Univerzita Jana Evangelisty Purkyně
v Ústí nad Labem
Přírodovědecká fakulta

Vliv předzpracování obrazu a augmentace dat na segmentaci rentgenových snímků

DIPLOMOVÁ PRÁCE

Vypracoval: Bc. Milan Gittler

Vedoucí práce: RNDr. Jiří Škvára, Ph.D.

Studijní program: Aplikovaná informatika

Ústí nad Labem 2024

Namísto žlutých stránek vložte digitálně podepsané zadání kvalifikační práce poskytnuté vedoucím katedry.
Zadání musí zaujímat právě dvě strany.

Zadání je nutno vložit jako PDF pomocí některého nástroje, který umožňuje editaci dokumentů (se zachováním elektronického podpisu).

V Linuxe lze například použít příkaz `pdftk`.

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a použil jen pramenů, které cituji a uvádím v přiloženém seznamu literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona c. 121/2000 Sb., ve znění zákona c. 81/2005 Sb., autorský zákon, zejména se skutečností, že Univerzita Jana Evangelisty Purkyně v Ústí nad Labem má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Jana Evangelisty Purkyně v Ústí nad Labem oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladu, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

V Ústí nad Labem dne 15. června 2024 Podpis: . . . . . .

Děkuji vedoucímu práce RNDr. Jiřímu Škvárovi, Ph.D.
za neocenitelné rady a pomoc při tvorbě diplomové práce.

Vliv předzpracování obrazu a augmentace dat na segmentaci rentgenových snímků

Abstrakt:

Hlavním cílem této diplomové práce je seznámit čtenáře s

Klíčová slova: lorem, ipsum, dolor, sit, amet

Impact of image preprocessing and data augmentation on segmentation of X-ray images

Abstract:

lorem ipsum dolor sit amet

Keywords: lorem, ipsum, dolor, sit, amet

Obsah

Úvod

1. Přehled metod předzpracování obrazu

Tato kapitola bude zaměřena na představení klíčových metod předzpracování obrazu, které hrají zásadní roli v procesu analýzy a zpracování rentgenových snímků. Předzpracování obrazu představuje kritický krok v řadě aplikací strojového učení a počítačového vidění, neboť ovlivňuje kvalitu a efektivitu následné analýzy. Metody předzpracování obrazu mohou výrazně zlepšit kvalitu dat a zvýšit přesnost detekce objektů, segmentace obrazu či klasifikace. Specifický výběr těchto technik je klíčový pro zjištění přesnosti a efektivity moderních metod počítačového vidění, včetně strojového učení a hlubokého učení, které jsou stále častěji aplikovány na širokou škálu problémů v oblasti zpracování obrazu. Zejména v kontextu rentgenových snímků může předzpracování pomoci překonat některé běžné výzvy, jako je nízký kontrast, šum, nebo artefakty, které mohou snížit kvalitu obrazu a tím ovlivnit diagnózu nebo automatickou analýzu.[1]

1.1. Klasické metody předzpracování obrazu

Následující sekce se zabývá představením klasických metod a technik, které jsou využívány pro úpravu a zlepšení kvality obrazových dat před jejich dalším zpracováním. Budeme se tedy věnovat různým přístupům filtrace obrazu, metodám redukce šumu či zaostření obrazu a také technikám prahování a binarizace, které jsou fundamentální pro analýzu a interpretaci obrazových dat. Přestože se jedná o metody, které mohou být považovány za základní, jejich správná aplikace a kombinace mohou výrazně zlepšit výslednou kvalitu obrazu a přispět k efektivnějšímu rozpoznávání vzorů a objektů v obrazových datech. Podrobně prozkoumáme každou z těchto metod, přičemž budeme klást důraz na jejich význam pro přípravu dat k dalšímu zpracování.

Vylepšení obrazu

Vylepšení obrazu je klíčovou technikou v předzpracování obrazu, zaměřenou na zlepšení vizuální kvality obrazových dat pro následné zpracování nebo analýzu. Vylepšení obrazu usiluje o zlepšení kontrastu, jasu a ostrosti, aby bylo zajištěno, že obrazová data jsou co nejvíce přístupná pro lidské vnímání nebo automatizované algoritmy. Jedním z klíčů k úspěšnému zlepšení obrazu je výběr vhodné metody a jejích parametrů, které musí být pečlivě nastaveny v závislosti na charakteristikách obrazových dat a konkrétním účelu zpracování.

Ekvalizace histogramu

Ekvalizace histogramu je fundamentální, ale mocná technika, která se používá pro zlepšení kontrastu v obrazových datech, zejména tam, kde původní obraz obsahuje špatně rozlišitelné detaily kvůli nedostatečnému rozsahu intenzit pixelů.

Cílem ekvalizace histogramu je aplikovat transformaci na původní histogram obrazu, $H(i)$, kde i představuje intenzitu pixelů v původním obrazu, tak, aby výsledný histogram měl uniformní rozložení. Tato transformace je založena na kumulativní distribuční funkci (CDF), $CDF(i)$, vypočítané z původního histogramu. $CDF(i)$ představuje součet pravděpodobností všech intenzit pixelů od nejnižší hodnoty až po intenzitu i a slouží jako mapovací funkce pro přiřazení nových intenzit pixelů ve výsledném obraze. Matematicky lze CDF definovat jako CDF(j) = $\sum_{i=0}^{j} P(i)$, kde $P(j)$ je pravděpodobnost výskytu intenzity j v původním obrazu, což se obvykle určí normalizací histogramu na celkový počet pixelů v obrazu. Nová intenzita pixelu i' pro každý pixel s původní intenzitou i ve výsledném obrazu je poté určena pomocí normalizované CDF, což zajišťuje, že všechny intenzity jsou rovnoměrně zastoupeny:

$$i' = (L-1) \cdot CDF(i) \tag{1.1}$$

kde: $L =$ je počet možných úrovní intenzity pixelů (např. $L = 256$ pro 8bitové obrazy) Tímto způsobem transformace zvýší kontrast obrazu tak, že "roztáhne"rozložení intenzit pixelů přes celý dostupný rozsah, což zvýrazní detaily a zlepší vizuální vnímání obrazu. Ekvalizace histogramu tak přináší výrazné zlepšení v oblastech s nízkým kontrastem a umožňuje lepší vizualizaci detailů, což je zásadní pro dalšího zpracování obrazu. [2] [3]

Algoritmus 1 Ekvalizace histogramu

1: **Vstup:** Původní obraz Img
2: **Výstup:** Obraz s ekvalizovaným histogramem Img_{eq}
3: Vypočítejte histogram H z Img
4: Vypočítejte kumulativní histogram CH z H
5: Normalizujte CH na rozsah intenzit pixelů obrazu
6: **for** každý pixel p v Img **do**
7: Nastavte $Img_{eq}[p]$ na hodnotu odpovídající $CH[Img[p]]$
8: **end for**
9: **return** Img_{eq}

Adaptivní ekvalizace histogramu

Adaptivní ekvalizace histogramu (AHE) představuje pokročilou metodu ekvalizace, která se snaží zlepšit kontrast obrazu lokálně, na rozdíl od globálního přístupu klasické ekvalizace histogramu. AHE algoritmus rozdělí původní obraz na malé, překrývající se bloky, nazývané dlaždice (tiles), a na každou z nich aplikuje ekvalizaci histogramu nezávisle. Tímto způsobem dokáže lépe zachytit lokální kontrastní charakteristiky obrazu a zvýraznit detaily v jednotlivých oblastech. Při překryvu

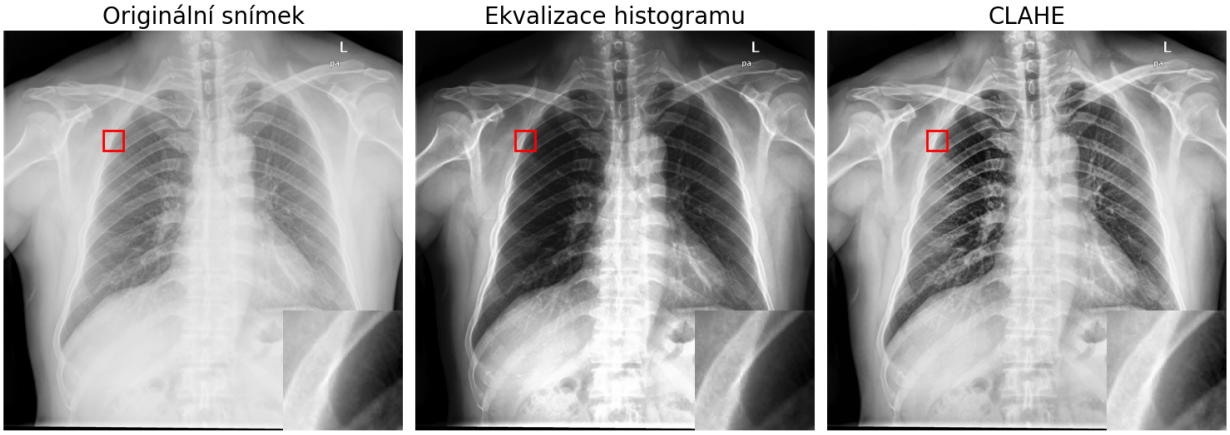
dlaždic se výsledné intenzity pixelů na okrajích vypočítají jako vážený průměr z odpovídajících intenzit získaných z každé příslušné dlaždice, což zajišťuje hladký přechod mezi dlaždicemi. [4]

Adaptivní ekvalizace histogramu s omezením kontrastu

Adaptivní ekvalizace histogramu s omezením kontrastu (CLAHE) byla vyvinuta s cílem předejít problémům spojeným s přílišným zvýrazněním šumu v homogenních oblastech obrazu, které nastává při použití AHE.

Základní myšlenka CLAHE spočívá v rozdělení obrazu na malé, kontextově závislé bloky a aplikování ekvalizace histogramu na každý z těchto bloků. Aby se zabránilo nežádoucímu efektu nadměrného zvýšení kontrastu, aplikuje se na histogram každého bloku proces „osekání“ (clipping), kdy hodnoty histogramu přesahující předem definovaný limit jsou sníženy na tento limit a přebytečné hodnoty jsou rovnoměrně rozděleny mezi ostatní úrovně intenzity. To vede k vytvoření vyváženějšího rozložení intenzit v obrazu a zajišťuje, že zvýšení kontrastu nevede k nežádoucímu zvýraznění šumu.

Jedním z klíčových aspektů CLAHE je výběr „clip limitu“, což je parametr, který omezuje míru zvýšení kontrastu. Tento limit se obvykle definuje jako násobek průměrné hodnoty histogramu a jeho správné nastavení je zásadní pro dosažení optimálního výsledku. [4]



Obrázek 1.1.1.: Ukázka metod ekvalizace histogramu na rentgenovém snímku. CLAHE demonstruje lepší zachování detailů a kontrastu v porovnání s tradiční ekvalizací histogramu.

Unsharp masking

Unsharp masking je technika určená k zlepšení ostrosti obrazu tím, že zvýrazňuje hrany a detaily, které jsou v původním obrazu méně patrné. Klíčovým krokem této metody je vytvoření tzv. masky ostrých detailů, která vznikne odečtením rozmazané verze obrazu od jeho původní

podoby. Rozmazaná verze obrazu se získává například pomocí Gaussova filtru, který je aplikován na původní obraz. Intenzita tohoto rozmazání určuje, jak silně budou hrany a detaily v masce zvýrazněny. Následující rovnice ukazuje získání výsledného, zaostřeného snímku pomocí techniky unsharp masking. Po získání masky ostrých detailů se tato maska přičte zpět k původnímu obrazu. Tento krok zvyšuje kontrast a ostrost obrazu, přičemž zvýrazňuje oblasti s výraznými rozdíly mezi sousedními pixely. [5]

Algoritmus 2 Unsharp Masking

1: **Vstup:** Původní obraz I, standardní odchylka Gaussova filtru σ, zesílení λ
2: **Výstup:** Vylepšený obraz I'
3: $G \leftarrow \text{Gauss}(I, \sigma)$ ▷ Vytvoření rozmazané verze obrazu pomocí Gaussova filtru
4: $M \leftarrow I - G$ ▷ Vytvoření masky ostrých detailů odečtením rozmazaného obrazu
5: $I' \leftarrow I + \lambda M$ ▷ Zvýšení ostrosti původního obrazu přičtením masky

V tomto procesu:

- σ určuje míru rozmazání při použití Gaussova filtru.
- λ je koeficient zesílení, který kontroluje míru, jakou jsou detaily zvýrazněny při přičítání masky k původnímu obrazu.

Při použití unsharp masking metody může vznikat nežádoucí halo efekt. Tento efekt se projevuje jako nežádoucí světelný okraj kolem kontrastních hran, což může vést k umělému a nepřirozenému vzhledu obrazu. Integrace filtru zachovávajícího hrany (edge-preserving filter) do procesu unsharp masking může být klíčová pro minimalizaci zmíněného efektu. [5]

Unsharp masking metoda je vhodná pro aplikace, kde je žádoucí zlepšit viditelnost detailů a hran. Důležitým aspektem při použití unsharp masking je správný výběr parametrů σ a λ, jelikož tyto hodnoty významně ovlivňují výsledný vzhled obrazu. Příliš vysoké hodnoty mohou vést k nežádoucím artefaktům, jako jsou halo efekty kolem hran, zatímco příliš nízké hodnoty mohou mít za následek nedostatečné zvýraznění detailů.

Gama korekce

Gamma korekce je technika používaná k úpravě jasu obrazových dat. Principem této metody je aplikace nelineární transformace na původní pixelové hodnoty obrazu, čímž se upravuje celková jasová křivka podle gamma funkce. Tato korekce umožňuje efektivněji využít dynamický rozsah zobrazovacích zařízení a zlepšit viditelnost detailů v tmavých i světlých oblastech obrazu bez ztráty informací v ostatních částech obrazového spektra. [6]

Aplikace gamma korekce se obvykle provádí podle vzorce:

$$I' = I^{\gamma} \tag{1.2}$$

kde: I = původní intenzita pixelu

I' = nová intenzita pixelu po aplikaci gamma korekce

γ = gamma faktor, který určuje míru korekce

Hodnota gamma faktoru menší než 1 zvýší jas tmavších oblastí obrazu, zatímco hodnota větší než 1 ztmaví světlejší oblasti a zvýší kontrast.

Redukce šumu

Redukce šumu ve snímcích je zásadní proces v digitálním zpracování obrazu, který má za cíl odstranit nebo minimalizovat vliv šumu přidaného během akvizice nebo přenosu obrazových dat. Šum může být způsoben různými externími faktory a může výrazně snížit kvalitu a čitelnost snímků, což má negativní dopad na jejich další analýzu a interpretaci.

Šum je obvykle definován jako náhodná variace intenzity obrazových bodů (pixelů), která se projevuje jako viditelné zrnění nebo textura na snímku. Tato náhodná variabilita může být důsledkem základních fyzikálních procesů, jako je povaha fotonu světla nebo tepelná energie v senzorech obrazu. Může vzniknout v okamžiku snímání nebo při přenosu obrazu.

Jednoduchý matematický model, kterým lze vyjádřit přítomnost aditivního šumu v obrazu, můžeme definovat jako:

$$f(x, y) = I(x, y) + N(x, y) \tag{1.3}$$

kde: $f(x, y)$ = je pozorovaná intenzita pixelu se šumem

$I(x, y)$ = je skutečná intenzita pixelu bez šumu

$N(x, y)$ = je hodnota šumu přidaná k pixelu

Redukce šumu je proto klíčovým krokem ve zpracování obrazu, který pomáhá zlepšit celkovou kvalitu snímků tím, že vyhlazuje obraz a odstraňuje nepřesnosti, avšak s opatrností, aby nedošlo k ztrátě důležitých detailů nebo hran v obrazu. Efektivní algoritmy pro redukci šumu se snaží dosáhnout rovnováhy mezi odstraněním šumu a zachováním nebo dokonce zvýrazněním důležitých vlastností obrazu, jako jsou hrany nebo textury. [7, 8]

Typy šumu ve snímcích

Existuje několik typů šumu, které se mohou ve snímcích vyskytovat. Tyto šumy se liší svým původem, charakteristikami a vlivem na kvalitu obrazu. Pro účinnou redukci šumu je klíčové rozpoznat, s jakým typem šumu máme co do činění, aby bylo možné zvolit nejvhodnější metodu pro jeho odstranění. Níže jsou uvedeny některé z nejčastějších typů šumu, které mohou ovlivnit digitální snímky.

- **Gaussovský šum** je pravděpodobně nejrozšířenějším typem šumu v digitálních obrazových systémech, způsobený především elektronickými fluktuacemi v senzoru a obvodech kamery.

Je charakterizován normálním rozdělením intenzit pixelů kolem skutečné hodnoty s určitou střední hodnotou a standardní odchylkou.

$$P(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(n-\mu)^2}{2\sigma^2}}$$

Gaussovský šum je obvykle považován za aditivní šum, což znamená, že jeho hodnota je nezávisle přidána k intenzitě každého pixelu. [7]

- **Impulzní šum (Šum soli a pepře)** se projevuje jako náhodné černé nebo bílé (nebo obojí) pixely rozptýlené po celém obrazu. Tento typ šumu může vzniknout kvůli chybám v senzoru nebo při přenosu dat. Na rozdíl od Gaussovského šumu je impulzní šum typicky nelineární a jeho redukce vyžaduje specifické nelineární filtrační techniky. [7]
- **Poissonův šum** je typ šumu, který je často přítomen v obrazových datech získaných z nízkých intenzit světla, jako jsou rentgenové snímky nebo mikroskopické snímky. Poissonův šum je důsledkem náhodného procesu počtu fotonů, které dopadají na senzor, a je charakterizován jako náhodný proces s diskrétním rozdělením pravděpodobnosti. Redukce Poissonova šumu vyžaduje specifické metody, které jsou schopny zacházet s diskrétními daty a zároveň zachovat důležité detaily v obraze. [7] Poissonův šum může být modelován jako:
$$P(n) \sim \frac{e^{-\lambda}\lambda^n}{n!}$$
- **Speckle šum** se řadí do skupiny multiplikativních šumů, které jsou často přítomny v ultrazvukovém a radarovém zobrazování. Tento šum je důsledkem koherence vlnění použitého pro snímání a interferencí mezi rozptýlenými vlnami. Speckle šum může být modelován pomocí Rayleighova nebo Gamma rozdělení v závislosti na povaze obrazových dat. [7]

Typy filtrů pro redukci šumu

Pro efektivní redukci šumu ve snímcích jsou k dispozici různé typy filtrů, které se liší svým principem a účinností v závislosti na konkrétním typu šumu. Níže jsou uvedeny některé z nejčastěji používaných filtrů pro redukci šumu ve snímcích.

- **Lineární filtrování**, také známo jako konvoluční filtrování, aplikuje lineární operátor na skupinu sousedicích pixelů v obrazu s cílem vyhladit nebo zvýraznit určité vlastnosti obrazu. Hlavní principem lineárních filtrů je konvoluce obrazu s jádrem filtru či maskou, což je malá matice daného rozměru, která určuje váhy pro každý pixel v okolí. Tyto váhy určují, jakým způsobem okolní pixely ovlivňují hodnotu pixelu výstupního obrazu. Hlavní výhodou lineárních filtrů je jejich jednoduchost a efektivita, což je dělá vhodnými pro rychlé a efektivní zpracování obrazu. Nicméně, lineární filtry mohou způsobovat neostrost hran and detailů ve snímcích. [7, 9]

$$I_{filtered}(x,y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} K(i,j) \cdot I(x-i, y-j) \tag{1.4}$$

kde: $I_{filtered}(x,y)$ = intenzita pixelu na pozici (x, y) ve filtrovaném snímku
$K(i,j)$ = koeficienty kernelu
$I(x-i, y-j)$ = intenzita pixelu v původním snímku v okolé pixelu (x, y)

- **Nelineární filtrování** je alternativní přístup k redukci šumu, který se zaměřuje na zachování hran a detailů v obraze. Nelineární filtry jsou schopny zacházet s nelineárními vztahy mezi pixely a jsou schopny lépe zachovat důležité strukturální vlastnosti obrazu. Nelineární filtry jsou proto obzvláště účinné při redukci impulzního či spekle šumu. [7, 9]

Gaussovský filtr

Gaussovský filtr reprezentuje jeden z nejčastěji používaných lineárních filtrů pro redukci šumu ve snímcích. Tento filtr je založen na aplikaci Gaussova jádra na obraz, což umožňuje odstranit vysokofrekvenční šum. Gaussovský filtr je založen na principu aplikace Gaussovy funkce na vstupní obraz, která počítá vážený průměr okolí každého pixelu, s nejvyšší váhou soustředěnou na samotný pixel a se zmenšujícími se váhami pro sousední pixely na základě jejich prostorové vzdálenosti od středového pixelu. Gaussovský filtr je založen na konvoluci obrazu s Gaussovým jádrem, které je definováno jako:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.5}$$

kde: $G(x,y)$ = hodnota Gaussova jádra na pozici (x, y)
σ = standardní odchylka Gaussova jádra

Gaussovský filtr je schopen efektivně redukovat Gaussovský šum a jednou z hlavních výhod je jeho přirozená odezva na vysokofrekvenční oblasti, které potlačuje bez vytváření kroužkouvých artefaktů, které jsou běžné u low-pass filtrů.

Přestože je Gaussovský filtr velmi účinný při redukci šumu, volba směrodatné odchylky σ je kritická. Příliš vysoká hodnota může vymazat důležité detaily spolu se šumem, zatímco příliš nízká hodnota nemusí šum dostatečně snížit. Navíc, izotropní povaha Gaussovského filtru znamená, že filtr rozmazává ve všech směrech stejnou měrou, což nemusí být žádoucí zejména tam, kde je potřeba zachování hran. [9]

Bilaterální filtr

Bilaterální filtrování je nelineární technika pro vyhlazování obrazu, která zachovává hrany a redukuje šum. Na rozdíl od tradičních lineárních filtrů, které aplikují uniformní konvoluci napříč

Algoritmus 3 Gaussovský filtr

1: **Vstup:** Původní obraz I, standardní odchylka σ
2: **Výstup:** Vyhlazený obraz $I_{smoothed}$
3: Vytvořte Gaussovo jádro G s rozměry $n \times n$ a σ
4: **Inicializuj** $I_{smoothed}$ jako prázdný obraz
5: **for** každý pixel p v I **do**
6: Aplikujte Gaussovo jádro na okolí pixelu p
7: Nastavte hodnotu pixelu p v $I_{smoothed}$ na vážený průměr hodnot v okolí
8: **end for**
9: **return** $I_{smoothed}$

obrazem, bilaterální filtr kombinuje dva hlavní koncepty: prostorovou blízkost a podobnost intenzity.

Podobně jako Gaussův filtr, bilaterální filtr používá Gaussovu funkci k vážení pixelů založeně na jejich prostorové vzdálenosti od centrálního pixelu. Tím pádem, čím jsou pixely blíže k centrálnímu pixelu, tím větší váhu mají. Unikátní pro bilaterální filtr je fakt, že váhy jsou také přizpůsobeny na základě rozdílu intenzit mezi sousedními pixely a centrálním pixelem. Pixely s podobnou intenzitou jako centrální pixel získávají vyšší váhu, což umožňuje zachování ostrých hran a detailů. Pro každý pixel ve zpracovávaném obrazu filtr vyhodnotí jeho sousedy a aplikuje váhy založené na dvou zmíněných kritériích.[10]

Matematický vzorec pro bilaterální filtr aplikovaný na pixel p v obrazu I lze vyjádřit jako:

$$BF[I](p) = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p - q||) \cdot G_{\sigma_r}(|I_p - I_q|) \cdot I_q \tag{1.6}$$

kde: $BF[I](p)$ = hodnota pixelu po aplikaci bilaterálního filtru

W_p = je normalizační faktor zajišťující, že součet vah je roven jedné

G_{σ_s} = prostorová Gaussova funkce, která snižuje vliv vzdálených pixelů na základě prostorové vzdálenosti ||p-q|| se standardní odchylkou σ_s

G_{σ_r} = rozsahová Gaussova funkce, která redukuje vliv pixelů q na základě rozdílu intenzit $|I_p - I_q|$ se standardní odchylkou σ_r

I_p = hodnota pixelu p

I_q = hodnota pixelu q

$$W_p = \sum_{q \in S} G_{\sigma_s}(||p - q||) \cdot G_{\sigma_r}(|I_p - I_q|) \tag{1.7}$$

Na rozdíl od Gaussova filtru, který aplikuje vyhlazování uniformně a může vést k rozmazání ostrých hran, bilaterální filtr nabízí sofistikovanější přístup k redukci šumu. Jeho schopnost

zachovat ostré hrany při současném snížení šumu v homogenních oblastech obrazu činí bilaterální filtr vysoce ceněným nástrojem v pokročilém zpracování obrazu. Avšak, s touto schopností přichází zvýšená výpočetní náročnost a potřeba pečlivého nastavení parametrů σ_s a σ_r, což jsou aspekty, které je třeba zvážit pro dosažení optimálních výsledků. [10]

Wienerův filtr

Wienerův filtr je adaptivní lineární filtr, který je často používán pro redukci aditivního šumu ve snímcích s cílem zachovat důležité detaily a strukturální vlastnosti obrazu. Wienerův filtr funguje na principu minimalizace střední kvadratické chyby (anglicky minimum mean squared error - MMSE)[1] mezi původním obrazem a obrazem se šumem. Matematicky je Wienerův filtr navržen tak, aby našel spektrální rovnováhu mezi inverzním filtrováním signálu a vyhlazováním šumu, s ohledem na výkonová spektra signálu a šumu. Odezva filtru je vypočítána pomocí výkonových spekter, což vede k optimálnímu vyvážení mezi inverzním filtrováním pro obnovu obrazu a potlačením šumu.

$$G(u,v) = \frac{1}{H(u,v)} \left[\frac{|H(u,v)|^2}{|H(u,v)|^2 + \frac{S_\eta}{S_f}} \right] \tag{1.8}$$

kde: $G(u,v)$ = odhad originálního obrazu

$H(u,v)$ = degradační funkce

S_η = výkonové spektrum šumu

S_f = výkonové spektrum originálního obrazu

Tento vztah představuje schopnost Wienerova filtru vyvažovat mezi inverzním filtrováním a vyhlazováním šumu s ohledem na degradační funkci a poměr signálu a šumu $\frac{S_\eta}{S_f}$[2]. Filtr je více nakloněn k inverznímu filtrování, kde je magnituda degradační funkce velká vzhledem k šumu (vysoké SNR), a více k vyhlazování v opačném případě. [11]

Mediánový filtr

Medianový filtr je robustní nelineární filtrační technika, který se používá k redukci impulzního šumu (šumu soli a pepře). Medianový filtr funguje tak, že posouvá okno po každém pixelu obrazu, shromažďuje hodnoty pixelů z okolí daného pixelu a nahrazuje středový pixel mediánem těchto hodnot. Okno, které se obvykle používá, je čtvercové (např. 3x3, 5x5) v závislosti na stupni šumu a požadované úrovni zachování detailů. Na rozdíl od průměrových filtrů, které také snižují šum, ale mají tendenci rozmazávat hrany, medianový filtr hrany zachovává. Dále také medianový filtr nezávisí na předpokladu, že šum je specifického typu nebo variance.

[1] MSE je metrika, která měří rozdíl mezi odhadovanými a skutečnými hodnotami. V případě Wienerova filtru se minimalizuje rozdíl mezi původním obrazem a obrazem se šumem.

[2] Poměr signálu a šumu (SNR) udává poměr mezi obrazem a úrovní šumu v daném obraze. Vysoká hodnota SNR

Algoritmus 4 Medianový filtr

1: **Vstup:** Původní obraz I, velikost okna k
2: **Výstup:** Obraz po redukci šumu $I_{filtered}$
3: **Inicializuj** $I_{filtered}$ jako prázdný obraz
4: **for** každý pixel p v I **do**
5: okolí ← extrahuj_okolí(I, p, k)
6: medián ← vypočti_medián(okolí)
7: Nastavte hodnotu pixelu p v $I_{filtered}$ na medián
8: **end for**
9: **return** $I_{filtered}$

Přestože je medianový filtr účinný, má svá omezení, zejména při vysokých úrovních šumu nebo když mají obrazy jemné strukturální detaily. Za těchto podmínek může filtr odstranit užitečné informace spolu se šumem. Navíc jeho výkon se může zhoršit, pokud není velikost okna správně zvolena, protože příliš velké okno může odstranit příliš mnoho detailů, zatímco příliš malé okno nemusí efektivně odstranit dostatek šumu.

Filtr ne-lokálních průměrů

Filtr ne-lokálních průměrů, anglicky Non-Local Means (NLM), představuje pokročilou techniku pro odstranění šumu z obrazu (převážně gaussovského šumu), která se vyznačuje výrazným zachováním detailů a hran. Na rozdíl od tradičních metod, které pracují s lokálními okolími pixelů, NLM vyhledává podobné fragmenty (patche) po celém obrazu a využívá jejich informace pro inteligentní vyhlazení cílového pixelu nebo oblasti, čímž se minimalizuje ztráta důležitých vizuálních prvků. NLM můžeme rozdělit na dvě varianty: pixelwise a patchwise NLM.

První varianta, pixelwise NLM, se zaměřuje na každý pixel individuálně. Pro každý pixel p ve vstupním obraze I se vypočítává jeho nová hodnota jako vážený průměr hodnot všech pixelů v obraze. Váhy závisí na podobnosti malých okolí (patchů) okolo každého pixelu a jsou určeny Euklidovskou vzdáleností mezi hodnotami pixelů v těchto okolích. Matematicky lze pixelwise NLM vyjádřit pomocí vztahu:

$$I_{new}(p) = \frac{\sum_{q \in I} w(p,q) \cdot I(q)}{\sum_{q \in I} w(p,q)} \tag{1.9}$$

kde: $w(p,q)$ = představuje váhu založenou na podobnosti mezi okolím pixelů p a q

$I(q)$ = hodnota pixelu q ve vstupním obraze

Druhá varianta, patchwise NLM, rozšiřuje přístup pixelwise metody tím, že jako základní jednotku pro výpočet váženého průměru bere celé patche namísto jednotlivých pixelů. Tím se zvyšuje

značí, že signál je silnější než šum, což znamená, že obraz má vysokou kvalitu s minimálním množstvím šumu. Naopak nízká hodnota SNR indikuje, že šum je dominantní a signál je obtížně rozeznatelný.

robustnost metody vůči šumu, jelikož se lépe využívá strukturální informace z obrazu. Vztah pro patchwise NLM může být zapsán obdobně, avšak s důrazem na porovnávání celých patchů.

Rozdíl mezi pixelwise a patchwise variantami spočívá v měřítku, na kterém se hodnotí podobnost mezi částmi obrazu. Zatímco pixelwise varianta se soustředí na detailní porovnání na úrovni pixelů, patchwise přístup poskytuje robustnější odhad tím, že zapojuje větší kontext okolí zkoumaného bodu.

Algoritmus NLM, ať už v pixelwise či patchwise variantě, nabízí vynikající schopnost odstraňování šumu při zachování ostrých hran a detailů. Jeho hlavní nevýhodou však zůstává vysoká výpočetní náročnost, zejména v důsledku nutnosti porovnávat velké množství oblastí napříč celým obrazem. To činí aplikaci NLM, především v reálném čase, výzvou. Optimalizace a využití paralelního zpracování však mohou tuto nevýhodu zmírnit. [12]

Lokálně adaptivní mediánový filtr

Lokálně adaptivní mediánový filtr je inovativní metodou redukce šumu, specificky navrženou pro typ šumu speckle. Tento typ šumu je výsledkem interferencí mezi koherentními vlnami odraženými od cíle, což vede k zrnitému vzoru v pořízeném obraze. Tento filtr využívá lokální statistiky pro detekci speckle šumu a jeho nahrazení mediánovou hodnotou z lokálního okolí.

Nejdříve je nutné zjistit celkový součet intenzit pixelů v určitém oblastním okně, které obklopuje centrální pixel. Tento součet je základem pro další výpočty statistik, jako je lokální průměr.

$$S(i,j) = \sum_{n=j-k}^{j+k} \sum_{m=i-k}^{i+k} d(m,n) \tag{1.10}$$

kde: k = poloměr posuvného okna okolo středového pixelu

$d(m,n)$ = hodnota pixelu na pozici (m, n)

Dále se určí průměrná hodnota a lokální standardní odchylka hodnot pixelů v posuvném okně. Lokální průměr pomáhá rozlišit oblasti s vysokým a nízkým šumem tím, že poskytuje základní hodnotu pro porovnání s ostatními pixely. Standardní odchylka je míra variability nebo rozptylu hodnot pixelů kolem lokálního průměru, která pomáhá identifikovat pixely s výrazně odlišnými intenzitami (tj. šum).

$$\mu(i,j) = \frac{S(i,j)}{N(i,j)} \tag{1.11}$$

kde: $S(i,j)$ = součet všech hodnot pixelů v posuvném okně

$N(i,j) = (2k+1)^2$ = počet pixelů v posuvném okně

$$\sigma(i,j) = \sqrt{\frac{\sum_{m=i-k}^{i+k} \sum_{n=j-k}^{j+k} (d(m,n) - \mu(i,j))^2}{N(i,j)}} \tag{1.12}$$

kde: $\mu(i,j)$ = lokální průměr hodnot pixelů v posuvném okně

Na základě definováné dolní a horní hranice pro rozpoznání validních pixelů v rámci posuvného okna se rozhodne, které pixely jsou považovány za validní, a které představují potenciální šum.

$$\begin{aligned} LB(i,j) &= \mu(i,j) - M \cdot \sigma(i,j) \\ UB(i,j) &= \mu(i,j) + M \cdot \sigma(i,j) \end{aligned} \tag{1.13}$$

kde: $\mu(i,j) =$ lokální průměr hodnot pixelů v posuvném okně

$\sigma(i,j) =$ lokální standardní odchylka hodnot pixelů v posuvném okně

$M \quad =$ uživatelem definovaný násobitel pro určení prahu šumu

$$l(m,n) = \begin{cases} 0 & \text{pokud } d(m,n) < LB(i,j) \text{ nebo } d(m,n) > UB(i,j) \\ 1 & \text{pokud } LB(i,j) \leq d(m,n) \leq UB(i,j) \end{cases} \tag{1.14}$$

kde: $LB(i,j),\ UB(i,j) =$ dolní a horní hranice pro validní hodnoty pixelů

Finální rovnice nahrazuje hodnotu centrálního pixelu, identifikovaného jako šum, mediánovou hodnotou z validních pixelů v okně. Tento proces zajišťuje, že hodnota pixelu reprezentuje jeho skutečné okolí a zlepšuje tak celkovou kvalitu obrazu redukcí šumu.

$$r(i,j) = \text{median } d(m,n) \mid l(m,n) = 1, i-k \leq m,n \leq i+k \tag{1.15}$$

kde: $l(m,n) =$ maska určující, zda je pixel na pozici (m,n) považován za validní nebo za šum

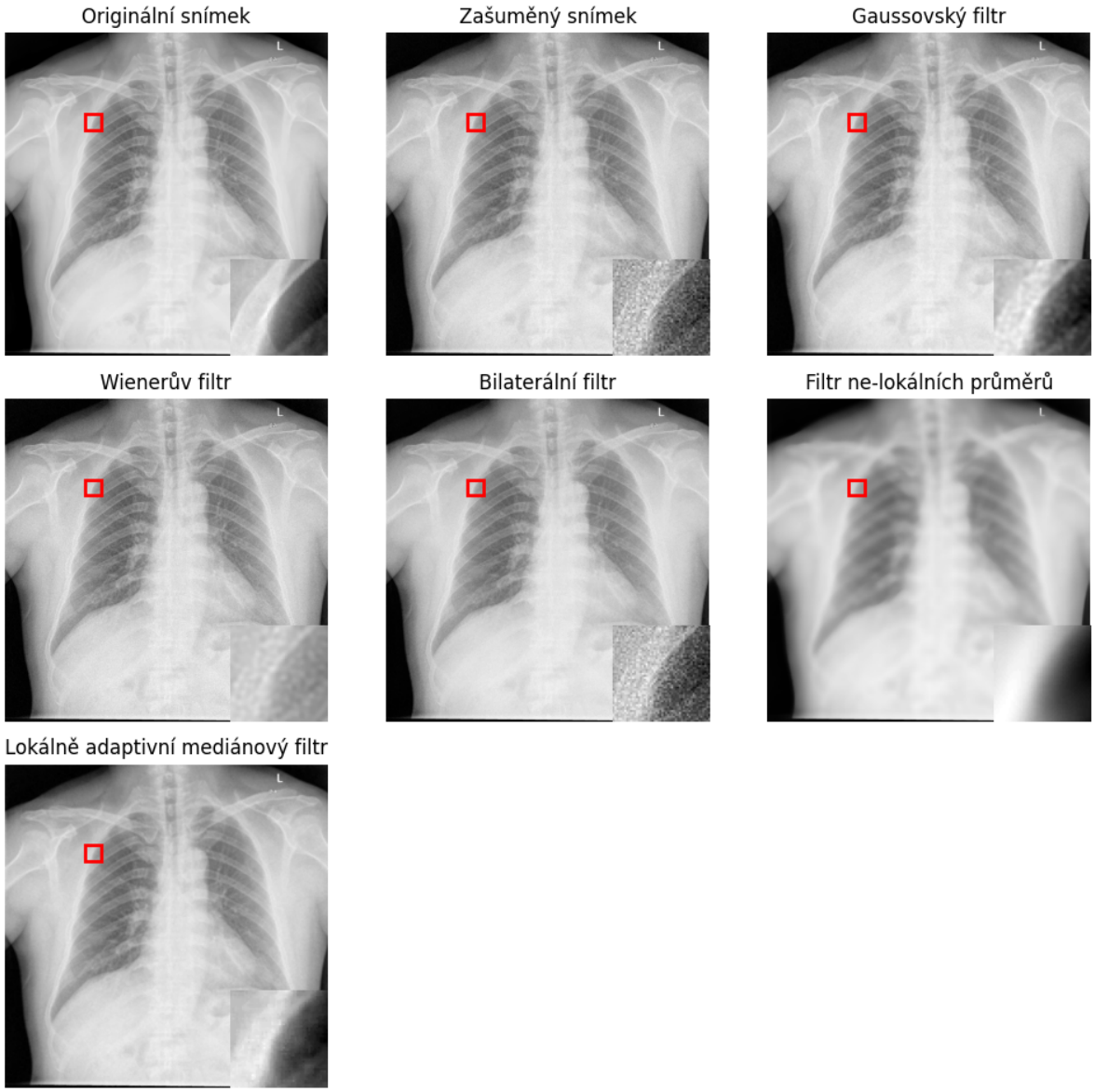
Lokálně adaptivní mediánový filtr se ukazuje jako výrazně efektivní při odstraňování speckle šumu, což je běžný problém v rentgenových a jiných druzích medicínských snímků. Využití lokálních statistik umožňuje filtru selektivně cílit na šumové pixely, zatímco zachovává ostré hrany a důležité detaily, což je klíčové pro zachování diagnostické hodnoty obrazu.

Nicméně, stejně jako většina pokročilých metod zpracování obrazu, i tento filtr má své nevýhody. Především je to jeho výpočetní náročnost, která může být limitující pro aplikace v reálném čase nebo při zpracování velkých datasetů. Navíc, efektivní využití filtru vyžaduje pečlivý výběr parametrů, což může vyžadovat pokročilé znalosti a experimentální ladění pro dosažení optimálních výsledků. [13]

1.2. Neuronové sítě pro předzpracování obrazu

Předchozí sekce byla věnována klasickým metodám předzpracování obrazu, kde byl brán důraz na techniky pro vylepšení obrazu a odstranění či redukci různých typů šumu ze snímků. Tato sekce se bude zabývat pokročilým oborem pro předzpracování obrazu, kterým jsou neuronové sítě.

Neuronové sítě představují významný skok v možnostech zpracování obrazu, především díky jejich schopnosti učit se a generalizovat na základě velkých datových sad. Na rozdíl od klasických



Obrázek 1.1.2.: Snímky demonstrují účinky popsaných filtrů na kvalitu obrazu v porovnání s originálním snímkem a snímkem po umělém přidání šumu, aby se simulovalo běžné zkreslení vznikající během akvizice obrazu.

metod, které aplikují pevně stanovená pravidla, se neuronové sítě dokáží přizpůsobit na základě konkrétních charakteristik a variací v datech, na kterých jsou trénovány. Tato adaptabilita je klíčová pro zpracování obrazů v dynamických a různorodých podmínkách, což zahrnuje vše od standardního zlepšování obrazu po složité úlohy jako je rozpoznávání objektů nebo segmentace obrazu.

Neuronové sítě, a zvláště konvoluční neuronové sítě (CNN), jsou vysoce efektivní v automatickém učení a rozpoznávání vzorců z obrazových dat. Tyto sítě simulují způsob, jakým lidský mozek zpracovává vizuální informace, což umožňuje extrakci a učení vysokého řádu charakteristik, které jsou následně využity pro klasifikaci nebo predikci. Díky této schopnosti se neuronové sítě stávají základem mnoha pokročilých systémů pro automatizované zpracování obrazu.

Využití neuronových sítí však také přináší výzvy, včetně potřeby velkého množství anotova-

ných trénovacích dat a výpočetně náročných modelů, které mohou vyžadovat specializovanou hardwarovou podporu pro efektivní trénink a nasazení.

Následující sekce bude zaměřena na nejčastěji využívané architekty neuronových sítí pro předzpracování obrazu, s důrazem na konvoluční neuronové sítě (CNN). [14]

Úvod do konvolučních neuronových sítí

Neuronové sítě jsou základním stavebním prvkem moderního strojového učení a mají zásadní význam pro rozvoj oblastí jako je zpracování přirozeného jazyka či obrazu, rozpoznávání objektů v obraze, při analýze signálu, v robotice a mnoha jiných důležitých oborech. Tyto systémy jsou inspirovány strukturou a funkcí lidského mozku a jsou navrženy tak, aby modelovaly složité vzory a vztahy v datech. Strukturálně jsou neuronové sítě složeny z vrstev neuronů, což jsou jednotky zpracovávající informace, které jsou propojeny spojeními, ohodnocenými vahami.

Hlavní komponenty neuronových sítí

Základem neuronové sítě je její architektura, která je definována několika klíčovými komponentami. Tyto komponenty jsou základem pro učení a zpracování dat v síti. V následujícím seznamu jsou jednotlivé komponenty podrobněji popsány:

- **Neuron** je elementární jednotka v neuronové síti, která simuluje chování neuronu v biologických systémech. Přijímá vstup $x = (x_1, x_2, x_3, ..., x_n)$, který je zpracován a následně vrácen na výstupu. Tento výstup je vstupen do dalšího neuronu nebo je součástí koncového výstupu ze sítě.

$$z = W \cdot x + b \tag{1.16}$$

  kde: W = reprezentuje váhy, jenž jsou parametry určující sílu signálu mezi neurony.

  b = bias, což je parametr posunutí, který upravuje výstup neuronu před aplikací aktivační funkce.

- **Vrstvy (Layers)** se skládají z neuronů a jsou základem architektury neuronové sítě. Každá vrstva plní specifickou funkci v procesu učení. Vrstvy mohou být kategorizovány do tří typů:
  - **Vstupní vrstva (Input layer)** přijímá surová data a připravuje je pro další zpracování v síti. Každý neuron této vrstvy přímo reprezentuje jednu vlastnost vstupního datového vektoru.
  - **Skyté vrstvy (Hidden layers)** jsou mezi vstupní a výstupní vrstvou a jejich úkolem je zpracování signálů přijatých od vstupní vrstvy. Komplexnost a hloubka těchto vrstev jsou klíčové pro schopnost sítě učit se složité vzory.

  - **Výstupní vrsta (Output layer)** je poslední vrstva, která generuje predikce určené neuronovou sítí. Například v klasifikačních úlohách může tato vrstva používat softmax funkci[3].

- **Aktivační funkce (Activation function)** je důležitou složkou neuronových sítí, která umožňuje modelování nelineárních vztahů. Funkce, jako je sigmoid, tanh a ReLU (Rectified Linear Unit), jsou běžně používány k výpočtu výstupu z neuronů na základě vstupu a vah. Například, ReLU, která je definována jako $ReLU(x) = max(0, x)$, je populární pro svou schopnost rychle konvergovat během tréninku a minimalizovat problém mizejících gradientů[4].
- **Ztrátová funkce (Loss function)**, někdy nazývaná penalizační funkce, poskytuje kvantitativní hodnocení toho, jak daleko jsou predikce neuronové sítě od skutečných hodnot. Během tréninku neuronové sítě je hlavním cílem minimalizace této funkce. Mezi běžné ztrátové funkce se řadí střední kvadratická chyba (MSE) pro regresní úkoly a cross-entropy pro klasifikační úkoly.
- **Optimalizační algoritmus (Optimizer)** je základem učení neuronových sítí, protože určují, jak se síť učí z dat tím, že iterativně upravují váhy a biasy k minimalizaci ztrátové funkce. Stochastic Gradient Descent (SGD) provádí tuto úpravu tak, že v každém kroku tréninku náhodně vybere vzorek dat a spočítá gradient ztrátové funkce pouze na základě tohoto vzorku: Adam a RMSprop jsou pokročilejší varianty, které se snaží zlepšit konvergenci SGD tím, že upravují rychlost učení pro každý parametr individuálně, což může vést k rychlejšímu a stabilnějšímu učení, zejména v komplexních sítích.

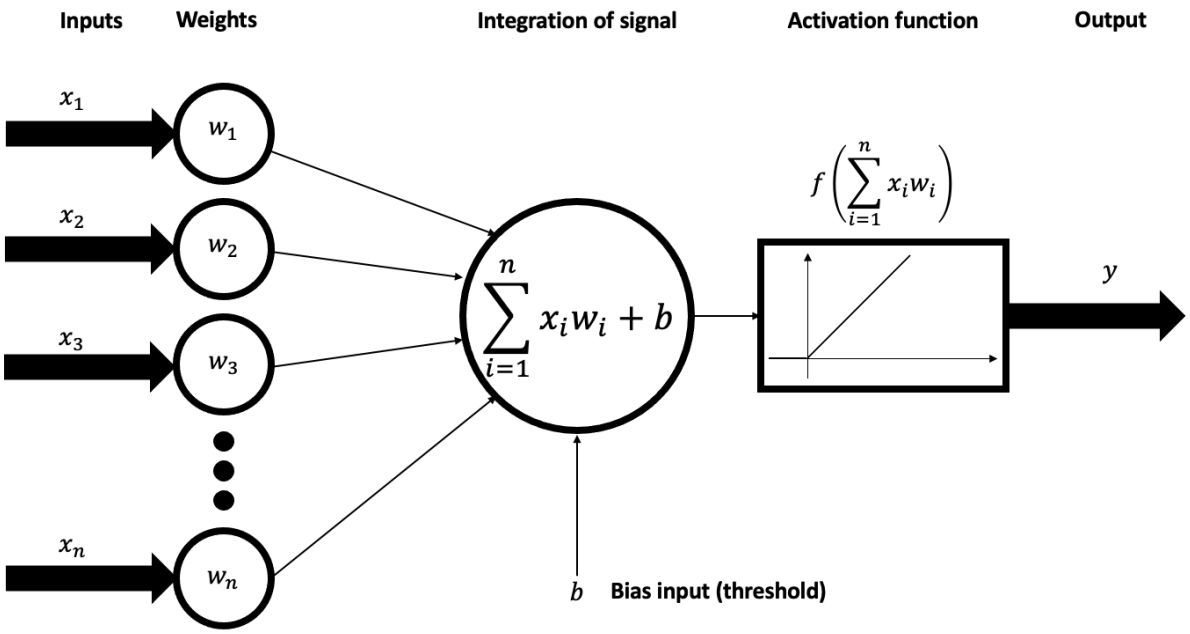
Základní komponenty neuronových sítí hrají zásadní roli ve funkcionalitě vytvořených modelů. Neurony a vrstvy formují strukturu sítě, zatímco aktivační funkce umožňují modelování nelineárních vztahů, což je nezbytné pro zpracování složitých vzorů a funkcí. Ztrátové funkce poskytují metriku pro hodnocení úspěšnosti sítě v předpovídání a jsou klíčové pro navigaci procesu učení. Optimalizační algoritmy řídí tento učící se proces tím, že iterativně upravují interní parametry sítě, aby se minimalizovala celková ztráta a zvýšila přesnost modelu. Každá z těchto komponent společně zajišťuje, že neuronové sítě se mohou učit efektivně, adaptovat se a poskytovat přesné predikce při řešení zadaných úloh. [14, 16]

Základy konvolučního zpracování v CNN

Jak již bylo zmíněno výše, konvoluční neuronové sítě (CNN) jsou specializovaným druhem neuronových sítí, které se používají především pro zpracování strukturovaných dat, jako jsou například obrazová data. CNN jsou známé svou schopností zachytit prostorové a časové závislosti v obraze

[3]Softmax funkce se často používá jako aktivační funkce v poslední vrstvě neuronové sítě při klasifikačních úlohách. Tato funkce převádí výstupní hodnoty neuronů na pravděpodobnostní rozdělení mezi různými třídami. Matematický vzorec pro softmax funkci je definován následovně: $\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$

[4]Mizející gradienty jsou problém, kdy gradient ztrátové funkce klesá exponenciálně rychle s každou vrstvou v síti, což způsobuje, že váhy v raných vrstvách se téměř neaktualizují. Tento jev výrazně zpomaluje trénink a může způsobit, že síť nedokáže efektivně učit složité vzory.



Obrázek 1.2.1.: Ilustrace neuronové jednotky, ukazující základní komponenty jako vstup, váhy, bias, aktivační funkci a výstup. (převzato z [15] pod licencí CC BY-SA 4.0)

pomocí aplikace příslušných filtrů. Úloha konvolučních operací je jádrem toho, co činí CNN tak efektivními pro úlohy, jako je rozpoznávání obrazu, klasifikace obrazu a detekce objektů.

Operace konvoluce zahrnuje posunutí jádra (kernelu) přes vstupní data (obvykle obraz) a výpočet skalárního součinu jádra s překrývajícími se části obrazu. Výsledek tvoří jednu položku ve výstupní mapě příznaků. Tento proces se opakuje posouváním jádra po všech pozicích vstupní matice. Mezi klíčové aspekty konvoluce patří:

- **Jádro konvoluce** Malé matice používané k detekci specifických prvků, jako jsou hrany nebo textury ve vstupním obraze. Každý filtr je navržen tak, aby zachytil určitý aspekt vstupních dat.
- **Krok jádra** Počet pixelů, o které posouváme filtr po vstupním obraze. To ovlivňuje velikost výstupní mapy prvků.
- **Padding** Existuje několik možností, jak umožnit jádru brát v potaz i hraniční prvky. Jednou z možnosti je přidání vrstev nul kolem vstupního obrazu. Další možností je použití zrcadlení (mirror padding), kde se okrajové prvky zrcadlově opakují, což pomáhá jádru zahrnout informace z okrajů obrazu bez ztráty dat.
- **Mapa příznaků** Výstup operace konvoluce se nazývá mapa příznaků. Představuje odezvu filtru v každé prostorové poloze. Použitím různých filtrů může síť CNN zachytit širokou škálu aspektů vstupního obrazu.

Dalším důležitým konceptem v CNN je poolování, které se obvykle používá po operaci konvoluce. Nejběžnější formou sdružování je tzv. max pooling, kdy výstupem sdružovacího jádra je maximální hodnota z části mapy, kterou pokrývá. Tato operace snižuje dimenzionalitu mapy příznaků a umožňuje síti být odolnou vůči malým změnám vstupního obrazu.

V hlubokých konvolučních neuronových sítích se vrství několik konvolučních a sdružovacích vrstev za sebou, což síti umožňuje učit se hierarchické rysy. Nižší vrstvy se mohou naučit rozpoznávat jednoduché rysy, jako jsou hrany, zatímco hlubší vrstvy mohou interpretovat složité aspekty, jako jsou objekty.

Konvoluce je základní operací v CNN, která umožňuje extrakci smysluplných rysů ze vstupních dat. Díky vrstvení a kombinaci konvolučních, aktivačních a sdružovacích vrstev se CNN mohou efektivně naučit hierarchii rysů, která je klíčová pro zvládání složitých úloh vizuálního rozpoznávání. Každá složka, od výběru filtrů až po použití poolingu, hraje rozhodující roli při definování schopnosti sítě přesně interpretovat a klasifikovat vstupní data. [14, 16]

Autoenkodéry v oblasti zpracování obrazu

Autoenkodéry jsou samostatnou třídou modelů neuronových sítí určených především k učení efektivního kódování dat. Tyto sítě toho dosahují komprimací vstupu do méně rozměrné reprezentace latentního prostoru a následnou rekonstrukcí výstupu z této komprimované podoby. Tento duální proces kódování a následného dekódování neslouží pouze ke kompresi dat, ale má zásadní význam pro to, aby síť byla nucena upřednostňovat a zachycovat nejvíce informativní rysy dat.

Struktura autoenkodérů

Autoenkodéry se dělí na tři hlavní části:

- **Enkodér** Proces kódování zahrnuje transformaci vstupních dat s vysokou dimenzí do komprimované podoby. Tento krok je klíčový, protože umožňuje modelu naučit se podstatu dat odfiltrováním šumu a redundance. Tím se síť naučí reprezentaci, která zachycuje základní vzory a struktury nezbytné pro rekonstrukci vstupních dat. Enkodér přebírá vstupní vektor x a převede jej na komprimovanou reprezentaci z, často označovaný jako latentní prostor:

$$z = \sigma(W_e \cdot x + b_e) \tag{1.17}$$

  kde: W_e = matice vah enkodéru.
  b_e = bias vektor enkodéru.
  σ = nelineární aktivační funkce.

- **Latentní prostor (bottleneck)** Srdcem autoenkodéru je jeho latentní prostor, v němž jsou vstupní data reprezentována ve své nejkomprimovanější podobě. Tato reprezentace poskytuje pohled na data v nižší dimenzi a zdůrazňuje jejich nejdůležitější atributy. Rozměrnost a konfigurace latentního prostoru jsou velmi důležité a ovlivňují, jak efektivně dokáže autokodér komprimovat a rekonstruovat data.

- **Dekodér** Proces dekódování se snaží co nejpřesněji rekonstruovat původní vstup z komprimované zakódované reprezentace. V této fázi se testuje kvalita naučených rysů a účinnost

kódování. Cílem není dosáhnout dokonalé rekonstrukce, ale spíše vytvořit aproximaci, která zachová nejvýznamnější atributy původních dat. Tato schopnost je užitečná zejména v aplikacích, kde je pochopení základní struktury dat cennější než reprodukce přesné kopie. Dekódovací část rekonstruuje vstupní data x' z latentní reprezentace z.

Autoenkodéry se používají v celé řadě aplikací, které přesahují rámec prosté komprese dat. Například při redukci šumu se naučí ignorovat náhodné odchylky ve vstupních datech a místo toho se soustředí na rekonstrukci smysluplného základního vzoru obrazu. Používají se také při detekci anomálií, kdy se naučí dobře reprodukovat typické případy dat, ale mají problémy s reprodukcí odlehlých hodnot, a tím identifikují anomální data. V oblasti zpracování obrazu se autoenkodéry také využívají pro segmentaci obrazu a extrakci rysů. Tyto schopnosti jsou klíčové pro rozpoznávání objektů a detailní analýzu obrazového obsahu. [17, 18]

1.3. Obecné postupy augmentace dat

Augmentace dat neboli rozšiřování dat je klíčovou technikou v oblasti strojového učení, zejména v oblastech zpracování obrazu a počítačového vidění. Zahrnuje umělé rozšíření velikosti a rozmanitosti datové sady použitím řady transformací, které jsou aplikovány na vstupní data. Tato sekce se bude zabývat konceptem augmentace dat, důvody použití se zdůrazněním na přínos i omezení při aplikace augmentace a v neposlední řadě budou popsány jedny z nejpoužívanějších technik rozšiřování dat.

Augmentace dat

Augmentace dat je mocný nástroj používaný ve strojovém učení ke zvětšení velikosti a rozmanitosti datových sad, což je zvláště výhodné při trénování modelů hlubokého učení. Podstata augmentace dat spočívá v aplikaci řady deterministických nebo náhodných transformací na trénovací vzorky s cílem vytvořit nová, rozmanitá data. Tento proces pomáhá simulovat variabilitu, s níž se model může setkat v reálných scénářích, a tím zlepšuje jeho zobecňovací schopnosti.

Rozšiřování dat zahrnuje několik kroků a technik, z nichž každá má za cíl upravit původní data tak, aby odrážela možné odchylky v reálném světě, aniž by se změnilo vlastní označení nebo význam dat. V případě obrázků to mohou být geometrické transformace, rozšíření barevného prostoru, jádrové filtry nebo zamíchání obrázků. Rozšířená data jsou pak vložena do procesu trénování spolu s původními daty. Takto rozšířená sada dat poskytuje modelu ucelenější sadu příkladů, z nichž se může učit, a pokrývá tak širší škálu variant, než jakou by mohla nabídnout samotná původní sada dat. Augmentace dat může být iterativní proces, při kterém se v různých trénovacích epochách používají různé kombinace transformací. Toto neustálé zavádění variací může zabránit tomu, aby se modely příliš zaměřovaly na konkrétní variace trénovacích dat a tím se zabraňovalo možnému přeučený trénovaného modelu[5]. [19, 20]

[5]Přeučení, nebo "overfitting", je jev, kdy model strojového učení příliš přesně odpovídá omezené sadě trénovacích

Přínos a omezení při rozšiřování dat

Rozšiřování dat může být dvousečná zbraň při využití ve strojovém učení, zejména v úlohách zahrnujících složité typy dat, jako jsou obrázky. Může sice významně zvýšit schopnost modelu schopnost zobecňovat na základě trénovacích dat, ale pokud není výběr augmentačních metod promyšlený, představuje také riziko. Tato podsekce se zabývá výhodami a omezeními rozšiřování dat a zdůrazňuje důležitost výběru vhodných strategií rozšiřování pro konkrétní úlohy.

Benefity rozšiřování dat

Zavedení realistických variací do trénovacích dat umožňuje modelům lépe se přizpůsobit a efektivně fungovat v široké škále scénářů, které mohou přesahovat podmínky zastoupené v původní datové sadě. Tato schopnost je nezbytná zejména v oblastech, kde je obtížné nebo finančně náročné získat rozmanité a rozsáhlé datové sady. Například ve zdravotnictví nebo v autonomních vozidlech, kde musí modely rozpoznávat a reagovat na nečekané situace, které nebyly přímo součástí trénovacích dat. Augmentace dat také zvyšuje efektivní velikost datové sady, což je klíčové pro minimalizaci přeučení modelu. Tím, že model není vystaven pouze omezenému počtu vzorů, se snižuje riziko, že by se model příliš zaměřil na irelevantní detaily, které nejsou užitečné pro generalizaci na nová data. Dále techniky jako vnášení šumu, ořezávání a rotace přispívají k zvýšení odolnosti modelu vůči různým změnám a zkreslením vstupů, což je kritické pro aplikace v reálném světě, kde jsou vstupní data často nepředvídatelná a různorodá. [19, 20]

Omezení rozšiřování dat

Přestože augmentace dat nabízí řadu výhod, je důležité přistupovat k jejímu využití opatrně. Nevhodně zvolené metody rozšiřování dat mohou vést k zavádějícím nebo irelevantním změnám ve vstupních datech, což může způsobit, že se modely naučí nesprávné vzorce a budou generovat chybné predikce na nových datech. Kromě předchozích popsaných omezení může zvětšení datové sady vést ke zvýšení výpočetní náročnosti, zejména ve fázích trénování modelů. Složité transformace a generování velkého objemu augmentovaných dat vyžadují výkonnější hardware a delší trénovací časy, což může být v některých situacích nepraktické nebo finančně náročné.

Například v projektu zaměřeném na detekci a klasifikaci vozidel v městské dopravě pomocí hlubokého učení byl učiněn pokus o zvětšení souboru dat použitím náhodného škálování snímků. Záměrem bylo zajistit, aby byl model odolný vůči změnám velikosti vozidla v důsledku vzdálenosti od kamery. Zvolený faktor škálování byl však příliš agresivní, neboť některá vozidla byla zmenšena na velikost, která neodpovídá reálným scénářům nebo byla zvětšena tak, že se nerealisticky překrývala s jinými objekty nebo hranicemi scény. Tato nesprávná aplikace vedla k několika problémům:

dat, což způsobuje, že má potíže s generalizací na nová, neviděná data.

- **Zmatení modelu** Model začal špatně klasifikovat malá vozidla, jako jsou motocykly či kola ale také příliš velká vozidla, jako jsou autobusy nebo nákladní vozidla, protože měřítko těchto vozidel v augmentovaných obrázcích nesouhlasilo s typickými měřítky pozorovanými v reálných dopravních podmínkách.
- **Prostorová nekonzistence** Přehnané měřítko narušilo prostorové vztahy a proporce ve scéně, které jsou klíčové pro kontextové porozumění v úlohách detekce objektů.
- **Neefektivnost tréninku** Model měl problémy během tréninku konvergovat, protože nadměrná variabilita ve velikostech objektů zavedená augmentací vedla k odchýlení při učení správných rysů, které definují různé typy vozidel ve standardních měřítcích.

Tento příklad zdůrazňuje důležitost aplikace transformací, které odrážejí realistické variace. Augmentace dat by měla směřovat k napodobení skutečných variací, se kterými se model setkává v praxi, místo zavádění nepravděpodobných nebo extrémních podmínek, které nepomáhají při smysluplném učení. [19, 20]

Zhodnocení použití rozšiřování dat

Výběr správných technik rozšiřování dat zahrnuje pochopení požadavků specifických pro daný úkol a potenciálních dopadů jednotlivých technik. Například úlohy, které vyžadují přesné prostorové povědomí, jako je analýza lékařských snímků nebo rozpoznávání obličejů, mohou být negativně ovlivněny agresivními geometrickými transformacemi, jako je rotace nebo převrácení podle osy. Naopak barevné variace a náhodné ořezávání mohou být vhodnější, aniž by byla ohrožena integrita vlastností objektu.

Závěrem lze říci, že ačkoli je rozšiřování dat neocenitelným nástrojem při vytváření výkonných modelů strojového učení, vyžaduje pečlivou implementaci, aby se zabránilo vnášení chyb a neefektivity do procesu trénování. Pochopení povahy úlohy a dat je klíčové pro plné využití potenciálu rozšiřování dat a zároveň pro zmírnění jeho rizik.

Přehled metod augmentace dat

V oblasti rozšiřování dat byly vyvinuty různé techniky pro zvýšení robustnosti a zobecnitelnosti modelů strojového učení. Tyto techniky lze obecně rozdělit do několika skupin podle typu transformace, kterou používají.

Geometrické transformace

Geometrické transformace jsou jednou z nejběžnějších metod augmentace dat. Patří sem rotace, zrcadlení, translace a škálování. Rotace obrazu kolem jeho osy umožňuje modelům se naučit zvládat různé úhly pohledu na objekty, což je užitečné zejména pro úkoly, jako je klasifikace a detekce objektů. Zrcadlení a převrácení obrazu po ose může rozšířit datovou sadu a zlepšit schopnost

modelů generalizovat. Translace umožňuje posunutí objektů v obraze, což může simulovat různé pozice objektů v reálném světě. Škálování umožňuje změnu velikosti objektů v obraze, což je užitečné pro modely, které musí pracovat s různými rozměry objektů. [20]

Transformace na úrovni pixelů

Transformace na úrovni pixelů zahrnují různé operace, které ovlivňují intenzitu a distribuci pixelů v obraze. Do této kategorie patří změna jasu a kontrastu, přidávání šumu, rozmazání a ostření masky. Změna jasu a kontrastu umožňuje modelům se naučit lépe se přizpůsobit různým podmínkám osvětlení. Přidávání šumu může zlepšit robustnost modelů vůči šumu ve vstupních datech a pomoci předejít přeučení. Rozmazání a ostření masky mohou být použity k manipulaci s hranami a texturami v obraze, což může pomoci modelům lépe identifikovat klíčové rysy objektů. [20]

Ořezávání obrazu

Další techniky augmentace dat zahrnují ořezávání obrazu, které se provádí ořízntutím náhodných obdélníkových oblastí v obraze. Tato metoda není jen o redukci velikosti a tvarů obrázků, ale jedná se o strategii, která nutí modely rozpoznat a rekonstruovat chybějící části, čímž se zlepšuje jejich schopnost generalizace. Náhodné ořezávání může modelům pomoci lépe se zaměřit na důležité části obrazu a zlepšit jejich detekční a klasifikační schopnosti tím, že je vyzývá k identifikaci a doplnění chybějících informací. Tato technika je zvláště účinná pro zvýšení odolnosti modelu vůči částečně zakrytým nebo neúplným vstupům, což je běžné v reálných scénářích, kde nemusí být objekty zcela viditelné. [20]

2. Měření vlivu předzpracování obrazu na výkonost neuronové sítě U-NET

V oblasti zpracování rentgenových snímků je výkon modelů hlubokého učení výrazně ovlivněn kvalitou a množstvím dat, na kterých jsou tyto modely trénovány. Následující kapitola se bude zabývat vlivem předzpracování rentgenových snímků a augmentace dat na účinnost segmentace pomocí modelů sítě U-Net jak pro binární, tak pro vícetřídní úlohy.

Kroky předzpracování, jako je normalizace, změna velikosti a odstraňování šumu, jsou zásadní pro zlepšení kvality vstupních dat a jejich přípravu pro trénování neuronových sítí. Augmentace dat, zahrnující techniky jako rotace, překlápění a škálování, dále rozšiřuje datovou sadu a umožňuje modelům lépe se zobecnit učením se z různých variací vstupních dat. Dále bude v kapitole prozkoumáno použití pokročilých neuronových sítí, detaily tréninkových a validačních procesů U-Net a metodologie použité k měření dopadu těchto technik. Výsledky z těchto experimentů jsou poté analyzovány, aby poskytly přehled o optimalizaci modelů U-Net pro lepší segmentační výkon v medicínských obrazových úlohách.

2.1. Datové sady a jejich rozbor

V této sekci budou představeny datové sady, na kterých bude měřen vliv předzpracování a augmentace dat na výkonnost modelů U-Net pro segmentaci lékařských rentgenových snímků. Pro účely této práce jsem zvolil dvě různé datové sady: jednu pro segmentaci plic a druhou pro segmentaci zubů. Tyto datové sady jsem vybral na základě jejich relevance pro typické úlohy lékařského pozorování a jejich různé úrovně obtížnosti, což umožňuje komplexně vyhodnotit robustnost a účinnost navržených technik pro předzpracování obrazu.

Datová sada pro segmentaci plic

Tato datová sada je složena ze dvou různých zdrojů, z nichž každá přispívá svou jedinečností a obohacuje tak celkovou různorodost dat.

- **První zdroj:** obsahuje 800 rentgenů hrudníku ve velmi vysoké kvalitě včetně odpovídajících segmentačních masek. Obrázky jsou dobře anotovány a poskytují jasné a přesné zobrazení

plic, což je ideální pro přesnou segmentaci. Tato datová sada je dostupná na stránkách Kaggle [1]."

- **Druhý zdroj:** zahrnuje přes 6 000 rentgenů hrudníku v nižší kvalitě ve srovnání s prvním zdrojem. Navzdory snížené kvalitě obrázků jsou tyto snímky stále vhodné pro úlohy segmentace plic a významně zvyšují velikost datové sady, čímž poskytují rozmanitější tréninková data. Tato datová sada je dostupná na stránkách Mendeley. [23]

Spojením těchto dvou datových sad bude docíleno rozmanitosti snímků využitím jak velmi kvalitních z první sady, tak méně kvalitních z druhé datové sady. Tím bude umožněno modelům sítě U-Net se přizpůsobit i na nečekané vstupy a zvýší se jejich schopnost generalizace. Vytvoříme tak komplexní datovou sadu, která bude efektivně sloužit k trénování a validaci U-Net modelů. Celkově se bude v sadě nacházet 5000 vybraných snímků hrudníku s přiloženými segmentačními maskami.

Datová sada pro segmentaci zubů

Jelikož je segmentace plic jednoduchý binární problém, bylo nutné zvolit si i složitější úlohu pro správně posouzení dopadu předzpracování a augmentace dat na síť U-Net. Vzhledem k mým předchozím zkušenostem a vysoké náročnosti segmentace jsem si jako další datovou sadu zvolil rentgenové snímky zubů. Tato datová sada je dostupná na stránkách Kaggle. [24]

Náročnost segmentace zubů

Segmentace zubů je velmi nárnočný problém z několika důvodů:

- Jedná se o vícetřídní segmentaci, neboť je nutné segmentovat až 32 různých zubů (v případě anomálií i více).
- Zuby mohou mít velmi různorodé tvary a nemusí být vždy dobře viditelné.
- Zuby postihují různá onemocnění, která mění jejich tvar nebo nebo způsobují jejich ztrátu. Je nutné taková onemocnění léčit například opravou zubů ve formě výplní, umělých korunek, endodoncie, rovnátek aj.

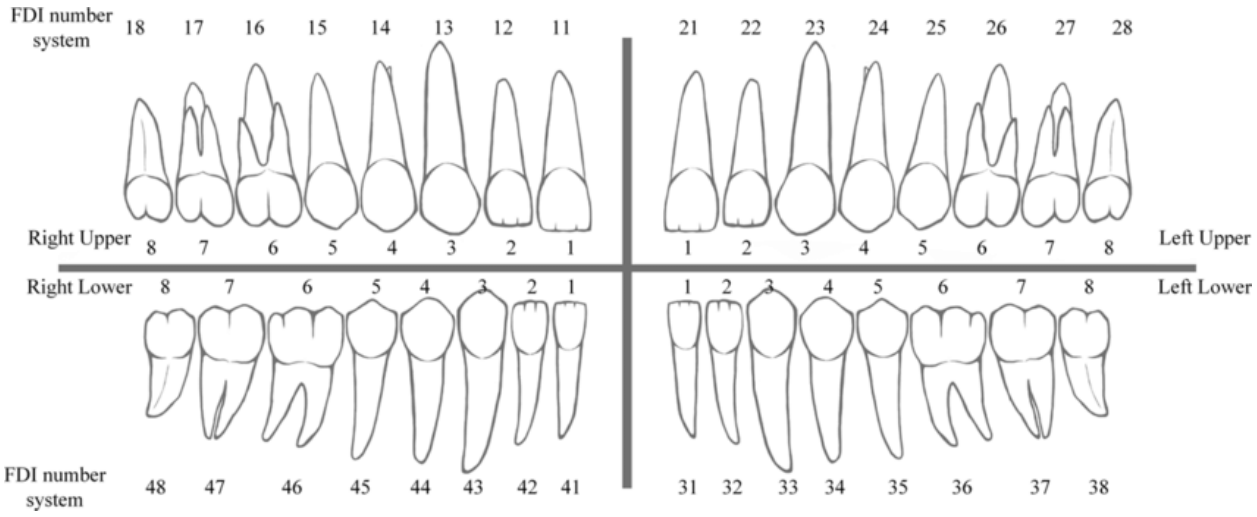
Všechny tyto faktory přidávají další variabilitu a složitost pro trénovaný model. Právě z těchto důvodů by měl být velmi patrný vliv předzpracování a augmentace na trénování a výkonnost modelů.

Popis datové sady

Datová sada zahrnuje téměř 600 rentgenových snímků zubů, z čehoz bude použito 500 vybraných snímků s jejich segmentačními maskami. Tyto snímky jsou v nižší kvalitě ve srovnání s rentgeny

[1] Pro více informací k datové sadě viz také články [21, 22].

plic. Snímky jsou anotovány jak pro dětské, tak i pro dospělé zuby, přičemž je použito stejné označení pro oba typy zubů, tedy například dospělý zub 11 má stejnou značku jako dětský zub 11. I když toto označení není ideální, pro naše účely nepředstavuje problém.



Obrázek 2.1.1.: Úkázka jednotlivých zubů s příslušným označením v podobě čísel jdoucím po směru hodinových ručiček. (převzato z [25])

Pro lepší orientaci je ústní dutina rozdělena do čtyř kvadrantů a zuby jsou označeny ve směru hodinových ručiček dvojčíslím, kde první číslo označuje číslo zubu a druhé kvadrant, ve kterém se zub nachází viz 2.1.1. Toto označení se zubů nazývá FDI a zahrnuje všechny důležité informace, což usnadňuje přesné a konzistentní anotování.

Aplikace šumu na snímky v datové sadě

Následující sekce je soustředěna na přidávání šumu do snímků ve vytvořené datové sadě, aby bylo možné vytvořit trénovací a validační sady pro vyhodnocování algoritmů pro redukci šumu. V první kapitole byly popsány nejčastěji se vyskytující typy šumu ve snímcích. V této práci se zaměřím na redukci tří nejčastějších typů šumů v retgenových snímcích: speckle šum, impulzní šum a Poissonův šum [26].

Ve vytvořené datové sadě se nachází 5000 rentgenů hrudníku a 500 rentgenů zubů. Na snímky budou aplikovány zmíněné typy šumů následovně:

Tabulka 2.1.: Detailní popis aplikace šumu na snímky v datové sadě

Část datasetu (%)	**Typ aplikovaného šumu**
0-20	Na snímky v tomto rozsahu bude aplikován šum typu speckle.
20-40	Další část bude ovlivněna typem šumu soli a pepře.
40-60	Tento rozsah bude zašuměn Poissonův šumem.
60-80	Zde bude aplikována náhodná kombinace dvou typů šumů.
80-100	Poslední část snímků bude narušena kombinací všech tří zmíněných šumů. Tyto snímky budou představovat největší míru komplexity šumu.

Tento postup umožňuje vytvoření různých úrovní obtížnosti při odstraňování šumu, což je klíčové pro vývoj a vyhodnocení robustních modelů pro redukci šumu.

Nejprve jsem aplikoval jednotlivé typy šumů samostatně na části datasetu (první tři řádky v tabulce 2.1), což vytváří relativně jednoduše odstranitelné šumové artefakty. Tímto způsobem můžu zhodnotit základní schopnosti algoritmů pro redukci šumu efektivně odstraňovat každý typ šumu zvlášť. Čtvrtý řádek v tabulce 2.1 představuje střední úroveň obtížnosti. Tento krok testuje algoritmy v situacích, kde musí řešit více šumových podmínek současně, což simuluje složitější a realističtější scénáře, jaké mohou nastat v praxi. Poslední řádek tabulky 2.1 simuluje nejnáročnější scénáře pro vytvořené algoritmy, které musí být schopny odstranit kombinaci různých typů šumů. Tyto snímky jsou klíčové pro testování a vývoj velmi robustních modelů, které mohou efektivně fungovat v nejrůznějších reálných situacích.

Tento stratifikovaný přístup k aplikaci šumu nejen zajišťuje důkladné testování algoritmů pro redukci šumu ve snímcích, ale také pomáhá při vývoji a ladění modelů, které jsou schopny se přizpůsobit různým úrovním šumové zátěže. Tímto způsobem dosáhnu komplexního hodnocení a vývoje efektivních technik pro zpracování rentgenových snímků v různých podmínkách.

Měření hodnoty šumu ve snímcích

Pro aplikaci šumu na snímky jsem využil knihovnu `numpy` [27], se kterou jsem implementoval funkce pro přidání speckle šumu, šumu soli a pepře a Poissonova šumu. Hlavním parametrem těchto funkcí pro regulaci šumu je nastavení intenzity, nikoli nastavení směrodatné odchylky (std). Abych mohl porovnat úroveň šumu ve snímcích, naprogramoval jsem funkci pro odhad směrodatné odchylky šumu ve snímku.

Pro aplikaci šumu na snímky jsem využil knihovnu `numpy`, se kterou jsem implementoval funkce pro přidání speckle šumu, šumu soli a pepře a Poissonova šumu. Hlavním parametrem těchto funkcí je intenzita šumu, nikoli jeho směrodatná odchylka (std). Použití intenzity jako parametru je praktické pro aplikaci různých úrovní šumu, ale pro analýzu a srovnání je často nutné znát skutečnou směrodatnou odchylku šumu v obraze. Proto jsem vyvinul následující funkci, která

odhaduje směrodatnou odchylku šumu v obraze pomocí extrakce segmentů (anglicky "patches") a analýzy hlavních komponent (PCA). Následující kód ukazuje implementaci zmíněné funkce[2].

Ukázka kódu 2.1: Funkce pro odhad směrodatné odchylky šumu ve snímku

```python
import numpy as np
from sklearn.feature_extraction.image import extract_patches_2d
from sklearn.decomposition import PCA
from typing import Tuple

def estimate_noise_in_image(
    image: np.ndarray, patch_size: Tuple[int, int] = (7, 7),
       num_patches: int = 1000,
    lower_percentile: float = 0.1, upper_percentile: float = 0.2
) -> tuple:
    # Extrahuje segmenty
    patches = extract_patches_2d(image, patch_size, max_patches=
       num_patches)

    # Přetvoří segmenty na 2D pole, kde každý řádek je segment
    patches = patches.reshape(patches.shape[0], -1).astype(np.
       float32)

    # Odebere průměr
    patches -= np.mean(patches, axis=1, keepdims=True)

    # PCA pro nalezení vlastních čísel
    pca = PCA(n_components=patch_size[0]*patch_size[1])
    pca.fit(patches)

    # Vypočítá indexy pro požadovaný percentilní v rozsahu vlastních čísel
    lower_index = int(lower_percentile * len(pca.
       explained_variance_))
    upper_index = int(upper_percentile * len(pca.
       explained_variance_))

    # Medián vlastních čísel ve specifikovaném percentilním rozsahu
    noise_variance = np.median(pca.explained_variance_[lower_index:
       upper_index])

    return np.sqrt(pca.explained_variance_), np.sqrt(noise_variance
       )
```

[2]Kompletní implementace funkcí pro přidání šumu lze nalézt v GitHub repozitáři této práce.

Tato funkce mi umožňuje kvantifikovat a porovnávat úrovně šumu a pomocí těchto odhadů mohu lépe analyzovat výkon algoritmů na různých úrovních šumu.

2.2. Návrh metod vhodných pro předzpracování rentgenových snímků

V této části se budu zabývat charakteristikou použitých knihoven a technologií, nastavením projektu pomocí nástroje Poetry a implementací různých standardních metod používaných pro odstranění šumu a zlepšení kvality obrazu. Zatímco teoretické pozadí a principy těchto metod byly popsány v teoretické části této práce, následující část se zaměřuje na praktické aspekty jejich implementace, včetně jakýchkoliv dalších algoritmů, které dosud nebyly zmíněny. Dále vysvětlím použití tzv. Ensemble Averaging postupu, který napomáhá ke zlepšení robustnosti algoritmů pro redukci šumu.

Struktura projektu, použité knihovny a technologie

Pro tento projekt jsem zvolil nástroj Poetry, který umožňuje snadnou správu závislostí a vytváření virtuálních prostředí. Poetry poskytuje pohodlný způsob, jak udržovat konzistenci balíčků a odděluje závislosti projektu od systémových knihoven. To usnadňuje sdílení projektu a zajišťuje, že všechny potřebné knihovny budou správně nainstalovány.

Instalace a konfigurace Poetry

Instalace nástroje Poetry začíná instalací pipx, což je nástroj umožňující instalaci a izolaci Python aplikací.

Ukázka kódu 2.2: Instalace pipx

```
python -m pip install pipx
```

Poté pomocí pipx je možné nainstalovat nástroj Poetry:

Ukázka kódu 2.3: Instalace Poetry

```
pipx install poetry
```

Po instalaci Poetry je nutné se přesunout do kořenového adresáře projektu a nainstalovat všechny závislosti definované v souboru pyproject.toml pomocí následujícího příkazu:

Ukázka kódu 2.4: Instalace virtuálního prostředí a knihoven

```
cd cesta/k/projektu
poetry install
```

Použité knihovny

V tomto projektu jsem použil několik důležitých knihoven. Nejvýznamnější z nich je **PyTorch Lightning**, která usnadňuje implementaci neuronových sítí a byla vybrána pro snadné a přehledné trénování modelů. Další důležité knihovny zahrnují již dříve zmíněný NumPy pro numerické výpočty, scikit-image pro zpracování snímků a Matplotlib pro vizualizaci dat. Následující tabulka ukazuje kompletní přehled použitých závislostí v této práci:

Knihovna	**Popis**
python	Základní programovací jazyk pro tento projekt.
torch	Hlavní knihovna pro implementaci neuronových sítí.
torchaudio	Knihovna pro zpracování audio dat v PyTorch.
torchvision	Knihovna pro zpracování obrazových dat v PyTorch.
lightning	Framework pro snadné trénování modelů v PyTorch.
pillow	Knihovna pro práci s obrazovými daty.
pandas	Knihovna pro manipulaci s daty a jejich analýzu.
pydantic	Knihovna pro validaci dat.
opencv-python	Knihovna pro práci s obrazovými daty a předzpracování.
torchmetrics	Knihovna pro výpočet metrik pro hodnocení modelů.
matplotlib	Knihovna pro vizualizaci dat.
mlflow	Nástroj pro sledování experimentů a modelů.
scikit-image	Knihovna pro zpracování snímků.
rich	Knihovna pro pokročilé logování.
pywavelets	Knihovna pro práci s waveletovou transformací.
numba	Knihovna pro akceleraci numerických výpočtů.

Tabulka 2.2.: Přehled použitých knihoven

Struktura projektu

Projekt je strukturován tak, aby umožňoval efektivní správu kódu a snadnou navigaci mezi různými částmi. V kořenovém adresáři projektu se nachází hlavní soubor **main.py**, který zahrnuje všechny funkce pro trénování, testování a tvorbu datasetů. Složka **common** obsahuje skripty pro vizualizaci, tvorbu datasetů a základní struktury tříd. Konfigurační soubory pro různé experimenty a nastavení filtrů jsou umístěny ve složce **configs**. Předzpracování dat, jak pomocí standardních metod, tak pomocí neuronových sítí, se nachází ve složce **preprocessing**. Implementace U-Net modelu a související skripty jsou umístěny ve složce **unet**. Tento strukturovaný přístup umožňuje

udržovat kód přehledný a organizovaný, což je klíčové pro efektivní vývoj.

Vybrané metody a jejich implementace

V této práci jsem se zaměřil na implementaci vybraných standardních metod pro odstranění šumu a zvýšení kvality obrazu. V následujícím seznamu jsou uvedeny použité metody spolu s jejich krátkým popisem:

- **CLAHE:** Zvyšuje lokální kontrast a zabraňuje nadměrnému zesílení šumu.
- **Mediánový Filtr:** Odstraňuje impulzní šum nahrazením každého pixelu mediánem jeho okolí.
- **Bilaterální Filtr:** Zachovává hrany při redukci šumu zohledněním prostorových i intenzitních rozdílů.
- **Filtr ne-lokálních průměrů:** Redukuje šum průměrováním podobných segmentů v celém obraze.
- **Lokálně adaptivní mediánový filtr:** Adaptivní medianový filtr, který přizpůsobuje prahovou hodnotu šumu na základě místní variance.
- **Redukce šumu pomocí waveletové transformace:** Rozkládá obraz na waveletové koeficienty a odstraňuje šum pomocí prahování.

Většina uvedených metod byla implementována s využitím knihoven `scipy` [28] a `skimage` [29], které poskytují optimalizované a efektivní řešení. Vzhledem k tomu, že dataset obsahuje 5500 snímků, je výpočetní efektivita klíčová pro zpracování v rozumném čase.

Některé z metod nebyly ve zmíněných knihovnách dostupné, bylo proto nutné je implementovat svépomocí. Jedná se o adaptivní medianový filtr a Frost filtr. Tyto metody jsem se snažil navrhnout tak, aby byly efektivní a rychlé, zejména Frost filtr bylo nutné implementovat s využitím knihovny `numba` [30], která umožňuje provádět náročné výpočty na grafické kartě, což výrazně zrychlilo zpracování. Níže je uveden příklad implementace Frostova filtru:

Ukázka kódu 2.5: Implementace Frost filtru s využitím Numba CUDA

```
1    @numba.jit(nopython=True)
2    def __compute_weights(
3        patch: npt.NDArray, N: int, initial_K: float, local_mean: np.
            float64, local_std: np.float64
4    ) -> npt.NDArray[np.float64]:
5        center_pixel = patch[N, N]
6        T_t0 = np.abs(center_pixel - local_mean) / local_std if
            local_std != 0 else 0
7        weights = np.zeros_like(patch)
8
9        for i in range(patch.shape[0]):
```

```
            for j in range(patch.shape[1]):
                current_pixel = patch[i, j]
                numerator = np.abs(current_pixel - center_pixel)
                abs_diffs = np.abs(patch - center_pixel)
                abs_diffs[N, N] = 0  # Exclude the center pixel from
                    the sum
                total_pixels = (2 * N + 1) ** 2 - 1
                denominator = np.sum(abs_diffs) / total_pixels
                Q_sh = numerator / denominator if denominator != 0 else
                     0
                K = initial_K * T_t0 * Q_sh
                d_sh = np.sqrt((i - N) ** 2 + (j - N) ** 2)
                weights[i, j] = np.exp(-K * d_sh)

        return weights


    def apply_frost_filter(
        image: npt.NDArray, window_size: int = 3, initial_K: float =
            1.0
    ) -> npt.NDArray:
        def adaptive_kernel(patch: npt.NDArray) -> np.float64:
            patch = patch.reshape(window_size, window_size)
            N = window_size // 2
            local_mean = np.mean(patch)
            local_std = np.std(patch)
            weights = __compute_weights(patch, N, initial_K, local_mean
                , local_std)
            weights /= np.sum(weights)
            return np.sum(weights * patch)

        filtered_image = generic_filter(image, adaptive_kernel, size=(
            window_size, window_size))
        filtered_image_uint8 = np.clip(filtered_image, 0, 255).astype(
            np.uint8)
        return filtered_image_uint8
```

Výzvy a úskalí standardních metod

Standardní metody pro odstranění šumu mají několik omezení, která mohou ovlivnit jejich účinnost v reálných aplikacích. Tato omezení jsou důležitá pro pochopení, proč samotné použití těchto metod nemusí být vždy dostatečné a proč je třeba hledat pokročilejší přístupy.

Většina standardních metod je navržena tak, aby se zaměřovala na specifické typy šumu. Například mediánový filtr je efektivní pro odstranění impulzního šumu, ale nemusí být účinný proti jiným typům. Toto zaměření na specifické typy šumu znamená, že pokud není typ šumu v obraze znám, může být obtížné vybrat správnou metodu pro jeho odstranění. V reálných aplikacích často není možné předem určit, jaký typ šumu bude v obraze přítomen, což výrazně omezuje účinnost těchto metod.

Dalším problémem je přítomnost více typů šumu současně v jednom obraze. Například, lékařské snímky mohou obsahovat kombinaci šumu typu speckle, Poissonova šumu a dalších artefaktů. Standardní metody, které jsou navrženy pro odstranění jednoho typu šumu, nemusí být schopny efektivně zvládnout takovou kombinaci. Použití jedné metody může zlepšit jednu část obrazu, ale zároveň zhoršit jinou.

Tento problém kombinace šumů vede k potřebě metod, které jsou schopny efektivně řešit více typů šumu současně, nebo metod, které kombinují výstupy různých filtrů pro dosažení lepších výsledků. Tato potřeba byla jedním z důvodů, proč jsem se rozhodl implementovat tzv. ensemble averaging.

Ensemble averaging

Vzhledem k omezením standardních metod pro odstranění šumu, jsem se rozhodl implementovat přístup zvaný ensemble averaging. Tento přístup kombinuje výsledky několika různých metod pro odstranění šumu a poskytuje tak robustnější a efektivnější řešení.

Hlavním důvodem pro použití ensemble averaging je překonání omezení jednotlivých metod. Jak již bylo zmíněno, každá standardní metoda má své silné a slabé stránky, a samotné použití jedné metody nemusí být dostatečné pro dosažení optimálních výsledků v reálných scénářích, kde může být přítomno více typů šumu.

Ensemble averaging využívá kombinaci několika metod k dosažení následujících výhod:

- **Robustnost:** Kombinací výsledků z různých metod lze dosáhnout robustnějšího odstranění šumu, které není závislé na jednom specifickém typu šumu.
- **Zlepšení kvality obrazu:** Průměrováním výsledků se minimalizují chyby a artefakty, které mohou vzniknout při použití jednotlivých metod.
- **Přizpůsobivost:** Tento přístup je přizpůsobivý různým typům šumu a jejich kombinacím, což je klíčové pro praktické aplikace.

Implementace ensemble averaging zahrnuje několik kroků. Každý obraz je zpracován několika různými metodami pro odstranění šumu, které jsou aplikovány samostatně, aby se zabránilo vzájemnému ovlivňování. Výstupy těchto metod jsou poté průměrovány za účelem vytvoření finálního obrazu s odstraněným šumem. Níže je ukázka kódu pro implementaci ensemble averaging:

Ukázka kódu 2.6: Implementace Ensemble Averaging metody

```
def apply_standard_preprocessing(
    image: npt.NDArray, transform_type: str, params: Dict | None =
        None
) -> Image.Image:
    preprocessing_method = getattr(standart_preprocessing, f"apply_
        {transform_type}")

    if preprocessing_method and callable(preprocessing_method):
        if params is None:
            return Image.fromarray(preprocessing_method(image))
        else:
            return Image.fromarray(preprocessing_method(image, **
                params))
    else:
        print("No preprocessing was applied!")
        return Image.fromarray(image)


def standard_preprocessing_ensemble_averaging(
    image: npt.NDArray[np.uint8], preprocessing_config: Dict[str,
        Dict[str, Any]]
) -> Image.Image:
    preprocessed_images: List[npt.NDArray[np.float64]] = []

    for method, params in track(
        sequence=preprocessing_config.items(),
        description="Applying preprocessing method...",
        total=len(preprocessing_config),
    ):
        image_copy = np.array(image, copy=True)
        processed_image = apply_standard_preprocessing(image_copy,
            method, params)
        preprocessed_images.append(np.array(processed_image))

    averaged_image = cast(npt.NDArray[np.float64], np.mean(
        preprocessed_images, axis=0))
    return Image.fromarray(averaged_image.astype(np.uint8))
```

Ensemble averaging nabízí několik výhod, které zlepšují kvalitu výsledného obrazu a jeho robustnost. Kombinace výsledků z různých metod pro odstranění šumu vede k lepšímu celkovému odstranění šumu, protože silné stránky jednotlivých metod se navzájem doplňují. Tento přístup zajišťuje, že výsledný obraz je méně náchylný k artefaktům a chybám, které mohou vzniknout při použití pouze jedné metody. Díky tomu je možné dosáhnout vyšší kvality obrazu, a to i v

přítomnosti různých typů šumu.

Nicméně ensemble averaging má také svá omezení. Jedním z hlavních problémů je výpočetní náročnost, protože každá metoda musí být aplikována samostatně na každý obraz, což výrazně zvyšuje dobu zpracování. Dalším omezením je složitost nastavení parametrů jednotlivých metod, protože různé snímky se mohou lišit a metody mohou vykazovat odlišné chování v závislosti na těchto rozdílech. Navíc kvalita výsledného obrazu závisí na výkonnosti jednotlivých metod, pokud tedy některá metoda selže, může to negativně ovlivnit celkový výsledek.

2.3. Návrh a implementace specializovaných modelů neuronových sítí

Následující sekce bude věnována návrhu a implementaci specializovaných modelů neuronových sítí pro odstranění šumu z rentgenových snímků. Sekce bude zahrnovat detailní popis dvou různých modelů neuronových sítí: konvoluční neuronovová síť pro redukci šumu (Denoising convolutional neural network DnCNN) a autoenkodér pro redukci šumu (Denoising autoencoder DAE). Popis modelů bude obsahovat podrobnosti o tom, jak jednotlivé modely fungují a bude také podrobně popsána jejich implementace, s důrazem na výsledky dosažené při odstraňování šumu.

Neuronová síť DnCNN

Konvoluční neuronovová síť pro redukci šumu (DnCNN) je pokročilý model, který se na rozdíl od tradičních metod zaměřuje na předvídání šumu přítomného v obraze, aniž by předvídal celý obraz. Tento nalazený šum je poté odečten od zašuměného vstupu, aby se získal čistý obraz. Tato metoda využívá residuální učení (residual learning), které umožňuje síti naučit se mapování ze zašuměného obrazu na samostatný šum. Koncept a implementace DnCNN byly představeny v článku "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising"[31].

Architektura DnCNN

Architektura DnCNN se skládá z několika konvolučních vrstev propojených aktivačními funkcemi ReLU a normalizačními vrstvami BatchNorm. Počáteční vrstva transformuje vstupní obraz na reprezentaci do vyšší dimenze. Následuje série vrstev, z nichž každá je následována vrstvou BatchNorm a aktivační funkcí ReLU. Tyto mezivrstvy umožňují modelu efektivně extrahovat rysy z obrazu a učit se složité vzory šumu. Poslední vrstvou je konvoluční vrstva, která předpovídá šum v obraze. Síť se může zaměřit na zachycení vysokofrekvenčních složek šumu, jejichž modelování je často náročnější. Tato konstrukce výrazně zlepšuje schopnost sítě odstraňovat z obrazu různé typy šumu, což vede k lepšímu výkonu ve srovnání s běžnými metodami. Níže je implementace této architektury:

Ukázka kódu 2.7: Implementace konvoluční neuronovové sítě pro redukci šumu

```
import torch
import torch.nn as nn
from torchmetrics import MetricCollection
from preprocessing.neural_networks.model import PreprocessingModel

class DnCNN(PreprocessingModel):
    def __init__(
        self,
        depth: int = 17,
        n_channels: int = 64,
        image_channels: int = 1,
        learning_rate: float = 1e-4,
        metrics: MetricCollection | None = None,
    ) -> None:
        super(DnCNN, self).__init__(learning_rate=learning_rate,
            metrics=metrics)
        layers = [
            nn.Conv2d(image_channels, n_channels, kernel_size=3,
                padding=1),
            nn.ReLU(inplace=True),
        ]

        for _ in range(depth - 2):
            layers.append(nn.Conv2d(n_channels, n_channels,
                kernel_size=3, padding=1))
            layers.append(nn.BatchNorm2d(n_channels))
            layers.append(nn.ReLU(inplace=True))

        layers.append(nn.Conv2d(n_channels, image_channels,
            kernel_size=3, padding=1))

        self.dncnn = nn.Sequential(*layers)

        def forward(self, x: torch.Tensor) -> torch.Tensor:
            predicted_noise = self.dncnn(x)
            return x - predicted_noise
```

Tréninkový proces

Tréninkový proces DnCNN zahrnuje několik klíčových kroků, které zajišťují, že se model efektivně naučí předvídat a odstraňovat šum ze snímků. Tento proces využívá ztrátovou funkci střední

kvadratické chyby (MSE) k měření rozdílu mezi předpokládaným šumem a skutečným šumem přítomným ve snímcích.

Níže je implementace třídy PreprocessingModel, která slouží jako základní třída pro modely redukující šum, včetně DnCNN. Tato třída definuje metody pro konfiguraci optimalizátoru, pro učení a testování spolu s metrikami pro ohodnocení modelu.

Ukázka kódu 2.8: Třída pro model redukující šum

```
from abc import ABC, abstractmethod
from typing import Any, Dict
import torch
import torch.nn.functional as F
import torch.optim as optim
from lightning import LightningModule
from torchmetrics import MeanSquaredError, MetricCollection

class PreprocessingModel(LightningModule, ABC):
    def __init__(
        self,
        learning_rate: float = 1e-4,
        metrics: MetricCollection | None = None,
    ) -> None:
        super(PreprocessingModel, self).__init__()
        self.learning_rate = learning_rate
        self.metrics = metrics or MetricCollection({"MSE":
            MeanSquaredError()})
        self.save_hyperparameters(ignore=["metrics"])

    @abstractmethod
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        pass

    def configure_optimizers(self) -> Dict[str, Any]:
        optimizer = optim.Adam(self.parameters(), lr=self.
            learning_rate)
        return {"optimizer": optimizer}

    def training_step(
        self, batch: Any, batch_idx: int
    ) -> torch.Tensor:
        noised_images, target = batch
        denoised_images = self(noised_images)
        loss = F.mse_loss(denoised_images, target)
        self.log("train_loss", loss, on_step=True, prog_bar=True,
```

```
                logger=True)
35              return loss
36
37          def validation_step(
38              self, batch: Any, batch_idx: int
39          ) -> None:
40              noised_images, target = batch
41              denoised_images = self(noised_images)
42
43              loss = F.mse_loss(denoised_images, target)
44              self.log("val_loss", loss, on_epoch=True, prog_bar=True,
                    logger=True)
45              self.log_metrics(denoised_images, target)
46
47          def log_metrics(
48              self, denoised_images, target
49          ) -> None:
50              metrics: Dict[str, Any] = self.metrics(denoised_images,
                    target)
51              for name, value in metrics.items():
52                  self.log(f"val_{name}", value)
```

Tato třída slouží jako základ pro model DnCNN, který tuto třídu zdědí, implementuje specifickou architekturu a dopředný průchod pro predikci šumu.

Návrh autoenkodéru pro redukci šumu

Autoenkodér pro redukci šumu (DAE) je model hlubokého učení navržený k redukci šumu ze snímků předpovídáním celého čistého snímku, nikoli pouze šumu jako v případě modelu DnCNN. Tento přístup umožňuje modelu naučit se komplexní vztahy v daném obraze, čímž může nejen efektivněji redukovat různé typy šumů, ale také vytvořit obraz věrnější svému původnímu zdrojovému snímku. DAE je složitější neuronová síť než DnCNN, protože obsahuje více vrstev a neuronů, což by mělo přispět k lepší schopnosti detekce šumu. Nicméně, vytvoření robustnější sítě znamená, že model bude potřebovat větší množství tréninkových dat. Jejich nedostatek může vést k tomu, že se model začne přizpůsobovat specifikám tréninkových dat a může ztrácet na obecném výkonu.

Architektura DAE

Architektura DAE se skládá ze dvou hlavních součástí: enkodéru a dekodéru. Obě části jsou konstruovány pomocí konvolučních a dekonvolučních vrstev s aktivačními funkcemi ReLU a

batch normalizací. Enkodér i dekodér obsahují každý šest vrstev, přičemž každá vrstva je navržena tak, aby postupně zpracovávala obrazové informace a detekovala složitější vzory ve snímcích.

Enkodér komprimuje vstupní obraz do nižší dimenzionální reprezentace pomocí postupně aplikovaných konvolučních vrstev. Každá vrstva extrahuje stále složitější rysy z obrazových dat. Dekodér rekonstruuje čistý obraz z komprimované reprezentace využitím dekonvolučních vrstev. Vrstvy dekodéru postupně zvětšují rozlišení a kombinují informace z enkodéru pomocí tzv. *skip connections*. Skip connections umožňují přenášet informace z odpovídajících vrstev enkodéru přímo do dekodéru, čímž napomáhají zachovat prostorové detaily a rysy z původního obrazu, které by mohly být ztraceny během komprimace.

Architektura DAE je detailně znázorněna na diagramu 2.3.1. Diagram ilustruje, jak jsou jednotlivé vrstvy enkodéru a dekodéru propojeny a také znázorňuje předávání dodatečných informací prostřednictvím skip connections. Enkodér začíná s konvoluční vrstvou s 64 filtry a postupně zvyšuje počet filtrů až na 512, zatímco dekodér postupně generuje obraz zpět do jeho původního rozlišení pomocí dekonvolučních vrstev. Níže je uvedena konkrétní implementace tříd autoenkodéru pro redukci šumu.

Ukázka kódu 2.9: Implementace autoenkodéru pro redukci šumu

```
import torch
import torch.nn as nn
from lightning import LightningModule
from torchmetrics import MetricCollection
from preprocessing.neural_networks.model import PreprocessingModel

class ConvBlock(LightningModule):
    def __init__(
        self, in_channels: int, out_channels: int, kernel_size: int
            = 3,
        padding: int = 1, stride: int = 1,
    ) -> None:
        super(ConvBlock, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, kernel_size=kernel_size,
                padding=padding, stride=stride,
            ),
            nn.ReLU(inplace=True),
            nn.Dropout(0.2),
            nn.BatchNorm2d(out_channels),
        )

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        return self.conv(x)
```

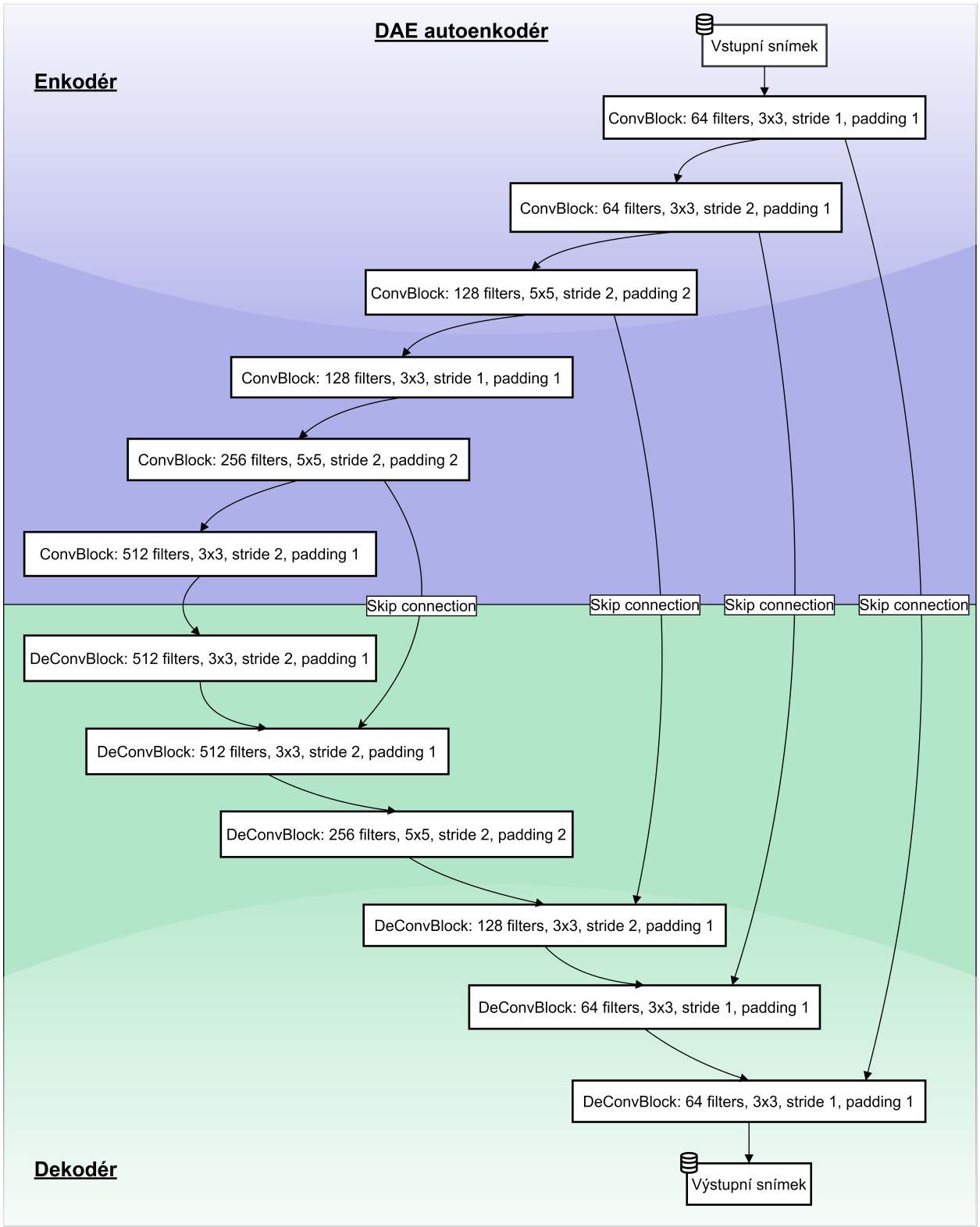
```
25
26      class DeConvBlock(LightningModule):
27          def __init__(
28              self, in_channels: int, out_channels: int, kernel_size: int
                    = 3,
29              stride: int = 1, padding: int = 1, output_padding: int = 1,
30          ) -> None:
31              super(DeConvBlock, self).__init__()
32              self.conv = nn.Sequential(
33                  nn.ConvTranspose2d(
34                      in_channels, out_channels, kernel_size=kernel_size,
                            stride=stride,
35                      padding=padding, output_padding=output_padding,
36                  ),
37                  nn.ReLU(inplace=True),
38                  nn.Dropout(0.2),
39                  nn.BatchNorm2d(out_channels),
40              )
41
42          def forward(self, x: torch.Tensor) -> torch.Tensor:
43              return self.conv(x)
44
45      class DenoisingAutoencoder(PreprocessingModel):
46          def __init__(
47              self, n_channels: int = 1, learning_rate: float = 1e-4,
48              metrics: MetricCollection | None = None,
49          ) -> None:
50              super(DenoisingAutoencoder, self).__init__(
51                  learning_rate=learning_rate, metrics=metrics
52              )
53              self.encoder_layers = nn.ModuleList([
54                  ConvBlock(n_channels, 64, 3, stride=1),
55                  ConvBlock(64, 64, 3, stride=2),
56                  ConvBlock(64, 128, 5, stride=2, padding=2),
57                  ConvBlock(128, 128, 3, stride=1),
58                  ConvBlock(128, 256, 5, stride=2, padding=2),
59                  ConvBlock(256, 512, 3, stride=2),
60              ])
61
62              self.decoder_layers = nn.ModuleList([
63                  DeConvBlock(512, 512, 3, stride=2),
64                  DeConvBlock(512, 256, 3, stride=2),
65                  DeConvBlock(256, 128, 5, stride=2, padding=2),
```

```
66                 DeConvBlock(128, 64, 3, stride=2),
67                 DeConvBlock(64, 64, 3, stride=1, output_padding=0),
68                 DeConvBlock(64, n_channels, 3, output_padding=0),
69             ])
70
71             self.adjust_layers = nn.ModuleList([
72                 nn.Conv2d(768, 512, kernel_size=1),
73                 nn.Conv2d(384, 256, kernel_size=1),
74                 nn.Conv2d(192, 128, kernel_size=1),
75                 nn.Conv2d(128, 64, kernel_size=1),
76             ])
77
78             # Encoder layers that will be connected to decoder -> n-1
79             self.skip_connection_layer_idxs = [4, 2, 1, 0]
80
81         def forward(self, x: torch.Tensor) -> torch.Tensor:
82             # Encoder
83             encoder_outputs = []
84             for encoder_layer in self.encoder_layers:
85                 x = encoder_layer(x)
86                 encoder_outputs.append(x)
87
88             # Decoder with skip connections
89             for i, decoder_layer in enumerate(self.decoder_layers):
90                 x = decoder_layer(x)
91                 if i < len(self.skip_connection_layer_idxs):
92                     x = torch.cat([x, encoder_outputs[self.
                        skip_connection_layer_idxs[i]]], dim=1)  # skip
                        connections
93                     x = self.adjust_layers[i](x)  # adjust channels
94
95             return x
```

Metriky pro zhodnocení modelů

Výsledky neuronových sítí DnCNN a DAE při redukci šumu z rentgenových snímků

2.4. Aplikace implementovaných metod a měření vlivu předzpracování na efektivitu učení segmentačních modelů



Obrázek 2.3.1.: Diagram zobrazující architekturu denoising autoenkodéru. (vytvořeno v [32])

3. Zhodnocení

4. Závěr

Seznam použitých zdrojů

1. BASAVAPRASAD, Benchamardimath; RAVINDRA, Hegadi. A STUDY ON THE IMPORTANCE OF IMAGE PROCESSING AND ITS APLLICATIONS. *International Journal of Research in Engineering and Technology* [online]. 2014, **03**(15), 155–160 [cit. 2024-03-16]. ISSN 23217308. Dostupné z DOI: 10.15623/ijret.2014.0315029.
2. KRIG, Scott. *Computer Vision Metrics*. 1. vyd. California: Apress Berkeley, 2014. ISBN 978-1-4302-5930-5. Dostupné z DOI: 10.1007/978-1-4302-5930-5.
3. *Computer Vision for X-Ray Testing: Imaging, Systems, Image Databases, and Algorithms*. 1. vyd. Cham: Springer International Publishing, 2015. ISBN 978-3-319-20747-6. Dostupné z DOI: 10.1007/978-3-319-20747-6.
4. Contrast Limited Adaptive Histogram Equalization. In: *Graphics Gems IV*. 1st Edition. San Francisco: Academic Press, 1994, s. 474–485. ISBN 978-0-12-336155-4.
5. JAYAKUMAR, Dontabhaktuni; PULLARAO, Bandi; PRADEEP, Vanukuru. An Algorithm on Generalized Un Sharp Masking for Sharpness and Contrast of an Exploratory Data Model. *International Journal of Advanced Engineering Research and Science*. 2016, **3**(9), 62–72. ISSN 23496495. Dostupné z DOI: 10.22161/ijaers/3.9.10.
6. CHAKI, Jyotismita; DEY, Nilanjan. *A Beginner's Guide to Image Preprocessing Techniques*. 1st Edition. Boca Raton: CRC Press, 2018. ISBN 9780429441134. Dostupné z DOI: 10.1201/9780429441134.
7. FAN, Linwei; ZHANG, Fan; FAN, Hui; ZHANG, Caiming. Brief review of image denoising techniques. *Visual Computing for Industry, Biomedicine, and Art* [online]. 2019, **2**(1) [cit. 2024-03-29]. ISSN 2524-4442. Dostupné z DOI: 10.1186/s42492-019-0016-7.
8. HAMBAL, Abdalla Mohamed; PEI, Zhijun; ISHABAILU, Faustini Libent. Image Noise Reduction and Filtering Techniques. *International Journal of Science and Research (IJSR)* [online]. 2017, **6**, 2033–2038 [cit. 2024-03-29]. Dostupné z: https://www.ijsr.net/getabstract.php?paperid=25031706.
9. MERY, Domingo. X-ray Image Processing. In: *Computer Vision for X-Ray Testing* [online]. 1. vyd. Springer Cham, 2015, s. 109–148 [cit. 2024-03-30]. ISBN 978-3-319-20746-9. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-319-20747-6_4.
10. PARIS, Sylvain; KORNPROBST, Pierre; TUMBLIN, Jack; DURAND, Frédo. *Bilateral Filtering: Theory and Applications* [online]. 1. vyd. Foundations a Trends, 2009 [cit. 2024-03-30]. ISBN 978-1-60198-251-3. Dostupné z DOI: 10.1561/0600000020.

11. WIENER FILTERS. In: *Advanced Digital Signal Processing and Noise Reduction* [online]. 2. vyd. Wiley, 2005, s. 178–204 [cit. 2024-03-30]. ISBN 0-470-84162-1. Dostupné z: https://www.roma1.infn.it/exp/cuore/pdfnew/ch06.pdf.

12. BUADES, Antoni; COLL, Bartomeu; MOREL, Jean-Michel. Non-Local Means Denoising. *Image Processing On Line* [online]. 2011, **1**, 208–212 [cit. 2024-04-04]. ISSN 2105-1232. Dostupné z DOI: 10.5201/ipol.2011.bcm_nlm.

13. QIU, Fang; BERGLUND, Judith; JENSEN, John R.; THAKKAR, Pathik; REN, Dianwei. Speckle Noise Reduction in SAR Imagery Using a Local Adaptive Median Filter [online]. 2013, **41**(3), 244–266 [cit. 2024-04-04]. ISSN 1548-1603. Dostupné z DOI: 10.2747/1548-1603.41.3.244.

14. GHOSH, Anirudha; SUFIAN, Abu; SULTANA, Farhana; CHAKRABARTI, Amlan; DE, Debashis. Fundamental Concepts of Convolutional Neural Network. *Recent Trends and Advances in Artificial Intelligence and Internet of Things* [online]. 2020, 519–567 [cit. 2024-04-18]. ISBN 978-3-030-32643-2. Dostupné z DOI: 10.1007/978-3-030-32644-9_36.

15. BRUNELLON. *Example of a neural network's neural unit* [online]. 2004. [cit. 2024-04-18]. Dostupné z: https://commons.wikimedia.org/wiki/File:Example_of_a_neural_network%5C%27s_neural_unit.png.

16. TEUWEN, Jonas; MORIAKOV, Nikita. Convolutional neural networks. *Handbook of Medical Image Computing and Computer Assisted Intervention* [online]. 2020, 481–501 [cit. 2024-04-18]. ISBN 9780128161760. Dostupné z DOI: 10.1016/B978-0-12-816176-0.00025-9.

17. BANK, Dor; KOENIGSTEIN, Noam; GIRYES, Raja. Autoencoders. *ArXiv*. 2020, **abs/2003.05991**. Dostupné také z: https://api.semanticscholar.org/CorpusID:260535551.

18. MICHELUCCI, Umberto. An Introduction to Autoencoders. *ArXiv*. 2022, **abs/2201.03898**. Dostupné také z: https://api.semanticscholar.org/CorpusID:245853675.

19. MIKOLAJCZYK, Agnieszka; GROCHOWSKI, Michal. Data augmentation for improving deep learning in image classification problem. *2018 International Interdisciplinary PhD Workshop (IIPhDW)* [online]. 2018, 117–122 [cit. 2024-04-20]. ISBN 978-1-5386-6143-7. Dostupné z DOI: 10.1109/IIPHDW.2018.8388338.

20. SHORTEN, Connor; KHOSHGOFTAAR, Taghi M. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data* [online]. 2019, **6**(1) [cit. 2024-04-20]. ISSN 2196-1115. Dostupné z DOI: 10.1186/s40537-019-0197-0.

21. JAEGER, Stefan; KARARGYRIS, Alexandros; CANDEMIR, Sema; FOLIO, Les; SIEGELMAN, Jenifer; CALLAGHAN, Fiona; XUE, Zhiyun; PALANIAPPAN, Kannappan; SINGH, Rahul K.; ANTANI, Sameer; THOMA, George; WANG, Yi-Xiang; LU, Pu-Xuan; MCDONALD, Clement J. Automatic Tuberculosis Screening Using Chest Radiographs. *IEEE Transactions on Medical Imaging*. 2014, **33**(2), 233–245. ISSN 1558-254X. Dostupné z DOI: 10.1109/tmi.2013.2284099.

22. CANDEMIR, Sema; JAEGER, Stefan; PALANIAPPAN, Kannappan; MUSCO, Jonathan P.; SINGH, Rahul K.; XUE, Zhiyun; KARARGYRIS, Alexandros; ANTANI, Sameer; THOMA, George; MCDONALD, Clement J. Lung Segmentation in Chest Radiographs Using Anatomical Atlases With Nonrigid Registration. *IEEE Transactions on Medical Imaging*. 2014, **33**(2), 577–590. ISSN 1558-254X. Dostupné z DOI: 10.1109/tmi.2013.2290491.

23. VIACHESLAV DANILOV. *Chest X-ray dataset for lung segmentation*. Mendeley, 2022. Dostupné z DOI: 10.17632/8GF9VPKHGY.1.

24. LOOP, Humans In The. *Teeth Segmentation on dental X-ray images*. Kaggle, 2023. Dostupné z DOI: 10.34740/KAGGLE/DSV/5884500.

25. CHEN, Hu; ZHANG, Kailai; LYU, Peijun; LI, Hong; ZHANG, Ludan; WU, Ji; LEE, Chin-Hui. A deep learning approach to automatic teeth detection and numbering based on object detection in dental periapical films. *Scientific Reports*. 2019, **9**. Dostupné z DOI: 10.1038/s41598-019-40414-y.

26. KUMAR, Nalin; NACHAMAI, M. Noise Removal and Filtering Techniques used in Medical Images. *Oriental journal of computer science and technology*. 2017, **10**(1), 103–113. ISSN 2320-8481. Dostupné z DOI: 10.13005/ojcst/10.01.14.

27. HARRIS, Charles R.; MILLMAN, K. Jarrod. Array programming with NumPy. *Nature*. 2020, **585**(7825), 357–362. Dostupné z DOI: 10.1038/s41586-020-2649-2.

28. VIRTANEN, Pauli; GOMMERS, Ralf; OLIPHANT. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, **17**, 261–272. Dostupné z DOI: 10.1038/s41592-019-0686-2.

29. VAN DER WALT, Stefan; SCHÖNBERGER. scikit-image: image processing in Python. *PeerJ*. 2014, **2**, e453.

30. LAM, Siu Kwan; PITROU, Antoine; SEIBERT, Stanley. Numba: A llvm-based python jit compiler. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, s. 1–6.

31. ZHANG, Kai; ZUO, Wangmeng; CHEN, Yunjin; MENG, Deyu; ZHANG, Lei. Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising. *CoRR*. 2016, **abs/1608.03981**. Dostupné z arXiv: 1608.03981.

32. HARRIS, Charles R.; MILLMAN, K. Jarrod; J., Stéfan. Array programming with NumPy. *Nature*. 2020, **585**(7825), 357–362. Dostupné z DOI: 10.1038/s41586-020-2649-2.

Seznam obrázků

Seznam tabulek

Sazba zdrojových kódů

A. Externí přílohy

Struktura repozitáře je následující:

BostonHousing	vypracovaná regresní úloha a data set Boston Housing
IrisFlowers	vypracovaná klasifikační úloha a data set Iris flowers
Obrázky	adresář s obrázky, které jsou zobrazeny v repozitáři
IntrusionDetection.rar	vypracovaná úloha Intrusion detection s data sety v souboru rar
README.md	jednoduchý popis repozitáře