Prepoznavanje osoba temeljem dlana primjenom dubokog učenja

Šćulac, Luka

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Prepoznavanje osoba temeljem dlana primjenom dubokog učenja

Rijeka, srpanj 2023.

Luka Šćulac
0069087959

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni prijediplomski studij računarstva

Završni rad

Prepoznavanje osoba temeljem dlana primjenom dubokog učenja

Mentor: prof. dr. sc Kristijan Lenac
Komentor: v. asist. dr. sc. Diego Sušanj

Rijeka, srpanj 2023.

Luka Šćulac
0069087959

Zahvala

Zahvaljujem se svom mentoru prof. dr. sc Kristijanu Lenacu te komentoru v. asist. dr. sc. Diegu Sušnju na iskazanom trudu i podršci tijekom pisanja ovoga rada i korisnim raspravama i savjetima.

Zahvaljujem se svim profesorima koji su me tijekom protekle tri akademske godine učili i proširili mi znanje u tehničkom svijetu.

Zahvaljujem se roditeljima i prijateljima za podršku i strpljenje tijekom studiranja.

Sadržaj

Poglavlje 1

Uvod

Tema ovog završnog rada je prepoznavanje osoba temeljem dlana primjenom dubokog učenja. U ovom radu detaljno ću istražiti koncept dubokog učenja i različite arhitekture koje omogućuju računalima da nauče prepoznavati osobe na temelju dlanova.

U prvom dijelu rada pružit ću pregled dubokog učenja, obuhvaćajući osnovne koncepte, principe i metode. Objasnit ću kako neuronske mreže s više slojeva rade zajedno s različitim algoritmima. Također ću istražiti različite vrste dubokih neuronskih mreža, poput konvolucijskih neuronskih mreža (CNN) i rekurentnih neuronskih mreža (RNN), koje su se pokazale iznimno učinkovitima u prepoznavanju i klasifikaciji slika.

U drugom dijelu rada fokusirat ću se na primjenu dubokog učenja u prepoznavanju osoba temeljem dlana. Detaljno ću opisati prethodno istrenirane arhitekture ili modele koji su dostupni javnosti i koji se koriste kao temelj za klasifikaciju slika ili podataka općenito. Takvi modeli već su naučili značajke iz velikih skupova podataka i omogućuju nam da iskoristimo njihovo naučeno znanje i primijenimo ih na naše vlastite podatke.

Nakon što detaljno objasnim ove arhitekture, pristupit ću eksperimentima i evaluaciji. Koristit ću odabrani skup podataka s uzorcima dlana za treniranje i testiranje razvijenih modela. Analizirat ću performanse modela, kao i njihovu sposobnost prepoznavanja osoba na temelju dlana.

U zaključku rada sumirat ću glavne rezultate istraživanja i eksperimenata. Ovaj rad pružit će analizu primjene računala za stvari koje su pojedinom čovjeku nemoguće te će istaknuti prednosti i nedostatke dubokog učenja u svrhu klasifikacije podataka.

Poglavlje 2

Duboko učenje

2.1 Što je duboko učenje

Duboko učenje (engl. *deep learning*)[1] je grana strojnog učenja koja se temelji na konceptu neuronskih mreža s više slojeva. To je tehnika koja omogućuje računalima da autonomno uče i donose zaključke na temelju podataka. Duboko učenje ističe se u analizi velikih i kompleksnih skupova podataka, poput teksta, zvuka i slika. Koristi proces učenja na temelju primjera, gdje modeli neuronskih mreža prolaze kroz velike skupove podataka kako bi naučili izgraditi interne reprezentacije podataka.

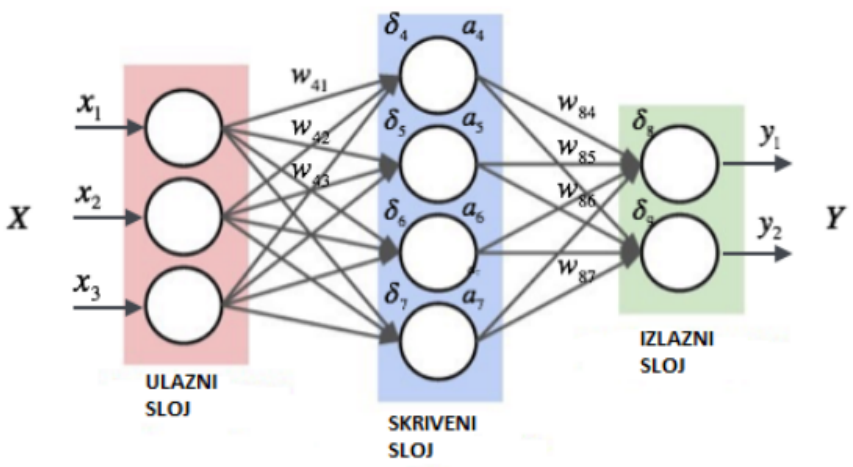
Kod strojnog učenja, za razliku od dubokog učenja, značajke iz ulaznih podataka se ručno izrađuju i biraju. Međutim, duboko učenje ima sposobnost automatskog izvlačenja značajki iz sirovih podataka putem upotrebe neuronskih mreža s više slojeva, poznatih i kao duboke neuronske mreže. Ove mreže su dizajnirane kako bi što vjernije oponašale ljudski mozak, sastavljene su od slojeva neurona koji obrađuju podatke i prenose informacije među sobom.

Jedna od prednosti dubokog učenja je njegova sposobnost obrade visokodimenzionalnih podataka, kao što su slike visoke rezolucije. Kroz učenje, duboke neuronske mreže mogu otkriti složene značajke koje ne možemo vidjeti golim okom, te omogućuju prepoznavanje objekata, obrisa ili lica. U kontekstu mog istraživanja, duboko učenje može se primjeniti na prepoznavanje osoba na temelju slike dlana.

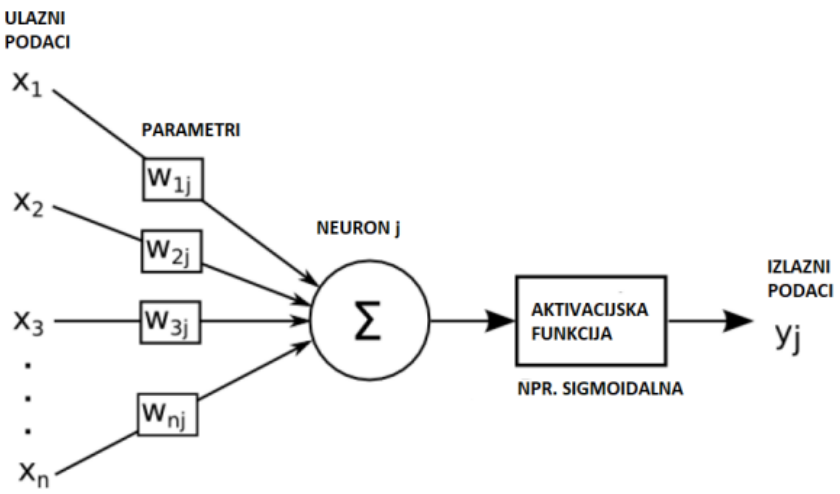
2.2 Neuronske mreže

Većina metoda dubokog učenja koriste varijacije arhitektura neuronskih mreža i zato se modele dubokog učenja često naziva i duboke neuronske mreže.

Riječ "duboke" se odnosi na broj skrivenih slojeva unutar neuronske mreže. Osim skrivenih postoje i ulazni te izlazni sloj koji se koriste kako bi se korigirali parametri podataka koje unosimo u neuronsku mrežu te modificiranje izlaznih podataka ovisno o našim potrebama. Na slici 3.13a možemo vidjeti grafički prikaz strukture neuronske mreže. Broj skrivenih slojeva može varirati i direktno pridonosi složenosti kompletne neuronske mreže.



Slika 2.1 Prikaz strukture neuronske mreže, [2]



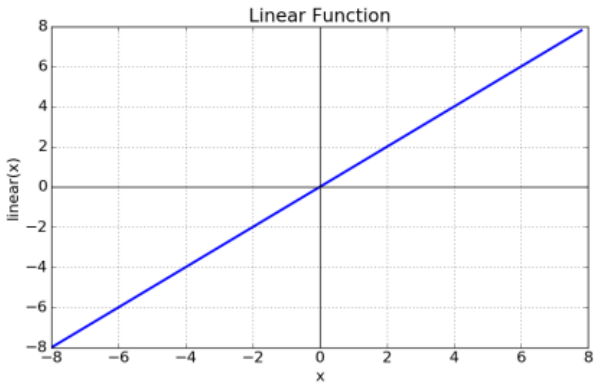
Slika 2.2 Prikaz jednog neurona, [2]

Na slici 3.13b vidimo strukturu i funkciju pojedinog neurona zajedno sa svojim ulazima i izlazima. Funkcija jednog neurona u neuronskoj mreži naziva se aktivacijska funkcija koja prima težinske vrijednosti i ulazne podatke neurona te na temelju toga generira izlazni signal. Najčešće korištene aktivacijske funkcije:

- **Linearna aktivacijska funkcija** koja ne obavlja nikakvu transformaciju nad ulaznim podacima. Izlazni signal jednak je zbroju težinskih vrijednosti pomnoženih s ulaznim podacima

$$f(x) = wx + b$$

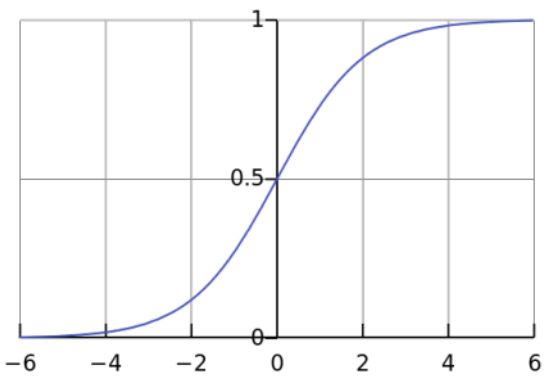
w je težinska vrijednost, x ulazni podatak, a b pomak neurona



Slika 2.3 Linearna funkcija, [2]

- **Sigmoidna aktivacijska funkcija** koja transformira ulazne podatke u rasponu između 0 i 1

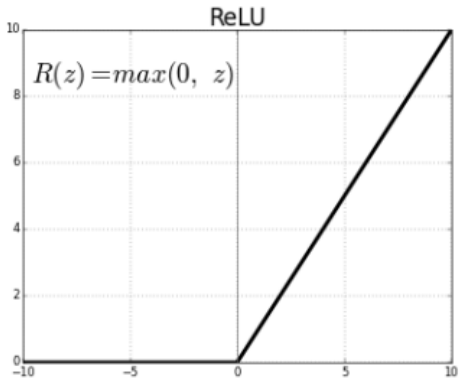
$$f(x) = \frac{1}{(1 + e^{-x})}$$



Slika 2.4 Sigmoidna funkcija, [2]

- **ReLu(*Rectified Linear Unit*) aktivacijska funkcija** koja daje izlazni signal jednak nuli za sve negativne ulazne vrijednosti dok pozitivne ulaze propušta

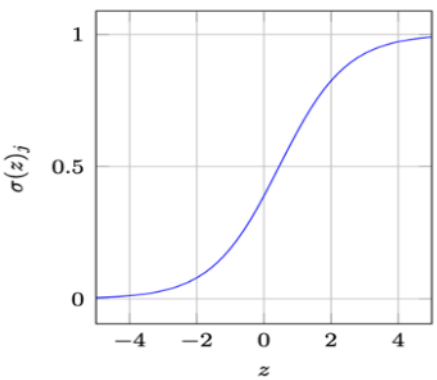
$$f(x) = max(0, x)$$



Slika 2.5 ReLu funkcija, [2]

- **Softmax aktivacijska funkcija** koja transformira ulazne vrijednosti u vjerojatnosti koje zbrojeno daju 1. Koristi se za vjerojatnost pripadanja svakoj klasi

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$



Slika 2.6 Softmax funkcija, [2]

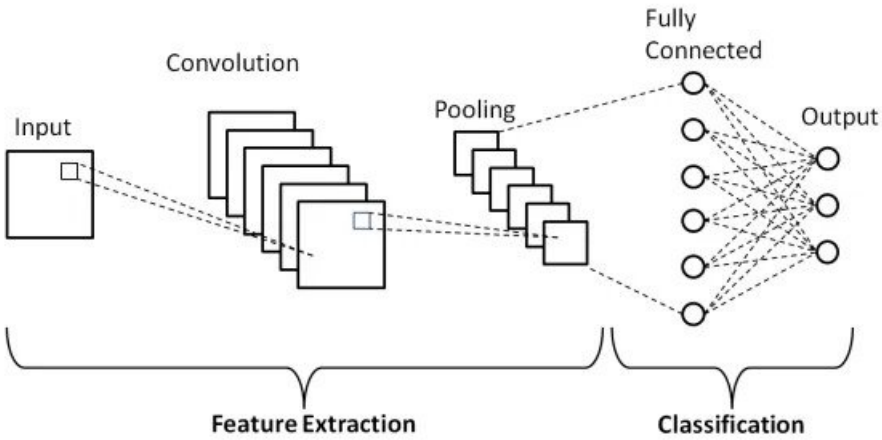
2.3 Konvolucijske neuronske mreže

Postoje različite vrste neuronskih mreža koje se koriste za različite primjene i na različitim vrstama podataka. Na primjer, rekurentne neuronske mreže često se koriste za obradu prirodnog jezika i prepoznavanje govora, dok se konvolucijske neuronske mreže (CNN) češće koriste za klasifikaciju i računalni vid. Prije CNN-a, za identifikaciju objekata na slikama koristile su se ručne, vremenski zathjevne, metode ekstrakcije značajki. Prije CNN-a, ekstrakcije značajki za identifikaciju objekata na slikama obavljale su se ručno što je zahtjevalo puno vremena. Međutim, konvolucijske neuronske mreže sada pružaju skalabilan pristup klasifikaciji slika i prepoznavanju objekata, koristeći principe linearne algebre, posebno matrično množenje, za identifikaciju uzoraka unutar slike. Međutim, zahtijevaju računalnu snagu i obično se koriste grafičke procesne jedinice (GPU) za treniranje modela.

Konvolucijske neuronske mreže[3] ističu se u usporedbi s drugim neuronskim mrežama zbog svoje izvrsne izvedbe s ulazima u obliku slika, govora ili audio signala. Imaju tri glavne vrste slojeva:

- Konvolucijski sloj (engl. *convolutional layer*)
- Sloj združivanja (engl. *pooling layer*)
- Potpuno povezani sloj (engl. *fully-connected layer*)

Konvolucijski sloj je prvi sloj u konvolucijskoj mreži. Iako konvolucijski slojevi mogu biti praćeni dodatnim konvolucijskim slojevima ili slojevima združivanja, potpuno povezani sloj je završni sloj. Vizualni prikaz slojeva vidimo na slici 2.7. Svakim slojem, konvolucijska neuronska mreža povećava svoju složenost i prepoznaje veće dijelove slike. Raniji slojevi se fokusiraju na jednostavne značajke poput boja i rubova. Kako podaci o slici napreduju kroz slojeve konvolucijske neuronske mreže, počinje prepoznavati veće elemente ili oblike objekta sve dok konačno ne prepozna ciljni objekt.



Slika 2.7 Slojevi CNN-a, [4]

2.3.1 Konvolucijski sloj

Konvolucijski sloj[3] je osnovna građevna jedinica konvolucijske neuronske mreže (CNN) i mjesto gdje se većina računanja odvija. Sastoji se od nekoliko komponenti, uključujući ulazne podatke, filter i značajnu mapu. Ulaz ima tri dimenzije kako bi odgovarao RGB slikama. Detektor značajki, poznat i kao jezgra ili filter, kreće se po poljima slike provjeravajući prisutnost određene značajke. Ovaj proces nazivamo konvolucijom.

Detektor značajki je dvodimenzionalni (2-D) niz težina koji predstavlja dio slike. Veličina jezgre obično je matrica dimenzija 3x3, što određuje veličinu recepcijskog polja. Jezgra se primjenjuje na područje slike, a između piksela ulaza i jezgre računa se skalarni produkt. Taj skalarni produkt se zatim pohranjuje u izlazni niz. Nakon toga, jezgra se pomakne za korak i postupak se ponavlja dok jezgra ne obiđe cijelu sliku. Konačni izlazni niz koji dobijemo iz serije skalarnih produkata naziva se značajna mapa, mapa aktivacija ili konvoluirana značajka.

Nakon svake operacije konvolucije, CNN primjenjuje transformaciju ReLU (engl. *Rectified Linear Unit*) na značajnu mapu, čime se uvodi nelinearnost u model.

2.3.2 Sloj združivanja

Slojevi združivanja[3], također poznati kao slojevi smanjivanja dimenzionalnosti, provode smanjivanje broja parametara ulaza. Slično kao i kod konvolucijskog sloja, operacija sažimanja provodi kretanje filtera preko cijelog ulaza, ali razlika je u tome što ovaj filter nema težine. Umjesto toga, jezgra primjenjuje funkciju agregacije na vrijednosti unutar recepcijskog polja i popunjava izlazni niz.

Postoje dva glavna tipa združivanja:

- **Maksimalno sažimanje (engl. *max pooling*):** Dok se filter kreće preko ulaza, odabire piksel s najvećom vrijednošću i šalje ga u izlazni niz. Ova metoda se češće koristi u usporedbi s prosječnim sažimanjem
- **Prosječno sažimanje (engl. *average pooling*):** Dok se filter kreće preko ulaza, izračunava prosječnu vrijednost unutar recepcijskog polja i šalje je u izlazni niz. Iako se u sloju sažimanja gubi puno informacija, on također ima nekoliko prednosti za CNN. Pomaže u smanjenju složenosti, poboljšava učinkovitost i smanjuje rizik od pretreniranja

2.3.3 Potpuno povezani sloj

Naziv sloja potpuno povezan (fully-connected layer) [3] opisuje sam sebe. Kao što je ranije spomenuto, vrijednosti piksela ulazne slike nisu izravno povezane s izlaznim slojem u djelomično povezanim slojevima. Međutim, u potpuno povezanom sloju, svaki čvor u izlaznom sloju izravno se povezuje s čvorom u prethodnom sloju.

Ovaj sloj obavlja zadatke klasifikacije na temelju značajki izvučenih iz prethodnih slojeva i njihovih različitih filtara. Dok konvolucijski i sažimajući slojevi obično koriste ReLu funkcije, potpuno povezani slojevi obično koriste softmax aktivacijsku funkciju kako bi pravilno klasificirali ulaze, proizvodeći vjerojatnost od 0 do 1.

Poglavlje 3

Opis praktičnog dijela

3.1 Baze podataka

Prvi korak pri početku rada na projektu bio je istraživanje i pronalazak baza podataka slika dlanova koje su sadržavale dovoljan broj raznolikih primjeraka. Pronalaženje prikladnih baza podataka bilo je ključno za dobivanje dovoljno raznolikog skupa podataka koji je potreban za trening i evaluaciju modela dubokog učenja.

Pri odabiru baza podataka, važno je uzeti u obzir nekoliko faktora. Prvo, baze podataka trebaju sadržavati slike visoke kvalitete s dovoljnom razlučivošću kako bi se omogućilo precizno izvlačenje karakteristika i značajki. Također, baze podataka trebaju imati dovoljan broj različitih primjeraka dlanova kako bi se modeli mogli naučiti razlikovati i prepoznati različite osobine i varijacije.

Pronalaženje baza bio je izazov te sam morao poslati veliki broj e-mailova tražeći dozvolu za korištenje podataka od pojedinih osoba ili o institucija koje imaju prava na te podatke. Na nekoliko upita sam dobio pozitivan odgovor od kojih su samo dve baze sadržavale dovoljno slika u zadovoljavajućoj kvaliteti.

3.1.1 11k Hands

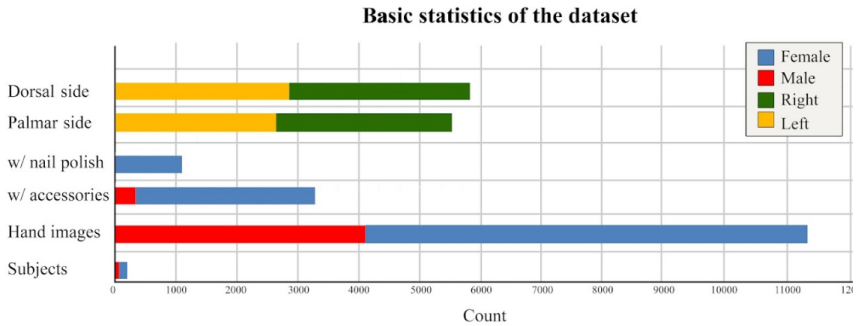
Prva baza podataka korištena u ovom radu je *11k Hands* [5]. Ova kolekcija sadrži 11,000 slika veličine *1600 x 1200*, koje obuhvaćaju 190 subjekata u dobi od 18 do 75

godina. Svaki subjekt je fotografiran s otvorenim i skupljenim dlanom, a u sklopu baze podataka mogu se pojaviti i neočekivani objekti poput prstena, sata, narukvice, laka za nokte ili tetovaže.

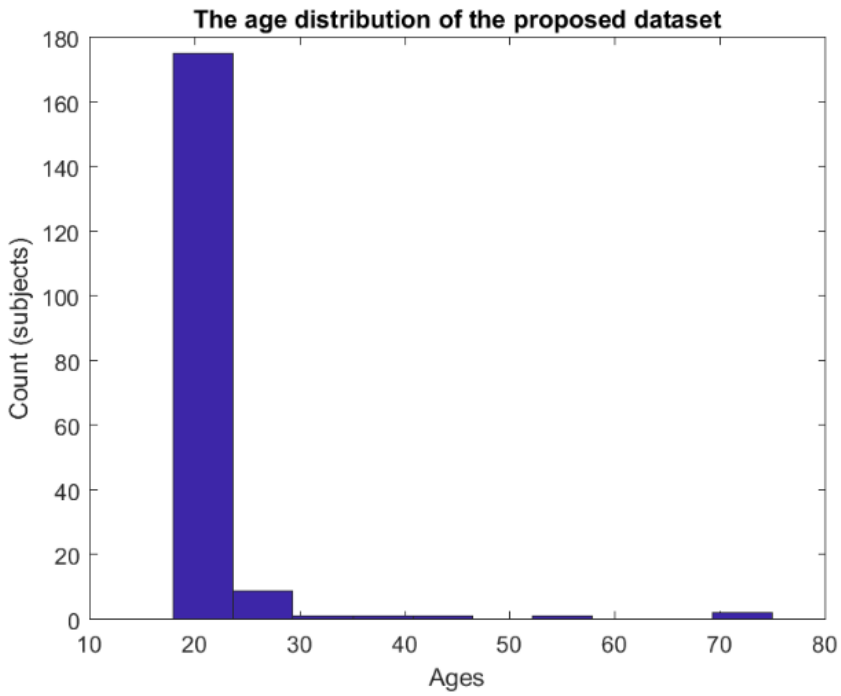
Baza podataka *11k Hands* pruža vrijedan skup podataka s raznolikim primjerima dlana. Ova raznolikost omogućava bolje učenje modela dubokog učenja, jer modeli imaju priliku naučiti razlikovati različite oblike, teksture i karakteristike dlanova. Osim toga, prisutnost neočekivanih objekata uključenih u fotografije pomaže modelima da nauče prepoznati i obrađivati potencijalne smetnje ili izazove koji se mogu pojaviti u stvarnom svijetu.

Korištenje baze podataka *11k Hands* pruža mogućnost istraživanja i evaluacije modela dubokog učenja za prepoznavanje osoba temeljem dlana. Kombinacija raznolikih primjera dlanova i prisutnost neočekivanih objekata omogućuje bolje razumijevanje i evaluaciju performansi modela u realnim uvjetima.

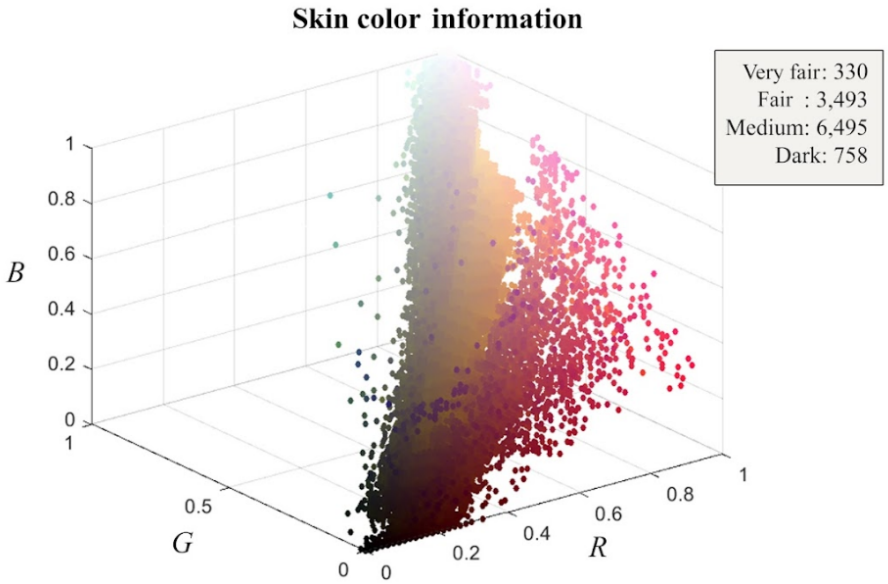
Iz baze je bilo potrebno ukloniti određen broj primjeraka koji su prikazivali dorsalnu stranu dlana.



Slika 3.1 Sastav baze podataka, [5]



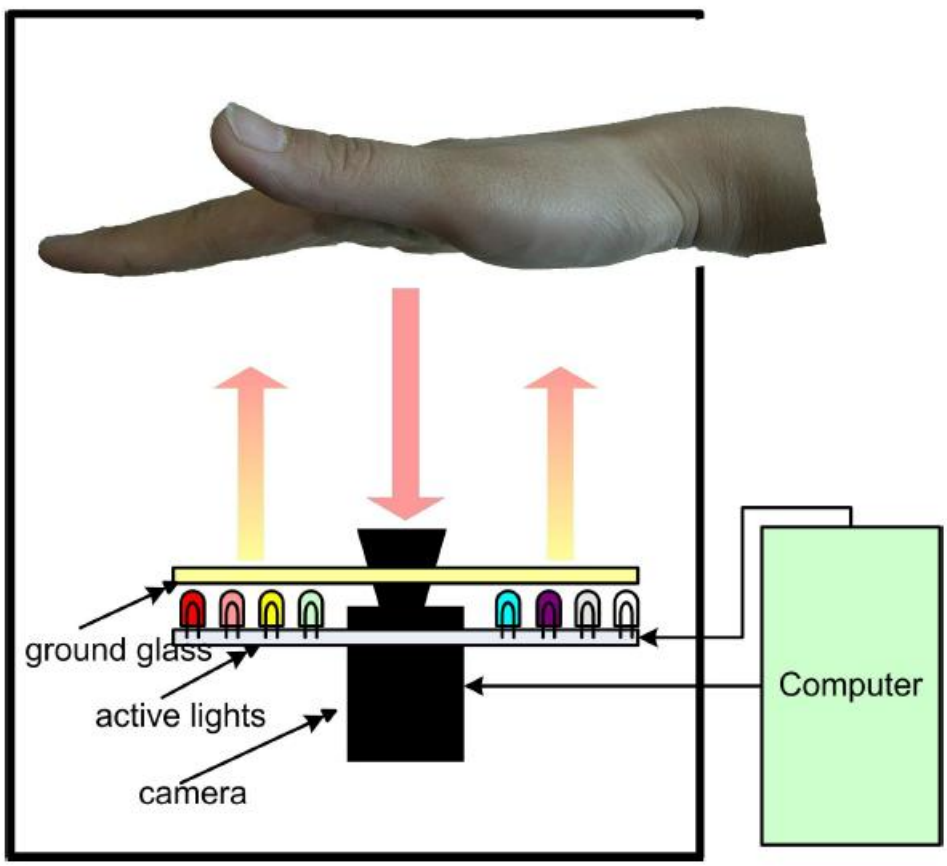
Slika 3.2 Raspodjela po godinama, [5]



Slika 3.3 Raspodjela po boji kože, [5]

3.1.2 CASIA-palmprint

Druga baza podataka kojoj sam dobio pristup je *CASIA Multi-Spectral Palmprint Image Database V1.0*. Slike u ovoj bazi su snimljene pomoću samorazvijenog uređaja za višespektralno snimanje prikazanog na slici 3.4.



Slika 3.4 Uređaj za višespektralno snimanje, [6]

[6] U istraživanjima biometrije, kombiniranje više modaliteta snimanja dokazano je kao obećavajući način poboljšanja performansi prepoznavanja. Prema elektromagnetskoj teoriji, Hertzove valove koji se kreću od vidljivog svjetlosnog spektra do bliskog infracrvenog spektra karakterizira sve veća prodiruća snaga u objekte. Za biometriju šake, višespektralni osvjetlivač može prodrijeti u potkožna tkiva na

različitim dubinama u područjima dlana i stvarati slike površinskih tekstura kože i hipodermičkih elemenata, uključujući žile dlana.

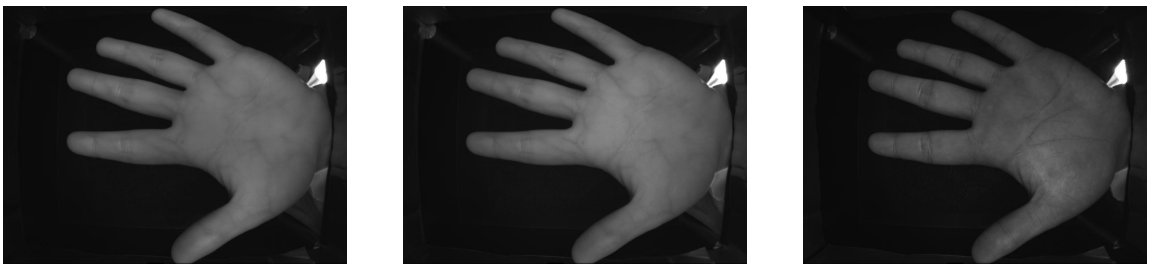
CASIA Multi-Spectral Palmprint Image Database sadrži 7,200 slika dlana snimljenih s 100 različitih osoba pomoću samorazvijenog uređaja za višespektralno snimanje, prikazanog na slici 1. Sve slike dlana su JPEG datoteke u 8-bitnom sivom tonu. Za svaku ruku, snimamo dvije sesije slika dlana. Vremenski interval između dviju sesija je više od jednog mjeseca. U svakoj sesiji postoje tri uzorka. Svaki uzorak sadrži šest slika dlana snimljenih istovremeno s šest različitih elektromagnetskih spektara. Valne duljine osvjetlivača koji odgovaraju tim spektrima su 460 nm, 630 nm, 700 nm, 850 nm, 940 nm i bijela svjetlost. Između dva uzorka dopušten je određeni stupanj varijacija položaja ruke. Time je povećana raznolikost unutarrazrednih uzoraka i simulirana praktična upotreba.

Za snimanje je korištena CCD(*charged-coupled device*) kamera[7] koja sadrži uređaj s naponski vezanim sklopom, koji je tranzistorski senzor svjetla na integriranom krugu. Ovaj uređaj manipulira električni signal u neku vrstu izlaza, uključujući digitalne vrijednosti.

(a) 460 *nm*

(b) 630 *nm*

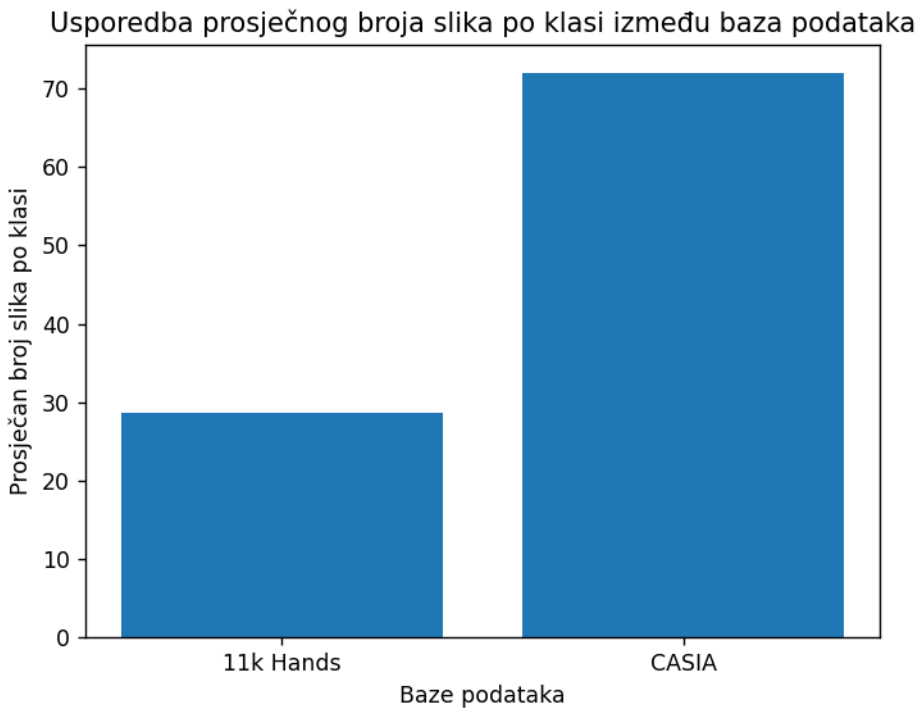
(c) 700 *nm*



(a) 850 *nm* (b) 940 *nm* (c) *white light respectively*

Slika 3.6

Na slikama 3.6 prikazani su rezultati snimaka dlanova koristeći višespektralni uređaj za snimanje.



Slika 3.7 Prosjek slika po klasi

3.2 Priprema podataka i okruženja

Prvi korak nakon pronalaska baza podataka bio je njihova priprema i uređivanje kako bi se moglo raditi s njima. Kada smo preuzeli baze podataka, bile su organizirane kao direktoriji u kojima su sve slike bile smještene na jednom mjestu. Svaka slika je bila označena određenim informacijama, kao što su informacije o tome je li riječ o lijevoj ili desnoj ruci, dlanu ili dorsalnom dijelu šake, te je li prisutno nešto poput smetnji ili dodataka na ruci.

U *11k Hands* bazi ti podaci su bili u obliku CVS*comma-separated values* datoteke što je format za spremanje tabličnih podataka. U tim datotekama podaci su organizirani u redove i stupce pri čemu je prvi red rezerviran za nazive stupaca.

Primjer prvog stupca:

```
id,age,gender,skinColor,accessories,nailPolish,aspectOfHand,
imageName,irregularities
```

Način organizacije podataka u CSV formatu olakšava rad s podacima jer omogućuje jednostavnu iteraciju po redovima i stupcima kako bi se dobile i manipulirale željene karakteristike za svaku sliku. Iteriranjem po redovima, možemo pristupiti pojedinačnim slikama i njihovim pripadajućim informacijama, kao što su oznake, položaj, prisutnost smetnji itd. Na taj način, možemo lako izvući specifične atribute ili podatke koje želimo analizirati ili koristiti u daljnjoj obradi.

3.2.1 Podjela podataka

Tipična podjela podataka kada je u pitanju duboko učenje je razvrstavanje na:[8]

- **Train -** Direktorij gdje se nalaze podaci na kojima se model trenira
- **Test -** skup podataka koji pruža referentnu vrijednost koja se koristi za evaluaciju modela. Koristi se samo nakon što je model potpuno treniran (koristeći skupove za treniranje i validaciju). Obično se koristi za usporedbu različitih modela

- **Validation -** koristi se za pružanje evaluacije modela koji je prilagođen skupu za treniranje, tj. pomoću ovog skupa podešavamo hiperparametre i na taj način neizravno utječemo na model

Slika 3.8 Podjela podataka, [9]

Količina podataka koju ćemo smjestiti u pojedini podskup ovisi o strukturi početne baze podataka koju smo preuzeli. Svaka baza ima različiti broj klasa i instanci te klase pa moramo sami procijeniti idealnu podjelu. Na slici 3.8 možemo vidjeti vizualni prikaz kako bi trebao izgledati omjer podataka u pojedinoj skupini.

Za uspješno treniranje modela, najveći dio podataka treba biti u skupu za treniranje (*train*) jer modelu je potreban relativno velik skup podataka za učenje. Također je važno da se podaci za treniranje i testiranje ne preklapaju, kako bi testiranje obuhvatilo podatke koje model dosad nije vidio. To omogućava preciznije mjerenje uspješnosti treninga modela.

U ovom radu korištena je podjela samo na skupove za treniranje (*train*) i testiranje (*test*), bez dodatnog *validation* skupa. Razlog tome je što su baze podataka koje sam koristio relativno male po broju slika po klasi, ponekad čak samo 2-3 slike. Stoga sam morao pažljivo rasporediti dostupne podatke za treniranje. Tijekom procesa treniranja primijetio sam da su rezultati dovoljno dobri i da mi dodatni *validation* skup nije bio potreban.

Ručna raspodjela podataka bi, naravno, bila nemoguća zbog velikog broja podataka koji se moraju premještati i preimenovati kako bi se stvorila baza podataka spremna za treniranje. Sve modifikacije koje sam morao obaviti nad podacima sam uspio uz pomoć skripti koje sam pisao u programskom jeziku Python.

Prvu modifikaciju koju sam obavio bila je podjela oba skupa podataka na tri, jedan u kojem su bile slike samo desnih dlanova, jedan samo sa lijevim dlanovima i jedan gdje su i lijevi i desni dlanovi. Ova podjela je provedena kako bih mogao analizirati rezultate treniranja i istražiti koliko su slični ili identični lijevi i desni dlanovi kod ljudi. Python skripta temeljila se na čitanju podataka iz CSV datoteke gdje za pojedinu sliku piše koji dlan se nalazi na njoj te lokaciju slike na računalu, sa te dvije informacije i jednom *if* petljom višesatno ručno modificiranje svodimo na svega par sekundi.

Druga modifikacija bila je preimenovati svaku sliku kako bi bez gledanja u CSV datoteku mogao očitati kojoj klasi slika pripada, koja je po redu instanca te klase te je li dlan na slici lijevi ili desni.

Sada je sve podatke trebalo podijeliti na *train* i *test* za što sam ponovo napisao Python skriptu u kojoj sam implementirao *skicit-learn*[10] funkciju *train_test_split* koja je napisana specifično za ovakve zadatke te ju je lako iskoristiti naredbom *from sklearn.model_selection import train_test_split*.

Krajnji zadatak bio je podatke unutar *train* i *test* direktorija podijeliti na direktorije klasa unutar kojih će biti slike dlanova koje pripadaju osobi te klase.

Konačna podjela podataka na kraju je izgledala ovako 3.9:

```
Dataset
├── train
|   ├── class_1
|   |   ├── image_1.jpg
|   |   └── ...
|   ├── class_2
|   |   ├── image_2.jpg
|   |   └── ...
|   └── ...
└── test
    ├── class_1
    |   ├── image_1.jpg
|   |   └── ...
    ├── class_2
    |   ├── image_1.jpg
|   |   └── ...
    └── ...
```

Slika 3.9 Izgled podjele podataka

Dogovor s mentorom i kolegama bio je da podjela podataka bude takva da 80% slika koristimo za treniranje modela, dok 20% slika koristimo za testiranje. Koristimo odgovarajući argument u navedenoj funkciji koja nasumično odabire taj postotak slika i smješta ih u mapu za treniranje, dok preostale slike premješta u mapu za testiranje.

3.3 Okruženje

3.3.1 Google Colaboratory

Treniranje modela na računalu obično zahtijeva značajne resurse radne memorije i procesorske snage. S obzirom na ograničene mogućnosti mog prijenosnog računala, odlučio sam koristiti Google Colaboratory kao okruženje za rad. Google Colab je besplatna online platforma koja pruža virtualno okruženje za izvršavanje Python koda. Ona omogućuje pristup GPU-ima *Graphics Processing Unit* i TPU-ovima (*Tensor Processing Units*) za ubrzano izvođenje računalno intenzivnih operacija, uključujući treniranje dubokih neuronskih mreža. Colab također pruža mogućnost suradnje i dijeljenja projekata s drugim korisnicima te integraciju s Google-ovim servisima za pohranu podataka poput Google Drive-a. Odabirom Google Colab-a kao okruženja za rad, mogu iskoristiti njihove računalne resurse i olakšati proces treniranja modela na ograničenom hardveru svog laptopa.

Colab kao besplatnu opciju nudi Tesla T4 GPU razvijenu od strane NVIDIA firme. [11] Tesla T4 posebno je dizajnirana za ubrzavanje paralelnih računalnih radnih opterećenja, što znači da je pogodna za strojno i duboko učenje, znanstvene simulacije i sve paralelne računalne operacije. T4 je temeljen na arhitekturi Turing, a ima 16GB GDDR6 VRAM-a, dok maksimalna propusnost memorije iznosi 320 GB/s. Još jedan potreban element kako bi omogućili upotrebu GPU-a za obavljanje treniranja je instalacija i postavljanje platforme CUDA. CUDA[12](*Compute Unified Device Architecture*) je paralelna računalna platforma i programski model razvijen također od strane NVIDIA-e. Omogućuje nam ubrzavanje aplikacija korištenjem GPU akceleratora. CUDA verzija koja je korištena u ovom radu je *12.0*.

3.3.2 PyTorch

Duboko učenje je složen posao koje nam uvelike olakšavaju već napisane funkcije i algoritmi u obliku gotovih knjižnica i platformi dostupnih javnosti. Neke od najpopularnijih i najkorištenijih su:

- TensorFlow
- Keras
- PyTorch
- scikit-learn
- XGBoost
- Caffe
- Theano

Po dogovoru s mentorom u radu je korištena PyTorch[13] knjižnica. PyTorch je kreiran od strane *Meta AI* i otvorenog je koda. PyTorch kao osnovne jedinice podataka koristi tenzore. [14] U strojnom učenu, tenzor se odnosi na dva različita koncepta koja organiziraju i predstavljaju podatke. Podaci se mogu organizirati u višedimenzionalno polje (M-polje) koje se neformalno naziva "podatkovni tenzor". Međutim, u strogo matematičkom smislu, tenzor je multilinearno preslikavanje iz skupa vektorskih prostora domene u vektorski prostor raspona. Podaci, poput slika, filmova, volumena, zvukova i odnosa između riječi i pojmova, pohranjeni u M-polju ("podatkovni tenzor"), mogu se analizirati umjetnim neuronskim mrežama ili tenzorskim metodama. Tenzorske metode mogu faktorizirati podatkovne tenzore na manje tenzore. Operacije nad podatkovnim tenzorima mogu se izraziti putem matričnog množenja i Kroneckerovog produkta. Izračunavanje gradijenta, važan aspekt algoritma propagacije unatrag, može se izvoditi pomoću biblioteka PyTorch i TensorFlow.

3.4 Pisanje koda

3.4.1 Dataloader

Prilikom pisanja koda, prvi korak je bio uspješno učitavanje podataka. PyTorch pruža klasu koja je upravo za tu svrhu pod imenom 'DataLoader'. DataLoader[15] je pomoćna klasa koja koristi za učitavanje podataka iz baza podataka i stvaranje mini-batcheva za treniranje dubokih modela učenja. Namijenjena je obradi velikih skupova podataka i izvođenju postupaka kao što su proširivanje podataka, miješanje i druge predobrade. Prednosti korištenja PyTorch DataLoader klase[15]:

- **Učinkovito učitavanje podataka:** Klasa DataLoader omogućava učinkovito učitavanje podataka omogućavajući korisniku da istovremeno koristi više CPU jezgri za paralelno učitavanje podataka. To može značajno smanjiti vrijeme učitavanja podataka, pogotovo za velike skupove podataka
- **Proširivanje podataka:** Klasa DataLoader pruža mogućnost proširivanja podataka, omogućavajući korisniku da primijeni razne transformacije na podatke tijekom učitavanja. To može pomoći povećanju veličine skupa za treniranje i poboljšanju generalizacije modela
- **Fleksibilnost:** Klasa DataLoader je izuzetno fleksibilna i može raditi s različitim formatima i izvorima podataka. Podržava učitavanje podataka iz datoteka, baza podataka i drugih vanjskih izvora, kao i predobradu podataka korištenjem prilagođenih funkcija
- **Miješanje podataka(*shuffling*):** omogućava miješanje podataka, što omogućava korisniku da miješa podatke za svaku epohu. To može pomoći u sprječavanju *overfitting-a* modela na skupu za treniranje i poboljšati njegovu generalizaciju

```
    train_loader = DataLoader(
    torchvision.datasets.ImageFolder(train_path, transform=transformer),
    batch_size=32,
    shuffle = True
)
```

3.4.2 Učitavanje podataka

U dubokom učenju modeli neuronskih mreža zahtijevaju određena pravila prilikom učitavanja, modificiranja i rad s podacima. Važno je da utvrdimo značenje naziva kao što su *sample*, *batch*, *epoch*, *shuffling* itd.

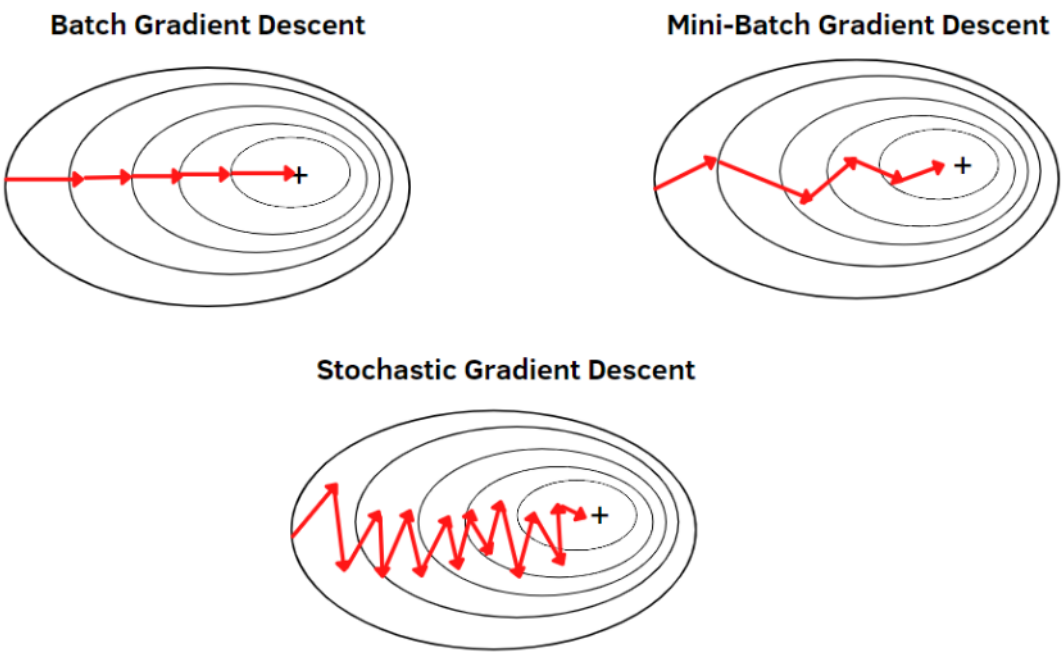
Uzorak(engl. *sample*)[16] je pojedinačan redak podataka. Sadrži ulazne vrijednosti koje se unose u algoritam i izlaznu vrijednost koja se koristi za usporedbu s predikcijom i izračunavanje pogreške. Skup podataka za treniranje sastoji se od mnogo redaka podataka, tj. mnogo primjeraka. Primjerak se također može nazvati instanca, promatranje, vektor ulaza ili vektor značajki.

Batch(engl. *grupa*) [16][15]je osnovna jedinica podataka koju koriste modeli dubokog učenja. Batch je uglavnom tenzor oblika (batch_size, input_shape). Veličina grupe, odnosno *batch_ size*, može se namjestiti kao argument prilikom kreiranja objekta DataLoader. Veličina grupe predstavlja broj podataka koji se šalju u model na treniranje odjednom. Veće veličine grupe mogu rezultirati bržim treniranjem modela, jer se više podataka obrađuje paralelno, ali zahtijeva više memorijskog prostora, pa ako memorija nije dovoljno velika, može doći do prekoračenja memorije. S druge strane, manje veličine batcha mogu omogućiti bolje generaliziranje modela. Na kraju grupe, predikcije se uspoređuju s očekivanim izlaznim varijablama i računa se pogreška što omogućuje ažuriranje težina modela na temelju manjeg broja primjera. Ovo može pomoći u izbjegavanju prenaučenosti (engl. *overfittinga*) i poboljšanju generalizacije modela. Međutim, manji batchevi mogu rezultirati sporijim treniranjem modela, jer se ažuriranje težina događa češće.

Na mom primjeru *batch_ size* veći od 32 je stvarao probleme sa memorijom pa sam se odlučio za taj broj. Kada se svi primjerci za treniranje koriste za stvaranje jedne grupe, algoritam učenja naziva se **gradijentni spust s potpunom grupom**

(engl. *batch gradient descent*). Kada je veličina grupe jedan primjerak, algoritam učenja naziva se **stohastički gradijentni spust** (engl. *stochastic gradient descent*). Kada je veličina grupe veća od jednog primjerka, ali manja od veličine skupa za treniranje, algoritam učenja naziva se **gradijentni spust s mini-grupom** (engl. *mini-batch gradient descent*).

Algoritam gradijentnog spusta[17] je optimizacijski algoritam koji se uglavnom koristi u strojnom učenju i dubokom učenju. Gradijentni spust prilagođava parametre kako bi se minimizirale određene funkcije do lokalnih minimuma. U linearnoj regresiji pronalazi težine i pomake, a u dubokom učenju se koristi u postupku unatrag propagacije. Cilj algoritma je identificirati modelne parametre poput težina i pomaka koji smanjuju pogrešku modela na skupu podataka za treniranje.



Slika 3.10 Prikaz tipova gradijentnog spusta, [17]

Gradijentni spust po grupama, također poznat kao obični gradijentni spust, računa pogrešku za svaki primjer unutar skupa podataka za treniranje. Međutim, model se ne mijenja dok se ne procijeni svaki primjer za treniranje. Cjelokupni postupak naziva se ciklus i jedna obuka.

Za razliku od toga, stohastički gradijentni spust (SGD) mijenja parametre za svaki primjer treniranja jedan po jedan za svaki primjer u skupu podataka za treniranje. Ovisno o problemu, to može učiniti SGD bržim od grupnog gradijentnog spusta. Jedna prednost je da redovna ažuriranja daju prilično točnu ideju o stopi poboljšanja.

Pošto mini-batch gradijentni spust kombinira ideje grupnog gradijentnog spusta s SGD-om, to je preferirana tehnika. On dijeli skup podataka za treniranje na upravljive grupe i ažurira ih odvojeno. Time se postiže ravnoteža između učinkovitosti grupnog gradijentnog spusta i izdržljivosti stohastičkog gradijentnog spusta.

Sve gore navedeni algoritmi su optimizatori. **Optimizator**[18] je algoritam ili metoda koji se koristi za minimiziranje funkcije pogreške (funkcije gubitka) ili za maksimiziranje učinkovitosti proizvodnje. Optimizatori su matematičke funkcije koje ovise o naučljivim parametrima modela, tj. o težinama i pristranostima. Optimizatori pomažu u određivanju kako promijeniti težine i stopu učenja neuronske mreže kako bi se smanjile gubitci. Postoji puno optimizatora a ovo su neki od popularnijih. Na ovom radu korišten je mini-batch gradijentni spust kao optimizator.

Miješanje podataka(engl. *shuffling*) je postupak nasumičnog preuređivanja podataka u skupu podataka. Miješanje je važno prilikom treniranja dubokih modela učenja jer sprječava model da se previše prilagodi redoslijedu podataka. Klasa DataLoader u PyTorchu omogućuje miješanje podataka za svaku epohu. Postavkom '`shuffle = True`' prilikom kreiranja objekta DataLoader-a omogućujemo miješanje čime klasa nasumićno mijenja indekse podataka i vraća podatke u nasumičnom redoslijedu za svaku epohu.

Kako moj skup podataka ima relativno puno klasa i malo slika unutar svake klase morao sam implementirati proširivanje podataka kako bi imao raznolikost prilikom treniranja. To se ostvaruje pomoću transformacija koje se primjenjuju prilikom učitavanja podataka. Neke od transformacija koje sam koristio su skaliranje slika da zadovoljim ulazni sloj modela kojeg sam koristio. Također sam sve slike pretvorio u crno-bijele (engl. *grayscale*) jer mi boje nisu bile potrebne, a složenost analize slike je manje kad za ulaz imamo jedan kanal umjesto 3. Još jedna transformacija koju sam primijenio je normalizacija što je postupak kojim se skalira svaki atribut (ili značajka) na raspon vrijednosti kako bi svi atributi imali sličan utjecaj na model. Za proširivanje podataka koristio sam '`RandomHorizontalFlip()`' i

'`RandomRotation(degrees=)`' kako bi se pojedine slike nasumično modificirale što modelu daju nove nepredviđene podatke i na taj način treniranje postaje raznoliko, a rezultati bolji.

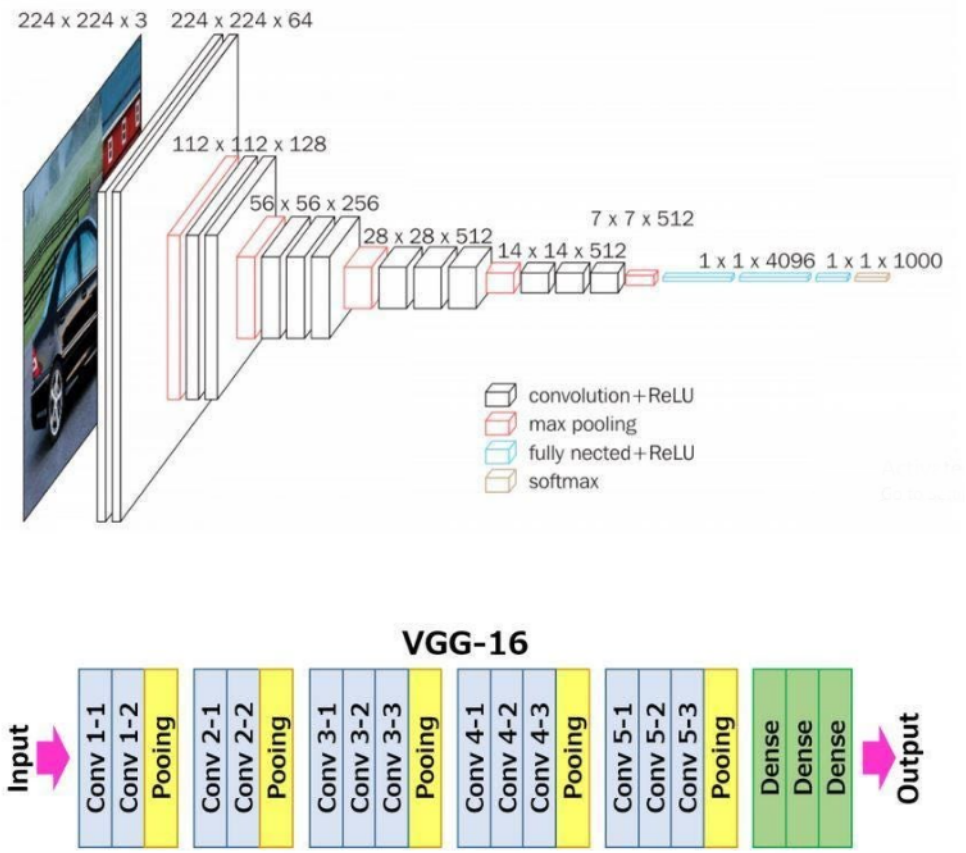
3.4.3 Odabir modela

Prilikom rada na projektu imao sam mogućnost samostalne izrade konvolucijske neuronske mreže ili iskoristiti već istrenirani model. Pred trenirani modeli su modeli dubokog učenja koji su već trenirani na velikom skupu podataka. Treniranje modela dubokog učenja zahtijeva velike količine podataka i računalne resurse te dugotrajno vrijeme. Pred trenirani modeli su već prošli kroz ovaj proces treniranja i naučili su prepoznavati obrasce u podacima. Često su trenirani na velikim skupovima podataka kao što su slike ili tekstualni podaci, te imaju sposobnost izvlačenja značajki iz tih podataka. Ovi modeli se zatim mogu iskoristiti za razne zadatke kao što su klasifikacija slika, prepoznavanje objekata, generiranje teksta ili prijevod. Prednost pred reniranih modela je ušteda vremena i resursa jer se ne moraju trenirati od početka. Umjesto toga, mogu se koristiti kao osnova za daljnje prilagođavanje ili fino podešavanje na manjem skupu podataka specifičnom za određeni zadatak. Lako ih je uključiti u svoj projekt pomoću knjižnica.

Za moje potrebe vizualne kasifikacije slika u istraživanju sam nailazio na par čestih modela od kojih sam se odlučio za dva. Većina modela koje sam trenirao koristila je *VGG16* model kao podlogu, a neki su koristili *ResNet50* kako bi mogao usporediti učinkovitost ta dva pred trenirana modela na mojem specifičnom zadatku i mojoj bazi podataka.

3.4.4 VGG16

VGG16[19][20] je vrsta konvolucijske neuronske mreže koja se smatra jednim od najboljih modela za računalni vid do danas. Kreatori ovog modela evaluirali su mreže i povećali dubinu korištenjem arhitekture s vrlo malim (3×3) konvolucijskim filtrima, što je pokazalo značajno poboljšanje u odnosu na prethodne konfiguracije. Oni su povećali dubinu na 16-19 slojeva težina, što rezultira približno 138 trenirajućih parametara. Koristi se za detekciju objekata i klasifikaciju podataka s algoritmom sposobnim da klasificira 1000 slika od 1000 rzličitih kategorija sa točnosti od 92.7%.



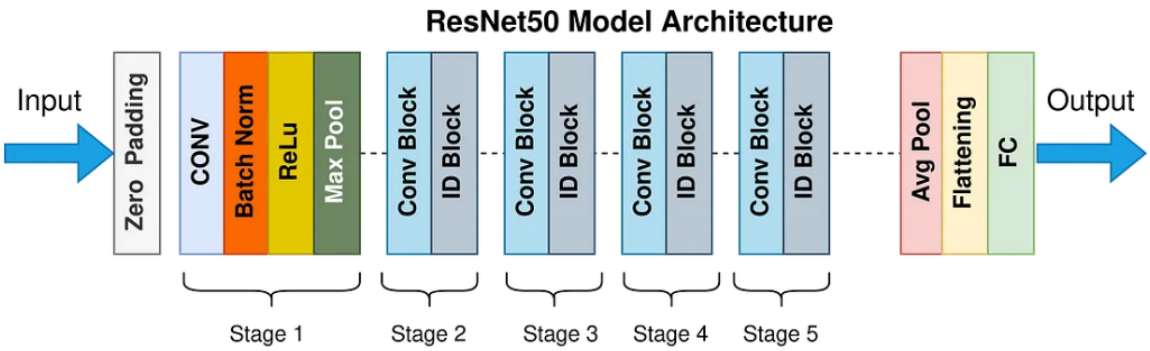
Slika 3.11 VGG16 arhitektura, [20]

Broj '16' u VGG16 referencira 16 slojeva koji imaju težine unutar mreže. U VGG16 modelu postoje trinaest konvolucijskih slojeva, pet slojeva maksimalnog grupiranja (engl. *Max Pooling*) i tri gusto povezana sloja (engl. *Dense*), što ukupno čini 21 sloj, ali ima samo šesnaest slojeva težina, odnosno slojeva s trenirajućim parametrima. Conv-1 sloj ima 64 filtra, Conv-2 ima 128 filtra, Conv-3 ima 256 filtra, Conv-4 i Conv-5 imaju 512 filtra kao što možemo vidjeti na slici 3.11.

Prilikom inicijalizacije modela bilo je potrebno modificirati prvi ulazni sloj jer je namješten da prima slike sa 3 kanala odnosno *RGB*. U slučaju crno-bijele slike koja ima jedan kanal linijim '`model.features[0] = nn.Conv2d(1, 64, kernel-size=3, stride=1, padding=1)`' mijenjamo atribut '*in- channels*' sa 3 na 1.

3.4.5 ResNet50

ResNet50[21] je popularna arhitektura konvolucijske neuronske mreže koja je riješila nekoliko problema s kojima su se suočavale moderne mreže. Jedan od ključnih koncepta u arhitekturi ResNet50 su "skip connections" (preskočene veze), koje su dodane kako bi se riješio problem nestajanja/eksplozije gradijenta prilikom propagacije unatrag. Skip connections omogućuju gradijentima da "prelijeću" slojeve bez značajnih promjena, čime se olakšava konvergencija modela. Arhitektura ResNet50, koju možemo vidjeti na slici 3.12, se sastoji od nekoliko dijelova, uključujući preprocesiranje ulaza, različite konfiguracije blokova slojeva (Cfg blocks) na različitim razinama, te potpuno povezan sloj. ResNet50 koristi 50 slojeva (16 Cfg blocks) za postizanje visoke preciznosti u prepoznavanju slika.



Slika 3.12 ResNet50 arhitektura, [21]

3.4.6 Overfitting

Pretreniranje modela (engl. *overfitting*)[22] je problem u strojnom učenju koji predstavlja model koji je dobro naučio podatke iz skupa za treniranje, ali slabo generalizira na novim, dosad neviđenim podacima iz testnog skupa. Kada model previše uči podatke za treniranje, on može "pametno" zapamtiti specifične karakteristike, šum ili nedostatke u tim podacima umjesto da nauči općenite zakonitosti koje se mogu primijeniti na nove primjere. To može rezultirati visokom točnošću na skupu za treniranje, ali niska točnost na skupu za testiranje ili stvarnim podacima.

Uzroci pretreniranja modela mogu biti:

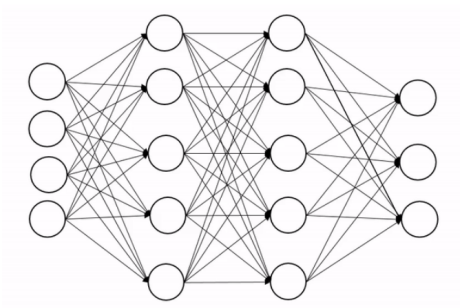
- **Prekomjerna složenost modela:** Ako model ima previše parametara u odnosu na količinu dostupnih podataka, postoji veća vjerojatnost da će model naučiti "pametno" zapamtiti podatke za treniranje umjesto da generalizira na nove primjere
- **Nedovoljno podataka:** Kada je dostupan mali skup podataka za treniranje, model može imati poteškoća u pronalaženju općenitih uzoraka i tendenciju da se nauči isključivo uz dostupne primjere

Posljedice pretreniranja modela mogu biti:

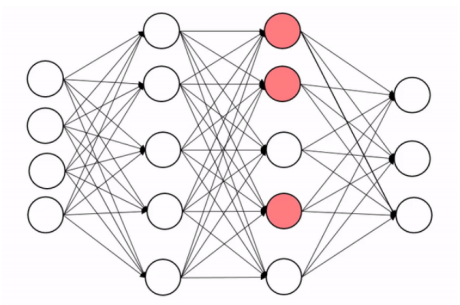
- **Loša generalizacija:** Model će pokazati lošu performansu na novim podacima koji nisu prisutni u skupu za treniranje
- **Preosjetljivost na šum:** Model će možda "pametno" naučiti šum u skupu za treniranje, što će rezultirati nepotrebnim prilagodbama i niskom točnošću na novim podacima

Overfitting se može riješiti ili smanjiti na više načina. Moguće je naći više podataka ili pojednostaviti model što nisu uvijek dostupne opcije. Postoje i tehnike regularizacije koje mogu ograničiti složenost modela i smanjiti rizik od pretreniranja. Regularizacija koju sam ja iskoristio u svom radu i koja mi je uvelike poboljšala rezultate treniranja i testiranja modela je '**dropout**'.

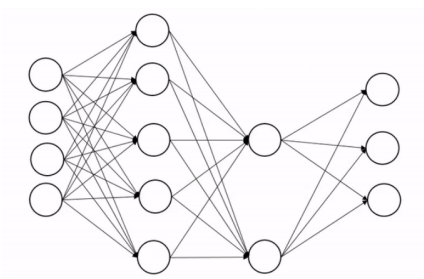
Dropout[23] je tehnika regularizacije koja se koristi u strojnom učenju, posebno u dubokom učenju, kako bi se spriječilo pretreniranje modela. Uključuje nasumično zanemarivanje određenih čvorova (neurona) u sloju tijekom treninga. Ideja iza dropouta je osigurati da se jedinice u sloju ne postanu previše ovisne jedne o drugima, čime se poboljšava sposobnost generalizacije modela. Dropout preventira overfitting unošenjem šuma i slučajnosti u proces treninga. Tijekom treninga, neki izlazi slojeva se ignoriraju, što čini da sloj izgleda kao da ima različiti broj čvorova i veza prema prethodnom sloju. Time se stvara robusniji model sprječavajući suadaptaciju slojeva mreže kako bi ispravili pogreške prethodnih slojeva. Vizualni prikaz zanemarivanja čvorova možemo vidjeti na slici 3.13.



(a)



(b)



(c)

Slika 3.13 Prikaz eliminiranja čvorova unutar sloja mreže

3.4.7 Epohe

Kada smo uključili potrebne knjižnice, učitali podatke i definirali pred trenirani model koji ćemo koristiti možemo započeti treniranje. Treniranje se odvija unutar epoha. Epoha je zapravo petlja unutar koje se iteracijom po učitanim podacima i pripadajućim alogritmima i metodama izvršava treniranje modela. Broj epoha je proizvoljan broj koji se definira prije pokretanja treniranja. Cilj je da broj epoha

bude dovoljan da točnost treniranja i testiranja dosegne konvergenciju. Dovoljan broj epoha da se postigne konvergencija točnosti varira, može biti 10, a može biti i 100. Do idealnog broja dolazi se isprobavanjem dok nismo zadovoljni rezultatom. Moj idealan broj epoha se ispostavio broj 30 s kojim sam u skoro svim slučajevima dolazio do konvergencije.

3.4.8 Kod treniranja

Prikaz djela koda koji trenira i testira model:

```
# Training
num_epochs = 30
best_accuracy = 0.0
best_f1_score = 0.0
best_recall_score = 0.0

for epoch in range(num_epochs):
    # Training phase
    model.train()
    train_loss = 0.0
    train_correct = 0
    train_predictions = []
    train_labels = []

    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
```

```
        optimizer.step()

        train_loss += loss.item() * images.size(0)
        _, predicted = torch.max(outputs.data, 1)
        train_correct += (predicted == labels).sum().item()

        train_predictions.extend(predicted.tolist())
        train_labels.extend(labels.tolist())

    train_loss = train_loss / len(train_loader.dataset)
    train_accuracy = train_correct / len(train_loader.dataset)
    train_f1_score = f1_score(train_labels, train_predictions,
    ↪  average='macro')
    train_recall_score = recall_score(train_labels,
    ↪  train_predictions, average='macro')

    # Testing phase
    model.eval()
    test_correct = 0
    test_predictions = []
    test_labels = []

    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            test_correct += (predicted == labels).sum().item()

            test_predictions.extend(predicted.tolist())
```

```
53              test_labels.extend(labels.tolist())
54
55      test_accuracy = test_correct / len(test_loader.dataset)
56      test_f1_score = f1_score(test_labels, test_predictions,
        ↪  average='macro')
57      test_recall_score = recall_score(test_labels, test_predictions,
        ↪  average='macro')
58
59      print(f'Epoch: {epoch+1}/{num_epochs}, Train Loss:
        ↪  {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, Train
        ↪  F1 Score: {train_f1_score:.4f}, Train Recall:
        ↪  {train_recall_score:.4f}, Test Accuracy: {test_accuracy:.4f},
        ↪  Test F1 Score: {test_f1_score:.4f}, Test Recall:
        ↪  {test_recall_score:.4f}')
60
61
```

Poglavlje 4

Rezultati treniranja

U ovom poglavlju ću predstaviti rezultate treniranja modela i iznijeti komentare, zanimljive usporedbe i zaključke na temelju dobivenih rezultata. Rezultati su bilježeni na kraju svake epohe i prikazani su u obliku postotaka za različite mjere, kao što su gubitak tijekom treniranja, odziv, točnost i F1 mjera, kako za trening tako i za testiranje. Analizirat ću ove rezultate kako bih dobio dublji uvid u performanse modela i njegovu sposobnost generalizacije na nepoznatim podacima. Također ću razmotriti potencijalne probleme ili nedostatke u modelu koji su se pojavili tijekom treniranja i testiranja te predložiti moguća poboljšanja.

Tablica 4.1 11k Hands, Lijevi dlanovi, VGG16

Epoch	**Train Loss**	**Train Accuracy**	**Test Accuracy**
1	5.4051	0.0186	0.0201
2	5.1495	0.0202	0.0201
3	5.1251	0.0217	0.0201
4	5.1227	0.0222	0.0201
5	5.1481	0.0191	0.0201
6	5.1790	0.0212	0.0201
7	5.2501	0.0156	0.0134
8	6.1886	0.0171	0.0117
9	5.1224	0.0181	0.0201
10	5.1158	0.0166	0.0201

Na tablici 4.1 vidimo prvi uspješno obavljeni trening. Možemo primijetiti da su rezultati veoma loši te da treniranje modela nije postiglo željene rezultate. Za ovako loše rezultate postoji više faktora: malen broj epoha, SGD optimizator nije bio uključen, nije obavljeno proširivanje podataka unutar transformacija, nije uključen 'dropout'.

Tablica 4.2 11k Hands, Lijevi dlanovi, ResNet50

Epoch	**Train Loss**	**Train Accuracy**	**Test Accuracy**
1	5.2870	0.0242	0.0201
2	4.6253	0.0368	0.0134
3	4.2480	0.0599	0.0435
4	3.9337	0.0932	0.0468
5	3.5755	0.1350	0.0702
6	3.2276	0.1874	0.0819
7	2.7973	0.2650	0.2291
8	2.3470	0.3582	0.2157
9	1.8396	0.4992	0.3512
10	1.3717	0.6302	0.1003

Tablica 4.3 11k Hands, Lijevi dlanovi, ResNet50

Epoch	**Train Loss**	**Train Accuracy**	**Test Accuracy**
41	0.8946	0.7070	0.7150
42	0.9141	0.7140	0.0794
43	0.9179	0.7149	0.5280
44	0.8631	0.7269	0.6667
45	0.8187	0.7278	0.6277
46	0.8478	0.7269	0.6947
47	0.7767	0.7453	0.0343
48	0.8354	0.7310	0.0296
49	0.7712	0.7476	0.6667
50	0.7423	0.7632	0.6713

Na primjeru 4.2 sam probao istu bazu trenirati pomoću ResNet50 modela. Rezultati su primijetno bolji jer sam uključio proširivanje podataka i 'dropout'. Također sam uključio i 'ADAM' optimizator. Iako su rezultati bolji konvergencija nije dosegnuta stoga sam u treniranju 4.3 povećao broj epoha na 50. Rezultat je bolji, ali konvergencija ponovo nije zadovoljena.

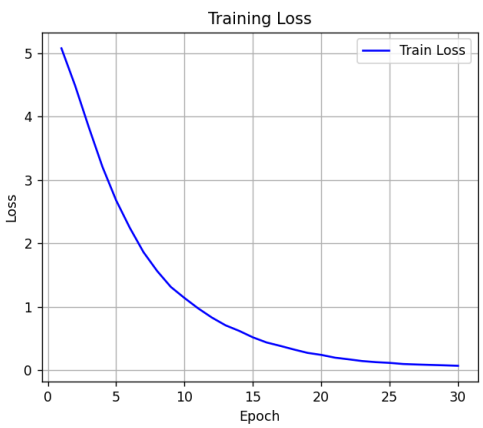
U narednim treninzima sam eksperimentirao s različitim brojevima epoha, veličinama grupa i transformacijama kako bih postigao bolje rezultate. Međutim, značajno poboljšanje sam primijetio kada sam promijenio optimizator iz 'ADAM' u 'SGD'.

Kada sam uspio postići konvergenciju točnosti blizu vrijednosti '1', krenuo sam trenirati sve baze podataka redom koristeći 30 epoha, što se pokazalo kao idealan broj.

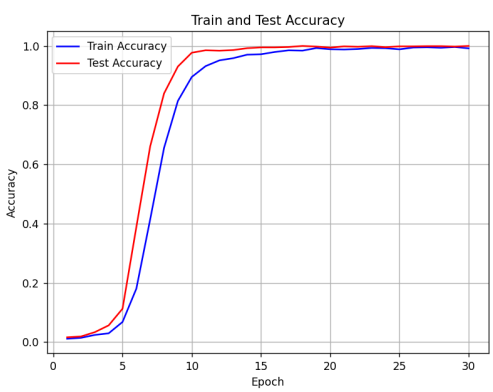
4.1 CASIA - lijevi i desni dlanovi

Tablica 4.4 CASIA, lijevi i desni dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score	Test Recall
1/30	4.6050	0.0111	0.0069	0.0107	0.0160	0.0042	0.0217
4/30	4.4381	0.0294	0.0201	0.0277	0.0562	0.0343	0.0588
7/30	2.2031	0.4147	0.4096	0.4135	0.6613	0.6558	0.6738
10/30	0.3509	0.8951	0.8942	0.8942	0.9771	0.9777	0.9778
13/30	0.1372	0.9578	0.9576	0.9575	0.9880	0.9884	0.9882
16/30	0.0632	0.9796	0.9795	0.9794	0.9951	0.9943	0.9950
19/30	0.0263	0.9928	0.9928	0.9928	0.9979	0.9979	0.9979
22/30	0.0351	0.9896	0.9894	0.9894	0.9972	0.9973	0.9975
25/30	0.0334	0.9886	0.9885	0.9884	0.9986	0.9982	0.9975
28/30	0.0248	0.9938	0.9939	0.9939	0.9993	0.9994	0.9993
28/30	0.0248	0.9938	0.9939	0.9939	0.9993	0.9994	0.9993
30/30	0.0218	0.9919	0.9919	0.9918	1.0000	1.0000	1.0000



Slika 4.1 Gubitak treniranja

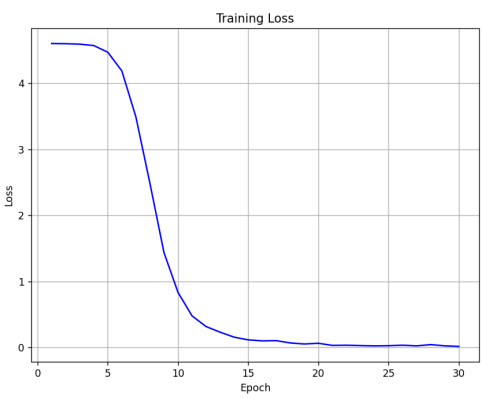


Slika 4.2 Točnost

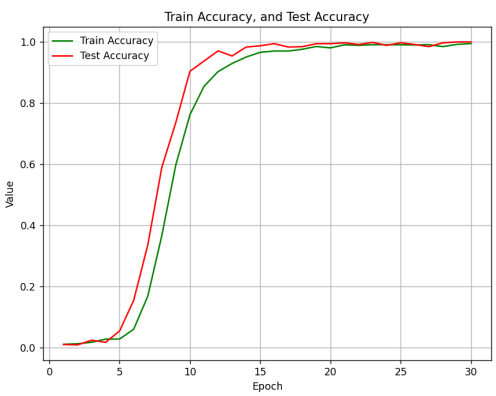
4.2 CASIA - lijevi dlanovi

Tablica 4.5 CASIA, lijevi dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score	Test Recall
1/30	4.6045	0.0117	0.0065	0.0115	0.0111	0.0031	0.0214
4/30	4.5734	0.0286	0.0131	0.0259	0.0181	0.0131	0.0249
7/30	3.4935	0.1690	0.1579	0.1675	0.3361	0.3046	0.3435
10/30	0.8365	0.7621	0.7593	0.7597	0.9042	0.8958	0.9103
13/30	0.2336	0.9297	0.9287	0.9287	0.9542	0.9549	0.9612
16/30	0.1022	0.9703	0.9695	0.9694	0.9944	0.9924	0.9952
19/30	0.0551	0.9852	0.9850	0.9850	0.9944	0.9948	0.9952
22/30	0.0365	0.9890	0.9889	0.9889	0.9917	0.9927	0.9937
25/30	0.0291	0.9907	0.9906	0.9904	0.9972	0.9969	0.9972
28/30	0.0457	0.9848	0.9848	0.9848	0.9972	0.9968	0.9976
30/30	0.0190	0.9948	0.9949	0.9948	1.0000	1.0000	1.0000



Slika 4.3 Gubitak treniranja

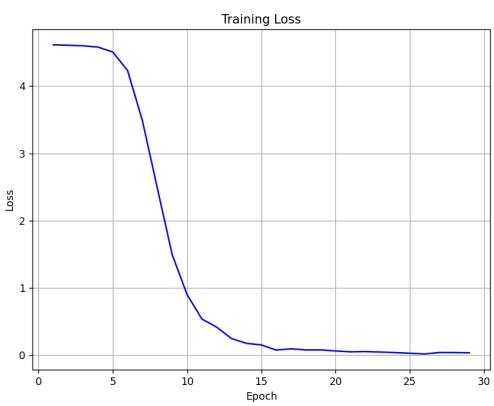


Slika 4.4 Točnost

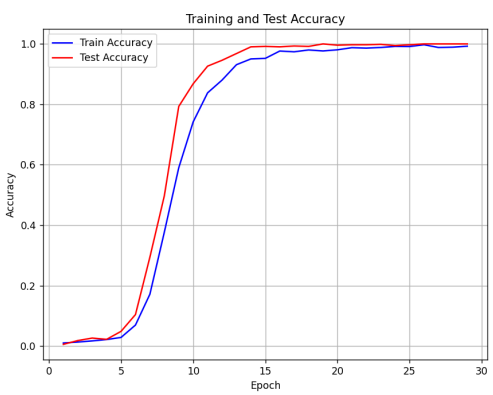
4.3 CASIA - desni dlanovi

Tablica 4.6 CASIA, desni dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score	Test Recall
1/30	4.6158	0.0101	0.0062	0.0097	0.0056	0.0014	0.0125
4/30	4.5826	0.0215	0.0097	0.0198	0.0222	0.0119	0.0399
7/30	3.4838	0.1716	0.1536	0.1675	0.2944	0.2638	0.3124
10/30	0.8984	0.7419	0.7372	0.7387	0.8681	0.8708	0.8812
13/30	0.2409	0.9312	0.9304	0.9303	0.9681	0.9660	0.9721
17/30	0.0702	0.9764	0.9756	0.9755	0.9903	0.9925	0.9927
19/30	0.0719	0.9799	0.9797	0.9795	0.9917	0.9916	0.9919
22/30	0.0436	0.9875	0.9873	0.9875	0.9972	0.9968	0.9968
25/30	0.0327	0.9917	0.9915	0.9914	0.9944	0.9947	0.9946
28/30	0.0337	0.9882	0.9879	0.9879	1.0000	1.0000	1.0000
30/30	0.0294	0.9924	0.9922	0.9923	0.9986	0.9990	0.9990



Slika 4.5 Gubitak treniranja

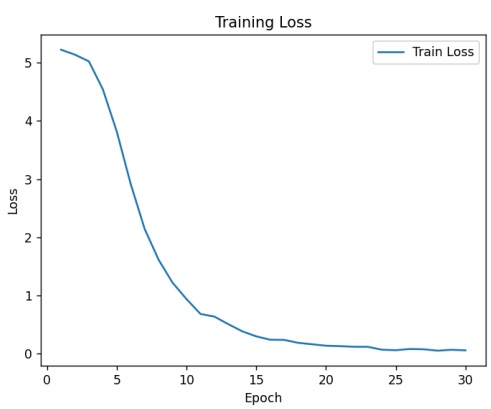


Slika 4.6 Točnost

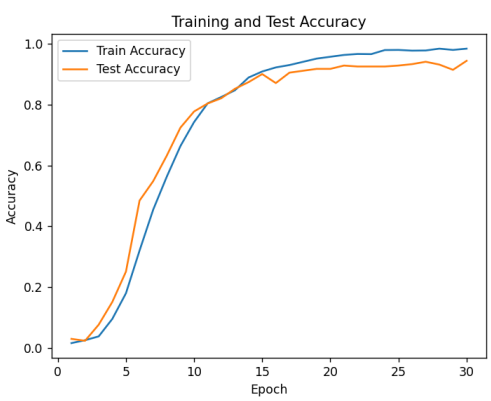
4.4 11k Hands - desni i lijevi dlanovi

Tablica 4.7 11k Hands, lijevi i desni dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score	Test Recall
1/30	5.2234	0.0157	0.0026	0.0064	0.0296	0.0010	0.0073
4/30	4.5429	0.0953	0.0236	0.0387	0.1511	0.0365	0.0771
7/30	2.1409	0.4546	0.3362	0.3451	0.5483	0.4042	0.4461
10/30	0.9365	0.7425	0.6594	0.6577	0.7773	0.6951	0.7219
13/30	0.5051	0.8471	0.7990	0.7961	0.8520	0.8132	0.8258
16/30	0.2385	0.9222	0.9013	0.9000	0.8707	0.8396	0.8456
19/30	0.1605	0.9512	0.9326	0.9352	0.9174	0.8876	0.8910
22/30	0.1177	0.9664	0.9613	0.9578	0.9252	0.9002	0.9070
25/30	0.0584	0.9797	0.9735	0.9736	0.9283	0.9070	0.9138
28/30	0.0507	0.9839	0.9815	0.9817	0.9315	0.9041	0.9102
30/30	0.0559	0.9839	0.9806	0.9798	0.9439	0.9191	0.9244



Slika 4.7 Gubitak treniranja

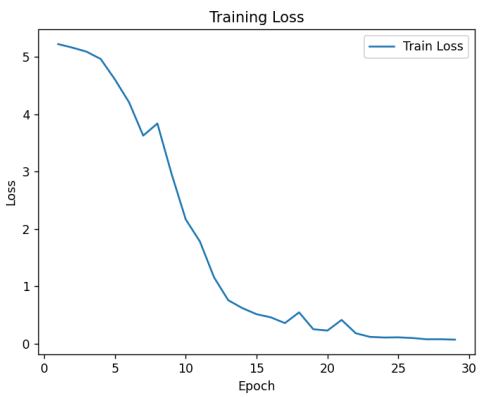


Slika 4.8 Točnost

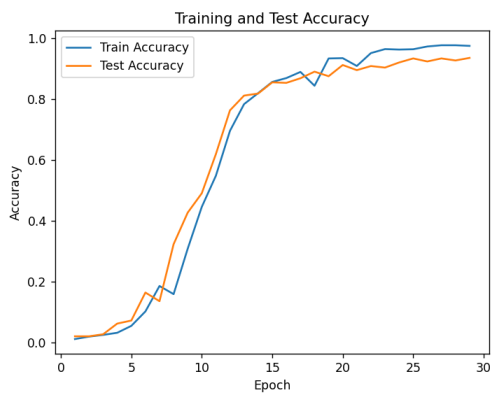
4.5 11k Hands - lijevi dlanovi

Tablica 4.8 11k Hands, lijevi dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score
1/30	5.2199	0.0111	0.0034	0.0201	0.0002	0.0053
4/30	4.9604	0.0317	0.0020	0.0619	0.0041	0.0242
7/30	3.6263	0.1854	0.1059	0.1355	0.0821	0.1139
10/30	2.1675	0.4453	0.3314	0.4900	0.3702	0.4014
13/30	0.7597	0.7829	0.7075	0.8110	0.7290	0.7587
16/30	0.4619	0.8685	0.8277	0.8528	0.8200	0.8225
19/30	0.2545	0.9330	0.9022	0.8746	0.8439	0.8508
22/30	0.1843	0.9506	0.9323	0.9080	0.8695	0.8853
25/30	0.1099	0.9622	0.9536	0.9197	0.8961	0.9049
28/30	0.0793	0.9763	0.9712	0.9331	0.9154	0.9197
30/30	0.0738	0.9743	0.9674	0.9348	0.9074	0.9153



Slika 4.9 Gubitak treniranja

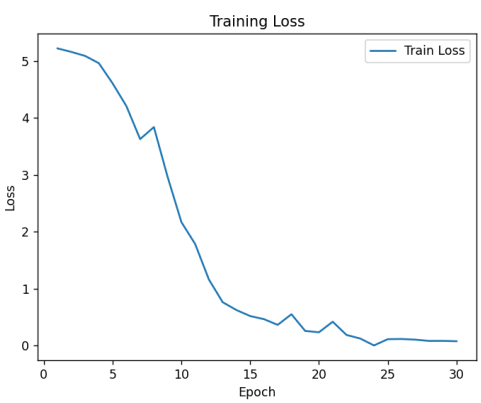


Slika 4.10 Točnost

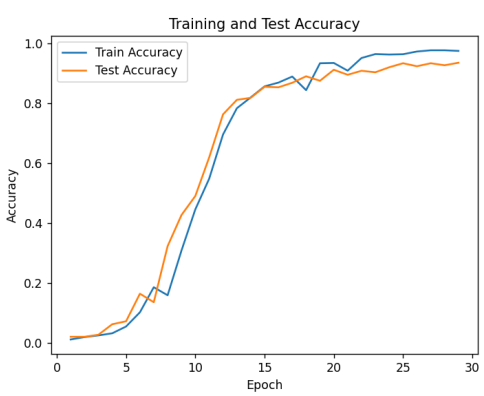
4.6 11k Hands - desni dlanovi

Tablica 4.9 11k Hands, desni dlanovi, VGG16

Epoch	Train Loss	Train Accuracy	Train F1 Score	Train Recall	Test Accuracy	Test F1 Score	Test Recall
1/30	5.2202	0.0157	0.0047	0.0064	0.0234	0.0002	0.0053
4/30	4.3097	0.1202	0.0362	0.0532	0.2259	0.0802	0.1272
7/30	1.6802	0.5578	0.4486	0.4504	0.5374	0.3862	0.4208
10/30	0.6746	0.8065	0.7485	0.7453	0.8520	0.7875	0.7979
13/30	0.2884	0.9111	0.8880	0.8864	0.8879	0.8425	0.8546
16/30	0.2330	0.9286	0.9170	0.9170	0.8972	0.8597	0.8703
19/30	0.1049	0.9650	0.9537	0.9518	0.9050	0.8647	0.8760
22/30	0.0813	0.9802	0.9748	0.9745	0.9143	0.8893	0.8935
25/30	0.0445	0.9876	0.9854	0.9842	0.9237	0.8883	0.9031
28/30	0.0723	0.9756	0.9659	0.9639	0.9190	0.8970	0.9037
30/30	0.0361	0.9936	0.9910	0.9914	0.9315	0.9007	0.9133



Slika 4.11 Gubitak treniranja



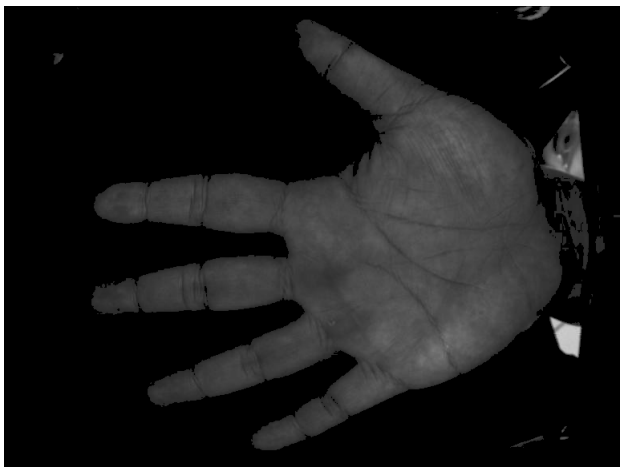
Slika 4.12 Točnost

4.7 Maskiranje

Unatoč iznimno dobrim rezultatima tijekom treninga, odlučio sam modificirati jedan skup podataka kako bih testirao utjecaj prepoznavanja dlana pomoću "OpenCV-a" i zacrnjivanja ostalih dijelova slike. Cilj mi je bio provjeriti hoće li model postići bolje rezultate kada se fokusira samo na značajke dlana, bez da ga ometaju karakteristike pozadine.

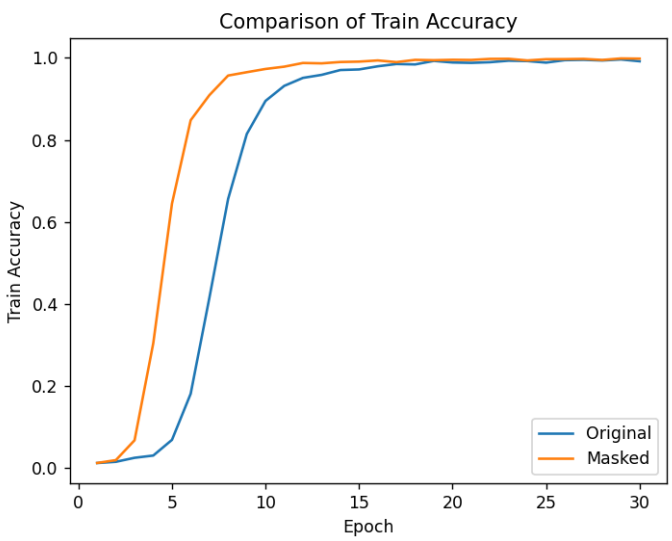
Slika 4.13 Original slika

Slika 4.14 Maska dlana

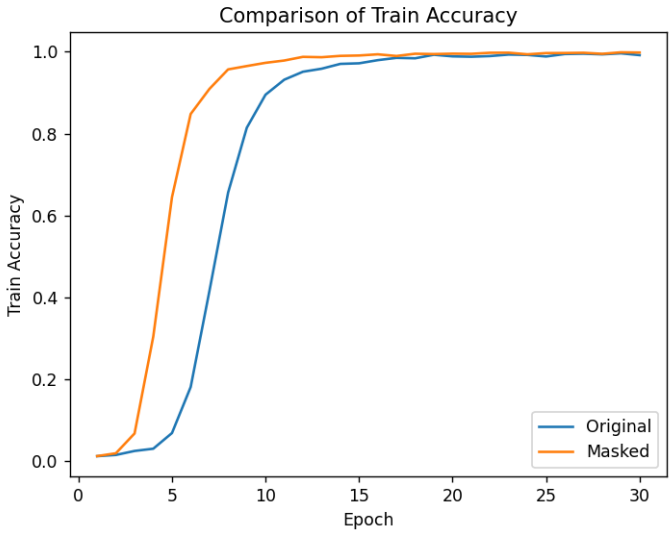


Slika 4.15 Kombinacija originalne slike i maske

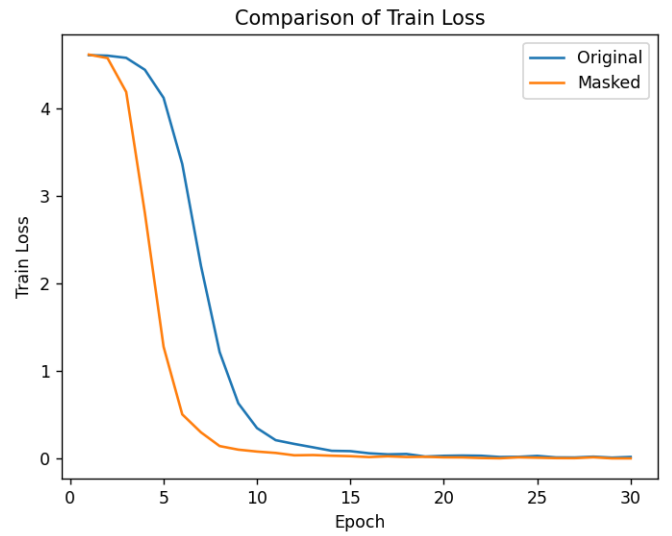
4.7.1 Usporedba



Slika 4.16 Usporedba točnosti treniranja



Slika 4.17 Usporedba točnosti testiranja



Slika 4.18 Usporedba gubitka treniranja

Kao što primjećujemo na slikama 4.16, 4.17 i 4.18 maskiranje dlana kako bi zanemarili pozadinu ima pozitivan učinak na treniranje modela. Model je u procesu treniranja brže počeo imati veću točnost treniranja i testiranja te je gubitak treniranja počeo puno brže opadati.

Poglavlje 5

Zaključak

U ovom istraživanju smo proučavali utjecaj algoritma dubokog učenja na performanse klasifikacije slika, specifičnije dlanova. Izvršili smo obuku konvolucijske neuronske mreže na skupu podataka koji se sastoji od slika različitih kategorija. Analizirali smo performanse modela tijekom epoha kako bismo pratili njegov napredak. Rezultati istraživanja pokazuju da je duboko učenje moćan alat za klasifikaciju slika. Tijekom obuke, model je postizao sve bolje rezultate kako su se epohe povećavale. Primijetili smo značajan napredak u točnosti, F1 mjeri i odzivu na trening i test skupovima podataka. To ukazuje na to da je model uspješno naučio značajke slika i generalizirao svoje rezultate na nepoznate primjere. Nadalje, primjećuje se da se performanse modela postepeno poboljšavaju tijekom prvih nekoliko epoha, a zatim se stabiliziraju. To ukazuje na to da je potrebno određeno vrijeme za model da nauči reprezentacije slika i da dosegnu svoj maksimalni potencijal. Također je zamjetno da maskiranjem slika u bazi podataka smanjujemo prostor promatranja samo na bitno područje i time dobivamo brži napredak modela.

U konačnici, rezultati istraživanja sugeriraju da je duboko učenje obećavajuća metoda za rješavanje problema klasifikacije slika. Daljnji rad može uključivati istraživanje drugih arhitektura neuronskih mreža, optimizaciju hiperparametara i primjenu različitih tehnika za povećanje performansi modela. Ovaj rad pruža osnovu za daljnja istraživanja u području dubokog učenja i klasifikacije slika te pruža korisne uvide u primjenu ovih tehnika na različite probleme u stvarnom svijetu.

Bibliografija

[1] MathWorks. (2023) What is deep learning? , s Interneta, https://www.mathworks.com/discovery/deep-learning.html

[2] I. Židov, "Uvod u neuronske mreže," 2018. , s Interneta, http://www.mathos.unios.hr/~mdjumic/uploads/diplomski/%C5%BDID03.pdf

[3] IBM. (N/A) What are convolutional neural networks? , s Interneta, https://www.ibm.com/topics/convolutional-neural-networks

[4] S. Balaji. (Aug 29, 2020) Binary image classifier cnn using tensorflow. , s Interneta, https://medium.com/techiepedia/binary-image-classifier-cnn-using-tensorflow-a3f5d6746697

[5] M. Afifi, "11k hands: gender recognition and biometric identification using a large dataset of hand images," *Multimedia Tools and Applications*, 2019. , s Interneta, https://doi.org/10.1007/s11042-019-7424-8

[6] H. Ying, Z. Sun, T. Tan, and C. Ren, "Multi-spectral palm image fusion for accurate contact-free palmprint recognition," in *Proceedings of IEEE International Conference on Image Processing*. USA: IEEE, 2008, pp. 281–284.

[7] W. Foundation. (2023) Charge-coupled device. , s Interneta, https://en.wikipedia.org/wiki/Charge-coupled_device#References

[8] J. Brownlee. (July 14, 2017) What is the difference between test and validation datasets? , s Interneta, https://machinelearningmastery.com/difference-test-validation-datasets/

[9] T. Shah. (Dec 03, 2017) Train, validation and test sets. , s Interneta, http://tarangshah.com/blog/2017-12-03/train-validation-and-test-sets/

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] S. Mahre. (April 2017) Nvidia t4 70w low profile pcie gpu accelerator. , s Interneta, https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-product-brief.pdf

[12] F. Oh. (September 10, 2012) What is cuda? , s Interneta, https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/

[13] PyTorch. (2023) Pytorch resources. , s Interneta, https://pytorch.org/features/

[14] W. Foundation. (2023) Tensor (machine learning). , s Interneta, https://en.wikipedia.org/wiki/Tensor_(machine_learning)

[15] T. Vo. (March 31, 2023) Pytorch dataloader: Features, benefits, and how to use it. , s Interneta, https://saturncloud.io/blog/pytorch-dataloader-features-benefits-and-how-to-use-it/

[16] J. Brownlee. (August 10, 2022) Difference between a batch and an epoch in a neural network. , s Interneta, https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

[17] A. Wagh. (July 26, 2022) Gradient descent and its types. , s Interneta, https://www.analyticsvidhya.com/blog/2022/07/gradient-descent-and-its-types/

[18] Musstafa. (Mar 27, 2021) Optimizers in deep learning. , s Interneta, https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0

[19] K. S. . A. Zisserman. (Apr 10, 2015) Very deep convolutional networks for large-scale image recognition. , s Interneta, https://arxiv.org/pdf/1409.1556.pdf

[20] R. G. (Sep 23, 2021) Everything you need to know about vgg16. , s Interneta, https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[21] S. Mukherjee. (Aug 18, 2022) The annotated resnet-50. , s Interneta, https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758

[22] IBM. (N/A) What is overfitting? , s Interneta, https://www.ibm.com/topics/overfitting

[23] P. Marimuthu. (August 11, 2022) Dropout regularization in deep learning. , s Interneta, https://www.analyticsvidhya.com/blog/2022/08/dropout-regularization-in-deep-learning/

Sažetak

U ovom završnom radu opisuje se koncept dubokog učenja i demonstrira implementacija treniranja konvolucijskih neuronskih mreža za klasifikaciju slika. Specifični cilj projekta bio je istražiti sposobnost računalnog sustava u prepoznavanju osoba na temelju karakteristika njihovih dlanova. Istraživanje je omogućilo bolje razumijevanje rada konvolucijskih neuronskih mreža i njihove primjene u području biometrije. Osim toga, istraživanje je pridonijelo razvoju praktičnih vještina u treniranju i evaluaciji modela dubokog učenja.

Ključne riječi — **duboko učenje, klasifikacija slika, konvolucijske neuronske mreže**

Abstract

This final paper describes the concept of deep learning and demonstrates the implementation of training convolutional neural networks for image classification. The specific objective of the project was to explore the ability of a computer system to recognize individuals based on the characteristics of their palms. The research has facilitated a better understanding of the workings of convolutional neural networks and their applications in the field of biometrics. Furthermore, the study has contributed to the development of practical skills in training and evaluating deep learning models.

Keywords — **deep learning, image classification, convolutional neural network**