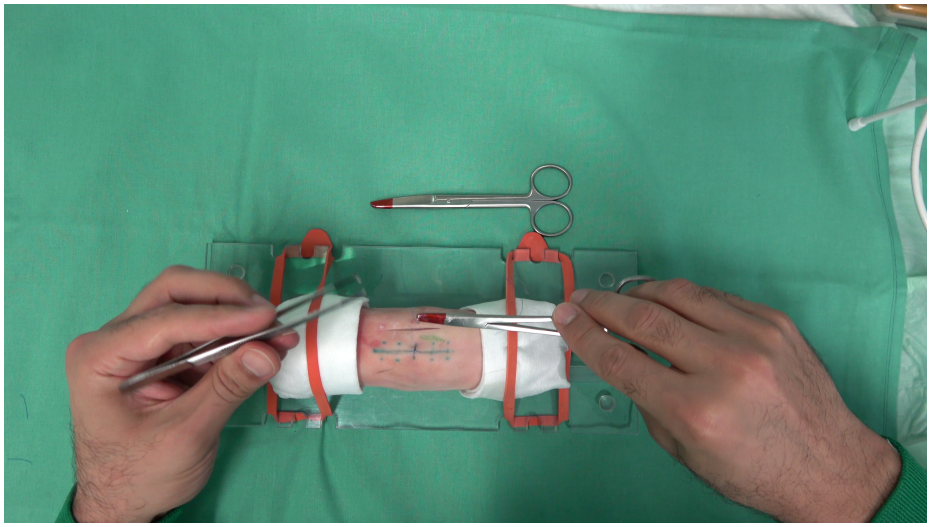
Obrázek 14: Průběh chyby v čase

6.3 Vyhodnocení výsledku vlastní metody

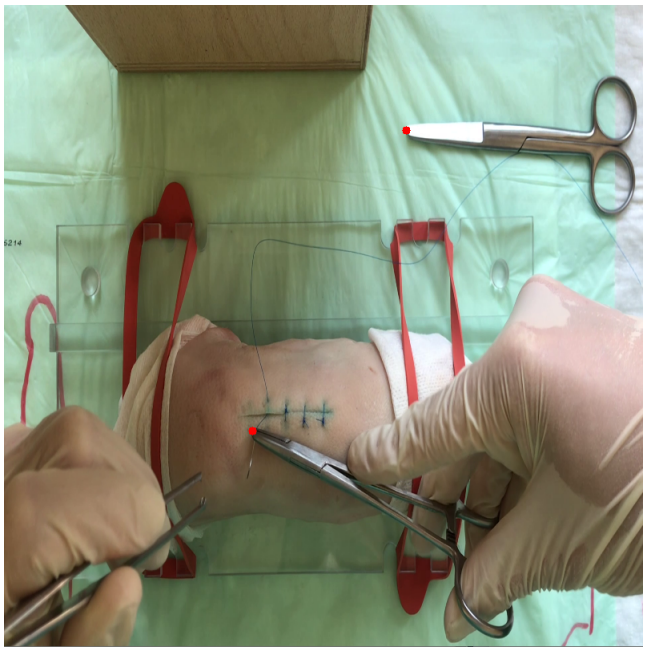
Celkem jsem prováděl tři způsoby vyhodnocování na čtyřech různých natrénovaných modelech. Hlavním cílem bylo vyhodnocení přesnosti algoritmu. Zároveň bylo potřeba posuzovat jeho spolehlivost, tedy jak často detekuje více bodů (obr(16)), případně nedetekuje žádné. V momentě, kdy síť detekovala více bodů, tak je to na grafech vždy vzdálenost toho nejbližšího ke špičce jehly. A zároveň když špička nebyla detekována, tak je to vyobrazeno jako hodnota -1. V tabulce (1) jsou ukázány doby trénování, průměrné chyby a průměrné přesnosti na testovacím datasetu. Síťě byly natrénovány na datatasetu, který obsahoval 7789 snímků v trénovací sadě a 300 snímků ve validační. Testovací sada obsahovala 500 po sobě jdoucích snímků. Důvodem volby testovací sady byly dobré světelné podmínky, zároveň se ve videu nachází jak rychlé, tak pomalé pohyby pod širokou škálou úhlů. Snímek číslo (15) je ukázka snímku z testovacího datasetu.

	Batch Size	**Rozlišení**	**Doba učení**	**Rychlost**	**Průměrná chyba**
ResNet34	8	512×512	36:42	2:48	5.82
ResNet34	2	800×800	1:13:13	3:23	29.41
ResNet34	1	1024×1024	1:56:36	4:48	9.77
ResNet101	1	800×800	2:48:56	6:34	NaN
MIL	-	1920×1080	0	0:58	187
KCF	-	1920×1080	0	0:16	856

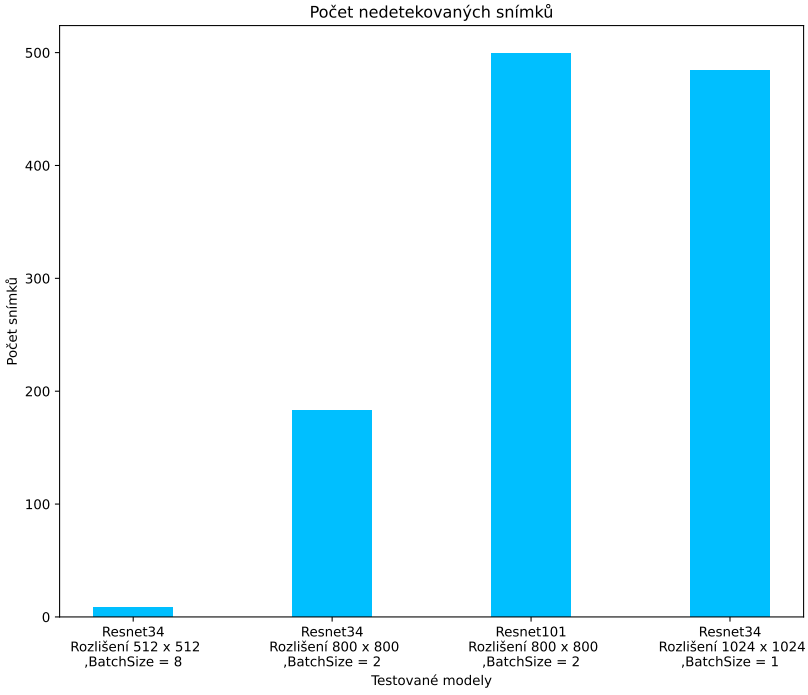
Tabulka 1: Tabulka doby trénování daných architektur
Rychlost říká dobu na vyhodnocení testovacího datasetu
HW: GTX 1060 Max-Q i7-7700HQ



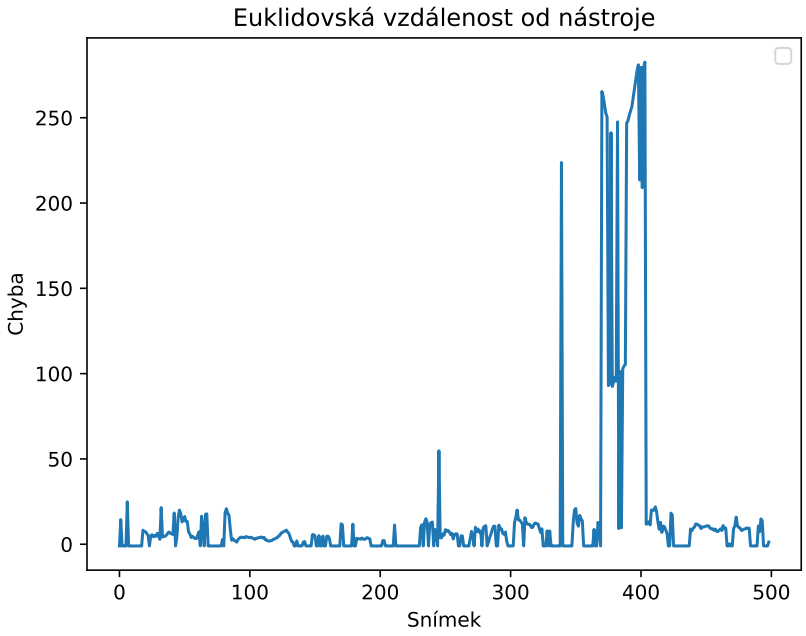
Obrázek 15: Ukázka rozpoznávaného snímku



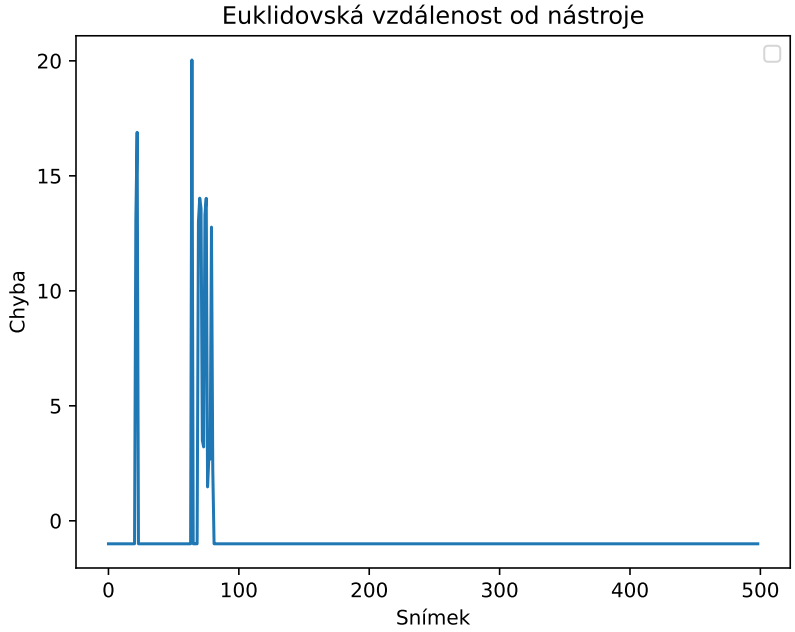
Obrázek 16: Ukázka chybného rozpoznání 2 špiček
Backbone ResNet34 rozlišení 800×800 pixelů



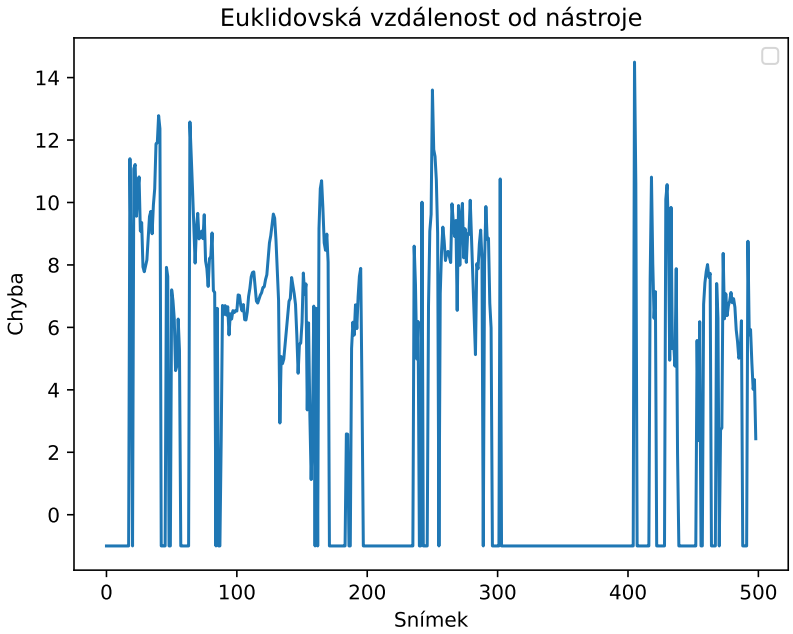
Obrázek 17: Četnost nedetekovaných snímků



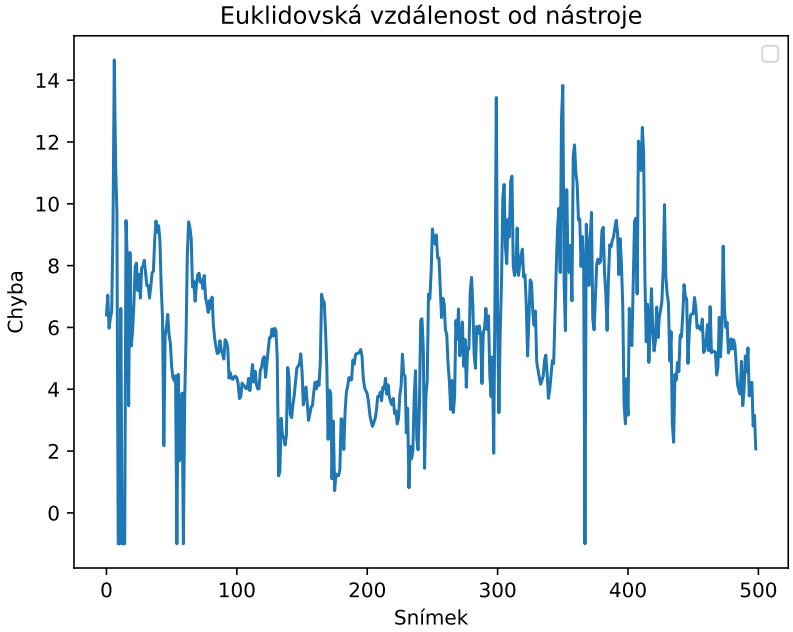
Obrázek 18: Chyba sítě na testovací sadě
Backbone ResNet34 rozlišení 800×800 pixelů batch size = 2



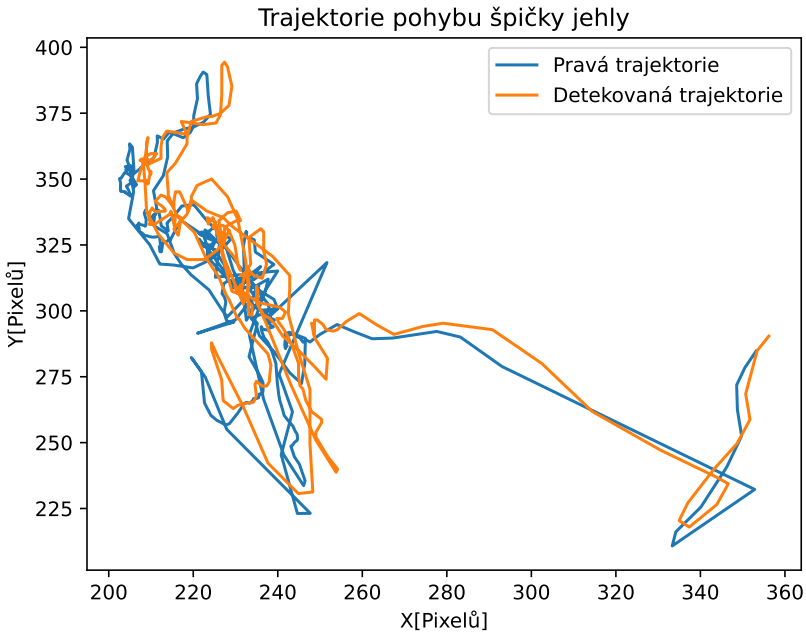
Obrázek 19: Chyba sítě na testovací sadě
Backbone ResNet34 rozlišení 1024×1024 pixelů batch size = 1



Obrázek 20: Chyba sítě na testovací sadě
Backbone ResNet34 rozlišení 800×800 pixelů batch size = 1



Obrázek 21: Chyba sítě na testovací sadě
Backbone ResNet34 rozlišení 512×512 pixelů batch size = 8



Obrázek 22: Trajektorie na testovací sadě
Backbone Resnet34 rozlišení 512×512 pixelů batch size = 8

Z výsledků můžeme vyvodit několik závěrů: zvyšováním rozlišení a složitosti sítě nevedlo k lepším výsledkům. Lze to vidět z grafu (19), že zvýšení rozlišení vedlo na větší složitost. Za další příčinu selhání považuji příliš malý batch size, který nepřispíval ke kvalitnímu učení sítě. Toto jsem otestoval s pomocí vytrénovaní jiné sítě s batch size = 2, kterou můžeme vidět na grafu (18). Její výsledky jsou výrazně lepší, hlavně bylo zásadně potlačeno vypadávání bodu, což lze vidět z grafu (17). Poměrně zajímavé byly výsledky sítě z grafu (21), kdy byla detekce velice přesná s minimem vynechávání. To mě vedlo k vykreslení trajektorie, která velice dobře kopíruje cestu špičky jehly (obr. (22)). Celkově výsledky považuji za uspokojivé, hlavně s porovnáním s klasickými metodami, a domnívám se, že přesnost je dostatečná pro další využití a rozšiřování práce.

Napadají mě hlavně dva směry, jak přesnost metody vylepšit. První z nich je rozšíření výstupu neuronové sítě o LSTM (Long short-term memory). Následný výstup v podobě dvou souřadnicových neuronů by umožňoval odstranění problému s vícenásobnými řešeními. Tato cesta by byla zajímavá, protože by do sítě přidala tzv. paměť předchozích stavů, čímž by byla zajištěna kontinuita mezi snímky. Další možné vylepšení by bylo využití Kalmanova filtru, kde by neuronová síť poslala výsledky a Kalmanův filtr by umožňoval filtrovat nesprávné a příliš rychlé pohyby. Dále by mohlo být zajímavé vyměnit jádro neuronové sítě, například místo ResNet34 využít VGG16. Toto by ovšem mělo za následek daleko větší výpočetní náročnost.

7 Závěr

Cílem této práce bylo navržení metody pro sledování špičky jehly v záznamu chirurgického šití. Tento nástroj vytváří ze záznamu výstup v podobě informace o poloze špičky nástroje. V optimálním případě je tato detekce provedena pro každý snímek v záznamu. Metoda by měla být přenositelná a snadno využitelná při vědecké činnosti, která bude následovat.

Navrhl jsem a implementoval metodu, která zpracovává záznam pomocí neuronové sítě na architektuře ResNet. Výstupem každého snímku je binární maska, která je pozitivní na pixelech, na kterých se nachází špička nástroje. Tyto oblasti se sdružují do shluků, jejichž střed považujeme za místo, kde se nachází nástroj. Metoda byla implementována v jazyce Python s frameworkem TensorFlow. Moje metoda byla porovnána s metodami, které jsou určeny pro sledování objektů a byla implementována v knihovně OpenCV. Přesnost metody se ukázala jako velice dobrá, konkrétně byla průměrná chyba od pravé špičky jehly 5.82 pixelu. Hlavní výhodou tohoto přístupu je možnost detekovat špičku i v jednotlivých snímcích. Práce se věnovala veškerým bodům, které byly zadány na začátku projektu.

Hlavním přínosem této metody je možnost automatického získávání trajektorie pohybu ve videu chirurgického šití. Díky tomu lze provádět hodnocení studentů při výuce chirurgie. Dále byl vyvinut software, který umožní jednoduché přetrénování sítě na jiné chirurgické nástroje v podobných podmínkách.

8 Literatura

[1] Where There Is No Doctor: A Village Health Care Handbook. 1993. London and Oxford: MACMILLAN EDUCATION, 1993. ISBN 0-333-51651-6.

[2] Chirurgické šití.cz [online]. Praha: PARO Úlehlová s.r.o. - Mgr. Tomáš Úlehla, 2015 [cit. 2022-05-12]. Dostupné z: http://chirurgickesiti.cz/chirurgicke-siti/

[3] B. Babenko, M. Yang and S. Belongie, "Visual tracking with online Multiple Instance Learning,"2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 983-990, doi: 10.1109/CVPR.2009.5206737.

[4] Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.

[5] HENRIQUES, Joao F., Rui CASEIRO, Pedro MARTINS a Jorge BATISTA. High-Speed Tracking with Kernelized Correlation Filters. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2015, 37(3), 583-596. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.2014.2345390

[6] BURKOV, Andriy. The Hundred-Page Machine Learning Book. Andriy Burkov, 2019. Connecticut: Andriy Burkov, 2019, 2019. ISBN 9781999579517.

[7] SANTOSH, KC, Swarnendu GHOSH a Nibaran DAS. Deep Learning Models for Medical Imaging. 1. Kolkata: Academic Press, 2021. ISBN 9780128236505.

[8] Ioffe, S., & Szegedy, C. (2015). Batch Normalization:
Accelerating Deep Network Training by Reducing Internal Covariate Shift (Verze 3). arXiv. https://doi.org/10.48550/ARXIV.1502.03167

[9] Kohler, J., Daneshmand, H., Lucchi, A., Zhou, M., Neymeyr, K., & Hofmann, T. (2018). Exponential convergence rates for Batch Normalization: The power of length-direction decoupling in non-convex optimization

[10] Xu, Rui & Wang, Xintao & Chen, Kai & Zhou, Bolei & Loy, Chen Change. (2020). Positional Encoding as Spatial Inductive Bias in GANs.

[11] PEIXEIRO, Marco. Introduction to Convolutional Neural Networks (CNN) with TensorFlow. Towardsdatascience [online]. towardsdatascience: towardsdatascience, 2019 [cit. 2022-05-17]. Dostupné z: https://towardsdatascience.com/introduction-to-convolutional-neural-networks-cnn-with-tensorflow-57e2f4837e18

[12] Gustineli, M. (2022). A survey on recently proposed activation functions for Deep Learning. arXiv preprint arXiv:2204.02921.

[13] BAHETI, Pragati. 12 Types of Neural Network Activation Functions: How to Choose?. Www.v7labs.com [online].

Microsoft: Microsoft, 2022, May 16 [cit. 2022-05-17].

Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions

[14] Yakubovskiy, P. (2019). Segmentation Models. In GitHub repository. GitHub. https://github.com/qubvel/segmentation_models

[15] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv.
https://doi.org/10.48550/ARXIV.1412.6980

[16] GOMEZ, Raul. Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names. In: Gombru.github.io/ [online]. GitHub: Blog, 2018 [cit. 2022-05-17]. Dostupné z: https://gombru.github.io/2018/05/23/cross_entropy_loss/

[17] Boris Sekachev, Nikita Manovich, Maxim Zhiltsov, Andrey Zhavoronkov, Dmitry Kalinin, Ben Hoff, TOsmanov, et al. "Opencv/cvat: V1.1.0". Zenodo, August 31, 2020. https://doi.org/10.5281/zenodo.4009388.

[18] Van der Walt, S., Schönberger, Johannes L, Nunez-Iglesias, J., Boulogne, Franccois, Warner, J. D., Yager, N., ... Yu, T. (2014). scikit-image: image processing in Python. PeerJ, 2, e453.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[20] S. Hare, A. Saffari and P. H. S. Torr, "Struck: Structured output tracking with kernels,"2011 International Conference on Computer Vision, 2011, pp. 263-270, doi: 10.1109/ICCV.2011.6126251.

[21] (2012). IEEE Trans. Pattern Anal. Mach. Intell., 34(7).

[22] Y. LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition,"

in Neural Computation, vol. 1, no. 4, pp. 541-551, Dec. 1989,

doi: 10.1162/neco.1989.1.4.541.