VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Kursinis darbas

Objektų atpažinimas ir sekimas kompiuterinės tomografijos vaizduose

(Object detection and tracking in computed tomography)

Atliko: 3 kurso Programų sistemų studentas
Paulius Milmantas (parašas)

Darbo vadovas:
Linas Petkevičius (parašas)

Vilnius
2020

Įvadas

Pastaraisiais metais į medicinos sritį labai sparčiai skverbiasi informacinės technologijos. Atliekant įvairias diagnostikas ar tiriant ligas, vaistus yra pasitelkiama programinių sistemų pagalba [Meh17]. Tačiau didžiąją radiologo darbo dalį sprendimus turi priimti jis pats be jokios programinės įrangos pagalbos, nors dažniausiai yra apibrėžti tam tikri ligų aptikimų algorimtai, kaip vėžinių lastelių aptikimas, lūžiai ir taip toliau [Wal19]. Mano darbo tikslas yra prisidėti prie programinės įrangos kūrimo radiologams ir sukurti programą, kuri aptinka plaučius, kad vėliau tai galėtų būti panaudota tolimesnei IT plėtrai medicinos srityje. Darbe yra siekiama sukurti modelį, kuris aptikis plaučius kompiuterinės tomografijos nuotraukose,

Šiandien viena iš labiausiai perspektyvių sričių informacinių technologijų srityje yra mašininio mokymosi algoritmai. Jie padeda duomenyse aptikti sunkiai pastebimus dėsningumus ir nuspėti išeities rezultatus su tam tikra klaidos tikimybe. Šiame darbe kalbėsiu apie vieną iš šių metodų: dirbtinius neuroninius tinklus. Autorius pasirinko juos, nes kompiuterinės tomografijos vaizdai turi labai daug informacijos ir juos apdoroti naudojant logines taisykles yra per daug sunku. Dabartinė mokslinė veikla, nagrinėjanti dirbtinių neuroninų tinklų veiklą, fokusuojasi ties sunkiai aptinkamos duomenų koreliacijos aptikimo automatizacijos [Kei10]. Tačiau net ir jiems ši užduotis yra per sudėtinga, jeigu tiriamos yra įvairios mutacijos arba nedažnos ligos, nes su kiekvienu nauju nematytu ligos atveju, yra reikalinga apmokyti modelį, o tai reikalauja daug duomenų.

Sprenžiamam uždaviniui yra daug alternatyvių sprendimo metodų. Pasirinktam mano dirbtinių neuroninių tinklų metodui, vienas iš galimų alternatyvų yra „sprendimo medžiai“. Tai yra prižiūrimo mokymosi metodas skirtas duomenų klasifikacijai ir duomenų regresijai. Šis metodas naudoja taisyklių rinkinį. Norint gauti mažą šio metodo paklaidą, turime aprašyti vis daugiau šių taisyklių. Taisyklės paprastai būna aprašomos naudojant „if, else“ aprašus [Seh18]. Mano nagrinėjamame atvejyje duomenys yra labai dideli ir įvairūs, todėl aprašyti visas šias taisykles užtruktų neproporcingai daug laiko ir tam tikrų radiologijos žinių.

Turinys

1. Dirbtinio neuroninio tinklo sudėtis

Dirbtiniai neuroniniai tinklai yra matematiniai modeliai, kurie iš duotų parametrų atlieka spėjimą apie galimą rezultatą [Kei10]. Modelio tikslumui didinti, jį reikia apmokyti jam reikia įvesties parametrus ir tikslius rezultatus. Dirbtiniai neuroninių tinklų modelio pavadinimas kilo iš biologinės modelio prigimties: modelį sudaro neuronai ir jie tarpusavyje saveikauja. Modelio apmokymas taip pat primena smegenų veiklą [Kei10]. Modeliui yra duodama daug duomenų pavyzdžių, norimi rezultatai ir jis aptinka sunkiai pastebimą koreliaciją tarp duomenų.

1.1. Bazinė struktūra

Daugelį dirbtinių neuronų tinklų sudaro panašios struktūrinės dalys:

1. Įvesties sluoksnis: tai dalis kuri priima įvestį ir perduoda kitiems sluoksniams. Šiame darbe yra naudojamos kompiuterinės tomografijos nuotraukos, kurios yra suvienodinamos į 300x300 dydį. Nuotraukose yra pašalinamas RGB kanalas, todėl kiekvieną pikselį nusako pilkos spalvos atspalvis, kuris yra intervale [0;1]. Vienai nuotraukai užkoduoti užtenka 90 000 taškų, todėl neuroninio tinklo įvesties sluoksnį sudaro 90 000 neuronų.

2. Išvesties sluoksnis: tai dalis, kuri naudoja aktyvacijos funkciją, kuri grąžina galutinį tinklo rezultatą: tikimybių rinkinį, kuris parodo kokia tikimybė, kad objektas atitinka tam tikrą klasę. Dažniausiai kiek uždavinyje yra klasių, tiek ir yra neuronų išvesties sluoksnyje. Mokymosi sparta ir šios tikimybės dydis priklauso nuo pačios aktyvacijos funkcijos.

3. Paslėptas sluoksnis: perduoda svorius iš praeito sluoksnio į sekantį. Prieš reikšmės perdavimą paslėptame sluoksnyje kiekvienas neuronas sudeda visų ankstesnio sluoksnio neuronų siunčiamas reikšmes. Prieš šios sumos siuntimą, neuronų reikšmės yra pakeičiamos atsižvelgiant į svorius tarp neuronų jungčių ir tendenciškumą.

4. Susijungimai, jų svoriai ir tendenciškumas: Šie du parametrai yra tinklo atsitiktinai sugeneruojami ir apmokant tinklą pagal duomenis, svoriai ir tendenciškumas yra automatiškai tinklo keičiami. Tendenciškumas apibrėžia, kaip tinklo išvestis yra nutolusi nuo tikrosios reikšmės. Tarp kiekvieno neurono, kuris yra susijungęs tarpusavyje, yra jungtys, per kurias yra paduodamos reikšmės kitiems neuronams sekančiuose sluoksniuose. Kaip reikšmė kis keliaujant per jungtis, nusako svoriai tarp jungčių. Tinklui mokantis šie svoriai yra keičiami tol, kols tinklo išvestis priartėja tikro atsakymo. Svoriai nurodo koks stiprus yra ryšys tarp neuronų, taip nurodant, kaip tinklo reikšmės kinta pačiame tinkle. Jeigu tinklui yra paduodama reikšmė, o pirmasis tinklo neuronas, kuris priima reikšmę turi mažą svorį, tai reiškia, jog galutiniams rezultatui ši reikšmė neturi daug įtakos. Jeigu svoris yra didelis, tai duota reikšmė turi daug įtakos rezultatui [Ala16]. Tinkluose tendenciškumas nusako, kaip turi būti koreguojamos tinklo reikšmės. Jeigu aktyvacijos funkcija grąžina blogą reikšmę, galima keisti tendenciškumo kintamąjį ir pakeisti galutinį rezultatą.

5. Aktyvacijos funkcija: tai funkcija, kokia turi būti neurono išvestis. Kiekvienas neuronas susumuoja praeito sluoksnio neuronų išvestis ir šią suma praleidžia pro aktyvacijos funkciją. Aktyvacijos funkcijos reikšmė yra toliau paduodama kitiems neuronams (atsižvelgiant taip pat ir į jungčių svorius ir į tendenciškumą).

6. Optimizatimo funkcija: optimizuoja tikslo funkciją. Tikslo funkcija yra taip pat vadinama nuostolių funkcija. Šios funkcijos pagrindinis tikslas yra keisti neuroninio tinklo reikšmes taip, kad tinklo išvestis būtų kuo artimesnė tikram rezultatui. Jeigu yra parinkta bloga funkcija, išvedamos gerų ir blogų spėjimų tikimybės gali būti artimos viena kitai ir taip modelis mokysis lėtai. Tačiau jeigu gera funkcija yra parenkama, tikimybės tarp gerų ir blogų reikšmių labai skiriasi ir taip modelis mokosi greičiau.

7. Nuostolių funkcija. Nuostolių funkcija apskaičiuoja nuostolį tarp dviejų reikšmių. Paprastai skaičiuojamas nuostolis tarp tinklo išvesties ir tikrosios reikšmės. Šios funkcijos rezultatas yra perduodamas optimizavimo funkcijai, todėl nuo nuostolių funkcijos gali priklausyti ir mokymosi geritis, bei modelio tikslumas.

8. Mokymosi taisyklė: apibrėžia, kaip tinkle keičiasi svoriai, kad tinklas išvestų norimus rezultatus. Taip pat galima nurodyti ir kaip keičiasi tendenciškumas, bei kada jį naudoti.

Turint šias bazines neuroninio tinklo dalis, galima jį pradėti mokyti. Paprastai mokymasis yra skiriamas į dvi dalis: priekinę ir galinę sklaidą. Kai tinklui yra paduodamos reikšmės, jos keliauja iš pradinio sluoksnio iki išvesties sluoksnio. Tai yra priekinė sklaida. Atgalinė sklaida vyksta, kai tinklas išveda rezultatą, patikrinama ar rezultatas yra teisingas ir atitinkamai tinklas reaguoja į tai. Inicijuojant tinklą svoriai yra nustatomi atsitiktinai. Kiekvienam duomenų rinkiniui yra stebima tinklo išvestis ir tikrinama kokia ji turėtų būti. Gauta reikšmė ir tikra reikšmė yra siunčiamos į nuostolių funkciją ir gaunamas nuostolis. Gautas nuostolis yra perduodama tinklui atgal, per praeitus sluoksnius, einant nuo išvesties sluoksnio. Šiame žingsnyje yra naudojama optimizavimo funkcija. Ji parodo kokie turėtų būti svorių ir tendenciškumo pokyčiai tinklai ir taip yra keičiami tinklo svoriai ir tendenciškumas atsižvelgiant į padarytą klaidą. Kai klaidos yra padaromos pakankamai mažos ir jos yra mažesnės už duotą ribinę reikšmę, yra laikoma, kad tinklas baigė mokymosi procesą [Hin18].

1.2. GPU naudojimas

Šiame darbe bus naudojamas GPU, atliekant tinklo treniravimą. Tai yra spartesnis būdas už CPU naudojimas, nes CPU turi kelis galingus branduolius, o GPU žymiai daugiau. Giliajame mokyme, naudojami aritmetiniai veiksmai yra ganėtinai paprasti, todėl juos apdoroti gali ir CPU ir GPU. Tačiau GPU turi daugiau branduolių, kas leidžia atlikti daugiau matematinių veiksmų paralizuotai, negul CPU. Taip yra pasiekiamas didesnis treniravimo našumas.

Didesniam treniravimo našumui pastebėti naudojant GPU, negu CPU, buvo atliktas paprastas bandymas: naudojant Google tensorflow struktūrą, apmokame sistemą atpažinti kur yra žmogaus

plaučiai kompiuterinės tomografijos vaizduose su CPU ir GPU. Vidutiniškai apdoruojant 60 megabaitų duomenų, vienam ciklui tinklas su CPU resursais užtrunka 6 sekundes, o naudojant GPU vidutiniškai vienas žingsnis užtrunka 0.3 sekundės.

Šiame darbe naudojamas Tensorflow karkasas naudoja NVIDIA CUDA sistemą [NVI20b]. CUDA yra paralelinio skaičiavimo platforma, naudojama su grafikos plokštėmis [NVI20a]. Ji kiekvieną skaičiavimą paleidžia ant atskiros gijos. Visi duomenys yra suskirstyti į vienos, dviejų ir trijų dimensijų blokus. Kiekvienams blogas gali turėti virš 512 gijų. Visos gijos kurios operuoja viename bloke dalinasi bendra GPU atmintimi. Jeigu norima optimizuoti programą ir paleisti jos skaičiavimus ant GPU, tada reikia suskaičiuoti kiek vyks paralelinių operacijų ir kiek iš jų dalinsis bendra atmintimi. Pagal apskaičiuotą kiekį nurodome kiek mes norime turėti blokų ir kiek kiekvienas blokas turės atskirų gijų. Prieš kiekvieną operaciją, siekiant pasiekti optimalų našumą, jeigu reikia įkeliame kintamuosiu į bendrą bloko atmintį. Po atliktų skaičiavimų išvestus rezultatus turime iškelti iš bendros atminties.

Tačiau GPU naudojimas nėra visur panaudojamas. GPU gerai atlieka slankaus kablelio operacijas (FLOPs) ir atminties pralaidumu. Jeigu problema reikalauja nepastovios atminties, jos sprendimo paleidimas ant GPU nebus efektyvus, nes nebus galima išnaudoti branduolių, dėl kintančios atminties. Taip pat problemos sprendimas nebus efektyvus, jeigu reikalaus nedaug aritmetiškai intensyvių operacijų.

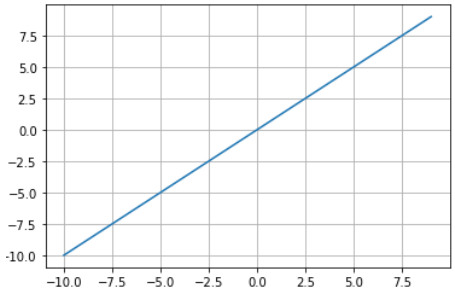
1.3. Aktyvacijos funkcijos

Aktyvacijos funkcijos yra būtinos neuroniniame tinkle, jeigu norime, kad tinklas grąžintų netiesines reikšmes. T.y. kad nuo tinklui duodamų parametrų, tinklo išvestis nepriklausytų tiesiogiai [Opp]. Neuroninio tinklo tikslas yra rasti tam tikrą koreliaciją tarp įvesties ir išvesties duomenų. Jeigu duomenys yra sudėtingų struktūrų, tiesinis tinklas koreliacijos neras, tačiau jeigu yra naudojamos aktyvacijos funkcijos, tinklas yra netiesinis, tada galima aptikti labiau kompleksyvią koreliaciją [Opp].

1.3.1. Tiesinė aktyvacijos funkcija

$$R(x, a) = x \cdot a$$

Tiesinės aktyvacijos funkcijos mių diapazonas yra labai didelis ir nėra dvejetainis, kaip dalis kitų funkcijų. Tačiau ši funkcija nėra plačiai naudojama, nes jos išvestinė yra konstanta, kas reiškia jog gautas gradientas nepriklauso nuo „x“. Taigi, jeigu spėjime yra klaida, vykdant atgalinę sklaidą mes reikšmę taisysime konstanta, neatsižvelgiant į pačią klaidą [Glo20].

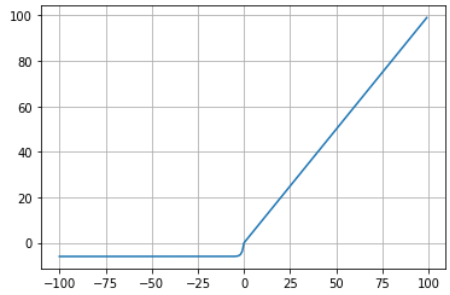


1 pav. Tiesinė lygtis

1.3.2. ELU

$$R(z) = \begin{cases} z; z > 0 \\ \alpha \cdot (e^z - 1); z \leq 0 \end{cases}$$

ELU funkcija vadinasi: „Exponential Linear Unit“. Ši funkcija paprastai konverguoja į nulį greičiau nei kitos ir taip duoda tikslesnius rezultatus. Skirtingai nuo kitų funkcijų, ši priima papildomai konstantą , kuri turi būti teigiamas skaičius. Ši funkcija yra labai panaši į RELU funkciją, tačiau ELU tampa tolygiai lėta, kol išvestis pasiekiama, o RELU staigiai sutolygėja [Glo20].



2 pav. ELU

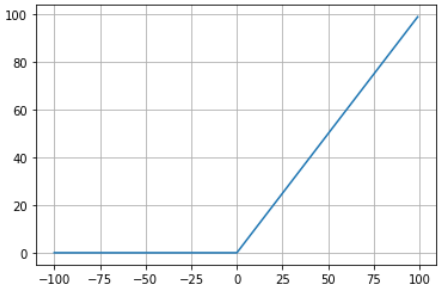
1.3.3. ReLU

$$R(z) = max(0, z)$$

ReLU funkcija vadinasi: „Rectified Linear Unit“. Ši funkcija pasiekia panašų efektyvumą kaip ir Sigmoid funkcija, tačiau ji reikalauja mačiau skaičiavimo resursų nei sigmoid ir tanh, nes minėta formulė yra paprastesnė, nei kitos [Glo20].

1.3.4. Sigmoid

$$R(x) = \frac{1}{1 + e^{-x}}$$



3 pav. ReLU

Sigmoid funkcijos parametrai priima realiuosius skaičius ir gražina reikšmę [0; 1]. Šios funkcijos geros savybės yra: tolydus gradientas, prognozuojamas išvesties diapazonas, netiesiškumas, visur diferencijuojama ir monotoniška. [Glo20]

1.3.5. Softmax

$$softmax(x)_i = \frac{exp(x_i)}{sum_{j=1}^{n} exp(x_j)}$$

Softmax funkcija priima parametrų skirstinį ir gražina tikimybių skirstinį, kurio bendra suma yra lygi vienam [Glo20].

1.3.6. Argumentacija už pasirinktą aktyvacijos funkciją

Šiame darbe bus naudojama Softmax aktyvacijos funkcija. Ši funkcija buvo pasirinkta, nes galima teigti, jog Softmax bando optimizuoti entropijos skirtumą tarp tikros reikšmės ir gautos. Entropija tarp tikros reikšmės „p“ ir spėjamos „q“ yra apskaičiuojama pagal:

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

Pagal entropijos lygtį galima teigti, jog Softmax funkcija bando optimizuoti entropiją tarp tikros reikšmės ir spėjamos [CS220]. Tai priveda prie apytiksliai stabilaus gradiento. Jeigu modelis klysta, jis dėl to gali greičiau pasitaisyti.

Kita priežastis, kodėl yra naudojama Softmax funkcija, yra naudojimui patogi išvestis. Jeigu norima aptikti kelias klases, kiekvienas išvesties neuronas išveda klasės tikimybę. Jeigu išvestis yra 0.6, tai tikimybė, kad tai ir yra ta klasė, yra 60%. Visų išvesties neuronų išvesties suma yra lygi 1.

2. Problemos su skaičiaja matematika

Neuroninius tinklus sudaro paprastai daug sluoksnių ir kiekviename sluoksnyje yra po daug neuronų. Kiekvienas neuronas yra sudėties funkcija. Funkcija apibrėžia m įvesties vienetų „x“, su „w“ svoriais. Visi šie neuroniai savo ir kaimyniniuose dviejuose sluoksniuose yra sujungti su visais kitais neuronais. Pereinant duomenims į kitą sluoksnį yra apskaičiuojamos visų neuronų reikšmės ir jos perduodamos toliau. Todėl tarp neuroniniame tinkle vyksta labai daug sudėties operacijų ir yra svarbu, kad neįvyktų jokią skaičiosios matematikos klaida, kaip reikšmės apvalinimo klaida.

$$y_k = \gamma \cdot (\sum_{i=1}^{m} w_{ki} x_i)$$

Neuroniniai tinklai atlieka daug skaičiavimų, todėl gali atsirasti klaidos su skaičiaja matematika: gradientinio nuolydžio „užstrigimas“, apvalinimo paklaida, blogas problemos apibrėžimas.

2.1. „Underflow“ ir „overflow“

Kai kompiuterinės programos vykdo daug skaičiavimų su labai dideliais arba mažais skaičiais atsiranda „underflow“ ir „overflow“ problemos dėl skaičių apvalinimo. „Underflow“ problema pasireiškia, kai funkcijos reikšmė yra labai arti nulio ir ji yra apvalinama į nulį. Kai kurie algoritmai gali „sulūžti“ dėl nulinio parametro (pvz. dalybos iš nulio). Kita problema yra „overflow“. Ji pasireiškia kai gauta reikšmė yra labai didelė ir ji yra apvalinama į begalybę [GBC16]. Vienas iš pavyzdžių kaip pasireiškia „overflow“ problema skaičiavimuose yra 4 pav. Šiame kode yra paimama didžiausia galima sveiko skaičiaus reikšmė ir ji yra didinama. Taip atsakymas yra apvalinamas į didžiausią galimą reikšmę ir yra prarandama ta pati reikšmė. Šios dvi problemos yra išspręstos naudojant dviejų bitų statuso registrą esantį kompiuterio procesoriuje. Vienas bitas yra nustatomas jeigu CPU įvykdo „overflow“, kitas, jeigu procesorius įvykdo „underflow“. Modernios programavimo kalbos, kaip Python, pagal šiuos du bitus gali pasakyti ar įvyko viena iš šitų klaidų.

```
int value = Integer.MAX_VALUE-1;
for(int i = 0; i < 4; i++, value++) {
    System.out.println(value);
}
```

4 pav. Overflow problema

2.2. Apribotas optimizavimas

Apribotas optimizavimas yra naudojamas norint optimizuoti funkciją turint nustatytus kintamuosius [GBC16].

Paprasčiausias šios problemos sprendimas yra didžiausio gradientinio nuolydžio metodo naudojimas. Pasirinkus pakankamai mažą mokymosi žingsnį ir pradinį tašką, kuris yra arti tikrosios reikšmės, galima pasiekti kitus neapribotus kintamuosius.

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta)$$

Sudėtingesnis šios problemos sprendimas yra pačios problemos perdarymas [GBC16].

2.3. Blogas problemos apibrėžimas

Funkcija yra blogai apibrėžta, jeigu keičiant funkcijos įvestį nedaug, funkcijos reikšmė pakinta ženkliai daugiau. Funkcija yra laikoma gerai apibrėžta, jeigu keičiant jos įvestį, jos reikšmė keičiasi proporcingai. Ši problema pasireiškia labiausiai, kai yra atliekami skaičiavimai su mažais skaičiais. Jeigu įvesties vietoje įvyksta apvalinimo klaida, funkcijos rezultatas gali grąžinti blogą reikšmę [GBC16].

3. Optimizavimo algoritmai

Tinklo siekiamas tikslas yra atliekant kuo mažiau mokymosi iteracijų, pasiekti norimą išvesties tikslumą. Šiam mokymosi greičiui ir tikslumui pasiekti ir naudojami optimizavimo algoritmai.

3.1. Gradientinis nuolydis

Gradientinio nuolydžio algoritmas Tai yra pati paprasčiausia optimizavimo funkcija.

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta)$$

Pagal algoritmą yra atliekama daug iteracijų. Kiekvienos iteracijos metu yra apskaičiuojama theta reikšmė. Ji yra gaunama iš esamos thetos atėmus mokymosi žingsnio konstantos ir nuostolių funkcijos gradientą parametro theta atžvilgiu. Kadangi žingsnis kurį atliekame kiekvienos iteracijos metu yra konstanta, jeigu pradinis parinktas taškas yra toli nuo lokalaus minimumo, tai prireiks daug iteracijų, kol jis bus pasiektas. Šiuo metodu taip pat kiekviena iteracija yra peržiūrimi visi duomenys, kad nėra gerai jeigu duomenų yra itin daug [Dos19]. Tokiu atveju reikėtų daug atminties. Šis algoritmas nebus naudojamas, nes nėra atsižvelgta į praeitas iteracijas ir minimumas yra per lėtai pasiekiamas, algoritmas.

3.2. Poaibio gradientinis nuolydis

Šis algoritmas yra panašus į paprastą gradientinį metodą, tačiau yra keli pakeitimai. Kai yra pereinama per visus duomenis, po kiekvieno duomenų elemento peržiūrėjimo, yra atnaujinami parametrai. Todėl po kiekvienos iteracijos, turime visų duomenų elementų pakeitimų sudėtį. Todėl minimumas yra pasiekiamas statesne trajektorija ir nereikalinga turėti daug atminties. Pagrindinis šio metodo trūkumas yra galimas jo lėtumas. Pereiti per visus duomenis užtrunka daug laiko ir nevisos duomenų elementų iteracijos prie parametrų pakeitimo daug neprisideda [Dab17].

3.3. Mažo poaibio gradientinis nuolydis

$$\theta = \theta - \eta \cdot \bigtriangledown_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

Abiejų minėtų algoritmų idėjų kombinacija išvedė mažo poaibio gradientinį metodą. Šis algoritmas suskirsto duomenis į poaibius, kurių elementų skaičius yra daugiau nei 1. Dažniausiai poaibių dydis yra pasirenkamas dvejetas pakeltas kokiu nors laipsniu, nes vaizdo plokštėms tokius poaibius apdoroja geriau. Šis algoritmas yra greitesnis nei poaibio gradientinis metodas, nes yra išvengiama mažų iteracijų, kurių duomenų elementas mažai prisideda prie parametrų pakeitimo. Taip pat šio metodo pasiekiamas tikslumas yra didesnis nei kitų minėtų algoritmų, nes atsitiktinai parenkami poaibiai skiriasi kiekvieno poaibių paskirstymo metu, taip sudarant daugiau variacijų ir

daugiau triukšmo. Kadangi bendrai yra apdorojama daugiau duomenų variacijų, šis metodas gali pasiekti didesnį tikslumą. Nors šis metodas gali pasiekti didesnį tikslumą ir yra greitesnis [Dab17].

3.4. Sunkumai taikant gradientinius metodus

1. Mokymosi žingsnio nurodymas. Tikslaus sprendimo nurodančio kokį žingsnį parinkti nėra, jis paprastai yra parenkamas empiriškai stebint konvergavimo greitį ir duomenų kiekį. Jeigu žingsnis yra parenkamas per didelis, tai optimizavimo funkcija gali diverguoti arba peršokti lokalų minimumą ir nutolti nuo jo, taip praiilginant procesą. Jeigu žingsnis yra parenkamas per mažas, tada konvergavimo greitis į lokalų minimumą yra labai mažas ir procesas trunka ilgai [Rud16].

2. Mokymosi žingsnio keitimo trigeris. Kuriant „trigerį“ reikia nurodyti arba epochų skaičių arba ribinę nuostolių funkcijos reikšmę, kada trigeris turi būti aktyvuotas. Po aktyvacijos, trigeris pakeičia mokymosi žingsnio dydį. Trigerio aktyvacijos laiko nurodymui irgi nėra jokios tikslios taisyklės, tai yra nurodoma empiriškai stebint mokymosi procesą. Jeigu aktyvacijos laikas yra nurodomas per anksti, mokymosi žingsnis yra sulėtinamas per greitai ir funkcija pradeda lėčiau diverguoti į minimumą. Jeigu žingsnis yra nurodomas per vėlai, tam tikru momentu žingsnis gali būti per didelis ir funkcija gali pradėti diverguoti arba peršokti lokalų minimumą ir tolti nuo minimumo [Rud16].

3. Duomenų rinkinio parinkimas. Jeigu daug duomenų turi tuos pačius požymius, pagal kuriuos tinklas klasifikuoja rezultatus, visų šių duomenų iteracija nėra būtina, nes jie tinklo svorių pokyčius įtakos labai nežymiai. Jeigu duomenų, su panašiais požymiais, yra daug, reikia paimti tik dalį jų ir tik tą dalį iteruoti. Jeigu duomenų, su panašiais požymiais, yra daug, tada tinklo vienos epochos trukmė gali užtrukti ilgiau ir optimizavimo funkcija lėčiau konverguos į lokalų minimumą [Rud16].

4. Optimizavimo funkcijos „užstrigimas“ ties lokaliu minimumu. Optimizavimo funkcijų tinklas yra rasti globalų minimumo tašką. Tačiau gradientų nuolydžiu pagrįstų algoritmų dažna problema: „užstrigimas“ ties lokaliu minimumu. Gradientas parodo kuria kryptimi funkcija leidžiasi žemyn. Ta kryptimi optimizavimo algoritmas eina. Pasiekus lokalų minimumą, kai gradientas pradeda didėti, optimizavimo algoritmai baigia darbą. Todėl jeigu yra blogai parenkamas pirmas taškas, optimizavimo algoritmai „užstrigs“ ties lokaliu minimumu [Rud16]. Tą išvengti galima prieš algoritmo pradžią įvertinti funkciją ir parinkti artimus pradinius taškus prie tikrųjų reikšmių. Kitas būdas to išvegti yra naudoti optimizavimo algoritmus, kurie nesiremia gradientų nuolydžiu. Vienas iš daugelio algoritmų, kuris nesiremia gradientais, yra banginių judesių pagrįstas optimizavimo algoritmas.

Minėtas banginių optimizavimo algoritmas buvo sukurtas stebint banginių judėjimą, jiems gaudant mažas žuvis. Gaudant žuvis banginiai negali padaryti status posūkio, todėl gaunant žuvis jie

juda aplink žuvis ir po kelių ratų jie pasiekia apskritimo centrą ir pagauna žuvis. Šis algoritmas yra pagrįstas būtent šiuo judėjimu [Sey16].

Randomly initialize the whale population.
Evaluate the fitness values of whales and find out the best search agent $\boldsymbol{X}^*$.
while $t < t_{max}$
  Calculate the value of a according to Equation (13)
  for each search agent
    if $h < 0.5$ then
      if $|\boldsymbol{A}| < 1$ then $\boldsymbol{X}(t+1) = \boldsymbol{X}^*(t) - \boldsymbol{A} \cdot \boldsymbol{D}$
      else if $|\boldsymbol{A}| \geq 1$ then $\boldsymbol{X}(t+1) = \boldsymbol{X}_{rand}(t) - \boldsymbol{A} \cdot \boldsymbol{D}''$
      end if
    else if $h \geq 0.5$ then
      $\boldsymbol{X}(t+1) = \boldsymbol{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \boldsymbol{X}^*(t)$
    end if
  end for
  Evaluate the fitness of $\boldsymbol{X}(t+1)$ and update $\boldsymbol{X}^*$
end while

5 pav. Banginių optimizavimo algoritmas [Sey16]

3.5. Bendros gradientinių metodų problemos

Gradientinio nuolydžio metodai yra pirmo laipsnio optimizavimo funkcijos, kas reiškia jog jos naudoja tik pirmo laipsnio išvestines. Kadangi pirma išvestinė rodo tik funkcijos statumą, bet ne išlenktumą, tai:

1. Jeigu antra išvestinė yra lygi nuliui, tai funkcija yra tiesinė. Todėl, mokymosi žingsnis yra lygus alpha [Dab17].

2. Jeigu antra išvestinė yra didesnė už nulį tai funkcijos išlenktumas eina į viršų. Todėl funkcijos žingsnis yra mažesnis už mokymosi žingsnį ir optimizavimo funkcija gali diverguoti [Dab17].

3. Jeigu antra išvestinė yra mažesnė už nulį, tai funkcija yra išlenkta į apačią. Todėl funkcijos žingsnis yra didesnis už mokymosi žingsnį [Dab17].

To pasekoje, optimizavimo funkcijos greitis gali būti lėtas arba ji gali diverguoti.

4. Nuostolių funkcijos

4.1. Nuostolių funkcijos pasirinkimas

Nuostolių funkcijos grąžina skirtumą tarp tinklo išvesties ir tikrų reikšmių. Pagal nuostolių reikšmes yra atitinkamai apskaičiuojami gradientai ir atitinkamai keičiami tinklo svoriai ir tendenciškumas.

4.2. Vidutinė kvadratinė paklaida (MSE)

$$Nuostolis = \frac{1}{N} \cdot [\sum (Y_{Gautas} - Y_{Tikras})^2]$$

MSE funkcija sumuoja skirtumų kvadratus ir randa bendrą nuostolių vidurkį. Ši funkcija yra dažnai numatyta įvairiuose MATLAB ir Python algoritmuose. Vienintelė neigiama šios funkcijos pusė yra tai, kad jeigu duoti duomenys yra pirmojo laipsio, tada negalima teigti, kad rezultatai tiesiogiai koreliuoja tarpusavyje, nes MSE yra antrojo laipsnio funkcija [Kat19].

4.3. Šaknis iš vidutinės kvadratinės paklaidos (RMSE)

$$Nuostolis = \sqrt{\frac{1}{N} \cdot [\sum (Y_{Gautas} - Y_{Tikras})^2]}$$

RMSE funkcija yra panaši į MSE, tik pridėta šaknies traukimo operacija. Šaknies traukimas paverčia RMSE pirmojo laipsnio funkcija, todėl ją galima naudoti su pirmojo laipsnio duomenimis [Kat19].

4.4. Huber nuostolių funkcija

$$Nuostolis_\delta = \begin{cases} 0.5 \cdot (f(x) - y)^2; |y - f(x)| \leq \delta \\ \delta \cdot |y - f(x)| - 0.5 \cdot \delta^2 \end{cases}$$

Huber nuostolių funkcija yra mažiau žinoma nei kitos ir mažiau naudojama praktikoje. Ji yra mažiau jautri deviacijai negul MSE funkcija. Ji taip pat yra diferencijuojama ties 0 [Kat19]. Šios funkcijos hiperparametras delta leidžia priartinti šią funkciją prie MAE arba MSE. Jeigu delta yra artimas nuliui, Huber funkcija priartėja prie MAE, jeigu delta artimas begalybei, priartėja prie MSE [Kat19].

5. Neuroninių tinklų modeliai

Objektų aptikimas nuotraukose, taikant dirbtinius neuroninius tinklus, gali būti kelių tipų: objektų klasifikacija, objektų lokalizacija, semantinė segmentacija, objektų segmentacija (instance segmentation).

Objektų klasifikacijos tinklas gražiną reikšmę nusakančią ar vaizde yra objektas ar jo nėra. Klasifikacijos uždavinį sprenžiant yra pradedama nuo duomenų normalizacijos. Visi vaizdai yra padidinami ar sumažinami iki tam tikro dydžio. Galima taip pat pašalinti vaizdų RGB kanalą ir padaryti juos bespalviais. Norint padidintį esamą duomenų rinkinį šiame darbe buvo naudotas Gauso filtras kiekvienai nuotraukai. Su kiekviena nuotrauka buvo sugeneruotas dar 5 papildomos su skirtingai Gauso filtro parametrais. Šis metodas taip pat padeda neuroniniam tinklui neprisirišti prie tam riktų pikselių reikšmių. Taip pat vaizdus galima pakreipti ir kitaip modifikuoti, tačiau plaučių aptikimo tomografiniuose vaizduose uždavinyje, tas nebuvo atliktas, nes organai dažniausiai išlaiko savo proporcijas. Po duomenų normalizacijos reikia žiūrėti duomenų parametrus ir pagal tai sukonstruoti pirmąjį tinklo sluoksnį. Paprastai pirmą sluoksnį nusako:

$$N_{Pirmas_sluoksnis} = X_{VaizdoPlotis} \cdot Y_{VaizdoAukkis} \cdot N_{SpalvKanalSkaiius}$$

Sprendžiant plaučių aptikimo kompiuterinės tomografijos nuotraukose uždavinį, buvo naudojamas RGB kanalų pašalinimas. Taip buvo sumažintas pirmasis tinklo sluoksnis tris kartus ir paleidžiant sprendimą prireikė mažiau atminties. Paskutinis tinklo sluoksnis priklauso nuo to, kokius mes rezultatus norime gauti. Jeigu norime pasakyti koks objektas yra nuotraukoje, galima naudoti tokį išvesties sluoksnio dydį, kiek yra klasių modelyje. Norint taip pat aptikti ir kur vaizde yra atpažintas objektas tinklo išvesties sluoksnio dydis turi būti:

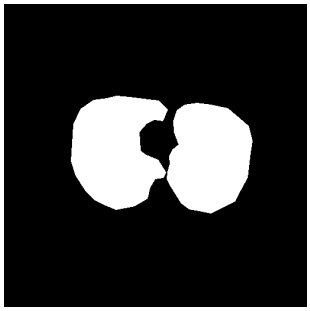
$$N_{Ivesties_sluoksnio_dydis} = C_{KlasiSkaiius} \cdot 5$$

Klasių skaičius yra dauginamas iš 5, nes norime gauti pačios klasės tikimybę ir tikėtinas viršutines ir apatines objekto x ir y reikšmes.Jeigu norime gauti ne tik x ir y reikšmes, bet tikslias objekto kiekvieno taško koordinates, kad tiksliai žinotume objekto ribas, reikia sudėtingesnio modelio, kuris vertintų kiekvieno vaizdo pikselio tikimybę, kad jis priklauso tam tikrai klasei. Šio modelio įvesties sluoksnis lieka toks pat kaip ir kitų, tačiau keičiasi išvesties sluoksnis. Jis tampa lygus:

$$N_{IvestiesSluoknis} = C_{KlasiSkaiius} \cdot X_{VaizdoPlotis} \cdot Y_{VaizdoAukkis}$$

Taip galima nusakyti kuriai klasei priklauso kiekvienas pikselis ir nusakyti, kokios yra kiekvienos klasės ribos vaizde.

Šiame darbe yra naudojamas Tensorflow karkaso Mask RCNN inception modelis, kuris naudoja objektų segmentavimą. Tinklo modelį ištreniruoti ir nusakyti kur yra tikslios plaučių ribos nuotraukose, buvo naudojami dvejetainių reikšmių „png“ failai. Vienas iš tokių failų pavyzdžių:



6 pav. Vaizdo kaukė.

Šiam uždaviniui tokio tipo failai buvo pasirinkti tam, nes reikia aptikti dvi klases: plaučiai ir ne plaučiai. Taip su skirtingomis pikselių reikšmėmis yra paduodamos kiekvieno pikselio klasės. Šie dvejetainiai vaizdų failai buvo gauti apibrėžiant plaučius kiekvienoje nuotraukoje ir eksportuojant pažymėtus ribinius taškus į „XML“ failus. Pagal duotas ribinių taškų X ir Y koordinates, buvo parašyta programa, kuri automatiškai pagal turimus duomenis sugeneruoja dvejetainius vaizdų failus.

5.1. Mask RCNN inception

Mask RCNN modelį sudaro dvi dalys: vaizdo regiono pasiūlymo tinklas (Regional proposed network) ir klasifikavimo modelis [Kha19]. Bendras algoritmas:

1. Nuotrauka yra praleidžiama per konvoliucinį neuroninį tinklą (R-CNN), kuris sugeneruoja požymių aibę, pagal kurią galima nuspėti, kad tai yra ieškomas objektas ir kur būtent jis yra vaizde.

2. Vaizdo regiono pasiūlos algoritmas naudoja CNN, kad sugeneruotų regionus vaizde pagal gautą požymių aibę.

3. Sulyginus gautis regionus gaunamos koordinatės, kurios apibrėžia objektą.

4. Naudojant regresijos modelius objekto apibrėžimas yra tikslinimas, kol gaunami optimalūs rezultatai.

5. Rezultatai yra persiunčiami į kitą CNN tinklą, kuris sugeneruoja vaizdų kaukes , atitinkančius klases.

Tinklas gauna vaizdo regionus, kurie turi didesnę tikimybę turėti objektą, naudodamas 9 inkarinių apibrėžimų koordinates. 3 inkariniai apibrėžimai yra skirtingų formų. Kiekvienam šiam apibrėžimui yra sukuriami dar 2 apibrėžimai, kurių formos yra vienodos, tačiau jų dydžiai skiriasi. Šiais 9 inkarais yra pereinama per vaizdą su kiekvienu inkaru ir kiekvienas inkaro apibrėžtas vaizdas yra analizuojamas: siunčiamas per CNN ir gaunama tikimybė, kad tai yra tam tikra klasė. Su kiekvienu uždaviniu yra patartina patiems apibrėžti kokios formos turi būti inkarai. [Gao17] Pavyzdžiui jeigu uždavinys yra aptikti plaučius kompiuterinės tomografijos nuotraukose, inkarai

turėtų būti būti didesni į aukštį ir mažesni į plotį ir 3 inkarai iš 9 turi būti kvadrato formos, nes jeigu analizuojamas plaučių viršus nuotraukose, tai jis yra apvalus, o ne pailgas.

Turint daug inkarų sudarytų vaizdų, reikia atskirti kurie vaizdai yra fonas, o kurie yra objektas. Foninių vaizdų negalima traukti į regresijos modelį, o objekto vaizdus reikia. Šis regresijos modelis yra naudojamas pataisyti inkaro sukurtą vaizdą, kad jis apibrėžtų tik objektą [Gao17].

Po šių veiksmų vaizdui yra sugeneruojama daug objektų apibrėžimų. Dalis šių apibrėžimų kertasi tarpusavyje. Apibrėžimai kurie kertasi tarpusavyje yra išsaugomi ir vėliau toliau analizuojami [Gao17]. Kiekvienam kirtimui yra apskaičiuojamas kirtimosi koeficientas:

$$K = \frac{N_{PirmasObjektas} \bigcap N_{AntrasObjektas}}{N_{PirmasObjektas} \bigcup N_{AntrasObjektas}}$$

Jeigu koeficientas yra lygus 1, tai reiškia jog inkarų apibrėžimai sutampa idealiai. Jeigu koeficientas yra mažesnis už arba lygus 0.5, tada apibrėžimai nėra naudojami. Jeigu koeficientas yra didesnis už 0.5, sankirta yra išsaugoma tolimesnei analizei. Atlikus paiešką, kiekvienam atskiram objektui yra parenkama tik sankirta su aukščiausiu sutapimo koeficientu ir kiti apibrėžimai apie einamąjį objektą yra pašalinami. Po šių žingsnių yra gaunami apibrėžimai aplink kiekvieną objektą, ir jų yra tik po vieną. Kiekvienas šis apibrėžimas yra siunčiamas per kitą CNN tinklą, kuris analizuoja kiekvieno pikselio reikšmę ir nustato tikslias objektų ribas [Gao17].

5.2. Vieno kadro aptikimas

Vieno kadro aptikimo (trumpinys SSD) algoritmas objektų aptikimui naudoja tik vieną kadrą. Objektų aptikui vaizdo įrašuose, objektams aptikti užtenka vieno kadro ir užtenka duomenis persiųsti per neuroninį tinklą tik vieną kartą. Šis algoritmas naudoja VGG16 architektūrą kaip pagrindą. Toks pasirinkimas buvo atliktas dėl VGG16 didelės spartos [For17]. Ieškomos vietos paieška naudojasi inkarų sistema: yra generuojami langai, vaizdui skanuoti, kurie yra įvairaus dydžio santykių. Kiekvienam sugeneruotam langui yra skaičiuojama tikimybė, kad tai yra koks objektas ir atstumą iki tikrosios reikšmės. Sugeneruoto lango nuostoliui apskaičiuoti yra naudojama L1-Norm. skaičiuojant nuostolius [For17]. Ši nuostolių funkcija yra mažiau tiksli negul naujesnė L2-Norm. tačiau ji buvo pasirinkta, nes ji yra nuolaidesnė kiekvienam pikseliui. T.y. jeigu keli pikseliai neatitinka objekto, o visi kiti atitinka, tai šie daro mažai įtakos galutiniam tinklo rezultatui.

5.3. Alternatyvūs objekto paieškos algoritmai

Minėti tinklų modeliai skiriasi savo neuroninių tinklų struktūra ir ieškomų regionų paieškos algoritmais. Minėti abu tinklai naudoja skirtingas inkarų paieškos algoritmo realizacijas. Egzistuoja ir daugiau paieškos būtų. Vienas iš būtų yra slenkančio lango algoritmas. Šis algoritmas pereina per visas galimas vaizdų kombinacijas. Vietoj laikantis tam sikro lango dydžio santykio,visi dydžiai yra pasirenkami ir išanalizuojami. Kadangi šis algoritmas tikrina daugiau sugeneruotų langų

kombinacijų, algoritmas veikia labai lėtai. Buvo sukurtas rašytinių lietuvių kalbos raidžių aptikimo modelis. Jis buvo padarytas iš 3 sluoksnių ir vienai 1920x1080 pikselių nuotraukai išanalizuoti prireikia vidutiniškai 20 minučių. Tačiau šį laiką galima sutrumpinti padidinant žingsnio dydį. Pavyzdžiui, padidinus žingsnio dydį vietoj 1 pikselio, naudojant 10-ties pikselių žingsnį, laikas vienai nuotraukai išanalizuoti vidutiniškai užtrunka apie 6 minutes.

6. Tyrimas

Objektų aptikimo kompiuterinės tomografijos nuotraukose uždaviniui spręsti, buvo naudotas Tensorflow karkasas. Tyrimas buvo padalintas į kelias dalis: literatūros analizę, modelio sukūrimą, treniravimą ir jo praktinį pritaikymą programoje.

6.1. Duomenų paruošimas

Kompiuterinės tomografijos nuotraukos buvo parsiųstos iš „https://www.cancerimagingarchive.net/“ puslapio. Jos kraštinės buvo sužymėtos naudojant „Make-sense“ duomenų žymėjimo programa. Kiekvienas pažymėtas taškas apibrėžiantis objektus buvo eksportuotas į JSON failą. Kadangi buvo reikalingi kaukių failai, o buvo turimas, tik JSON, buvo parašyta kita programa, kuri išanalizuoja JSON failą ir sukuria kaukes pagal tai. Šiame darbe buvo sužymėtos 50 nuotraukų. Kiekvienai nuotraukai buvo pritaikytas Gauso filtras su 5 skirtingais parametrais. Taip buvo gauta 5 kartus daugiau duomenų, negu buvo sužymėta iš pradžių.

Turint vaizdus ir jų kaukes, reikia šiuos duomenis suspausti į vieną failą. Tensorflow karkasas treniruojant modelį priima „.pb“ failus mokant modelį. Duomenų suspaudimui buvo parašyta tam atskira programa kuri naudoja „object_detection“ direktorijoje esančias pagalbines programas. Galutinis duomenų paruošimo rezultatas: sukurti trys atskiri „.pb“ failai, kuriuose yra pateikti treniravimo, testavimo ir validavimo duomenys.

6.2. Modelio sukūrimas ir treniravimas

6.2.1. Pasiruošimas

Prieš naudojamo modelio pasirinkimą apmokymą reikia atlikti keletą paruošiamųjų žingsnių. Pirma reikia parsisiųsti Tensorflow karkaso sukurtą modelių rinkinį, kuriame yra pateikti modeliai ir kiti modeliavimo sprendimai, pagreitinantys darbą. Šiame rinkinyje yra naudojamos dvi direktorijos: „object_detection“ ir „slim“. „object_detection“ direktorijoje yra pateiktas karkasas, su kuriuo yra lengva kurti naujus objektų aptikimo modelius ir juos apmokyti. Šis rinkinys turi ir jau apmokytų modelių pagal COCO duomenų bazę [Ten]. Kitą direktorija „slim“, yra skirta mokymui ir patikrinimui kaip veikia vaizdų klasifikacijos modeliai, kaip Inception ir VGG. Parsisiuntus Tensorflow karkaro modelių rinkinį, reikia parsisiųsti Google Protobuf produktą. Minėtoje „object_detection“ direktorijoje esančiame karkase, yra naudojami „.proto“ failai, kurie turi būti sukompiliuoti į „.py“ failus. Google Protobuf programa atlieka šį kompiliavimą.

6.2.2. Kūrimas ir treniravimas

Paruošta „object_detection“ direktorija jau turi paruoštus modelius, kurie yra ištreniruoti naudojant COCO duomenų bazę. Kadangi kompiuterinės tomografijos nuotraukose yra reikalingas

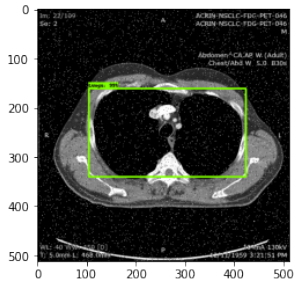
didelis tikslumas ir organai yra susispaudę vienas tarp kito taip, kad sunkiai matosi organų ribos, negalima naudoti duomenų žymėjimui stačiakampius. Reikia naudoti kaukes kiekvienam objektui sužymėti. Vienas iš modelių, kuris naudoja kaukes, yra „mask_rcnn_inception_v2_coco“.

Nusistačius koks modelis bus naudojamas, reikia jį sukonfigūruoti. Naudojamo failo „pipeline.config“ faile reikia nurodyti visus tinklo parametrus. Vienas iš jų yra klasių skaičius. Šis skaičius nurodo tinklo išeities sluoksnio dydį. Kadangi šiame darbe yra dirbama tik su plaučių aptikimu, klasių yra tik 1. Yra nurodomas įvesties vaizdų modifikavimas. Visų jų dydžiai yra pakeičiami į 800x1365 pikelių dydį. Tai reiškia, kad tinklo įvesties sluoksnį sudarys 1.092.000 neuronų. Šis skaičius yra lygus vaizdo dydžio parametrų sandaugai. Toliau yra nustatomas vaizdo slenkančio branduolio parametrai: žingsnio dydis, kuris yra lygus 2 ir branduolio dydis, lygus 2. Konfigūraciniame faile taip pat yra nurodoma naudojama aktyvacijos funkcija, kuri tyrime yra parinkta „SOFTMAX“. Tiek žingsnių užtenka modelio apibrėžimui.

Tinklui treniruoti, Tensorflow karkasas „object_detection“ direktorijoje turi paruošęs programa „legacy/train.py“, kuri tikriną nurodytą konfiguracinį failą, duotus suspaustas duomenis, modelį ir treniruoja tinklą. Kas tam tikrą epochų skaičių, tinklo svoriai ir kitos tinklo reikšmės yra išsaugomos „.pb“ faile, kuris vėliau yra naudojamas, kai norima užkrauti tinklo reikšmes ir atlikti tolimesnį tinklo treniravimą arba vykdyti objektų aptikimą vaizduose.

7. Rezultatai

Naudojant Tensorflow karkasą ir „mask rcnn inception v2“ tinklo modelį, buvo sėkmingai ištreniruotas tinklas aptikti plaučius su 0.02 nuostoliais. Tikslaus tikslumo procentais negalima nusakyti, nes tai yra ne objektų klasifikavimo, bet aptikimo uždavinys. Treniravimas užtruko apie 20 minučių, naudojant Google Colab GPU. Užteko 31240 epochų apmokyti plaučių aptikimą. Treniravimui buvo sužymėtos 50 nuotraukų. Naudojant žymėjimo programą, buvo sukurti kaukių failai, kurie tiksliai apibrėžia plaučių ribas. Naudojant įvairias modifikacijas: gauso filtras pritaikytais 5-iomis skirtingomis tikimybėmis, buvo gautos 298 nuotraukos. Kadangi nuostoliai nuo 2500 epochos stabiliai laikėsi ir nustojo mažėti, reiškia norint pasiekti dar didesnį tinklo tikslumą, reikia sužymėti daugiau duomenų. Galutinis programos rezultatas pavaizduotas pav. programa. Programos kodas naudoti duomenys yra patalpinti internete, adresas pateiktas priejų skiltyje. Šis darbas įrodo, jog pasiekti aukštą objektų aptikimą kompiuterinės tomografijos nuotraukose, nereikia turėti daug duomenų ir daug resursų. Tai įmanoma padaryti ir naudojant Google Cloud platformą, kuri suteikia nemokamą prieigą prie GPU resursų.



7 pav. Programa.

Išvados

Pasitelkiant Tensorflow karkasą ir neuroninių tinklų pagalba galima sukurti programą, kuri aptinka objektus kompiuterinės tomografijos nuotraukose. Apmokyti tinklą nereikia daug duomenų, vienai klasei apmokyti užtenka 50-ties nuotraukų. Jos yra modifikuojamos: pridedamas įvairių stiprumo triukšmas. Po maždaug 3000 epochų, tinklas aptiko plaučius su 0.02 nuostoliais naudojant MSE nuostolių funkciją.

Literatūra

[Ala16] Jay Alammar. A visual and interactive guide to the basics of neural networks, 2016. URL: http://jalammar.github.io/visual-interactive-guide-basics-neural-networks/ (tikrinta 2020-04-30).

[CS220] CS231n. Linear classification: support vector machine, softmax, 2020. URL: https://cs231n.github.io/linear-classify (tikrinta 2020-04-30).

[Dab17] Imad Dabbura. Gradient descent algorithm and its variants, 2017. URL: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3 (tikrinta 2020-04-30).

[Dos19] Sanket Doshi. Various optimization algorithms for training neural network, 2019. URL: https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6 (tikrinta 2020-04-30).

[For17] Eddie Forson. Understanding ssd multibox — real-time object detection in deep learning, 2017. URL: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab (tikrinta 2020-05-23).

[Gao17] Hao Gao. Faster r-cnn explained, 2017. URL: https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8 (tikrinta 2020-04-30).

[GBC16] Ian Goodfellow, Yoshua Bengio ir Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Glo20] ML Glossary. Activation functions, 2020. URL: https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html (tikrinta 2020-04-30).

[Hin18] Knut Hinkelmann. Neural networks, 2018. URL: http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf (tikrinta 2020-04-30).

[Kat19] Chayan Kathuria. Regression — why mean square error?, 2019. URL: https://towardsdatascience.com/https-medium-com-chayankathuria-regression-why-mean-square-error-a8cad2a1c96f (tikrinta 2020-04-30).

[Kei10] N.L.W. Keijsers. Neural networks, 2010. URL: http://www.sciencedirect.com/science/article/pii/B9780123741059004937 (tikrinta 2020-06-05).

[Kha19] Renu Khandelwal. Computer vision: instance segmentation with mask r-cnn, 2019. URL: https://towardsdatascience.com/computer-vision-instance-segmentation-with-mask-r-cnn-7983502fcad1 (tikrinta 2020-04-30).

[Meh17] Rao DS. Mehta VK Deb PS. Application of computer techniques in medicine, 2017. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6257447/.

[NVI20a] NVIDIA. Cuda zone, 2020. URL: https://developer.nvidia.com/cuda-zone (tikrinta 2020-04-30).

[NVI20b] NVIDIA. Gpu support, 2020. URL: https://www.tensorflow.org/install/gpu (tikrinta 2020-04-30).

[Opp] Artem Oppermann. Activation functions in neural networks. URL: https://www.deeplearning-academy.com/p/ai-wiki-activation-functions (tikrinta 2020-05-23).

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. arXiv: 1609.04747 [cs.LG].

[Seh18] Shirag Sehra. Decision trees explained easily, 2018. URL: https://medium.com/@chiragsehra42/decision-trees-explained-easily-28f23241248 (tikrinta 2020-04-30).

[Sey16] Andrew Lewis Seyedali Mirjalili. The whale optimization algorithm, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0965997816300163?casa_token=yI9V7so44bIAAAAA:y_34ijm8Gc4JBSknYd7cH2hqfk2K4h6dpYxZBTR7RGCEa89wP2b5Ohl_FqhK8t8gG9r4qdmGCg (tikrinta 2020-04-30).

[Ten] TensorFlow. Tensorflow model garden. URL: https://github.com/tensorflow/models/tree/master/research (tikrinta 2020-05-30).

[Wal19] Michael Walter. 4 key reasons ai won't replace radiologists, 2019. URL: https://www.radiologybusiness.com/topics/artificial-intelligence/4-key-reasons-ai-wont-replace-radiologists (tikrinta 2020-05-31).