Prepoznavanje osoba temeljem uha primjenom dubokog učenja

Barjaktarić, Karlo

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

Prepoznavanje osoba temeljem uha primjenom dubokog učenja

Rijeka, rujan 2023.

Karlo Barjaktarić

0069089628

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

Prepoznavanje osoba temeljem uha primjenom dubokog učenja

Mentor dr. sc. Kristijan Lenac

Komentor v. asist. dr. sc. Diego Sušanj

Rijeka, rujan 2023.

Karlo Barjaktarić

0069089628

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 20. ožujka 2023.

Zavod: **Zavod za računarstvo**
Predmet: **Algoritmi i strukture podataka**
Grana: **2.09.04 umjetna inteligencija**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Karlo Barjaktarić (0069089628)**
Studij: Sveučilišni prijediplomski studij računarstva

Zadatak: **Prepoznavanje osoba temeljem uha primjenom dubokog učenja / Ear recognition using deep learning**

Opis zadatka:

U radu je potrebno istražiti i usporediti primjene metoda dubokog učenja za identifikaciju osoba temeljem uha. Potrebno je proučiti te implementirati odabrane modele dubokog učenja. Ispitati implementirana rješenja na odabranom skupu podataka.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Prof. dr. sc. Kristijan Lenac

Dr. sc. Diego Sušanj (komentor)

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2023.

Karlo Barjaktarić

Zahvala

Zahvaljujem se mentoru dr. sc. Kristijanu Lencu i komentoru v. asist. dr. sc. Diegu Sušnju na pruženoj pomoći i smjernicama tijekom izrade ovog rada.
Također, zahvaljujem se svojoj obitelji i svima koji su me podržavali za vrijeme studiranja na preddiplomskom studiju.
Zahvaljujem se Fakultetu računarstva i informatike Sveučilišta u Ljubljani čiji su resursi korišteni.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. UVOD

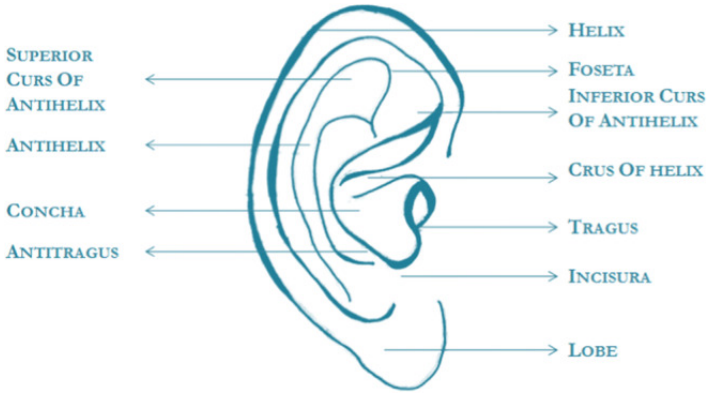
Potraga za sigurnim i pouzdanim sustavom za identificiranje potaknula je istraživanje i razvoj brojnih područja među kojima su računalna vizija (eng. *computer vision)* i inteligentni sustavi. Većina sustava koristi biometriju za identifikaciju ljudi. Razlog tome je fizička nepromjenjivost dijelova ljudskog tijela tijekom vremena. Biometrija je znanstvena i tehnološka disciplina koja se bavi proučavanjem i mjerenjem fizioloških i bihevioralnih karakteristika pojedinaca. To uključuje lice, šarenicu, otisak prsta, dlana, geometriju šake, glas i potpis, što su najkorišteniji podaci. Većina ovih biometrijskih podataka zahtijeva neposrednu suradnju subjekta kako bi se sakupili za identifikaciju. Mana takve nužnosti najbolje se uočila u vremenu pandemije COVID-19 kada je cijela ljudska zajednica bila primorana nositi maske i držati sigurnosni razmak. Sustavi za prepoznavanje lica doživljavali su poraze i postojala je potreba za preradom postojećih sustava. Tada se uho, kao jedna od nespomenutih karakteristika, pokazala kao savršena predispozicija za potrebe identifikacije. Dakle, beskontaktni, nekooperativni biometrijski sustav kao što je biometrija uha bila je potreba koja se od tada znatno razvila.

Naravno, povijest prepoznavanja korištenjem uha seže od drevnih civilizacija, gdje su se pojedinci međusobno identificirali na temelju karakterističnih obilježja uha. Međutim, tek je u digitalnoj eri to područje dobilo znanstvenu utemeljenost. Tijekom posljednjih nekoliko desetljeća, istraživači su iskoristili snagu računalnih tehnika za razvoj robusnih i preciznih sustava za prepoznavanje uha.

Metoda prikupljanja podataka ljudskog uha je beskontaktna i nenametljiva te ne uključuje kooperativnost čovjeka kojeg pokušavamo prepoznati. Biometrija uha može poslužiti kao dopuna drugim biometrijskim modalitetima u sustavima automatskog prepoznavanja i pružiti naznake identiteta kada su druge informacije nepouzdane ili čak nedostupne. U nadzornim aplikacijama, kada prepoznavanje lica može imati poteškoća s profilnim licima, uho može poslužiti kao izvor informacija o identitetu čovjeka. Automatizirana ljudska identifikacija pomoću slika ušiju sve se više proučava radi moguće komercijalne primjene.

Ljudsko uho ima stabilnu strukturu od rođenja i jedinstveno je za svakog pojedinca. Mnoge medicinske studije pokazale su da se značajne promjene u obliku uha događaju prije 8. godine i nakon 70. godine. Raspodjela boja uha je gotovo ujednačena. Položaj uha je gotovo u sredini profila lica. Jednostavna digitalna kamera kao što je CCTV kamera prikladna je za snimanje slika uha. Ono sadrži stabilne i pouzdane informacije kao i strukturu oblika koja ne pokazuje drastične promjene s godinama. *Slika 1.1.* prikazuje vidljivu strukturu ljudskog uha i njegove različite morfološke komponente uključujući: spiralu, antiheliks, tagus, antitragus, režanj, školjku i druge dijelove. Dok je

vidljiva vanjska struktura relativno jednostavna, promjene između dva uha su dovoljno očite čak i kod jednojajčanih blizanaca. Također, slika ljudskog uha ima jednoliku distribuciju boja za površinu uha i slike uha su nepromjenjive u odnosu na izraze lica. Stoga je analiza slika uha za izdvajanje takvih jedinstvenih i prepoznatljivih obilježja za identifikaciju pojedinaca i provjeru njihovog identiteta aktivna istraživačka tema i nova inteligentna biometrijska aplikacija. Longitudinalne studije iz Indije i Europe otkrile su da se duljina uha kod ljudi povećava s godinama, dok širina i struktura ostaju relativno konstantne [1].



Slika 1.1. Dijelovi ljudskog uha

Glavni dio ovog rada fokusirat će se na prikupljanje, strukturiranje, imenovanje skupa podataka te učenje i klasifikaciju, koristeći se pritom neuronskom mrežom koja je glavni alat treniranja kako bi se postigla visoka točnost u prepoznavanju ušiju, odnosno njihovih subjekata. Dotaknut će se najvažnijih metoda i tehnika za provođenje tih procesa te na konkretnom primjeru ilustrirati koncepte prepoznavanja uha. Sagledat će dobivene rezultate i pokušati otkriti faktore koji su utjecali na ishod.

2. DUBOKO UČENJE

Duboko učenje je vrsta strojnog učenja i umjetne inteligencije koja oponaša način na koji ljudi stječu određene vrste znanja. Modeli dubokog učenja mogu naučiti obavljati zadatke klasifikacije i prepoznavati uzorke na fotografijama, zvuku, tekstu i drugim različitim podacima. Također se koristi za automatizaciju zadataka koji inače zahtijevaju ljudsku inteligenciju, kao što je opisivanje slika ili transkripcija audio datoteka. Duboko učenje važan je element znanosti o podacima, uključujući statistiku i modeliranje prema predikciji. Izuzetno je koristan znanstvenicima koji se bave podacima koji imaju zadatak prikupljanja, analiziranja i tumačenja velikih količina podataka. Duboko učenje postiže točnost prepoznavanja na višim razinama nego ikad prije. Nedavni napredak u dubokom učenju poboljšan je do točke gdje duboko učenje nadmašuje ljude u nekim zadacima kao što je klasificiranje objekata na slikama. Zahtijeva velike količine imenovanih podataka. Na primjer, razvoj automobila bez vozača zahtijeva milijune slika i tisuće sati videa. Također zahtijeva značajnu računalnu snagu. GPU-ovi visokih performansi imaju paralelnu arhitekturu koja je za to veoma učinkovita. U kombinaciji s klasterima ili računarstvom u oblaku, to omogućuje razvojnim timovima da smanje vrijeme treninga za mrežu dubokog učenja s tjedana na sate ili manje.

Duboko učenje omogućuje računalu učenje na primjeru. Dobra usporedba toga je malo dijete koje uči govoriti. Dijete uči što je mačka, a što nije, pokazujući na životinju i izgovarajući riječ „mačka“. Roditelj potvrđuje ili negira djetetovu opservaciju/klasifikaciju. Kako dijete nastavlja pokazivati na životinje, postaje svjesnije osobina koje posjeduju sve mačke. Ono što dijete radi, a da to ni ne zna, jest razjašnjavanje složene apstrakcije: pojam mačke. Ono to čini izgradnjom hijerarhije u kojoj je svaka razina apstrakcije stvorena znanjem koje je stečeno iz prethodnog sloja hijerarhije. Programi dubokog učenja imaju više slojeva međusobno povezanih čvorova, pri čemu se svaki sloj nadograđuje na prethodni kako bi poboljšao i optimizirao predviđanja i klasifikacije. Izvodi nelinearne transformacije svog ulaza i koristi ono što nauči za stvaranje statističkog modela kao izlaza. Iteracije se nastavljaju dok izlaz ne dosegne prihvatljivu razinu točnosti. Broj slojeva obrade kroz koje podaci moraju proći je ono što je inspiriralo pridjev „duboko“.

Kao što je već rečeno, duboko učenje je specijalizirani oblik strojnog učenja. Tijek rada strojnog učenja započinje ručnim izdvajanjem relevantnih značajki iz slika. Značajke se zatim koriste za stvaranje modela koji kategorizira objekte na slici. Uz tijek rada dubokog učenja, relevantne značajke automatski se izdvajaju iz slika. Osim toga, duboko učenje izvodi "end-to-end“ učenje – gdje se mreži daju neobrađeni podaci i zadatak koji treba izvršiti, kao što je klasifikacija, i ona uči kako to učiniti automatski. Još jedna ključna razlika je da se algoritmi dubokog učenja skaliraju s podacima, dok

plitko učenje konvergira. Plitko učenje odnosi se na metode strojnog učenja koje se nalaze na određenoj razini izvedbe kada se doda više primjera i podataka za treniranje. U strojnom učenju ručno se biraju značajke i klasifikator za sortiranje slika. Uz duboko učenje, ekstrakcija značajki i koraci modeliranja su automatski.

Poput ljudskog mozga koji ima milijune međusobno povezanih neurona koji rade zajedno kako bi naučio informacije, duboko učenje uključuje neuronske mreže izgrađene od više slojeva softverskih čvorova koji rade zajedno. Modeli dubokog učenja treniraju se pomoću velikog skupa označenih podataka i arhitektura neuronskih mreža. Duboko učenje može se koristiti za nadzirano, nenadzirano kao i za ojačano strojno učenje. Koristi razne načine za njihovu obradu.

Nadzirano strojno učenje je tehnika strojnog učenja u kojoj neuronska mreža uči predviđati ili klasificirati podatke na temelju označenih skupova podataka. Ovdje unosimo obje ulazne značajke zajedno s ciljnim varijablama. Neuronska mreža uči predviđati na temelju cijene ili pogreške koja proizlazi iz razlike između predviđenog i stvarnog cilja, ovaj proces je poznat kao povratno širenje. Algoritmi dubokog učenja kao što su konvolucijske neuronske mreže, rekurentne neuronske mreže koriste se za mnoge nadzirane zadatke kao što su klasifikacija i prepoznavanje slika, analiza osjećaja, jezični prijevodi itd.

Nenadzirano strojno učenje je tehnika strojnog učenja u kojoj neuronska mreža uči otkrivati obrasce ili grupirati skup podataka na temelju neoznačenih skupova podataka. Ovdje nema ciljnih varijabli, dok stroj mora sam odrediti skrivene obrasce ili odnose unutar skupova podataka. Algoritmi dubokog učenja poput autokodera i generativnih modela koriste se za nenadzirane zadatke kao što su klasteriranje, smanjenje dimenzionalnosti i otkrivanje anomalija.

Ojačano strojno učenje je tehnika strojnog učenja u kojoj agent uči donositi odluke u okruženju kako bi maksimizirao funkciju nagrade. Agent stupa u interakciju s okolinom poduzimanjem radnji i promatranjem dobivenih nagrada. Duboko učenje može se koristiti za učenje politika ili skupa radnji koje maksimiziraju kumulativnu nagradu tijekom vremena. Algoritmi dubokog učenja kao što su *Deep Q* mreže i *Deep Deterministic Policy Gradient* (DDPG) koriste se za jačanje robotike, uspješnosti igranje igrica itd [2].

2.1. Metode dubokog učenja

Za stvaranje jakih modela dubokog učenja mogu se koristiti različite metode. Ove tehnike uključuju pad stope učenja (eng. *learning rate decay*), prijenos učenja (eng. *transfer learning*), treninga od nule (eng. *training from scratch*), ekstrakcija značajki i odustajanje (eng. *dropout*).

Stopa učenja je parametar koji definira sustav ili postavlja uvjete za njegov rad prije procesa učenja i koji kontrolira koliku će promjenu model doživjeti kao odgovor na procijenjenu pogrešku svaki put kada se težine modela promijene. Stope učenja koje su previsoke mogu rezultirati nestabilnim procesima treniranja ili učenjem suboptimalnog skupa težina. Stope učenja koje su premale mogu proizvesti dugotrajan proces treninga koji ima potencijal da zapne.

Metoda opadanja stope učenja je proces prilagođavanja stope učenja radi povećanja performansi i smanjenja vremena treninga. Najlakše i najčešće prilagodbe stope učenja tijekom treninga uključuju tehnike za smanjenje stope učenja tijekom vremena.

Prijenos učenja uključuje usavršavanje prethodno uvježbanog modela te zahtijeva sučelje za unutarnje dijelove već postojeće mreže. Prvo, korisnici dodaju postojećoj mreži nove podatke koji sadrže prethodno nepoznate klasifikacije. Nakon što se naprave prilagodbe na mreži, novi zadaci mogu se izvoditi s specifičnijim sposobnostima kategorizacije. Prednost ove metode je što zahtijeva mnogo manje podataka od drugih, čime se vrijeme izračuna smanjuje na minute ili sate.

Trening od nule zahtijeva od programera prikupljanje velikog, imenovanog skupa podataka i konfiguriranje mrežne arhitekture koja može naučiti značajke i model. Ova tehnika je posebno korisna za nove aplikacije, kao i aplikacije s mnogo izlaznih kategorija. Međutim, općenito gledano, to je rjeđi pristup jer zahtijeva neumjerene količine podataka, zbog čega treninga traje danima ili tjednima.

Ekstrakcija značajki je više specijalizirani pristup dubokom učenju koji koristi mrežu kao ekstraktora značajki. Budući da su svi slojevi zaduženi za učenje određenih značajki iz slika, te značajke možemo povući iz mreže u bilo kojem trenutku tijekom procesa treninga. Te se značajke zatim mogu koristiti kao ulazne informacije za model strojnog učenja kao što su potporni vektorski strojevi (SVM).

Metodom odustajanja pokušava se riješiti problem preadaptacije (eng. overfitting) u mrežama s velikom količinom parametara. Preadaptacija je čest problem u strojnom i dubokom učenju gdje model postaje izrazito vezan za šum i nedosljednosti podataka u treningu, zbog čega ima lošu izvedbu na prethodno neviđenim podacima, dok se odlično ponaša s materijalima iz trening faze. Dokazano je da metoda odustajanja može poboljšati izvedbu neuronskih mreža na zadacima učenja pod nadzorom u područjima kao što su prepoznavanje govora, klasifikacija dokumenata i računalna biologija [3].

2.2. Nedostaci i limitacije

Osim brojnih prednosti i koristi koje sustavi dubokog učenja nude, postoje i nedostaci koje treba spomenuti.

Oni uče kroz opažanja, što znači da znaju samo ono dostupno u podacima na kojima su trenirali. Ako korisnik ima malu količinu podataka ili oni dolaze iz jednog specifičnog izvora koji nije nužno reprezentativan za šire funkcionalno područje, modeli ne uče na način koji je moguće generalizirati.

Pitanje pristranosti također je veliki problem za modele dubokog učenja. Ako model trenira na podacima koji sadrže pristranosti, model reproducira te pristranosti u svojim predviđanjima. Ovo je bio mučan problem za programere dubokog učenja jer modeli uče razlikovati na temelju suptilnih varijacija u elementima podataka. Programeru često nisu eksplicitno razjašnjeni čimbenici koje određuje da su važni. To znači, na primjer, da model prepoznavanja uha može donositi odluke o karakteristikama ljudi na temelju stvari kao što su rasa ili spol, a da programer toga nije svjestan.

Kao što je već bilo spomenuto, stopa učenja postaje veliki izazov za modele dubokog učenja. Ako je stopa previsoka, tada model konvergira prebrzo, dovodeći do neoptimalnog rješenja. Ako je stopa preniska, proces može zapeti, a još je teže doći do rješenja.

Hardverski zahtjevi za modele dubokog učenja također stvaraju ograničenja. Višejezgrene grafičke procesorske jedinice (GPU) visokih performansi i druge slične procesorske jedinice potrebne su kako bi se osigurala poboljšana učinkovitost i smanjena potrošnja vremena. Međutim, ove jedinice su skupe i troše velike količine energije. Ostali hardverski zahtjevi uključuju RAM i pogon tvrdog diska ili SSD pogon temeljen na RAM-u.

Zahtijeva velike količine podataka. Nadalje, snažniji i točniji modeli trebaju više parametara, koji zauzvrat zahtijevaju više podataka.

Nedostatak višezadačnosti. Jednom istrenirani, modeli dubokog učenja postaju nefleksibilni i ne mogu se nositi s istovremenim obavljanjem više zadataka. Oni mogu pružiti učinkovita i točna rješenja, ali samo za jedan specifičan problem. Čak bi i rješavanje sličnog problema zahtijevalo ponovno osposobljavanje sustava.

Nedostatak obrazloženja. Svaka aplikacija koja zahtijeva razmišljanje, kao što je programiranje ili primjena znanstvene metode. Dugoročno planiranje i manipulacija podacima nalik na algoritam potpuno je izvan onoga što trenutne tehnike dubokog učenja mogu učiniti, čak i s velikim količinama podataka [3].

2.3. Primjena

Primjena dubokog učenja provodi se u industrijama od automatizirane vožnje do medicinskih uređaja. Automobilska industrija koristi duboko učenje za automatsko otkrivanje objekata kao što su znakovi, stop i semafori. Osim toga, koristi se za otkrivanje pješaka, što pomaže u smanjenju nesreća. U zrakoplovstvu i obrani duboko učenje koristi se za identifikaciju objekata sa satelita koji lociraju interesna područja i identificiranje sigurnih ili nesigurnih zona za trupe. Istraživačima raka u medicini služi za automatsko otkrivanje stanica raka. U industrijskoj automatizaciji duboko učenje pomaže u poboljšanju sigurnosti radnika oko teških strojeva automatskim otkrivanjem kada su ljudi ili predmeti unutar nesigurne udaljenosti od strojeva. Ono se koristi i u automatiziranom prevođenju sluha i govora. Na primjer, uređaji za kućnu pomoć koji reagiraju na vaš glas i znaju vaše preferencije pokreću aplikacije dubokog učenja [3].

3. NEURONSKA MREŽA

Neuronska mreža niz je algoritama koji nastoje prepoznati temeljne odnose u skupu podataka kroz proces koji oponaša način na koji funkcionira ljudski mozak. U tom smislu, neuronske mreže se odnose na sustave neurona, organske ili umjetne prirode. Funkcionira slično neuronskoj mreži ljudskog mozga. "Neuron" u neuronskoj mreži je matematička funkcija koja prikuplja i klasificira informacije prema specifičnoj arhitekturi. Mreža ima veliku sličnost sa statističkim metodama kao što su prilagodba krivulje i regresijska analiza. Neuronske mreže mogu se prilagoditi promjenjivom unosu tako da mreža generira najbolji mogući rezultat bez potrebe za redizajniranjem izlaznih kriterija. Koncept neuronskih mreža, koji svoje korijene vuče iz umjetne inteligencije, brzo dobiva na popularnosti u razvoju sustava trgovanja. Neuronske mreže, u svijetu financija, pomažu u razvoju takvih procesa kao što su predviđanje vremenskih nizova, algoritamsko trgovanje, klasifikacija vrijednosnih papira, modeliranje kreditnog rizika i konstruiranje vlastitih pokazatelja i cjenovnih derivata [4].

3.1. Povijest neuronskih mreža

Ideju modeliranja biološke funkcije mozga zasnivaju autori McCulloch i Pitts 1943., predlažući uporabu logičkih sklopova za opisivanje neuro-aktivnosti. Kako je u to doba razvoj računala bio tek u fazi osnovne izvedbe, trebalo je proći neko vrijeme da se razvije model koji bi računalnim programom mogao djelomično opisivati biološku aktivnost te vrste. Krajem pedesetih godina osnove modernih neuronskih mreža za implementaciju računalnim programima postavio je eksperimentalni psiholog Rosenblatt predlažući model „Perceptron“ (1958). Usprkos tomu što je u svojem radu i kasnijem izvješću Rosenblatt koristio ideju Perceptron modela s više međuslojeva, krajem šezdesetih godina autori Minsky i Papert oštro kritiziraju njegov rad zasnovan na jednostavnom Perceptron modelu, što u konačnici dovodi do znatnog smanjenja novčane potpore za istraživanja na tom području. U svojoj knjizi Minsky i Papert uzimaju za primjer Perceptron bez međuslojeva, te pokazuju kako se tim modelom ne može riješiti jednostavan problem opisivanja logičke XOR funkcije. Sažetak njihove kritike sadržan je u riječima: „Kažeš kako će računala biti svjesna, a ne mogu riješiti XOR problem“. Autori su bili izrazito oštri prema Rosenblattu te su ideju Perceptrona okarakterizirali bezvrijednom, što je eksplicitno izrečeno u citatu iz uvoda na četvrtoj stranici njihove knjige „*Most of this writting ... is without scientific value and we will not usually refer by name to the works we criticize*“. Samog Rosenblatta kritika je teško pogodila, zbog čega je

napustio istraživanja, da bi dvije godine kasnije, na svoj 43. rođendan, tragično preminuo u pomorskoj nesreći u zaljevu Chesapeake [5].

Henry J. Kelleyu pripisuju se zasluge za razvoj osnova kontinuiranog modela povratne propagacije 1960. Stuart Dreyfus je 1962. razvio jednostavniju verziju temeljenu samo na lančanom pravilu. Iako je koncept povratnog širenja (širenje pogrešaka unatrag u svrhu treninga) postojao ranih 1960-ih, bio je nespretan i neučinkovit te će postati koristan tek 1985. Najraniji napori u razvoju algoritama dubokog učenja potječu od Alexeya Grigoryevicha Ivakhnenka i Valentina Grigor′evicha Lape koji su 1965. koristili modele s polinomnim aktivacijskim funkcijama, koje zatim su statistički analizirani. Najbolje statistički odabrane značajke iz svakog sloja proslijeđene su na sljedeći sloj.

Prve "konvolucijske neuronske mreže" koristio je Kunihiko Fukushima 70-ih godina. Fukushima je dizajnirao neuronske mreže s višestrukim skupnim i konvolucijskim slojevima. Godine 1979. razvio je umjetnu neuronsku mrežu, nazvanu „Neocognitron“, koja je koristila hijerarhijski, višeslojni dizajn. Ovaj je dizajn omogućio računalu da "nauči" prepoznavati vizualne obrasce. Mreže su nalikovale modernim verzijama, ali su bile uvježbane strategijom pojačanja ponavljajuće aktivacije u više slojeva, koja je s vremenom dobivala na snazi. Osim toga, dizajn Fukushime omogućio je ručno podešavanje važnih značajki povećanjem "težine" određenih spojeva. Tijekom 1970-ih nastupilo je prvo „zatišje“ umjetne inteligencije, rezultat obećanja koja se nisu mogla održati.

Godine 1989., Yann LeCun pružio je prvu praktičnu demonstraciju povratnog širenja u Bell Labsu. Kombinirao je konvolucijske neuronske mreže s povratnim širenjem na očitane "rukom pisane" znamenke. Taj je sustav na kraju korišten za očitavanje brojeva rukom pisanih čekova. U to vrijeme započelo je drugo „zatišje“ umjetne inteligencije (1985.-90.). Razni pretjerano optimistični pojedinci preuveličali su "neposredan" potencijal umjetne inteligencije, prekršivši očekivanja i razljutivši ulagače. Ljutnja je bila toliko jaka da je fraza Umjetna inteligencija dosegla status pseudoznanosti. Godine 1995. Dana Cortes i Vladimir Vapnik razvili su stroj potpornih vektora (sustav za mapiranje i prepoznavanje sličnih podataka). LSTM (duga kratkotrajna memorija) za rekurentne neuronske mreže razvili su 1997. Sepp Hochreiter i Juergen Schmidhuber. Sljedeći značajan evolucijski korak za duboko učenje dogodio se 1999. godine, kada su računala počela postajati brža u obradi podataka i kada su razvijeni GPU-i. Brža obrada, s GPU-ovima koji obrađuju slike, povećala je računalne brzine za 1000 puta u razdoblju od 10 godina.

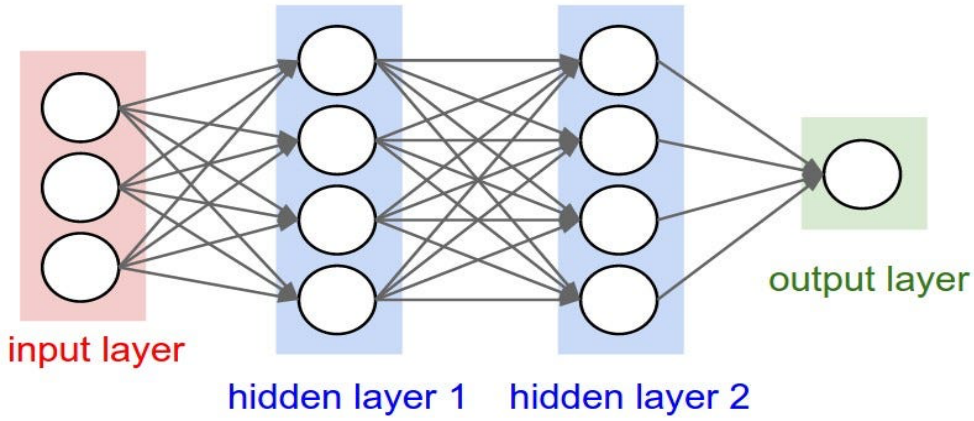
Oko 2000. godine pojavio se problem nestajanja gradijenta. Otkriveno je da "značajke" formirane u nižim slojevima nisu bile naučene od strane viših slojeva, jer nikakav signal učenja nije stigao do ovih slojeva. Dva rješenja korištena za rješavanje ovog problema bila su prethodni trening sloj po sloj i razvoj dugog kratkoročnog pamćenja. Godine 2001., istraživačko izvješće META grupe

(sada pod imenom Gartner) opisalo je izazove i mogućnosti rasta podataka kao trodimenzionalne što je označilo poziv za pripremu uzleta „Big Data“ koja je uslijedila. Godine 2009. Fei-Fei Li, profesor umjetne inteligencije na Stanfordu, pokrenuo je „ImageNet“, skupivši besplatnu bazu podataka s više od 14 milijuna označenih slika potrebnih za treniranje neuronskih mreža.

Do 2011. brzina GPU-a značajno je porasla, što je omogućilo treniranje konvolucijskih neuronskih mreža "bez" sloj-po-sloj prethodnog treninga. Jedan primjer je AlexNet, konvolucijska neuronska mreža čija je arhitektura pobijedila na nekoliko međunarodnih natjecanja tijekom 2011. i 2012. 2012. godine Google Brain objavio je rezultate neobičnog projekta poznatog kao „*The Cat Experiment*“. Projekt slobodnog duha istraživao je poteškoće "učenja bez nadzora". Duboko učenje koristi "nadzirano učenje", što znači da se konvolucijska neuronska mreža trenira pomoću označenih podataka. Korištenjem nenadziranog učenja, konvolucijska neuronska mreža dobiva neoznačene podatke, a zatim se od nje traži da traži ponavljajuće obrasce. Deset milijuna "neoznačenih" slika nasumično je preuzeto s YouTubea i na kraju treninga otkriveno je da jedan neuron u najvišem sloju snažno reagira na slike mačaka i jedan koji reagira na ljudska lica. Učenje bez nadzora ostaje značajan cilj u području dubokog učenja. Generativna kontradiktorna neuronska mreža (GAN) predstavljena je 2014. Uz GAN, dvije neuronske mreže igraju jedna protiv druge u igri. Cilj igre je da jedna mreža oponaša fotografiju i prevari protivnika da povjeruje da je prava. Protivnik, naravno, traži mane. Igra se sve dok gotovo savršena fotografija ne prevari protivnika [6, 7].

3.2. Kako neuronska mreža radi?

Neuronska mreža ima mnogo slojeva. Svaki sloj obavlja određenu funkciju, a što je mreža složena, to je slojeva više. Zato se neuronska mreža naziva i višeslojni perceptron. Najčišći oblik neuronske mreže, koji je također poznat kao sloj čvora, ima tri sloja: ulazni sloj, skriveni sloj i izlazni sloj. Kao što nazivi sugeriraju, svaki od ovih slojeva ima određenu svrhu. Ovi slojevi se sastoje od čvorova. Može postojati više skrivenih slojeva u neuronskoj mreži prema zahtjevima. Ulazni sloj preuzima ulazne signale i prenosi ih na sljedeći sloj. Prikuplja podatke iz vanjskog svijeta. Svaki od pojedinačnih čvorova može se usporediti s jedinstvenim linearnim regresijskim modelom. Skriveni sloj obavlja sve pozadinske zadatke izračuna. Mreža može imati i nula skrivenih slojeva. Međutim, neuronska mreža ima barem jedan skriveni sloj. Izlazni sloj prenosi konačni rezultat izračuna skrivenog sloja. Ovaj koncept može se vidjeti na *Slici 3.1.*



Slika 3.1. Slojevi neuronske mreže

Neuronska mreža se sastoji od mnogo perceptronskih slojeva zbog čega ima naziv „višeslojni perceptron“. Ovi se slojevi nazivaju i skriveni slojevi gustih slojeva. Sastoje se od mnogo perceptronskih neutrona. Oni su primarna jedinica koja zajedno radi na formiranju perceptronskog sloja. Ovi neuroni primaju informacije u skupu ulaza. Kombinacijom ovih numeričkih ulaza s pristranošću i grupom utega proizvodi se jedan izlaz. Svaki neuron uzima u obzir težine i pristranost. Zatim, kombinirana funkcija koristi težinu i pristranost da bi dala izlaz (modificirani ulaz). Djeluje kroz sljedeću jednadžbu (3.1):

$$kombinacija = pristranost + ponderi \times ulazi \; (kombinacija = pristranost + težine \times ulaz) \quad (3.1)$$

Nakon toga, aktivacijska funkcija proizvodi izlaz sa sljedećom jednadžbom (3.2):

$$izlaz = aktivacija\,(kombinacija) \quad (3.2)$$

Ova funkcija određuje kakvu će ulogu imati neuronska mreža. One čine slojeve mreže. Sljedeće su prevladavajuće funkcije aktivacije.

3.2.1. Linearna funkcija

Izlaz linearne funkcije je samo kombinacija neurona (3.3):

$$aktivacija = kombinacija \quad (3.3)$$

Hiperbolička tangentna funkcija je najpopularnija aktivacijska funkcija među neuronskim mrežama. To je sigmoidna funkcija i nalazi se između -1 i +1 (3.4):

$$aktivacija = tangens\,(kombinacija) \qquad (3.4)$$

3.2.2. Logistička funkcija

Logistička funkcija je prilično slična hiperboličkoj tangentnoj funkciji jer je također vrsta sigmoidne funkcije. Međutim, razlikuje se jer se nalazi između 0 i 1 (3.5):

$$aktivacija = 11 + e - kombinacija \qquad (3.5)$$

3.2.3. Funkcija rektificirane linearne jedinice

Drugi naziv za funkciju ispravljene linearne jedinice je ReLU. ReLU je jednak kombinaciji kada je jednak ili veći od nule, a negativan je ako je kombinacija niža od (negativne) nule.

Koraci rada neuronske mreže:

1. Informacije se unose u ulazni sloj koji ih prenosi na skriveni sloj
2. Međusobne veze između dva sloja nasumično dodjeljuju težine svakom ulazu
3. Pristranost se dodaje svakom unosu nakon što se težine pojedinačno pomnože s njima
4. Ponderirani zbroj se prenosi u aktivacijsku funkciju
5. Aktivacijska funkcija određuje koje čvorove treba aktivirati za ekstrakciju značajki
6. Model primjenjuje funkciju aplikacije na izlazni sloj za isporuku izlaza
7. Težine se prilagođavaju, a izlaz se širi unazad kako bi se pogreška svela na minimum

Model koristi funkciju troška za smanjenje stope pogreške, stoga je potrebno mijenjati težine s različitim modelima treninga.

1. Model uspoređuje izlaz s izvornim rezultatom
2. Ponavlja postupak radi poboljšanja točnosti

Model prilagođava težine u svakoj iteraciji kako bi se povećala točnost izlaza [8].

3.3. Vrste neuronskih mreža

3.3.1. Rekurentna neuronska mreža (RNN)

U ovoj mreži, izlaz sloja se sprema i prenosi natrag na ulaz. Na ovaj način čvorovi određenog sloja pamte neke informacije o prošlim koracima. Kombinacija ulaznog sloja umnožak je zbroja težina i značajki. Ponavljajući proces neuronske mreže počinje u skrivenim slojevima. Ovdje svaki čvor pamti neke od informacija svog prethodnog koraka. Model zadržava neke informacije iz svake iteracije, koje kasnije može koristiti. Sustav sam uči kada je njegov ishod pogrešan. Zatim koristi tu informaciju za povećanje točnosti svog predviđanja u povratnom širenju. Najpopularnija primjena RNN-a je tehnologija pretvaranja teksta u govor [8].

3.3.2. Konvolucijska neuronska mreža (CNN)

Ova se mreža sastoji od jednog ili više konvolucijskih slojeva. Konvolucijski sloj prisutan u ovoj mreži primjenjuje konvolucijsku funkciju na ulaz prije nego što ga prenese na sljedeći sloj. Zbog toga mreža ima manje parametara, ali postaje dublja. CNN-ovi se široko koriste u obradi prirodnog jezika i prepoznavanju slika [8].

3.3.3. Neuronska mreža radijalne osnovne funkcije (RBFNN)

Ova neuronska mreža koristi funkciju radijalne baze. Ova funkcija uzima u obzir udaljenost točke od središta. Ove mreže se sastoje od dva sloja. Skriveni sloj kombinira značajke s funkcijom radijalne baze i prenosi izlaz na sljedeći sloj. Sljedeći sloj izvodi isto dok koristi izlaz prethodnog sloja. Neuronske mreže radijalne bazične funkcije koriste se u elektroenergetskim sustavima [8].

3.3.4. *Feedforward* neuronska mreža (FNN)

Ovo je najčišći oblik umjetne neuronske mreže. U ovoj mreži podaci se kreću u jednom smjeru, tj. od ulaznog sloja prema izlaznom sloju. U ovoj mreži izlazni sloj prima zbroj umnožaka ulaza i njihovih težina. U ovoj neuronskoj mreži nema povratnog širenja. Ove mreže mogu imati mnogo ili nijedan skriveni sloj. Lakše ih je održavati i nalaze primjenu u prepoznavanju lica [8].

3.3.5. Modularna neuronska mreža

Ova mreža posjeduje nekoliko mreža koje funkcioniraju neovisno. Svi oni obavljaju specifične zadatke, ali ne djeluju jedni na druge tijekom procesa izračunavanja. Na taj način modularna neuronska mreža može izvršiti vrlo složen zadatak s puno većom učinkovitošću. Te je mreže zahtjevnije održavati u usporedbi s jednostavnijim mrežama (kao što je FNN), ali također daju brže rezultate za složene zadatke [8].

3.4. Prednosti

3.4.1. Izvedba na problemima s mnogo varijabli

Za problem sa strogim skupom pravila i zahtjeva te s ograničenim unosima, stroju je lako pronaći odgovor. Glavni primjer ovdje je kalkulator. Pravila matematike nikad se ne krše i relativno ih je jednostavno slijediti. Unesimo dvije varijable (dva realna broja) i lako ćemo dobiti njihov zbroj. Ali prepoznavanje govornih obrazaca ili dijagnosticiranje bolesti zahtijevaju mnogo više varijabli. Strojevi trebaju "razumjeti" ne samo što traže, već i kako se to razlikuje od šuma i kako se na nju može utjecati na različite načine. Neuronske mreže su izrazito dobre u rješavanju ovih velikih problema - ponekad čak i bolje od ljudi [9].

3.4.2. Inženjering značajki

Neuronske mreže također su nevjerojatno dobre u pronalaženju točnih značajki koje se pripisuju problemu, poznato kao inženjering značajki. Recimo da pokušavamo naučiti algoritam kako igrati i pobijediti igru „Go“, kao što je Google učinio. Go je igra s praktički neograničenim mogućnostima poteza i bez jasnog načina određivanja je li potez "dobar" ili "loš", posebno u ranoj fazi igre. Da bi stroj učio učinkovito, mora biti sposoban naučiti kako prepoznati što čini potez manje ili više vjerojatnim da će ga približiti pobjedi. Neuronske mreže mogu to učiniti. Mogu stvoriti nove kategorije za razmatranje i primijeniti ih na svoj rad [9].

3.4.3. Primjenjivost

Neuronske mreže također imaju moć fleksibilnosti. Jednom uspostavljeni, mogu se primijeniti na gotovo sve, bilo da pomažu ljudima da uoče probleme koji ometaju njihovu produktivnost ili

poboljšavaju obrasce zračnog prometa za glađe letove. Temeljna funkcija neuronske mreže je učinkovito naučiti nešto, pa ako postoji sustav koji može naučiti prepoznavati uzorke, mogao bi prepoznati uzorke u gotovo svim domenama [9].

3.5. Mane

3.5.1. Zahtjevi za podatke

Za početak, sve neuronske mreže moraju proći kroz razdoblje "učenja" u kojem počinju prepoznavati obrasce i usavršavati se. Iako smo u mogućnosti "podučavati" strojeve učinkovitije nego ikad prije, još uvijek postoji ogroman zahtjev za podacima prije nego što ti algoritmi počnu biti učinkoviti. Ovisno o primjeni, to bi moglo zahtijevati 10 000 zasebnih skupova podataka ili više. Ovo bi moglo značajno povećati vrijeme koje je potrebno da neuronska mreža postane učinkovita ili ograničiti moguće primjene [9].

3.5.2. Visoke cijene i „princip crne kutije“

Neuronske mreže također su skupe i dugotrajne za razvoj. Računalni procesi potrebni za rukovanje svim tim varijablama i svim tim dolaznim skupovima podataka zahtijevaju CPU i GPU snagu izvan opsega normalnog sustava. To ga čini obeshrabrujućim pothvatom za neke inženjere i povećava cijenu funkcionalnog sustava, čineći ga težim za korištenje u predviđene svrhe. Kao što možete zamisliti, stvarnost razvoja neuronske mreže je mnogo dublja i kompliciranija nego što se može implicirati jednostavnom, sveobuhvatnom definicijom. Nevjerojatno je teško naučiti kako razviti neuronsku mrežu, a mnogi inženjeri koji započnu putovanje na kraju odustanu. Povrh toga, zbog zamršenosti neuronskih mreža, često transparentnost da vidimo kako naši algoritmi dolaze do svojih zaključaka manjka. Može se utvrditi jesu li njihovi nalazi točni, ali ne i točno vidjeti kako su došli do tih odgovora, što ih čini još tajanstvenijim, čak i za profesionalce [9].

3.5.3. Dugoročni potencijal.

Neuronske mreže već su odgovorne za značajan napredak u području umjetne inteligencije, ali u smislu dugoročnog potencijala, možda nemaju toliko snage kao druge mogućnosti, poput metoda kernela ili čak klasične umjetne inteligencije. Postoji čvrsta granica učinkovitosti ili kompliciranosti neuronskih mreža, a ta gornja granica obeshrabruje mnoge istraživače [9].

3.6. Budućnost

Slabosti neuronskih mreža mogle bi se lako nadoknaditi ako bismo ih mogli integrirati s komplementarnom tehnologijom, poput simboličkih funkcija. Teži dio bi bio pronaći način da ovi sustavi rade zajedno kako bi proizveli zajednički rezultat, a inženjeri već rade na tome.

Sve ima potencijal za povećanje u smislu snage i složenosti. Uz tehnološki napredak, možemo učiniti CPU-e i GPU-e jeftinijima i/ili bržima, omogućujući proizvodnju većih, učinkovitijih algoritama. Također možemo dizajnirati neuronske mreže sposobne za obradu više podataka ili bržu obradu podataka, tako da može naučiti prepoznavati uzorke sa samo 1.000 primjera, umjesto s 10.000. Nažalost, možda postoji gornja granica koliko možemo napredovati u tim područjima - ali tu granicu još nismo dosegnuli, pa ćemo joj vjerojatno težiti u bliskoj budućnosti.

Umjesto da napreduju okomito, u smislu brže procesorske snage i veće složenosti, neuronske mreže bi se mogle, i vjerojatno hoće, širiti i horizontalno, upuštajući se u raznovrsnije primjene. Stotine industrija mogle bi aktivno koristiti neuronske mreže za učinkovitije djelovanje, ciljanje nove publike, razvoj novih proizvoda ili poboljšanje sigurnosti potrošača, a ipak se „kriminalno“ nedovoljno koriste. Šire prihvaćanje, veća dostupnost i veća kreativnost inženjera i trgovaca imaju potencijal za primjenu neuronskih mreža u više aplikacija.

Tehnološki optimisti uživali su u proricanju slavne budućnosti neuronskih mreža, ali one možda neće još zadugo biti dominantan oblik umjetne inteligencije ili rješavanja složenih problema. Za nekoliko godina, čvrsta ograničenja i ključne slabosti neuronskih mreža mogle bi spriječiti njihovo provođenje. Umjesto toga, programeri i potrošači mogu gravitirati prema nekom novom pristupu, pod uvjetom da jedan postane dovoljno pristupačan. S dovoljno potencijala da postane dostojni nasljednik. Teško je reći hoće li se razvoj neuronske mreže nastaviti unedogled ili će umjesto nje doći neka nova, učinkovitija tehnologija, ali u svakom slučaju, ovaj iskorak u području umjetne inteligencije zaslužuje našu pažnju [9].

4. SKUPOVI PODATAKA

Podaci za trening su vrsta podataka koja se koristi za trening nove aplikacije, modela ili sustava kroz različite metode ovisno o izvedivosti i zahtjevima projekta. Za potrebe umjetne inteligencije ili strojnog/dubokog učenja takvi podaci malo su drugačiji, jer su označeni određenim tehnikama kako bi bili prepoznatljivi računalu koje pomaže strojevima da razumiju objekte. Kod strojnog učenja skupovi podataka su ključni čimbenik koji navodi stroj da prepozna objekte ili određene obrasce i napravi točno predviđanje kada se koristi u stvarnom svijetu, a ne simulaciji. U osnovi, postoje tri vrste podataka za trening koji se koriste u razvoju modela strojnog učenja i svaki podatak ima svoju važnost i ulogu u izgradnji modela.

Podaci za trening su temeljni skup podataka koji se koristi za trening modela strojnog učenja. Sadrži ulazne primjere uparene s njihovim odgovarajućim željenim izlazima, omogućujući modelima da nauče obrasce i odnose unutar podataka, čime se poboljšava njihova sposobnost točnih predviđanja na novim, neviđenim podacima.

Podaci za validaciju koriste se tijekom procesa treninga za pedantno podešavanje parametara modela i procjenu njegove izvedbe. Usporedbom predviđanja modela sa stvarnim ishodima u ovom skupu podataka, mogu se napraviti prilagodbe kako bi se spriječila preadaptacija i poboljšala generalizacija na nove podatke.

Podaci za testiranje služe kao konačna procjena za istrenirani model strojnog učenja. Razlikuju se i od podataka treninga i validacije i koriste se za procjenu prediktivnih mogućnosti modela u stvarnom svijetu. Testiranje pomaže u procjeni koliko će model dobro funkcionirati na novim, neviđenim podacima u praktičnim primjenama [10].

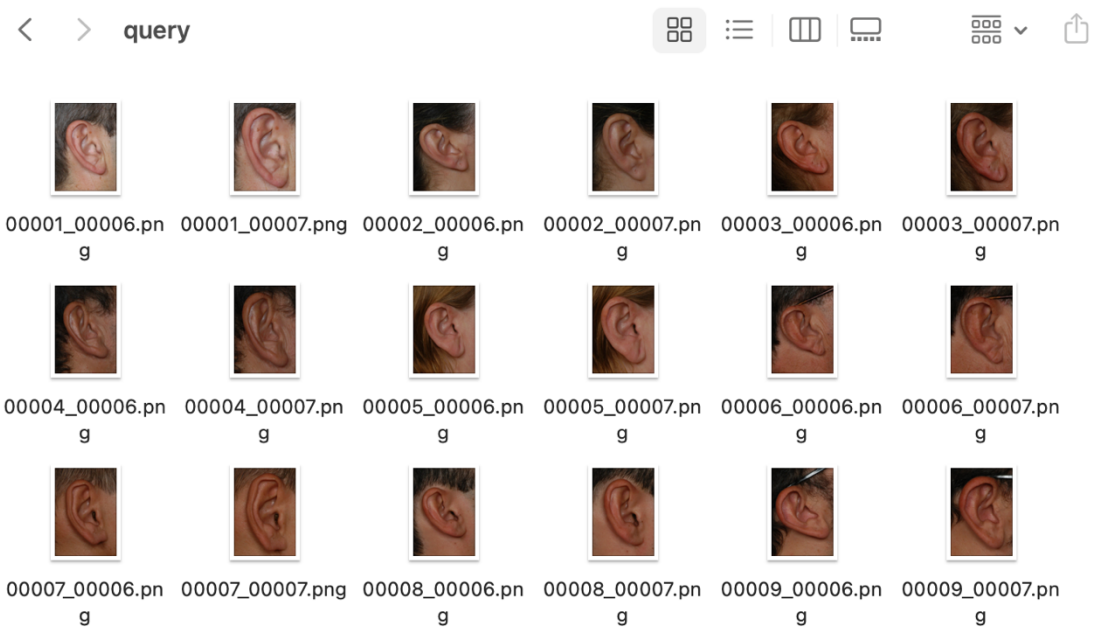
U svrhu ovog rada skupovi podataka podijeljeni su na podatke za trening („gallery“) i podatke za testiranje („query“) zbog malog broja podataka u skupovima. Dva su skupa podataka, dobivena iz različitih izvora i svaki s različitom brojnošću podataka, u ovom slučaju – fotografija. Važno je napomenuti da je u oba skupa princip podijele na podatke za trening i testiranje bio 70:30. Prvi skup podataka naziva „AMI dataset“ dobiven od španjolskog sveučilišta Las Palmas sadrži ukupno 700 fotografija, pri čemu svaki subjekt ima 7 predodijeljenih, a subjekata je 100. Slična situacija je i s drugim skupom naziva „AWE dataset“ koji je dobiven na korištenje od strane Sveučilišta u Ljubljani, Fakulteta računarstva i informatike. On također ima 100 subjekata, ali svakom subjektu je pridodano 10 fotografija. Za razliku od prvog skupa podataka gdje su sve fotografije jednake dimenzije, 492 x 702 i tipa .jpg, u drugom skupu to nije slučaj. Fotografije su proizvoljnih dimenzija i .png proširenja.

4.1. Priprema podataka

Prije nego opišem postupak pripreme skupova podataka, važno je naglasiti koji je konačni cilj same pripreme, odnosno kakav naziv zahtjeva svaka fotografija i u koju mapu ona treba biti spremljena. Dogovorena notacija je sljedećeg formata: *xxxxx_yyyyy.png,* pri čemu x-ovi odgovaraju broju subjekta čije uho slika prikazuje, a y-i rednom broju slike x-tog subjekta. Ako je potrebno označiti četvrtu sliku sedmog subjekta, to će izgledati ovako: *00007_00004.png.* Kao što sam već spomenuo, omjer raspodijele fotografija je 7:3. Takav omjer odgovara brojnosti fotografija u AWE datasetu s obzirom da svaki subjekt ima 10 fotografija*,* ali ne i u AMI datasetu gdje je 7 fotografija po subjektu*.* Stoga je raspodjela u njemu napravljena u omjeru 5:2 što je matematički najbliže osnovnom omjeru.

AMI dataset preuzet je u zip formatu nakon čega je prebačen u standardnu mapu. U toj mapi fotografije su bile raspoređene u 4 zasebne mape koje sam onda prebacio u jednu zajedničku. Prije promjene nazivlja, fotografije su imale naziv u obliku 000_front_ear.jpg. Dakle, subjekti su započinjali od 0 i neki brojevi u redoslijedu do 100 su nedostajali što je rezultiralo brojem subjekata koji premašuje broj 100. Kako bi izbjegao manualno razvrstavanje i mijenjanje naziva fotografija, napravio sam skriptu u Pythonu. U njoj sam definirao putanju do pojedinih mapa i postavio logiku da prolazi redom po svih fotografijama, mijenja im ekstenziju u .png i nakon svakih sedam prijeđenih fotografija, indeks subjekta se mijenja, počevši od nule. Definirao sam programu sintaksu po kojoj ih mora imenovati, a ta sintaksa objašnjena je u prošlom odlomku. Nakon što su sve fotografije ispravno imenovane i spremljene u zajedničku mapu, sljedeći korak bio ih je raspodijeliti u omjeru 5:2. To sam također učinio uz pomoć skripte [11].

S AWE datasetom postupio sam na sličan način kao i sa prvim. No, samo preuzimanje nije bilo dostupno prije odobrenja vlasnika, stoga se ovom prilikom zahvaljujem Fakultetu računarstva i informatike u Ljubljani na ukazanoj prilici da mi njihov skup podataka posluži u odrađivanju zadatka i pisanju ovog rada. Fotografije su u ovom slučaju već bile raspoređene u mape za svakog subjekta, stoga zadatak skripte bio iterirati po mapama i dodjeljivati definirani naziv. Na samom kraju, sve fotografije u zajedničkoj mapi raspoređene su u omjeru 7:3 u posebne mape koje odjeljuju podatke za trening od podataka za testiranje. Primjer pripremljenog skupa podataka može se vidjeti na *Slici 4.1.* [12].



Slika 4.1. Prikaz podataka za testiranje AMI dataseta

5. PREDOBRADA PODATAKA

U prošlom poglavlju bio je opisan proces pripreme skupova podataka kako bi oni bili spremni za konačan proces dubokog učenja i prepoznavanja. No prije toga, postoji još jedan proces kroz koji je potrebno proći, a naziva se predobrada podataka. Taj postupak služi kako bi pripremljene podatke, fotografije, pozvali u program i sistematski ih strukturirali po subjektima te na taj način omogućili stroju da na svoj jedinstven način procesuira podatke. Prije samog objašnjenja kako je to učinjeno, nužno je napraviti kratak uvod o jeziku koji je sve to omogućio.

5.1. PyTorch

Godine 2016. Facebookov istraživački tim za umjetnu inteligenciju stvorio je platformu za strojno učenje otvorenog koda pod nazivom PyTorch. Izgrađen je na Lua pisanoj biblioteci Torch, koja je okvir za znanstveno računarstvo. Zbog svoje jednostavnosti, prilagodljivosti i dinamičnog računalnog grafikona, PyTorch je brzo narastao i svrstao se među najpopularnije okvire dubokog učenja. Namjera mu je olakšati istraživačima i programerima stvaranje i treniranje neuronskih mreža pomoću jednostavnog i razumljivog Python API-ja. Veliki projekti strojnog učenja imaju koristi od sposobnosti PyTorcha da upravlja distribuiranim treningom na brojnim GPU-ovima i čvorovima, kao i nizom alata i mogućnosti. Također, ima značajnu i aktivnu zajednicu korisnika i suradnika, što je potaknulo stvaranje brojnih korisnih biblioteka i proširenja [13].

5.2. Učitavanje podataka

Za programiranje koristio sam radnu okolinu naziva „Jupyter Notebook“. Jupyter Notebook je web aplikacija otvorenog koda koja omogućuje stvaranje i dijeljenje dokumenata koji sadrže kod, jednadžbe, vizualizacije i narativni tekst. Upotreba uključuje čišćenje i transformaciju podataka, numeričku simulaciju, statističko modeliranje, vizualizaciju podataka, strojno učenje i još mnogo toga. Za razliku od Python skripti, Jupyter Notebook skripte se izvršavaju na nelinearan način. To znači da se blokovi koda mogu izvršavati proizvoljnim redoslijedom umjesto izvršavanja svakog bloka od vrha do dna. Može se pokrenuti lokalno bez pristupa internetu ili se može instalirati na udaljeni poslužitelj i pristupati joj putem interneta.

Nakon pokretanja Jupyter Notebooka, kreirao sam dvije „bilježnice“ za svaki skup podataka.

Po svoj strukturi su gotovo identični. Ono malo po čemu se razlikuju su upravo klase za učitavanje podataka o kojima će sada biti riječ. Međutim, ta razlika nije u strukturi funkcija i načinu na koji oni obavljaju zadaću učitavanja, već o specifičnosti nazivlja tj. brojnosti podataka u svakom od skupova, što rezultira različito definiranim kvantitativnim svojstvima skupova. Jednostavnije objašnjeno, jedna će učitavati 7 fotografija, a druga 10 fotografija za svakog pojedinog subjekta. Za početak je potrebno uvesti „pytorch“ i ostale potrebne knjižnice, a neke od njih su:

```
import os
import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
from pathlib import Path
import torchvision
from torchvision import transforms
import torch.nn as nn
```

Ukratko, to su uvozi koji omogućuju razne funkcionalnosti, od funkcionalnosti ovisnih od operacijskom sustavu pa sve do baratanja s fotografijama, transformacija, postavljanja putanje do istih i izgradnje neuronskih mreža.

Dvije klase naziva „TrainDataset“ i „TestDataset“ su klase napisane nakon uvoza i služe za učitavanje skupa podataka prema njihovoj specifičnoj nomenklaturi. S obzirom da su nomenklature zajedničke u oba odjeljka (podatci za trening i podaci za testiranje), neću ih pojedinačno objašnjavati, već ću pomoću jednog odjeljka i njegove klase objasniti oba u srži. Klase se sastoje od nekoliko ključnih metoda, a to su inicijalizacija, metoda za vraćanje ukupnog broja učitanih fotografija i metoda za vraćanje podatka pod određenim indeksom.

Prilikom poziva klase za učitavanje, važno je uputiti putanju do skupa podataka kao argument:

```
train_dataset = TrainDataset(root_dir='./Posloženidatasetovi/AMI/gallery')
```

Konstruktor prima tu putanju i tako zna gdje se fotografije nalaze. U inicijalizacijskoj metodi (konstruktoru) moramo definirati koliko subjekata postoji i koliko fotografija odgovara svakom subjektu te od kojeg broja počinju. To se radi na sljedeći način:

```
def __init__(self, root_dir):
        self.root_dir = root_dir
```

```
        self.subjects = range(1, 101)     # 100 subjekata
        self.start_image_number = 1.      # počinju od broja 1
        self.num_images = 5  # svaki subjekt ima 5 fotografija za trening
```

U ovoj metodi još se definiraju transformacije koje će biti opisane u zasebnom odjeljku kasnije. Metoda za vraćanje ukupnog broja fotografija jednostavno pomnoži broj subjekata s brojem fotografija za svakog subjekta:

```
def __len__(self):
        return len(self.subjects) * self.num_images
```

Sljedeća stvar je objasniti programu kako znati koje fotografije pripadaju kojem subjektu. To je implementirano u sklopu metode za vraćanje podatka pod određenim indeksom. Pruženi se indeks koristi u računanju ID-a subjekta i broja fotografije (broj s desne strane povlake). Četvrti red u kodu predstavlja konstrukciju naziva fotografije prema izračunatim varijablama. To omogućuje programu da pronađe traženi podatak:

```
def __getitem__(self, idx):
        subject_id = idx // self.num_images + 1
        image_number = idx % self.num_images + self.start_image_number
        image_name =
f"{str(subject_id).zfill(5)}_{str(image_number).zfill(5)}.png"
        image_path = os.path.join(self.root_dir, image_name)
```

U ovom trenutku program može učitati fotografiju i dodijeliti joj oznaku koja je od ID-a subjekta umanjena za 1. To je uobičajena praksa kod dubokog učenja da oznake subjekata kreću od 0, a ne od 1 jer omogućuje kompatibilnost sa sintaksom Pythona koji koristi indeksiranje od 0 za podatkovne strukture poput lista i polja i olakšava postupak koji se zove „one-hot encoding“ (pretvara kategoričke podatke, kao što su oznake klasa ili kategorije, u binarni vektorski format, gdje je svaka kategorija predstavljena jedinstvenim bitom u vektoru):

```
        image = load_image(image_path)
                label = subject_id - 1
```

Nakon što se fotografija učitala, potrebno je provesti postupak transformacije [13].

5.3. Transformacije

U *init* metodi klasa definirano je nekoliko transformacija koje su primijenjene na skup podataka. Te su transformacije ključne za pripremu podataka i osiguravanje njihove prikladnosti za trening modela dubokog učenja, posebice konvolucijskih neuronskih mreža (CNN).

```
self.image_transform = transforms.Compose([
            Resize((224, 224)),                  # promijeni veličinu
            Grayscale(num_output_channels=3),      # provodi grayscaling
            RandomHorizontalFlip(),            # nasumično vodoravno okretanje
            ToTensor(),                         # pretvori u tenzor
])
```

Promjena veličine na fiksnu veličinu nije nužno uvijek, ali je važno za kompatibilnost, učinkovitost, smanjenje korištenja potrebne memorije i sl. Neuronske mreže obično očekuju da ulazni podaci imaju dosljedne dimenzije. Mnoge tehnike optimizacije rade učinkovitije kada su podaci dosljedne veličine. Promjena veličine slika na iste dimenzije osigurava da značajke izdvojene iz različitih slika imaju slične razmjere i prostorne rezolucije. Ovu transformaciju nije bilo potrebno provesti kod AMI dataseta zbog jednakih dimenzija fotografija u cijelom skupu.

Transformacija u sive tonove (*eng. grayscaling)* se primjenjuje za pretvaranje slika u boji u sive tonove. Slike u sivim tonovima sadrže samo informacije o intenzitetu, pojednostavljujući podatke uz zadržavanje bitnih značajki. Ova transformacija smanjuje složenost ulaznih podataka, olakšavajući modelu učenje uzoraka i smanjujući zahtjeve za računanjem. *Num_output_channels* označuje koliko bi izlazna slika trebala imati kanala boja. Postavljanje *num_output_channels* na 3 u *grayscale* transformaciji način je da se osigura kompatibilnost s modelima i operacijama koje očekuju fotografije u boji s tri kanala. Učinkovito replicira jednokanalne informacije u sivim tonovima u sva tri kanala boja, tako da na kraju dobivamo fotografiju u sivim tonovima koja ima tri identična kanala.

Nasumično vodoravno okretanje unosi raznolikost u skup podataka pružajući modelu varijacije iste slike. Na primjer, vodoravno okrenuta slika mačke i dalje predstavlja mačku, ali izgleda drugačije. Ova tehnika pomaže spriječiti prekomjernu adaptaciju izlaganjem modela različitim točkama gledišta i orijentacijama istog objekta.

Transformacija *ToTensor* pretvara slikovne podatke iz izvornog formata (obično „NumPy“ polje) u PyTorch tenzore. Okviri dubokog učenja poput PyTorcha besprijekorno rade s tenzorima. Ova

transformacija također normalizira vrijednosti piksela na raspon (0, 1), što je standardna praksa kako bi se osiguralo dosljedno skaliranje podataka [14].

Ovako primjenjuje transformacije i konačno vraća fotografiju i oznaku (labelu):

```
image = self.image_transform(image)
return image, label
```

5.4. Dataloader

U PyTorchu, „DataLoader“ je ugrađena klasa koja pruža učinkovit i fleksibilan način za učitavanje podataka u model za trening i klasifikaciju. Posebno je koristan za rukovanje velikim skupovima podataka koji ne mogu stati u memoriju, kao i za izvođenje povećanja podataka i pretprocesiranja.

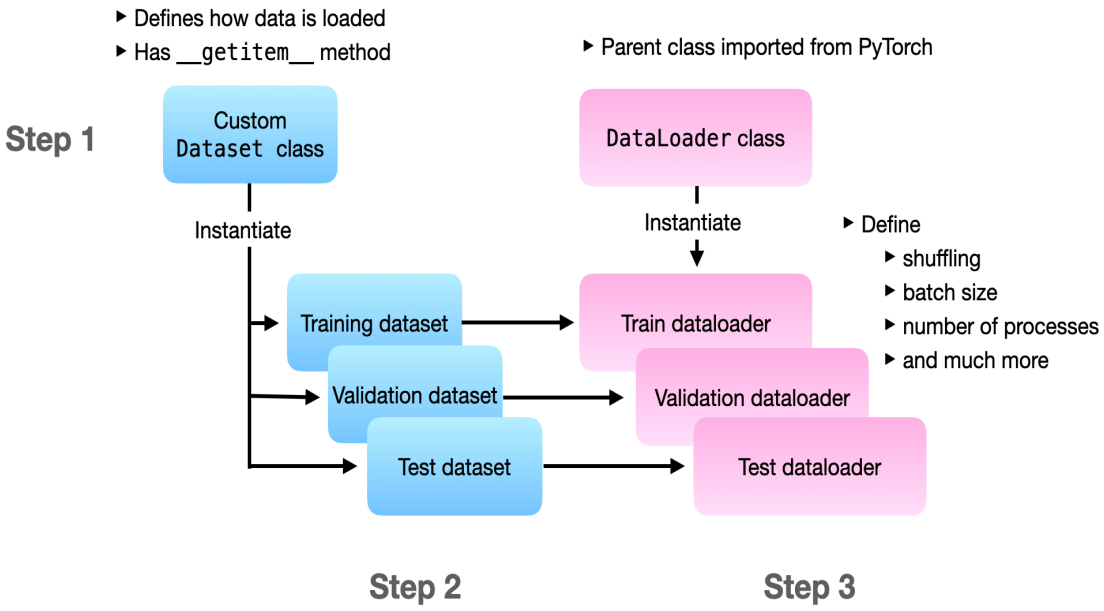
Klasa DataLoader radi stvaranjem objekta skupa podataka koji se može ponavljati i iteriranjem u serijama, koji se zatim unose u model za obradu. Objekt skupa podataka može se stvoriti iz različitih izvora, uključujući *NumPy* nizove, PyTorch tenzore i prilagođene izvore podataka kao što su CSV datoteke ili direktoriji slika.

Kako bismo mogli koristiti klasu DataLoader, prvo moramo definirati objekt skupa podataka i navesti sve potrebne transformacije ili korake pretprocesiranja. Na primjer, možda ćemo trebati promijeniti veličinu slika, normalizirati vrijednosti piksela ili izvršiti povećanje podataka. Nakon što je definiran objekt skupa podataka, može se stvoriti DataLoader objekt prosljeđivanjem objekta iz skupa podataka i navođenjem veličine serije (eng. *batch size*) i svih drugih relevantnih parametara. Ovako izgleda poziv Dataloadera na primjeru skupa podataka za trening:

```
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

U varijabli *train_loader* pohranjen je prilagođeni učitavač podataka *train_dataset* iz kojeg je potrebno stvoriti mini-serije za trening. U ovom kontekstu, to je PyTorch objekt koji sadrži označene i transformirane podatke. Zatim se kao argumenti još daju parametri *batch_size* koji određuje broj uzoraka podataka koji će se učitati u memoriju i obraditi zajedno kao serija te *shuffle* koji označava treba li se skup podataka promiješati prije stvaranja mini-serija. Miješanjem podataka osigurava se da model ne vidi primjere istim redoslijedom tijekom svake epohe treninga, što može spriječiti model da nauči redoslijed podataka umjesto osnovnih obrazaca.

Na *Slici 5.1.* grafički je prikazan cijeli proces koji je objašnjen u ovom poglavlju. Treba uzeti u obzir, kao što je već rečeno, da je validacija i testiranje u ovom kontekstu jedna cjelina [13].



Slika 5.1. Proces predobrade podataka

6. TRENING I KLASIFIKACIJA

6.1. Opis korištenih mreža

Tijekom godina, istraživači teže stvaranju dubljih neuronskih mreža, dodajući više slojeva, za rješavanje složenih zadataka i za poboljšanje točnosti klasifikacije/prepoznavanja. No, vidjelo se da kako dodajemo više slojeva neuronskoj mreži, postaje ih teško uvježbati i točnost se počinje zasićivati, a zatim također opada. Ovdje ResNet dolazi u pomoć i pomaže riješiti ovaj problem.

ResNet, skraćeno od rezidualni blokovi (eng. *Residual Network*), specifična je vrsta neuronske mreže koju su 2015. godine predstavili Kaiming He, Xiangyu Zhang, Shaoqing Ren i Jian Sun u svom radu "Deep Residual Learning for Image Recognition". Prva stvar koja je primjetno drugačija je da postoji izravna veza koja preskače neke slojeve (mogu se razlikovati u različitim modelima). Ova se veza naziva "vezom za preskakanje" i jezgra je preostalih blokova. Zbog ove veze za preskakanje, izlaz sloja sada nije isti. Bez korištenja ove veze preskakanja, ulazni x se množi s težinama sloja nakon čega se dodaje izraz pristranosti. Zatim, ovaj izraz prolazi kroz aktivacijsku funkciju, f() i dobivamo izlaz koji izgleda kao: $H(x) = f(wx + b)\ ili\ H(x) = f(x)$

Sada s uvođenjem veze za preskakanje, izlaz se mijenja u: $H(x) = f(x) + x$

Mali problem se pojavljuje s ovim pristupom kada se dimenzije ulaza razlikuju od izlaza, što se može dogoditi s konvolucijskim i skupnim slojevima. U ovom slučaju, kada su dimenzije f(x) različite od x, možemo uzeti dva pristupa:

- Spoj za preskakanje je podstavljen dodatnim nula ulazima za povećanje dimenzija.
- Metoda projekcije koristi se za usklađivanje dimenzija što se postiže dodavanjem 1×1 konvolucijskih slojeva ulazu pa izlaz izgleda: $H(x) = f(x) + w1.x$

Veze za preskakanje u ResNetu rješavaju problem nestajanja gradijenta u dubokim neuronskim mrežama dopuštajući ovaj alternativni prečac kroz koji gradijent teče. Drugi način na koji te veze pomažu je dopuštanje modelu da nauči funkcije identiteta što osigurava da će viši sloj raditi barem jednako dobro kao niži sloj, a ne lošije. Recimo da imamo plitku mrežu i duboku mrežu koja preslikava ulaz x u izlaz y pomoću funkcije H(x). Želimo da duboka mreža radi barem jednako dobro kao plitka mreža i da ne degradira performanse kao u slučaju običnih neuronskih mreža (bez rezidualnih blokova). Jedan od načina da se to postigne je ako dodatni slojevi u dubokoj mreži nauče funkciju identiteta i stoga je njihov izlaz jednak ulazima koji im ne dopuštaju degradaciju performansi čak i s dodatnim slojevima. Vidjelo se da rezidualni blokovi slojevima iznimno olakšavaju učenje funkcija identiteta. To je vidljivo iz gornjih formula. U običnim mrežama izlaz je: $H(x) = f(x)$. Dakle,

da bi se naučila funkcija identiteta, f(x) mora biti jednak x. Dok u slučaju ResNeta, sve što trebamo je učiniti f(x)=0 i dobit ćemo x kao izlaz koji je također ulaz (6.1):

$$H(x) = f(x) + x;\ f(x) = 0 \qquad (6.1)$$

$$H(x) = x$$

U najboljem slučaju, dodatni slojevi duboke neuronske mreže mogu bolje aproksimirati preslikavanje x u izlaz y i značajno smanjuje pogrešku. Stoga se očekuje da ResNet radi jednako ili bolje od običnih dubokih neuronskih mreža [15].

Sljedeća korištena mreža je VGG. VGG je kratica za „Visual Geometry Group“ što je standardna arhitektura duboke konvolucijske neuronske mreže (CNN) s više slojeva. Njezina struktura je jednostavnija od ResNetove jer koristi niz konvolucijskih slojeva s malim 3x3 filtrima, a ne koristi „veze za preskakanje“. S druge strane, obično ima veliki broj parametara, što ju čini računalno skupim za trening. Iako postiže dobre performanse na mnogim zadacima, nisu tako računalno učinkoviti kao novije arhitekture. Zato ResNet postiže bolje performanse s manje parametara u usporedbi s VGG-om. ResNet mreža koristi 34-slojnu običnu mrežnu arhitekturu inspiriranu VGG-19 u koju je zatim dodana veza prečaca [16].

6.2. Definicija modela i optimizatora

Prije nego trening započne, potrebno je definirati nekoliko parametara, od kojih su najvažniji model neuronske mreže i optimizator. Na samom početku provjerava se dostupnost GPU-a, jedinice za grafičku obradu. Ovo je važna točka jer trening modela dubokog učenja može biti znatno brža na GPU-u s obzirom na rad na CPU-u. Ako je GPU dostupan, ispisuje se njegov naziv koristeći *torch.cuda.get_device_name():*

```
if torch.cuda.is_available():
        print("GPU available " + torch.cuda.torch.cuda.get_device_name())
```

Sljedeća linja označuje odabir uređaja ako je GPU dostupan, inače se vraća CPU na korištenje. Ovaj će se uređaj koristiti za izvođenje izračuna tijekom treninga i testiranja:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Sada dolazimo do odabira modela. U ovom slučaju to je unaprijed istreniran ResNet-18 model kojeg se učitava iz PyTorch model središta pomoću *torch.hub.load()*. Unaprijed uvježbani model u kontekstu dubinskog učenja odnosi se na model neuronske mreže koji je uvježban na velikom skupu podataka, obično za određeni zadatak kao što je klasifikacija slika, otkrivanje objekata ili obrada prirodnog jezika. Izraz "predtreniran" podrazumijeva da je ovaj model prošao proces treninga prije nego što je postao dostupan drugima za korištenje. Argument *weights=True* ukazuje na to da se prethodno istrenirane težine za model trebaju preuzeti ako su dostupne. Ovaj se model često koristi kao ekstraktor značajki ili se fino podešava za specifične zadatke:

```
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', weights=True)
```

Zatim slijedi poziv *model.to(device)* koji premješta cijeli model (uključujući njegove slojeve i parametre) na prethodno odabrani uređaj (GPU ili CPU). Time se osigurava da će se svi proračuni koji se odnose na model izvoditi na odabranom uređaju.

Definicija funkcije gubitka, u ovom slučaju, *Cross-Entropy Loss*, obično se koristi za zadatke klasifikacije. Metoda *.to(device)* koristi se za premještanje funkcije gubitka na isti uređaj kao i model. Konačno, definira se optimizator (u ovom slučaju je Adam) za ažuriranje težine modela tijekom treninga. Adamov optimizator je adaptivni algoritam optimizacije stope učenja koji kombinira prednosti dviju drugih popularnih metoda optimizacije: AdaGrad i RMSprop. Adaptivne stope učenja i momentum u Adamu čine ga prikladnim za širok raspon zadataka dubokog učenja. Postao je popularan izbor za trening neuronskih mreža jer često konvergira brzo i učinkovito:

```
loss_fn = nn.CrossEntropyLoss()
loss_fn.to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

6.2.1. Stopa učenja

Stopa učenja *lr* ima veliku ulogu u procesu treniranja. U kontekstu treniranja neuronskih mreža, ona je hiperparametar koji kontrolira veličinu koraka pri kojem optimizator ažurira parametre modela tijekom treninga. To je jedan od najkritičnijih hiperparametara za podešavanje pri uvježbavanju modela dubokog učenja. Neuronske mreže se treniraju pomoću optimizacijskih

algoritama, često onih temeljenih na gradijentu poput stohastičkog gradijentnog spuštanja (SGD). U svakoj iteraciji treninga (seriji), model izračunava gradijente funkcije gubitka s obzirom na svoje parametre (težine i pristranosti). Ti gradijenti pokazuju smjer i veličinu promjena potrebnih za smanjenje gubitka. Stopa učenja određuje veličinu koraka poduzetih za ažuriranje parametara modela. Veća stopa učenja znači veće korake, dok niža stopa učenja znači manje korake. Ako je stopa učenja previsoka, optimizator bi mogao premašiti minimum funkcije gubitka. To može dovesti do nestabilnog treninga, divergencije ili konvergencije do suboptimalnog rješenja. Ako je stopa učenja preniska, optimizator poduzima vrlo male korake, što može dovesti do vrlo spore konvergencije. U nekim slučajevima, optimizator može zapeti u lokalnom minimumu. U praksi je uobičajeno koristiti planer stope učenja (eng. *scheduler*) koji prilagođava stopu učenja tijekom treninga. Na primjer, može se započeti s relativno visokom stopom učenja za postizanje velikog početnog napretka, a zatim je postupno smanjivati, kako trening napreduje, za osiguravanje konvergencije. To može pomoći u ravnoteži brze konvergencije na početku s finim podešavanjem kako se proces optimizacije nastavlja.

Odabir odgovarajuće stope učenja je ključan. Često zahtijeva eksperimentiranje i podešavanje hiperparametara kako bi se pronašla stopa učenja koja omogućuje modelu da brzo konvergira do dobrog rješenja bez odstupanja. Neki napredni optimizatori, poput Adama kojeg sam koristio u ovom treningu, automatski prilagođavaju efektivnu stopu učenja za svaki parametar na temelju povijesti gradijenata. Ova prilagodljiva stopa učenja može dovesti do brže konvergencije i manje osjetljivosti na izbor fiksne stope učenja.

Za potrebe AWE dataseta kreirao sam planer stope učenja kako bi dobio bolje rezultate. U planeru sam definirao da se nakon svakih 25 epoha, stopa učenja smanji za 50%:

```
scheduler = lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.5)
```

Epoha označuje jedan potpuni prolaz kroz cijeli skup podataka za trening. Tijekom epohe, parametri modela (težine i pristranosti) ažuriraju se korištenjem podataka o treningu kako bi se smanjila funkcija gubitka i poboljšala njezina izvedba.

I na samom kraju, prije iteracija za trening i validaciju, inicijalizirat će se *Tensorboard Summary Writer*. To je alat za vizualizaciju koji se koristi za praćenje napretka treninga, uključujući metriku, krivulje gubitka i sl. Program za pisanje se inicijalizira s direktorijem dnevnika koji sadrži vremensku oznaku za trenutno izvođenje [17, 18].

6.3. Trening za jednu epohu

Trening za jednu epohu sadržana je u funkciji naziva *train_one_epoch()* koja kao argumente prima trenutni indeks epohe i *writer* varijablu u koju je spremljen *Summary Writer.*
Inicijalno se postavljaju varijable gubitka *running_loss* i *last_loss* koje akumuliraju gubitak u serijama unutar epohe i daje sliku prosječnog gubitka za jednu seriju.

Nakon toga postavlja se petlja koja iterira kroz skup podataka za trening koristeći train_loader. Funkcija *enumerate* koristi se za praćenje indeksa serije (*i*) i podataka (eng. *data*) u svakoj seriji. Svaka serija raspakira podatke na ulaze (eng. *inputs*) i oznake (eng. *labels*) i premjesti ih na odabrani uređaj (GPU/CPU). Prije izračuna gradijenata za trenutnu seriju, treba se osigurati da su gradijenti iz prethodnih serija očišćeni kako bi se spriječilo nakupljanje. Zatim će se pozvati model kako bi izradio predviđanja, izlaze na temelju ulaza. Nakon toga slijede izračuni gubitaka usporedbom predviđenih i stvarnih oznaka. Uz to, za optimizaciju, neophodno je izračunati gradijente gubitka s obzirom na parametre modela koje onda optimizator „Adam“ koristi za prilagođavanje težina modela. Gubitak za trenutnu seriju treba se pribrojiti dosadašnjem gubitku, a pomoću *.detach()* sprječava se povratno širenje budući da nas zanima samo vrijednost gubitka [18].

```
for i, data in enumerate(train_loader):
                inputs, labels = data
                inputs = inputs.to(device)
                labels = labels.to(device)
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = loss_fn(outputs, labels)
                loss.backward()
                optimizer.step()
                running_loss += loss.detach().item()
                accuracy = calculate_accuracy(outputs, labels)
```

Zadnja linija u kodu izračunava točnost pozivanjem funkcije *calculate_accuracy()* i slanjem predikcija i stvarnih oznaka kao argumente funkcije:

```
def calculate_accuracy(outputs, labels):
        _, predicted = torch.max(outputs, 1)
        correct = (predicted == labels).sum().item()
        total = labels.size(0)
```

```
        accuracy = correct / total * 100
        return accuracy
```

Nadalje, pomoću if-petlje provjerava je li indeks serije višekratnik broja 1000 te, u slučaju da jest, ispisuje izvješće o napretku u *writer* i kalkulira prosječni gubitak po seriji. On nam daje ideju o tome koliko dobro model radi na ovom podskupu podataka. Print linija ispisuje broj serije, prosječni gubitak za tu seriju i točnost u postotcima. U tb_x varijablu se izračunava i sprema jedinstveni identifikator za trenutnu seriju za potrebe zapisivanja u TensorBoardu:

```
if i % 1000 == 0:
        last_loss = running_loss / 1000
        print('  batch {} loss: {} accuracy: {:.2f}%'.format(i + 1,
                last_loss, accuracy)
        tb_x = (epoch_index * len(train_loader)) + i
        tb_writer.add_scalar('Loss/train', last_loss, tb_x)
        tb_writer.add_scalar('Accuracy/train', accuracy, tb_x)
        running_loss = 0.0
```

Konačno, funkcija *train_one_epoch()* vraća prosječni gubitak i točnost za cijelu epohu. Ove se vrijednosti koriste za praćenje i analizu tijekom treninga:

```
        return last_loss, accuracy
```

6.4. Validacija/predviđanje

U posljednjem dijelu usredotočit ću se na validaciju koja je izrazito važna točka cijelog ovog rada, jer u konačnici, validacija je ultimativni zadatak koji nam govori kako bi se model ponašao u stvarnom svijetu, na „živim“ primjerima. Ono što je naučio treba primjenjivati, bez dodatne mogućnosti učenja tijekom procesa predviđanja. Dio koda koji će dati sliku kako validacija funkcionira je for-petlja koja prolazi po epohama, odnosno izvršit će se onoliko puta koliko je definiran broj epoha.

Na početku petlje, prvo se ispisuje redni broj trenutne epohe, a zatim pomoću *model.train()* postavlja model u način rada za trening jer će u sljedećoj liniji pozvati funkciju za trening train_one_epoch() koja je bila tema prošlog potpoglavlja i koja će vratiti prosječni gubitak i točnost

za cijelu epohu. U PyTorchu se određeni slojevi, poput odustajanja (eng. dropout) i normalizacije, ponašaju drugačije tijekom treninga i validacije. Postavljanjem modela u „trening“ način rada, eksplicitno mu dajemo do znanja da koristi ove slojeve onako kako su definirani za trening. Zatim se inicijaliziraju varijable za pohranu gubitka i točnosti (running_vloss, running_vaccuracy) u postupku validacije i postavlja model u način rada za validaciju: *model.eval()*. U ovom načinu rade odustajanje je onemogućeno, a normalizacijski slojevi koriste statistiku populacije umjesto skupne statistike. Ovo osigurava da se model ponaša dosljedno za klasifikaciju.

Prije samog postupka predviđanja, privremeno se onemogućuje izračun gradijenata za tenzore unutar 'with' bloka kako bi se smanjila potrošnja memorije jer oni nisu potrebni tijekom validacije. For-petlja iterira po skupu podataka za testiranje koristeći *test_loader*, objekt koji puni podatke i pruža serije tih podataka. Sljedeće tri linije raspakiravaju serije podataka u ulaze i oznake i premještaju ih na odabrani uređaj (GPU/CPU). Zatim se prosljeđuju ulazi u model u svrhu dobivanja predviđanja. Sljedeće četiri linije izračunavaju gubitak i točnost na način da postoje gubitci i točnosti za trenutnu seriju i akumulirani gubitci i točnosti preko cijelog testnog skupa podataka. Kako bi se dobio prosječan gubitak i prosječna točnost potrebno je akumulativni broj podijeliti s brojem uzorka u skupu. Na samom kraju, broj epoha se inkrementira za jedan, a uvjet koji provjerava je li trenutni gubitak niži od najnižeg gubitka validacije viđenog do sada ažurira najniži gubitak ako se utvrdi da je tako [18].

```
with torch.no_grad():
        for i, vdata in enumerate(test_loader):
                vinputs, vlabels = vdata
                vinputs = vinputs.to(device)
                vlabels = vlabels.to(device)
                voutputs = model(vinputs)
                vloss = loss_fn(voutputs, vlabels)
                running_vloss += vloss.item() * vinputs.size(0)
                vaccuracy = calculate_accuracy(voutputs, vlabels)
                running_vaccuracy += vaccuracy * vinputs.size(0)
        avg_vloss = running_vloss / len(test_dataset)
        avg_vaccuracy = running_vaccuracy / len(test_dataset)
```

Dobra je praksa bilježiti spomenuti gubitak i točnost pomoću Tensorboarda i ispisati napredak nakon svake epohe u konzolu:

```
print('LOSS train {} valid {} accuracy: {:.2f}%'.format(avg_loss,
                avg_vloss, avg_vaccuracy))
writer.add_scalars('Training vs. Validation Loss',
                {'Training': avg_loss, 'Validation': avg_vloss},
                epoch_number + 1)
writer.add_scalars('Training vs. Validation Accuracy',
                 {'Training': accuracy, 'Validation': avg_vaccuracy},
                 epoch_number + 1)
```

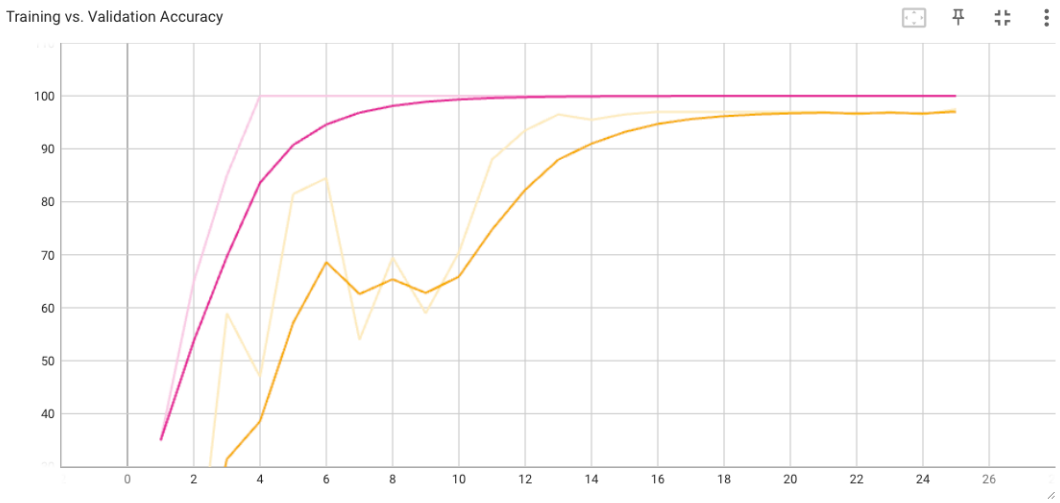
7. REZULTATI I USPOREDBA

Nakon mnogo podešavanja parametara kako bi dobio najbolji mogući ishod, dobio sam rezultate koji će biti prikazani u sljedećim potpoglavljima, po skupovima podataka. Treba imati na umu da je vrijeme mijenjanja tih postavki vrlo kratko, a da je vrijeme potrebno za treniranje modela zaista dugo. Ukupno vrijeme koje mi je bilo potrebno za isprobavanje većine kombinacija i treniranje dva skupa podataka iznosi otprilike 50 sati. U ovo se ne ubraja priprema skupova podataka, predobrada i pisanje koda. Ukupni rezultati mogu se vidjeti u *Tablici 6.1. i 6.2.*

7.1. AMI skup podataka

7.1.1. ResNet-18

Definitivno najuspješniji model koji je bio korišten u ovom radu. Najvišu točnost klasifikacije koju je model ostvario je točnost od 97.5 % u 25. epohi. Točnost u treningu već je dosegla 100% u 4. epohi. Iako sam napomenuo da ovom skupu podataka ne treba mijenjati dimenzije, ipak sam primijenio *Resize(224, 224)* kao uobičajnu praksu za generalizaciju i kompatibilnost. Ostali parametri (veličina serije = 32 i stopa učenja = 0.001) nisu mijenjani. Rezultantni graf dobiven *Tensorboardom* prikazan je na *Slici 6.1.* Rozom je prikazana točnost treninga, a žutom validacije. X os označuje epohe, a Y os točnost.

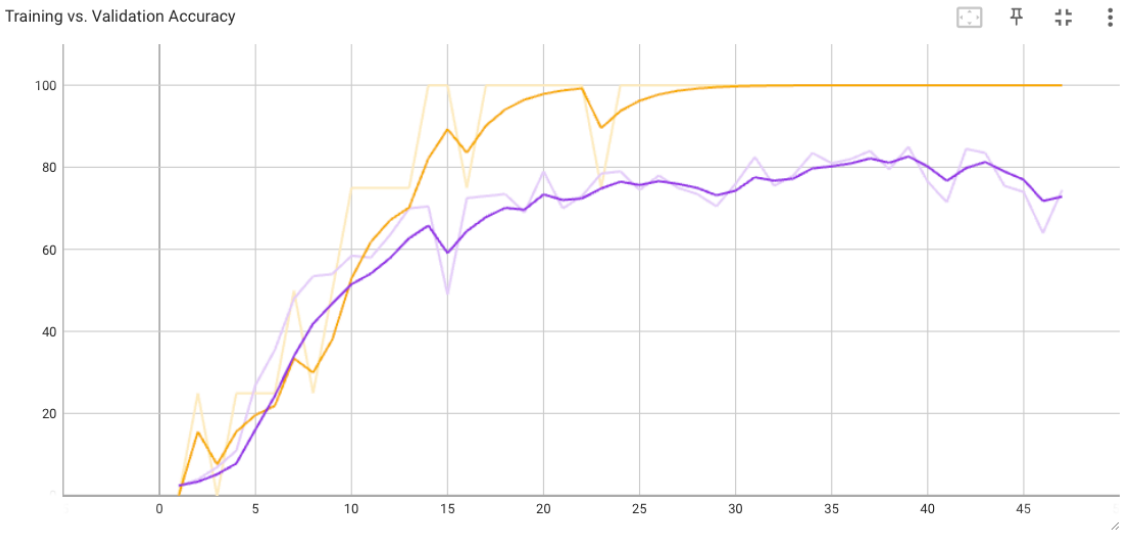


Slika 6.1. Usporedba napretka točnosti AMI(ResNet)

U potrazi za rezultatima dobivenim na istom ovom skupu podataka radi usporedbe, pronašao sam članak objavljen sredinom 2022. godine u kojem je rečeno da je stupanj prepoznavanja dosegao 97% što se podudara s mojim rezultatima [19].

7.1.2. VGG-16

Iako je i ovaj model ostvario dobre rezultate, oni nisu uspjeli nadmašiti ResNet-18, što je i očekivano. Naime, za VGG potrebno je mnogo finije namjestiti parametre za što treba imati veće znanje i iskustvo. VGG ima veću vjerojatnost da se u ranoj fazi preadaptira, pogotovo ako su skupovi podataka mali, kao u ovom slučaju. Najviša ostvarena točnost klasifikacije kod ovog modela je 85 % u 39. epohi. Točnost u treningu dosegla je 100% u 14. epohi. Ovdje sam također primijenio *Resize(224, 224)* i usto smanjio veličinu serije s 32 na 16 i stopu učenja na 0.0001 što je rezultiralo dužem vremenu treniranja, ali i bržoj konvergenciji. Rezultantni graf dobiven *Tensorboardom* prikazan je na *Slici 6.2.* Žutom je prikazana točnost treninga, a ljubičastom validacije. X os označuje epohe, a Y os točnost.

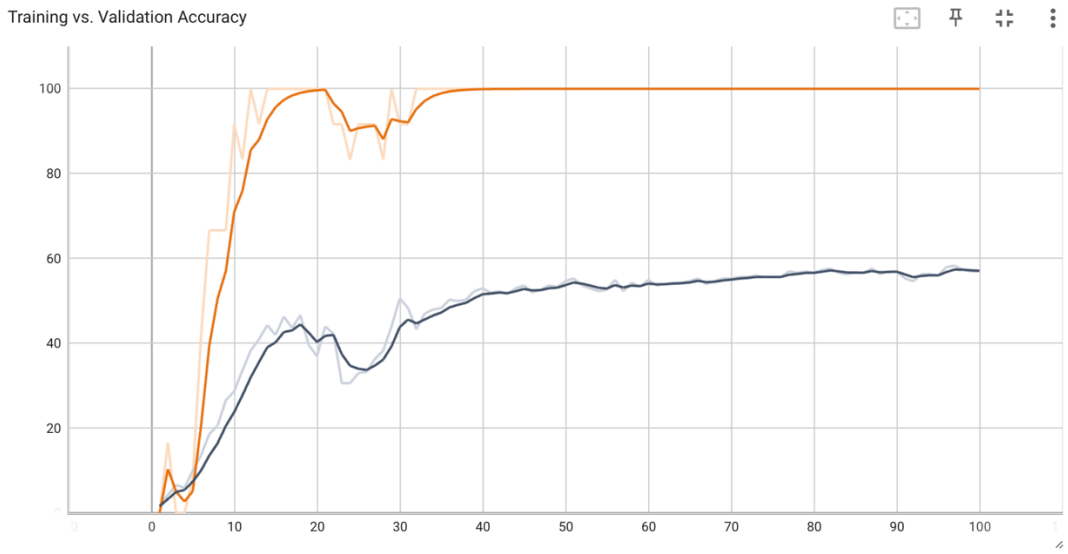


Slika 6.2. Usporedba napretka točnosti AMI(VGG)

7.2. AWE skup podataka

7.2.1. ResNet-18

AWE skup podataka je već, na prvi pogled, mnogo lošije kvalitete od AMI-a, što se odrazilo na rezultate. Osim toga, fotografije izrazito variraju u dimenzijama što može otežati trening, čak i kad je primijenjena *Resize()* transformacija. Najviša ostvarena točnost klasifikacije kod ovog modela je 58 % u 96. epohi. Točnost u treningu dosegla je 100% u 12. epohi. Svi parametri i transformacije, osim veličine serije koja je ovdje postavljena na 16, su jednaki onima iz AMI-ResNet treninga. Rezultantni graf dobiven *Tensorboardom* prikazan je na *Slici 6.3.* Žutom je prikazana točnost treninga, a crnom validacije. X os označuje epohe, a Y os točnost.



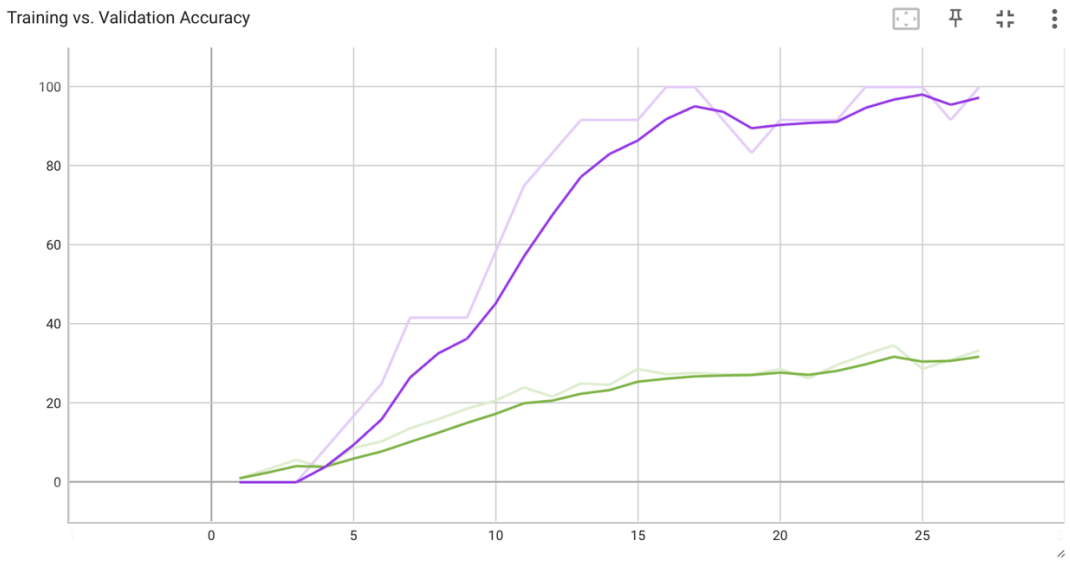
Slika 6.2. Usporedba napretka točnosti AWE(ResNet)

U članku koji govori o metodi prepoznavanja osobe temeljem uha na principu fuzije značajki otkrivena je točnost od 72.7% na AWE skupu podataka, što govori da taj princip uvelike utječe na rezultat koji prelazi ovaj po uspješnosti za oko 14% [20].

7.2.2. VGG-16

Kao i u slučaju prošlog skupa podataka s VGG-om, ostvareni rezultat nije nadmašio ResNet model. Najviša ostvarena točnost klasifikacije kod ovog modela je 35% u 24. epohi. Točnost u treningu dosegla je 100% u 18. epohi. Svi parametri i transformacije jednaki su onima iz AMI-VGG

treninga. Rezultantni graf dobiven *Tensorboardom* prikazan je na *Slici 6.4.* Ljubičastom je prikazana točnost treninga, a zelenom validacije. X os označuje epohe, a Y os točnost.



Slika 6.4. Usporedba napretka točnosti AWE(VGG)

Trening		Broj epoha	Gubitak	Točnost
ResNet-18	AMI	25	3.86×10^{-7}	100%
	AWE	100	1.24×10^{-7}	100%
VGG-16	AMI	47	8.35×10^{-8}	100%
	AWE	27	5.9×10^{-8}	100%

Tablica 6.1. Prikaz rezultata za trening

Validacija		Broj epoha	Gubitak	Točnost
ResNet-18	AMI	25	0.11	97.5%
	AWE	100	2.02	58%
VGG-16	AMI	47	0.96	85%
	AWE	27	4.32	35%

Tablica 6.2. Prikaz rezultata za validaciju

8. ZAKLJUČAK

U području, gdje je potraga za preciznošću i sigurnošću najvažnija, duboko učenje se pokazalo kao nužna sila. Među različitim biometrijskim modelima, prepoznavanje uha, iako relativno manje poznato u usporedbi s otiscima prstiju ili prepoznavanjem lica, dobilo je zamah kao put koji obećava. Uz integraciju tehnika dubokog učenja, ovo još nedovoljno istraženo područje burno se istražuje, nudeći prednosti u točnosti, robusnosti i prilagodljivosti.

Prepoznavanje uha privuklo je pozornost zbog svojih jedinstvenih karakteristika. Uho, sa svojom složenom strukturom i relativno stabilnim karakteristikama, predstavlja izvrstan biometrijski identifikator. Za razliku od crta lica, izrazi lica ili starenje uglavnom ne utječu na uho, što ga čini idealnim izborom za sustave prepoznavanja.

Duboko učenje, grana strojnog učenja, kataliziralo je evoluciju sustava za prepoznavanje uha. Konvolucijske (CNN), rekurentne (RNN) i druge neuronske arhitekture promijenile su način na koji se percipiraju i koriste biometrijski podaci. Snaga dubokog učenja leži u njegovoj sposobnosti da automatski uči hijerarhijske značajke iz neobrađenih podataka, sposobnosti koja se savršeno uklapa sa složenošću slika uha.

Značajan napredak u biometrijskom prepoznavanju proizlazi iz napretka u tehnikama predobrade i „povećanja“ podataka. Tehnike poput izrezivanja, rotacije i podešavanja svjetline ojačale su modele. *Grayscale* pretvorba, na primjer, ne samo da smanjuje računalno opterećenje, već također povećava otpornost modela na varijacije osvjetljenja. Osim toga, promjena veličine slika u dosljedan format, često 224x224 piksela, postala je uobičajena praksa. Iako se može činiti kontraintuitivnim, ovaj proces standardizira ulaze i omogućuje uspješnije korištenje moćnih unaprijed istreniranih modela kao što su ResNet ili VGG.

Korištenje unaprijed istreniranih modela dubinskog učenja nedvojbeno je promijenilo pravila igre u području prepoznavanja. Modeli kao što su ResNet, VGG i MobileNet, u početku trenirani na skupovima podataka velikih razmjera, donose prednost prijenosa učenja. Finim podešavanjem ovih modela s manjim skupovima podataka slika, u ovom slučaju uha, istraživači mogu postići najsuvremenije rezultate uz značajno smanjene troškove računanja. Paradigma ovakvog učenja demokratizirala je dubinsko učenje dopuštajući čak i onima s ograničenim podacima da iskoriste moć najsuvremenijih arhitektura.

Biometrijski podaci podižu brigu o privatnosti na potpuno novu razinu koju uobičajeno prikupljanje podataka samo dotiče. Digitalni identiteti mogu se krivotvoriti, a anonimnost na webu može se održati do određenog stupnja. Problem s biometrijskim podacima je taj što ih je nemoguće

krivotvoriti i prikriti, a trajni su. Ne možemo promijeniti strukturu uha, oka, otisak prsta ili crte lica. Dok mnoge tvrtke još uvijek daju korisnicima mogućnost korištenja svima poznate lozinke za otključavanje svojih telefona, biometrijska alternativa polako dobiva sve više na snazi i jednog će dana postati norma jer je teško argumentirati njezinu praktičnost i sigurnost. Tu leži pitanje tko bi, ako uopće itko, trebao imati pristup takvim podacima. Današnji tehnološki divovi imaju sve naše podatke u svojim rukama, što se neće promijeniti širenjem metoda biometrijske provjere. Doista, to će samo povećati mogućnosti ciljanja.

Uspostavljanjem ravnoteže između inovacija i etičke odgovornosti, može se otključati puni potencijal prepoznavanja pomoću ušiju i iskoristiti ga za sigurniju i jednostavniju budućnost [21].

LITERATURA

[1] Ramar Ahila Priyadharshini, „A deep learning approach for person identification using ear biometrics“, s Interneta, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7594944/, 11.8.2023.

[2] Alexander S. Gillis, „Deep learning“, s Interneta, https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network, 11.8.2023.

[3] Mathlab, „What is deep learning“, s Interneta, https://www.mathworks.com/discovery/deep-learning.html, 12.8.2023.

[4] Joules Garcia, „What is a neural network?“, s Interneta, https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature, 13.8.2023.

[5] Reni Banov, Anđa Valent, Judita Anušić, „Neuronske mreže za početnike“, s Interneta, https://hrcak.srce.hr/file/425148, 15.8.2023.

[6] Jaspreet, „A concise history of neural networks“, s Interneta, https://towardsdatascience.com/a-concise-history-of-neural-networks-2070655d3fec#:~:text=The%20idea%20of%20neural%20networks,McCulloch%20and%20mathematician%20Walter%20Pitts, 15.8.2023.

[7] Keith D. Foote, „A brief history of deep learning“, s Interneta, https://www.dataversity.net/brief-history-deep-learning/, 15.8.2023.

[8] upGrad, „Neural network tutorial“, s Interneta, https://www.upgrad.com/blog/neural-network-tutorial-step-by-step-guide-for-beginners, 18.8.2023.

[9] Frank Landman, „Everything you need to know about the future of neural networks“, s Interneta, https://readwrite.com/everything-you-need-to-know-about-the-future-of-neural-networks/, 19.8.2023.

[10] BecomingHumanAIMagazine, „What is training dana, its types and why it is important“, s Interneta, https://becominghuman.ai/what-is-training-data-its-types-and-why-it-is-important-f998424c3c9, 20.8.2023.

[11] Esther Gonzalez, „AMI ear database“, s Interneta, https://webctim.ulpgc.es/research_works/ami_ear_database/, 20.8.2023.

[12] University of Ljubljana, „Ear recognition research“, s Interneta, http://awe.fri.uni-lj.si/datasets.html, 20.8.2023.

[13] Shittu Olumide Ayodeji, „What is PyTorch dataloader?“, s Interneta, https://www.educative.io/answers/what-is-pytorch-dataloader#, 25.8.2023.

[14] John Reilly, „Data transformation in machine learning: Why you need it and how to do it with AI“, s Interneta, https://www.akkio.com/post/data-transformation-in-machine-learning#:~:text=Data%20transformation%20is%20also%20known,able%20to%20make%20accurate%20predictions, 27.8.2023.

[15] Great Learning Team, „Introduction to ResNet or Residual Network“, s Interneta, https://www.mygreatlearning.com/blog/resnet/, 28.8.2023.

[16] Gaudenz Boesch, „VGG very deep convolutional networks (VGGNet) – What you need to know“, s Interneta, https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/#:~:text=and%20many%20more.-,What%20is%20VGG%3F,16%20and%2019%20convolutional%20layers, 1.9.2023.

[17] Deepchecks, „Learning rate in machine learning“, s Interneta, https://deepchecks.com/glossary/learning-rate-in-machine-

learning/#:~:text=The%20learning%20rate%2C%20denoted%20by,network%20concerning%20the%20loss%20gradient%3E, 2.9.2023.

[18] PyTorch, „Welcome to PyTorch tutorials“, s Interneta, https://pytorch.org, 3.9.2023.

[19] Maha Sharkas, „Ear recognition with ensemble classifiers“, s Interneta, https://link.springer.com/article/10.1007/s11042-022-13252-w 4.9.2023.

[20] Xuebin Xu, Yibiao Liu, Chenguang Liu, „A future fusion human ear recognition method based on channel features and dynamic convolution“, s Interneta, https://www.mdpi.com/2073-8994/15/7/1454#:~:text=We%20performed%20simulations%20on%20the,existing%20human%20ear%20recognition%20methods, 4.9.2023.

[21] Adnan Kayyali, „What are the ethical issues in biometrics?“, s Interneta, https://insidetelecom.com/what-are-the-ethical-issues-in-biometrics/#:~:text=Ethical%20Issues%20in%20Biometrics%20and,degree%20if%20you%20know%20how, 5.9.2023.

SAŽETAK

Ovaj rad opisuje kako kreirati model koji će imati sposobnost točne klasifikacije kada mu se kao ulaz uputi fotografija uha subjekta čija je obilježja trebao naučio u trening fazi te su objašnjene prednosti ovakvog tipa podataka nad ostalim biometrijskim tipovima. Na samom početku, pokriveni su pojmovi duboko učenje i neuronska mreža koji su osnova za razumijevanje funkcioniranja treninga i validacije modela. Korišteni podaci potječu iz dva neovisna skupa podataka koji sadrže fotografije ljudskih uši, a koje su sistematski imenovane prema opisanim pravilima. Rad se dotiče predobrade tih podatka i potrebnih/proizvoljnih transformacija provedenim na njima prije nego su predane modelima. Korišteni pretrenirani modeli su ResNet-18 i VGG-16, a cjelokupni kod pisan je u PyTorchu, platformi otvorenog koda za strojno učenje. ResNet-18 ostvario je bolje rezultate na spomenutim skupovima podataka od VGG-16, a razlozi se kriju u njihovoj arhitekturi, parametrima i obilježjima samih skupova podataka.

Ključne riječi: uho, duboko učenje, neuronska mreža, skup podataka, ResNet, VGG

ABSTRACT

This paper describes how to create a model that will be capable of accurate classification when it is given a photo of the subject's ear as input, the features of which should have been learned in the training phase, and the advantages of this type of data over other biometric types. At the very beginning, the concepts of deep learning and neural network are covered, which are the basis for understanding the functioning of model training and validation. The data come from two independent datasets containing photographs of human ears, which were systematically named according to the described rules. The paper concerns the preprocessing of data and the necessary/arbitrary transformations performed on them before they are submitted to the models. The pretrained models that are used are ResNet-18 and VGG-16. The entire code is written in PyTorch, an open-source platform for machine learning. ResNet-18 achieved better results on the mentioned datasets than VGG-16 and the reasons lie in their architecture, parameters and characteristics of the datasets themselves.

Keywords: ear, deep learning, neural network, dataset, ResNet, VGG