SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 532

SUSTAV ZA DETEKCIJU I PRAĆENJE DLANA

Lovro Glogar

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 532

SUSTAV ZA DETEKCIJU I PRAĆENJE DLANA

Lovro Glogar

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 532

Pristupnik:	**Lovro Glogar (0036521732)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Zoran Kalafatić

Zadatak: **Sustav za detekciju i praćenje dlana**

Opis zadatka:

Detekcija i praćenje dlana u videu ima zanimljive primjene u izgradnji naprednih sučelja za interakciju između čovjeka i računala. U okviru završnog rada treba proučiti pristupe detekciji i praćenju objekata u slikama. Odabrati prikladan pristup te programski ostvariti sustav za detekciju i praćenje dlana. Pripremiti skup slika za učenje i testiranje oblikovanog sustava. Analizirati dobivene rezultate u pogledu točnosti detekcije te računske zahtjevnosti. Radu priložiti izvorni i izvršni kôd razvijenih postupaka, ispitne slike i rezultate, uz potrebna objašnjenja i dokumentaciju.

Rok za predaju rada: 10. lipnja 2022.

Zahvaljujem mentoru prof.dr.sc. Zoranu Kalafatiću, prijateljima i obitelji.

Sadržaj

Uvod

Računalni vid se prvi puta pojavljuje 60-ih godina dvadesetog stoljeća u početkom razvoja umjetne inteligencije. Cilj područja je razvijanje teorijskih i algoritamskih temelja primjenom kojih se postiže razumijevanje scene, tj. izlučivanje informacije o svijetu na temelju slike ili slijeda slika. Na početku, ideja računalnog vida je bila oblikovanje 3D svijeta na temelju slika. Razvijene su brojne tehnike kao što su ekstrakcija rubova, detekcija pokreta, optičko prepoznavanje znakova (engl. Optical Character Recognition (OCR)) te statističke metode od kojih je značajnija metoda detekcije pomoću svojstvenih vrijednosti i vektora.

Dolaskom interneta pojavljuje se mogućnost prikupljanja velikog broja podataka koji tada mogu biti korišteni u strojnom učenju. Danas, računalni vid je usko povezan za poljem strojnog učenja, posebno dubokog učenja.

Cilj ovoga rada je razviti sustav za detekciju i praćenje dlana čovjeka. U tu svrhu koristimo se poznatim programskim sučeljem Tensorflow Object Detection te važnim konceptom prijenosa znanja (engl. transfer learning).

Najprije su opisani osnovni koncepti strojnog učenja pomoću primjera linearne regresije. Zatim je predstavljen temeljni koncept dubokog učenja, umjetne neuronske mreže. Opisan je koncept perceptrona i njegova uloga u nastanku umjetnih neuronskih mreža. Ukratko je opisan algoritam unazadne propagacije. Predstavljene su konvolucijske neuronske mreže i njihovo značenje u računalnom vidu. Predstavljano je područje dubokog učenja i neki od osnovnih dubokih arhitektura korištenih u sklopu ovoga rada te je definiran prijenos znanja. Dalje je opisana implementacija sustava počevši s kratkim predstavljanjem korištenih tehnologija. Opisan je postupak sakupljanja skupa podataka, njegovo označavanje te njegova priprema za treniranje. Opisan je cjeloviti postupak pripreme za treniranje te samo treniranje. Na kraju iskazani su rezultati za dva razvijena modela te je dan zaključak.

1. Strojno učenje

Strojno učenje je najpopularnije područje umjetne inteligencije, ali i jedno od najpopularnijih područja u računarstvu današnjice. Nastanak interneta i početak informacijskog doba omogućuje nagli rast strojnog učenja početkom 21. stoljeća, posebice dubokog učenja.

Ovo poglavlje predstavlja osnovne koncepte strojnog učenja.

1.1. Osnove i motivacija

Strojno učenje je programiranje računala tako da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera ili prethodnog iskustva. Raspolažemo modelom koji je definiran do na neke parametre, a učenje se svodi na izvođenje algoritma koji optimizira parametre modela na temelju podataka ili prethodnog iskustva [1]. Dakle, cilj strojnog učenja je sastaviti matematički model koji na temelju ulaznih podataka i nekih svojih parametara može precizno izvesti neki traženi rezultat. Ono što takav model zapravo radi je generalizacija na temelju podataka nad kojima je učio.

Mnogo je razloga zašto bi nekim problemima odlučili pristupiti strojnim učenjem. Jedan od tih razloga je sama količina dostupnih podataka. Pretpostavimo da imamo ogromnu količinu strukturiranih podataka iz kojih je potrebno izvući neke zaključke ili neko novo znanje. Tradicionalan pristup problemu bi brzo doveo do slijepe ulice. Međutim primjenom strojnog učenja relativno bezbolno možemo izvući puno znanja.

Sama složenost algoritma je također jedan od razloga primjene strojnog učenja. Naime, neki problemi su presloženi da bi ih riješili algoritamski ili ljudi uopće nemaju ideju kako bi ih riješili na tradicionalan način. Na primjer, problem raspoznavanja objekata u slikama. Ljudima je to trivijalan zadatak, međutim za računalo to je izuzetno težak problem. Ovakvim problemima se bavi područje računalnog vida.

Glavni zadatci strojnog učenja su klasifikacija i regresija. Klasifikacija se koristi za predviđanje diskretnih vrijednosti, na primjer, određivanje je li se na slici nalazi dlan ili ne. Regresija se koristi za predviđanje kontinuiranih vrijednosti. Na primjer, određivanje vjerojatnosti nekog događaja.

1.2. Treniranje i predikcija

Strojno učenje na najvišoj razini možemo podijeliti na nadzirano, nenadzirano i podržano strojno učenje. Kod nadziranog učenja naši podatci su strukturirani u obliku (ulaz, izlaz), tj. svakom ulazu pridijeljena je pripadna oznaka. Dakle, model uči na temelju ulaza i zna što bi trebao biti izlaz za dotični ulaz. Nenadzirano učenje znači da podatci nisu označeni odnosno labelirani. Podržano učenje predstavlja učenje optimalne strategije na temelju pokušaja s odgođenom nagradom. U ovom radu bavimo se nadziranim učenjem. Neke od najpoznatijih metoda nadziranog učenja su linearna ili nelinearna regresija, logistička regresija i skup potpornih vektora. Definirajmo dalje neke bitne pojmove.

Instanca podataka se naziva primjer i strukturirana je nizom značajki $x_1, x_2, \dots, x_n$. Označenom primjeru pridružena je još oznaka y. Niz značajki prikazujemo vektorom $\boldsymbol{x}$. Model definira odnos između vektora značajki $\boldsymbol{x}$ i oznake y. Treniranje modela je proces učenja nad danim označenim primjerima. Predikcija je primjena treniranog modela na neoznačene primjere kako bi se dobila predikcija oznake y, y'.

Proučimo i detaljnije opišimo treniranje modela strojnog učenja na primjeru linearne regresije. Linearna regresija je proces modeliranja linearnog odnosa između *n* ulaznih podataka i jednog izlaza y'.

$$y' = \boldsymbol{w}\boldsymbol{x} + b \tag{1}$$

Vrijednost y' predstavlja predviđenu oznaku, vektor $\boldsymbol{w}$ predstavlja težine značajki ulaznog vektora $\boldsymbol{x}$ i vrijednost b označava pristranost (engl. bias) modela. Treniranje ovakvog modela se svodi na traženje optimalnih vrijednosti težina modela (1). Taj postupak počinje određivanjem funkcije gubitka. Funkcija gubitka je pogreška koju model čini na svakom primjeru x uz točnu oznaku y. Kod regresije najčešće se koristi funkcija kvadratnog gubitka.

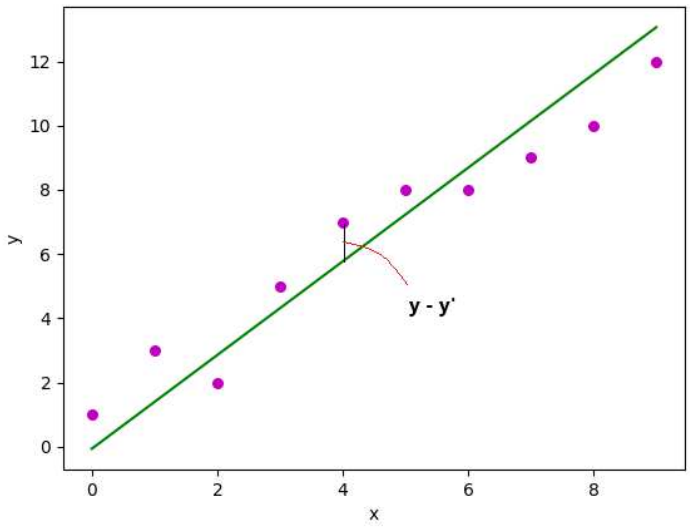
$$L(y, y') = (y - y')^2 \tag{2}$$

Gdje je y prava vrijednost oznake, a y' predviđena vrijednost oznake. Dalje je potrebno odrediti funkcije pogreške. Funkcija pogreške predstavlja očekivanu vrijednost gubitka modela. Regresija najčešće koristi funkciju srednjeg kvadratnog odstupanja (engl. Mean Square Error – MSE).

$$MSE = \frac{1}{N} \sum_{(x,y)\epsilon D} (y - y')^2 \tag{3}$$

Gdje je D skup primjera za treniranje, N broj primjera u skupu za treniranje D. Na kraju trebamo postupak koji će nam dati optimalne težine $\boldsymbol{w}$ takve da minimiziramo izraz (3).

Dakle, trebamo naći minimum izraza (3). Odnosno, želimo minimizirati pogrešku modela. Sada relativno jednostavnom analizom dolazimo do optimalnih vrijednosti težine $\boldsymbol{w}$ te nalazimo optimalni model (1). Slika 1.1 prikazuje primjer rezultata ovakvog postupka pri čemu imamo samo jednu značajku x i 10 označenih primjera. Napominjem da $\frac{1}{N}$ u izrazu (3) inače zamjenjujemo sa $\frac{1}{2}$ zbog jednostavnosti pri izračunu težina.



Sl. 1.1 Linearna regresija

1.3. Prenaučenost i unakrsna provjera

Glavni problem strojnog učenja je prenaučenost. Želimo da naš model jako dobro generalizira, tj. da daje vrlo dobre predikcije na neviđenim podatcima. Međutim, ako model daje gotovo savršene predikcije na cijelom skupu podataka onda se vrlo vjerojatno prilagodio na specifičnosti tog skupa i vrlo loše generalizira.

Kako bi izbjegli problem prenaučenosti, koristi se metoda unakrsne provjere. Unakrsna provjera znači treniranje tako da skup podataka podijelimo na skup za treniranje i skup za testiranje. Skup za treniranje služi za učenje, a skup za testiranje služi za računanje pogreške. Drugi, slični, pristup je dijeljenje skupa podataka na skup za treniranje, validaciju i testiranje. U ovom slučaju, model uči na skupu za treniranje, računa pogreške na skupu za validaciju i na kraju učenja model se testira na skupu za testiranje.

Sl. 1.2 Skup podataka podijeljen na skup za treniranje, validaciju i testiranje

Prenaučenost se ublažava i regularizacijom. Regularizacija je termin koji se koristi za kažnjavanje složenosti modela. Kao što je pokazano na primjeru linearne regresije, cilj modela je minimizirati funkciju pogreške. Uvođenjem regularizacije nastoji se minimizirati kombinacija pogreške i funkcije složenosti. Jedan poznati regularizacijski izraz je takozvana L_2 regularizacija koja ograničava magnitudu težina značajki $\boldsymbol{w}$.

$$L_2 = \sum_{i=1}^{n} w_i^2 \tag{4}$$

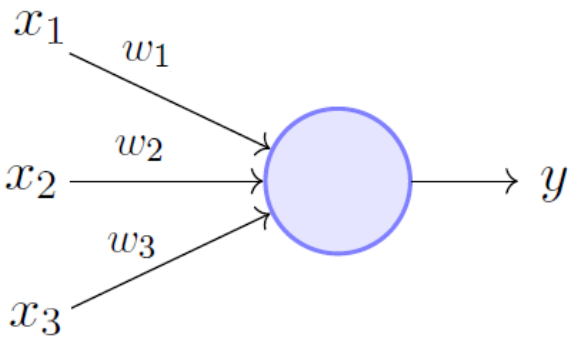
2. Umjetne neuronske mreže

Umjetne neuronske mreže (engl. Aritificial Neural Networks - ANN) ime su dobile jer su u početku zamišljene kako bi znanstvenici, posebice neuroznanstvenici, dobili bolji uvid u funkcionalnost ljudskog mozga [1]. Ljudski mozak se sastoji od oko 10 milijardi neurona (10^9) i oko 1 bilijarde veza (10^{15}). Može se reći da je mozak vrlo kompleksno, nelinearno, paralelno računalo. Ova struktura se pokušava modelirati pomoću ANN-a. ANN su u samom srcu dubokog učenja. One su prilagodljive, snažne i skalabilne, što ih čini idealnim alatom za rješavanje nekih od najkompleksnijih problema strojnog učenja, kao što su raspoznavanje govora i raspoznavanje objekata [2].

U ovome poglavlju predstavljena je jednostavna arhitektura neuronskih mreža. Zatim je ukratko predstavljen algoritam po kojemu neuronske mreže uče, propagacija unatrag. Te su opisane konvolucijske neuronske mreže.

2.1. Arhitektura

Umjetni neuron je vrlo jednostavna strukturu koju prvi predlažu neurofiziolog Warren McCulloch i matematičar Walter Pitts 1943. godine. Taj model umjetnog neurona radio je na principu binarne logike. Jednu od najjednostavnijih arhitektura ANN-a predstavio je Frank Rosenblatt 1957. godine pod nazivom perceptron.



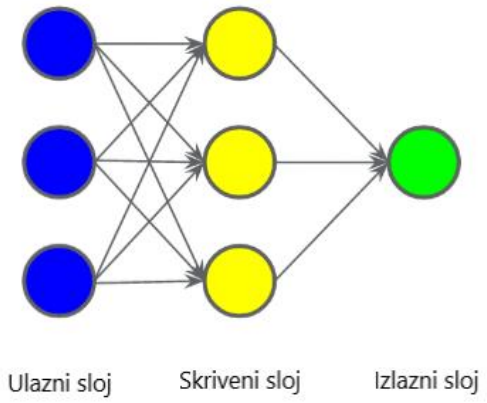
Sl. 2.1 Perceptron

Perceptron je vrlo jednostavna struktura. Neuron predstavlja matematičku funkciju. Ulaz u neuron su varijable $x_1, x_2, \dots, x_n$, svakoj varijabli x_i je dodijeljena težina w_i, izlaz neurona je definiran težinskom sumom ulaza i prijenosnom funkcijom. Prijenosna funkcija je oblika:

$$y = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i + w_0 > 0 \\ 0, & inače \end{cases} \quad (5)$$

To je jednostavna stepenasta funkcija koja poprima vrijednost 0 kada je težinska suma manja ili jednaka nuli i 1 inače. Slobodna težina w_o označava pristranost i dodaje se kako bi model bolje generalizirao.

Slaganje ovakvih perceptrona u slojeve daje strukturu neuronske mreže.



Sl. 2.2 Jednostavna neuronska mreža

Svaka mreža se sastoji od ulaznog sloja, izlaznog sloja i proizvoljnog broja skrivenih slojeva.

2.2. Unazadna propagacija

Algoritam unazadne propagacije temeljni je algoritam po kojemu ANN-ovi uče. Ulazni podatak se šalje na ulazni sloj mreže. Ulazni sloj te podatke jednostavno prosljeđuje prvom skrivenom sloju. Prvi skriveni sloj računa izlaze svih neurona u sloju i šalje ih sljedećem sloju. Ovaj postupak se ponavlja dok izlazni sloj ne primi svoj ulaz iz zadnjeg skrivenog sloja te računa i daje izlaz mreže. Ovaj dio algoritma se naziva prolazak unaprijed. Važno je napomenuti da se neki međurezultati prolaska unaprijed čuvaju za danji rad.

Sljedeći korak algoritma je gradijentni spust. Cilj algoritma je pronaći optimalne vrijednosti težina svih veza mreže. U tu svrhu, algoritam dalje računa funkciju pogreške E na temelju dobivenog izlaza i izlaza definiranog primjerom. Kako bi sad znali kako promijeniti vrijednosti težina, računamo razinu promjene funkcije E po svim težinama w_i, odnosno računamo gradijente $\frac{dE}{dw_i}$. Ti gradijenti se računaju primjenom lančanog pravila i prolaska

kroz mrežu, ali ovaj put u suprotnom smjeru. Pomoću tih derivacija onda mijenjamo težine mreže prema izrazu

$$w_{ij} = w_{ij} - \alpha \cdot \frac{dE}{dw_i} \tag{6}$$

U ovom izrazu α je pozitivna konstanta koja se naziva faktor učenja ili brzina učenja (engl. learning rate). Što je α veći, to je brzina treniranja mreže brža jer se težine brže približavaju vrijednostima za koje će funkcija pogreške dostići minimum. Međutim, u tom slučaju moguće je da se težine koje se nauče budu sub-optimalne ili da cijeli postupak treniranja bude nestabilan odnosno da divergira. Što je α manji to će vrijeme treniranja biti dulje i moguće je da se algoritam zaglavi. Postoji poboljšanje gradijentnog spusta koje koristi dinamički promjenjiv faktor učenja α koje se koristi kako bi olakšali treniranje i sam postupak odabira faktora učenja.

Najjednostavnija inačica gradijentnog spusta koristi cijeli skup za treniranje za računanje gradijenta te samo jednom se ažuriraju težine. Bolji pristup je takozvani mini-batch stohastički gradijentni spust (engl. Mini-batch Stochastic Gradient Descent) kod kojeg se skup za treniranje dijeli na manje dijelove (engl. batch) te se računaju gradijenti i ažuriraju težine za svaki dio.

2.2.1. Prijenosne funkcije

Prethodno opisani algoritam unazadne propagacije s modelom umjetnog neurona kojem je prijenosna funkcija neka stepenasta funkcija ne bi funkcionirao jer stepenasta funkcija nema gradijente s kojima bi mogli raditi. Također, želimo modelirati i nelinearne modele. Opisani model prima ulaz i primjenjuje niz linearnih transformacija kako bi dobio izlaz. Ako nam je cilj napraviti nelinearnu funkciju, slično kako ljudski mozak funkcionira, onda linearnu prijenosnu funkciju zamjenjujemo nelinearnom funkcijom. Dakle prijenosna funkcija se primjenjuje na svaku težinsku sumu neurona i predaje kao izlaz neurona danjim slojevima.

Neke od najpopularnijih prijenosnih funkcija su sigmoidalna funkcija

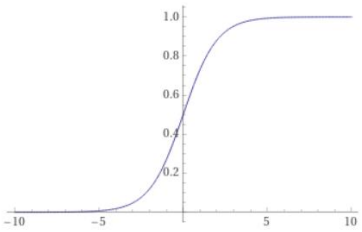
$$f(x) = \frac{1}{1+e^{-x}}, \tag{7}$$

tangens hiperbolni

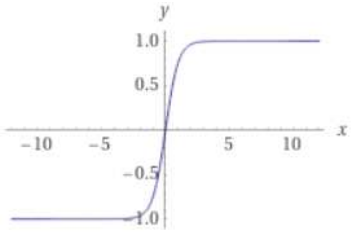
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{8}$$

te ReLU (engl. rectified linear units)

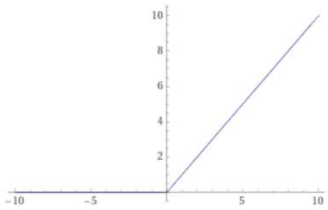
$$f(x) = \max(0, x) \tag{9}$$



Sl. 2.3 Sigmoidalna funkcija



Sl. 2.4 Tangens hiperbolni



Sl. 2.5ReLU

Valja spomenuti i funkciju Softmax koja se koristi kod klasifikacijskih modela koji na izlazu daju vjerojatnost da primjer pripada klasi. Ako imamo n klasi onda izlazni sloj mreže ima n vrijednosti. Tih n vrijednosti u sumi moraju dati 1 jer se radi o vjerojatnostima. Kako bi se ovo postiglo, na izlaznom sloju koristi se prijenosna funkcija Softmax.

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{10}$$

Ona svakoj klasi i pridružuje vrijednost $S(y_i)$ koja predstavlja vjerojatnost klase i.

Definirajmo još neke ključne pojmove vezane uz treniranje umjetnih neuronskih mreža.

- Veličina serije (engl. batch size) – označava broj primjera iz ulaznog skupa podataka za treniranje koji se propagiraju kroz mrežu unaprijed prije nego što se za sve njih računa funkcija pogreške i započinje propagacija unatrag
- Epoha (engl. epoch) – predstavlja jedan prolaz algoritma kroz cijeli skup podataka za učenje. Broj epoha definira koliko puta model prođe kroz cijeli skup podataka tijekom treniranja.

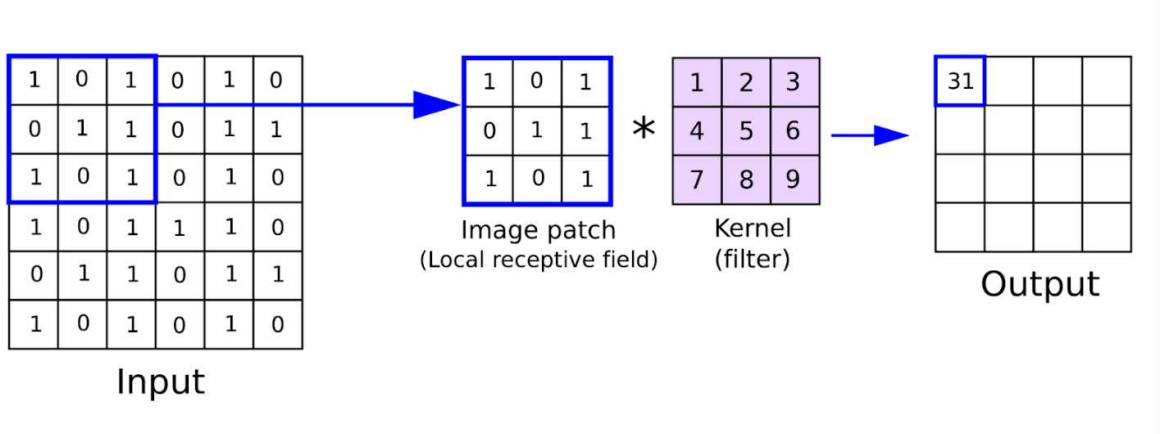
2.3. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (engl. Convolutional neural networks – CNNs) su osnovna gradiva jedinica dubokih modela za računalni vid. Najčešće se koriste u klasifikaciji slika ili prepoznavanju objekata na slikama. Međutim, upotrebljavaju se i za ostale probleme, kao što su raspoznavanje govora ili za obradu prirodnog jezika [2]. CNN se uglavnom sastoje od nekog broja konvolucijskih slojeva popraćenih slojem sažimanja te potpuno povezanim

slojevima. Potpuno povezani sloj je skriveni sloj kojem je svaki neuron povezan sa svakim neuronom iz prijašnjeg sloja. Slika 2.2 prikazuje skriveni sloj koji je potpuno povezan.

2.3.1. Konvolucijski sloj

U matematici, konvolucija je operacija koja pomoću dvije funkcije radi treću funkciju koja opisuje kako je oblik prve modificiran drugom. U strojnom učenju, posebno računalnom vidu, konvolucija se najčešće koristi za raspoznavanje uzorka na slikama. Zamislimo da imamo sliku nekih dimenzija, ta slika može biti prikazana matricom istih dimenzija čiji elementi predstavljaju vrijednosti boja piksela te slike. Konvolucija je primjena neke manje matrice, zvane filtrom, po cijeloj slici. Slika 2.1 prikazuje jedan korak konvolucije ulazne slike. Na slici je filtar matrica dimenzija 3x3 dok je ulaz neka matrica dimenzija 6x6. Konvolucijski sloj primjenjuje filtar na dio ulaza definiran klizećim prozorom. Primjenjivanje filtra je uglavnom množenje matrice tako da se množi element na poziciji (i, j) matrice klizećeg prozora s elementom na poziciji (i, j) matrice filtra, te se sumiraju svi takvi međurezultati u konačnu vrijednost.

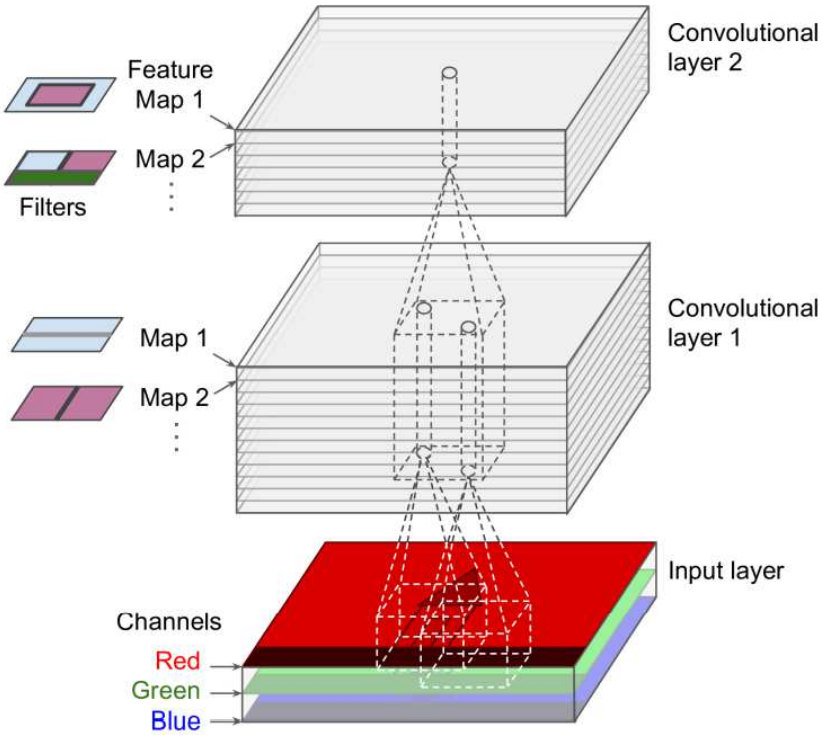


Sl. 2.6 Konvolucija

Konvolucijski sloj je najbitniji dio CNN-a. Konvolucijski sloj primjenjuje filtar na cijelu sliku i daje izlaz. Jedan neuron konvolucijskog sloja, koji je direktno nakon ulaznog sloja, je povezan s jednim uzorkom ulaza, kojem su dimenzije jednake dimenzijama filtra. Gledajući ponovno sliku 2.6, uokvireni neuron na izlazu je povezan s 3x3 označenim dijelom ulaza. Na isti način, sljedeći konvolucijski sloj povezan s prvim konvolucijskim slojem ima neurone povezane s nekim prozorom prvog konvolucijskog sloja. Svaki neuron konvolucijskog sloja primjenjuje isti filtar kako bi na izlazu dao takozvanu mapu značajki.(engl. feature map). Također, kako bi sačuvao dimenzije ulazne slike, konvolucijski sloj uglavnom radi nadopunu (engl. padding) nad ulazom. Na primjer, nadopuna nulama,

koja oko slike ulaza doda okvir nula. Konvolucijski sloj uglavnom pri svakom koraku prozor pomiče za jedno mjesto, no moguće je podesiti takozvani korak (engl. stride) sloja na način da se prozor pomiče s više mjesta.

Opisano do sada je rad konvolucijskog sloja s jednim filterom. Ono što konvolucijski sloj zapravo radi je da primjenjuje više filtera i time na izlaz daje veći broj mapi značajki. S time na umu, jedan neuron konvolucijskog sloja zapravo na ulaz prima sloj prozora na mape značajki. Odnosno, svaki neuron svake mape značajki jednog konvolucijskog sloja prima ulaz od svake grupe neurona svake mape značajki konvolucijskog sloja ispred. Ako konvolucijski sloj prima ulaz iz ulaznog sloja onda je to najčešće mapa od 3 značajke gdje svaka predstavlja udio crvene, zelene odnosno plave boje. Ovo je najlakše shvatiti gledajući sliku 2.7. Računanje mape značajki se onda svodi na težinsko sumiranje svih vrijednosti prozora mapi značajki ispred trenutnog sloja.

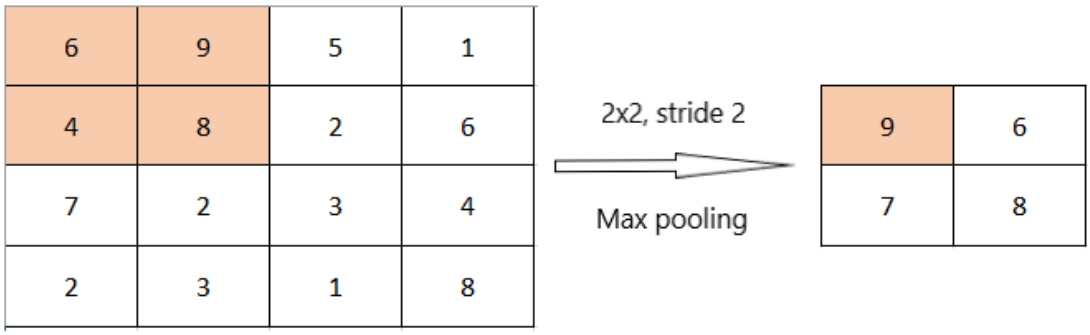


Sl. 2.7 Prikaz konvolucijskog sloja [2]

2.3.2. Sloj sažimanja

Sloj sažimanja (engl. pooling layer) se često koristi uz CNN-e. Njegov cilj je sažimanje mapi značajki. To se radi zbog raznih razloga kao što su smanjenje zauzete memorije i

smanjenje broja parametara mreže, što u konačnici može puno pomoći s generalizacijom. Postupak sažimanja je prikazan slikom 2.8.



Sl. 2.8 Postupak sloja sažimanja

Sloj sažimanja definiran je veličinom prozora sažimanja ili filterom, korakom sažimanja (engl. stride) i načinom sažimanja. Veličina prozora sažimanja definira koliki dio originalne slike će sloj sažimanja gledati pri svakom koraku rada. Korak sažimanja definiran je isto kao i kod konvolucijskog sloja. Način sažimanja opisuje operaciju koja se provodi na prozorom. Na primjer, maksimalno sažimanje (engl. max-pooling) znači da prozor kao izlaz daje maksimalnu vrijednost od svih vrijednosti prozora na ulazu. Na slici 2.8 je prikazan jedan korak sažimanja koji radi s filterom veličine 2x2, s korakom sažimanja vrijednosti 2 te maksimalnim sažimanjem.

3. Duboko učenje

Duboko učenje (engl. Deep Learning) je podgrana strojnog učenja koja se bavi isključivo umjetnim neuronskim mrežama, posebice vrlo velikim i dubokim modelima. Ono postaje vrlo popularno područje istraživanja zbog razvoja novih tehnika i modeliranja, pojave velikih skupova podataka te velikog razvoja hardware-a odnosno procesorske moći računala.

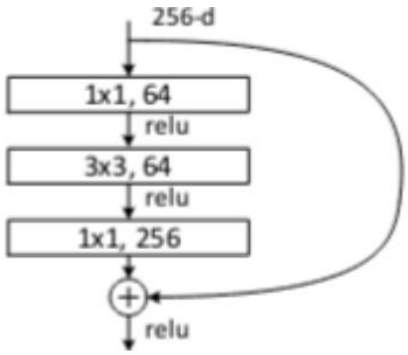
Zbog potreba ovoga rada u nastavku poglavlja ukratko su predstavljene samo neke poznate arhitekture dubokih modela koje se koriste u računalnom vidu te je opisan prijenos znanja.

3.1. Duboki modeli

Duboku modeli korišteni u sklopu ovoga rada su modeli koji služe za detekciju objekata u slikama. Takvi modeli se uglavnom sastoje od dvije cjeline. Prva je ekstraktor značajki odnosno CNN koji na izlaz daje mapu značajki. Druga cjelina je detekcijska mreža koja prima mapu značajki od ekstraktora značajki.

3.1.1. ResNet

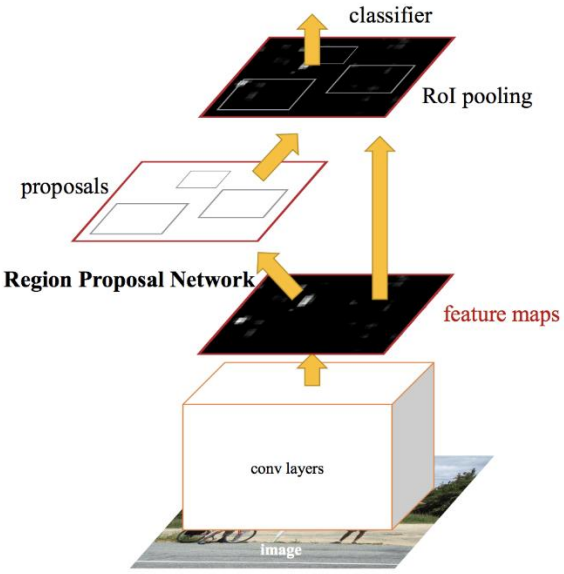
Problem kod dubokih modela su takozvani nestajući gradijenti – kako se gradijenti propagiraju prema ranijim slojevima tako se gradijent sve više i više smanjuje. ResNet (engl. Residual Neural network) je arhitektura čija je glavna ideja propagacija unaprijed kako bi se ublažio problem nestajućih gradijenata. ResNet to čini tako da uvodi predikciju prethodnika na način prema slici 3.1.



Sl. 3.1 ResNet predikcija prethodnika

3.1.2. CNN bazirana na regijama

Glavna ideja CNN-a baziranog na regijama (engl. Region Based CNN – R-CNN) je da se iz slike generira određeni broj regija nad kojima će se vršiti klasifikacija. Te regije se dalje predaju konvolucijskoj mreži koja klasificira. Glavni nedostatak ovakvog rješenja je brzina i vrijeme treniranja. Kako bi se model ubrzao, nastaju brze R-CNN (Fast R-CNN) koje nakon odabira regija ne šalju regije u CNN direktno nego se na konvolucijski sloj šalje samo originalna slika i tada se mapa značajki koristi za klasifikaciju regija. Još daljnje je brža R-CNN (Faster R-CNN) koja ne koristi selektivni odabir regija nego se daje mreži da sama nauči regije. Za to se koristi odvojena mreža „Region Proposal Network".



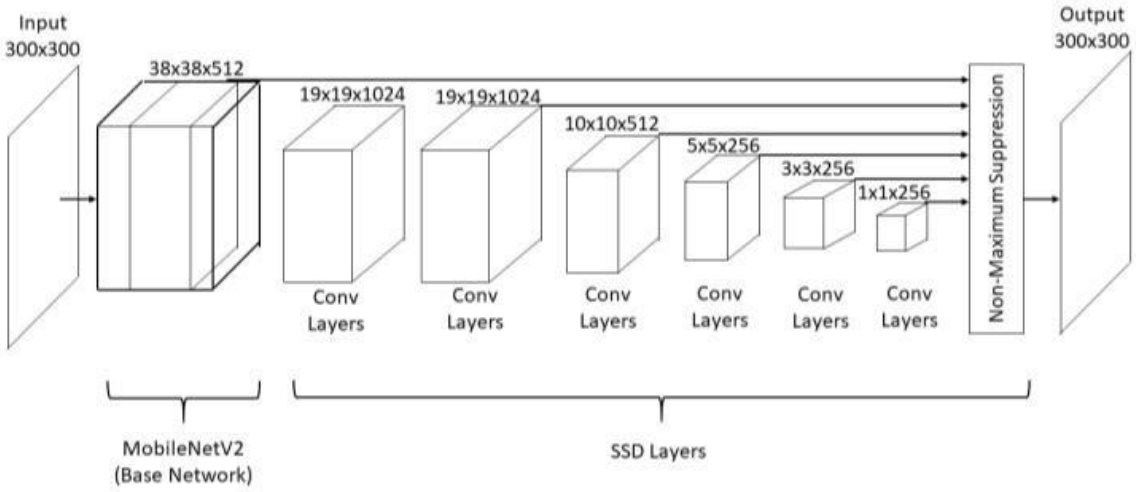
Sl. 3.2 Faster R-CNN – Mapu značajki može dati neki model kao što je ResNet

3.1.3. Single shot detector – SSD

Za razliku od modela kao što su R-CNN ili Fast(er) R-CNN detekciju dijele na dva dijela – identificiranje regija i zatim klasifikacija regija – SSD model detekciju radi u jednom koraku. Taj pristup ih generalno čini bržima od R-CNN-a, ali za tu brzinu plaćaju na točnosti. U suštini, SSD dijeli sliku na mrežu ćelija i u svakoj ćeliji se provodi detekcija. Svakoj ćeliji se pridjeljuje brojčana vrijednost koja označava koliko dobro objekt pripada nekoj ćeliji. Određenim postupkom se onda dobiva završna detekcija iz skupa detekcija koje se preklapaju. Uz SSD se uglavnom koristi neka druga mreža kao bazna mreža modela koja služi kao ekstraktor značajki.

3.1.4. MobileNet

MobileNet je arhitektura razvijena primarno za klasifikaciju pomoću mobilnih uređaja. Dakle, to je vrlo računalno ne zahtjevna metoda. Glavna ideja MobileNet-a je dubinski separirana konvolucija koja značajno smanjuje broj parametara mreže u odnosu na klasičnu CNN.



Sl. 3.3 MobileNet SSD arhitektura – MobileNet je bazna mreža modela, SSD radi detekciju

3.2. Prijenos znanja

Ideja prijenosa znanja je vrlo jednostavna – iskoristiti već naučeni model za novi, jednostavniji zadatak. Posljednjih nekoliko slojeva istrenirane mreže se odrežu i preostali slojevi se spoje s početkom mreže za novi zadatak te se takva mreža trenira za novi problem. Ovaj rad je temeljen na ovoj ideji. Iskorištavaju se mreže trenirane za detekciju objekata Faster R-CNN ResNet i SSD MobileNet kako bi se napravio model koji detektira dlan čovjeka.

4. Sustav za detekciju i praćenje dlana

Prisjetimo se, cilj ovoga rada je dizajnirati i napraviti sustav za detekciju i praćenje dlana. Problem detekcije objekata se može riješiti primjenom više različitih metoda računalnog vida. Neke od metoda su metoda lokalnih binarnih značajki (engl. Local Binary Patterns), metoda histograma orijentiranih vektora (engl. Histogram of Oriented Gradients) te primjena dubokog učenja. Kao što se već da naslutiti, za razvoj traženog sustava u radu se primjenjuje duboko učenje tj. duboke umjetne neuronske mreže. Rješavanje problema detekcije objekata do nedavno ne bi bio laki zadatak zbog samog hardware-a potrebnog za treniranje jednog dubokog modela. Međutim, danas postoje razne biblioteke i programska sučelja koja značajno olakšavaju cjelokupan proces izrade sustava, od prikupljanja podataka do treniranja i testiranja modela. Također, današnje grafičke kartice su se pokazale kao izvanredan alat pri treniranju dubokih modela.

Ideja implementacije traženog sustava je korištenje sučelja Tensorflow Object Detection za treniranje modela koji će služiti za detekciju dlana. Praćenje dlana će se implementirati tako da se uzima u stvarnom vremenu slika s kamere te se primjenjuje trenirani model na svaku sliku te u stvarnom vremenu crta rezultat modela na slici, odnosno položaj dlana ako ga je model prepoznao.

U narednom poglavlju daje se uvid u korištene biblioteke i programska sučelja korištena za rješavanje problema, opisuje se postupak prikupljanja i pripreme skupa podataka potrebnih za treniranje modela te se opisuje sam postupak treniranja modela.

4.1. Programska podrška i korištene biblioteke

4.1.1. Tensorflow

Tensorflow je besplatna open-source biblioteka razvijena u Google-u i služi za strojno učenje i umjetnu inteligenciju. Bitan dio Tensorflow-a je API za duboko učenje Keras. Keras je vrlo jednostavno, fleksibilno i moćno sučelje razvijeno za potrebe brzog testiranja i rješavanja problema strojnog učenja s fokusom na duboko učenje. Velika prednost Tensorflow-a je činjenica da je koristi Python kao „front-end“ rada, ali u pozadini koristi visoko optimizirani i brzi C++.

4.1.2. Tensorflow Object Detection API

Tensorflow Object Detection je programsko sučelje razvijeno nad Tensorflow-om koje služi za jednostavno konstruiranje i treniranje modela za detekciju objekata. Zbog toga je ono ključni dio ovoga rada. Ono što čini ovo sučelje toliko popularnim i moćnim je velika jednostavnost korištenja te relativno velik broj već treniranih modela nad poznatim skupovima podataka kao što su COCO skup podataka, KITTI skup podataka te Open Images skup podataka. Ti trenirani modeli onda mogu služiti kao potpora za nove modele dobivene već opisanom tehnikom prijenosa znanja. Ti modeli dostupni su svima na Tensorflow Object Detection API Model Zoo [4] dok je sam API dostupan na [5].

4.1.3. OpenCV

OpenCV (CV - Computer Vision) je biblioteka korištena za računalni vid i strojno učenje. Tijekom implementacije sustava, OpenCV je korišten za prikupljanje skupa podataka te testiranje treniranog modela u stvarnom vremenu.

4.2. Prikupljanje i priprema skupa podataka

4.2.1. Prikupljanje podataka

Kako bi trenirali model za detekciju dlana potreban je skup podataka koji sadrži označene slike dlanova osoba. Najveći i najpoznatiji takvi skupovi podatka su *EgoHands Dataset* [6] te *Hand Dataset* [7]. Međutim ovi skupovi podatak nisu prikladni za problem detekcije dlana. Prvi je skup slika ruku iz egocentrične perspektive i stoga teže detektira dlan a lakše šaku. Drugi skup iz sličnih razloga teže detektira dlan.

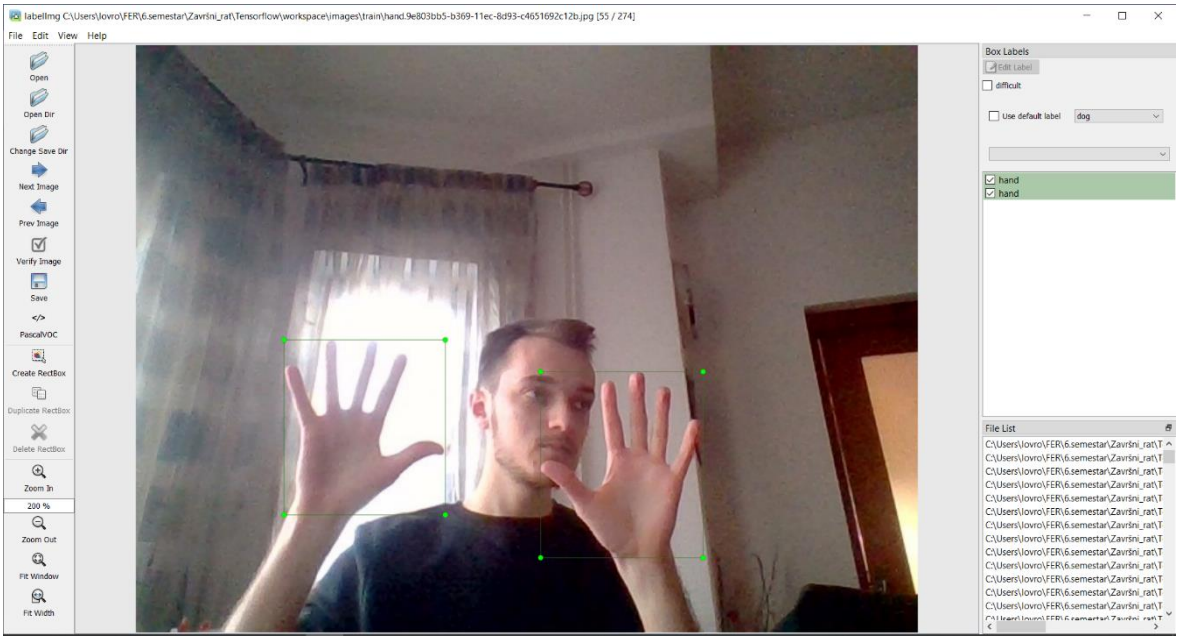
Posjetimo se da se modeli treniraju prijenosom znanja i stoga nam nije potreban veliki skup podatak kako bi trenirali modele. Zbog toga sami stvaramo vlastiti skup podataka. Ovo jednostavno ostvarujemo pomoću OpenCV biblioteke. Pokrećući skriptu danu u nastavku, s kamere računala svake dvije sekunde se očituje slika i sprema za kasnije označavanje.

```
cap = cv2.VideoCapture(0)
time.sleep(5)
numberImgs = 100
for imgnum in range(number_imgs):
    ret, frame = cap.read()
    imagename = os.path.join(IMAGES_PATH, "hand", "hand" + "." +
                             "{}.jpg".format(str(uuid.uuid1())))
    cv2.imwrite(imagename, frame)
    cv2.imshow("frame", frame)
    print("Sleeping")
    time.sleep(2)
```

Sl. 4.1 Skripta za sakupljanje skupa podataka

4.2.2. Označavanje podataka – LabelImg

Skup podatak dobiven u prijašnjom koraku sastoji se od slika koje je dalje potrebno označiti. To činimo open-source alatom LabelImg [8].



Sl. 4.2 Sučelje alata LabelImg

Sučelje alata je prikazano na slici 4.2. Odabere se Open Dir i kao put odabere put od skupa podataka. Odabere se Save Dir i kao put odabere put do direktorija u koji želimo spremati .xml datoteke koje će svaka pripadati jednoj slici iz skupa podataka i sadrži informaciju o tome što je na slici označeno i kojom labelom. Na primjeru na slici, označena su dva objekta i oba su označena labelom 'hand'. Prolazi se kroz sve slike iz Open Dir direktorija i označuju se cijeli skup podataka. Kada smo označili skup podataka, dijelimo ga na skup za treniranje i testiranje.

4.2.3. TFRecord

Kako bi upotrijebili Tensorflow Object Detection API za treniranje, potrebno je dalje skup podataka (posebno skup za treniranje, validaciju i testiranje) pretvoriti u format s kojim API radi. To je poseban format skupa podataka zvan TFRecord. TFRecord je jednostavan format u kojemu je svaki primjer poredan u nizu jedan do drugoga sa svim informacijama koje treba imati – veličina, ime, parametri bounding box-a oznake itd. Time su svi podatci potrebni za treniranje na jednom mjestu, tj. možemo efikasno dohvaćati pojedine primjere i trenirati model. Jednostavno generiranje TFRecord-a za treniranje i testiranje dobivamo pomoću skripte preuzete sa [9]. Skripta uzima slike i njihove pripadne .xml datoteke i pretvara ih u primjere TFRecord-a i sve te primjere sprema u jednu datoteku.

4.3. Treniranje modela

Kako bi mogli primijeniti prijenos znanja, potreban nam je model treniran nad problemom sličnim problemu detekcije dlana. U tu svrhu, preuzimamo modele sa stranice [4]. Konkretno, u ovome radu koristimo modele Faster R-CNN ResNet50 640x640 i SSD MobileNet 320x320. Oba modela su originalno trenirana na COCO17 skupu podataka [10].

Postupak konfiguracije je sljedeći. Prvo je potrebno stvoriti prazan direktorij u koji će se spremati kontrolne točke tijekom treniranja. Zatim je potrebno kopirati *pipeline.config* datoteku koja se nalazi u svakom modelu preuzetom sa [4]. Ovu datoteku koristi Object Detection API i u njoj se nalaze sve bitne informacije koje možda budu potrebne za treniranje novih modela. U nastavku izdvajamo najbitnije dijelove *pipline.config* datoteke.

```
model {
  ssd {
    num_classes: 90
    …
  }
…
```

```
train_config {
  batch_size: 128
  data_augmentation_options {
    …
  }
  data_augmentation_options {
    …
  }
  sync_replicas: true
  optimizer {
   …
  }
  fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"
  num_steps: 50000
  startup_delay_steps: 0.0
  replicas_to_aggregate: 8
  max_number_of_boxes: 100
  unpad_groundtruth_tensors: false
  fine_tune_checkpoint_type: "classification"
  fine_tune_checkpoint_version: V2
}
train_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
eval_config {
  …
}
eval_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
```

Datoteka 4.1. Dio pipeline.config datoteke za unaprijed trenirani model SSD MobileNet

Ovu datoteku onda modificiramo na sljedeći način. Potrebno je `num_classes` postaviti na broj klasa koje želimo da naš novi model razazna. U našem slučaju to je samo jedna klasa. Zatim je moguće, ali ne i potrebno, modificirati vrijednost `batch_size` koja veličinu serije postavlja na željenu vrijednost. Napominjem sada da zbog arhitekture Faster R-CNN ResNet modela, ovaj parametar mora biti jednak broju grafičkih kartica koje se koriste za treniranje. Dalje je potrebno modificirati puteve `train_input_reader`-a te puteve `eval_input_reader`-a. Prvi traži put do TFRecord datoteke koja sadrži skup podataka za treniranje, a drugi put do TFRecord datoteke koja sadrži skup podatak za validaciju. Svakom je još potrebno postaviti `label_map_path` koji predstavlja put do .pbtxt datoteke koja u sebi sadrži informacije o oznakama koje model treba koristi. U našem slučaju ona izgleda ovako:

```
item {
  name:'hand'
  id:1
}
```

Datoteka 4.2. label_map.pbtxt datoteka

I zadnje što treba modificirati je `fine_tune_checkpoint_type` i postaviti na vrijednost `'detection'`.

Ovime smo spremni započeti treniranje modela. Iz datoteke u kojem nam se nalazi Object Detection API izvodimo naredbu:

```
python Tensorflow/models/research/object_detection/model_main_tf2.py --
model_dir=<put_do_dirktorija_novog_modela> --
pipeline_config_path=<put_do_modificirane_pipline.config_datoteke> --
num_train_steps=<broj_koraka_treniranja>
```

Napominjem da ovdje `num_train_steps` predstavlja broj koraka treniranja. Broj koraka treniranja u ovom slučaju definiran je kao jedan batch. Odnosno, broj koraka treniranja podijeljen s veličinom serije daje broj epohi treniranja ovakvog modela. Razvijeni Faster R-CNN ResNet model je treniran nad 20,000 koraka dok je SSD MobileNet treniran nad 18,000 koraka.

5. Rezultati

5.1. Sustav za praćenje

Nakon treniranja modela, implementiran je sustav praćenja tako da se svaka slika dobivena s kamere predaje modelu koji daje predikciju i pomoću izlaza se crta bounding-box oko pronađenog dlana. U nastavku je dan dio kôda.

```
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import config_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder

configs = config_util.get_configs_from_pipeline_file(<put_do_.config_datoteke)
model_config = configs['model']
detection_model = model_builder.build(
                model_config=model_config, is_training=False)

ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(MODEL_PATH, <kontrolna_tocka>)).expect_partial()

def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)

    return detections, prediction_dict, tf.reshape(shapes, [-1])
```

Sl. 5.1. Učitavanje modela

Slika 5.1 prikazuje učitavanje modela u program, model se učitava od zadnjeg kontrolne točke treniranja. Također je definirana funkcija koja vraća predikciju modela nad danom slikom.

Dalje je prikazan dio koda kojim se pokreće kamera te se uzima slika s kamere na koju se crta bounding-box rezultata.

```
category_index = label_map_util
                    .create_category_index_from_labelmap(<put_do_label_map_datoteke)

import cv2
cap = cv2.VideoCapture(0)

import numpy as np

while True:
    ret, image_np = cap.read()

    image_np_expanded = np.expand_dims(image_np, axis=0)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections, predictions_dict, shapes = detect_fn(input_tensor)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
          image_np_with_detections,
          detections['detection_boxes'][0].numpy(),
          (detections['detection_classes'][0].numpy() + label_id_offset).astype(int),
          detections['detection_scores'][0].numpy(),
          category_index,
          use_normalized_coordinates=True,
          max_boxes_to_draw=200,
          min_score_thresh=.5,
          agnostic_mode=False)

    cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))
    if cv2.waitKey(25) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Sl. 5.2. Čitanje kamere te crtanje bounding-box-a dobivenog predikcijom

Sustav za praćenje je mogao biti implementiran i na drugačije načine. Na primjer, u slučaju više dlanova na slici – što je i razumna pretpostavka – mogao je biti definiran zahtjev da sustav različite dlanove može razlikovati, tj. da sustav za praćenje dodjeli posebne oznake dlanovima kako bi ih se moglo razlikovati. U tom slučaju bilo bi potrebno na neki način u svakom idućem trenutku na temelju prijašnjeg trenutka odrediti koji dlan pripada kojoj oznaci, odnosno na neki način implementirati povezivanje detekcija iz slike u sliku.

5.2. Metrike uspješnosti

Prije nego vidimo rezultate pojedinih mreža testiranih na skupu za testiranje, definiramo metrike koje su korištene kao mjera performansi modela. To su preciznost (engl. Precision - P) i odziv (Recall - R). U svrhu iskaza tih mjera, definiramo: True Positive (TP), False Positive (FP) i False Negative (FN). True/False predstavlja da se radi o primjeru koji je točno klasificiran ili ne. Positive/Negative označava da se radi o primjeru na koji je

klasifikator predvidio da se pozitivan (npr. neki objekt se nalazi na slici) ili negativan (neki objekt se ne nalazi na slici). S ovime definiramo:

TP – označava broj primjera koji su točno klasificirani i pozitivni su

FP – označava broj primjera koji su netočno klasificirani i pozitivni su

FN – označava broj primjera koji su netočno klasificirani i negativni su

U kontekstu ovoga rada, primjer se klasificira pozitivnim ako je omjer presjeka stvarnog i predviđenog bounding-box-a dlana i unije stvarnog i predviđenog bounding-box-a dlana viši od x. Inače, ovdje se taj omjer naziva Intersection over Union (IoU), a x se naziva IoU threshold. TP je onda definiran kao broj primjera na kojima se nalazi dlan te je $IoU \geq x$, FP je broj primjera na kojima se nalazi dlan, ali je $IoU < x$, i FN je broj primjera na kojima se ne nalazi dlan, ali je model detektirao nešto.

S ovime na umu, možemo definirati preciznost i odziv. Preciznost je definirana kao udio točno definiranih pozitivnih primjera u skupu svih pozitivno definiranih primjera.

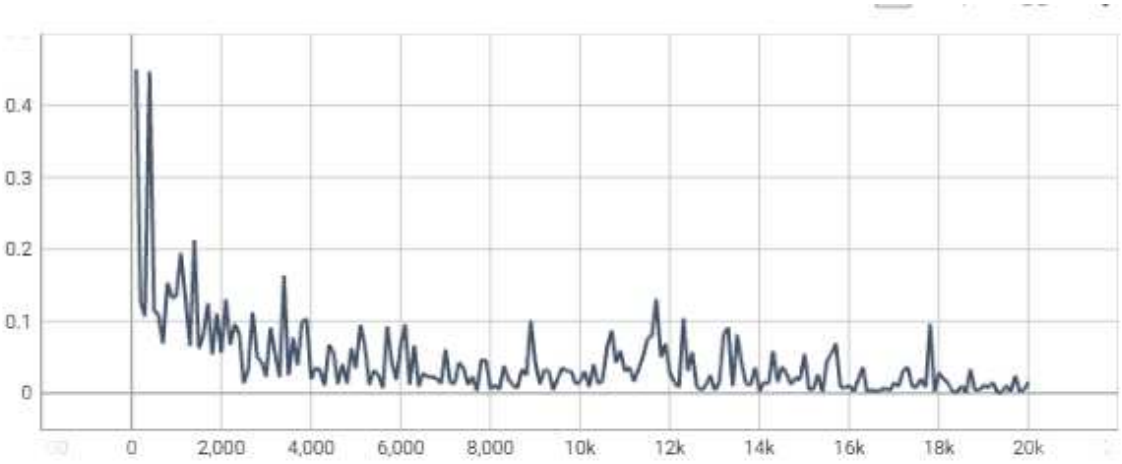
$$P = \frac{TP}{TP+FP} \tag{11}$$

Odziv je definiran kao udio točno klasificiranih primjera u skupu svih primjera koji su u stvarnosti pozitivni.

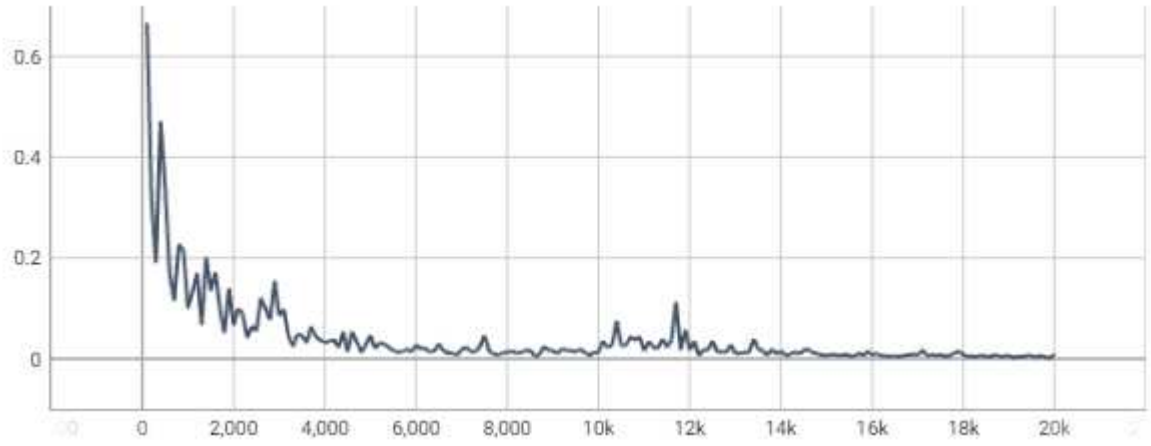
$$R = \frac{TP}{TP+FN} \tag{12}$$

5.3. Faster R-CNN ResNet

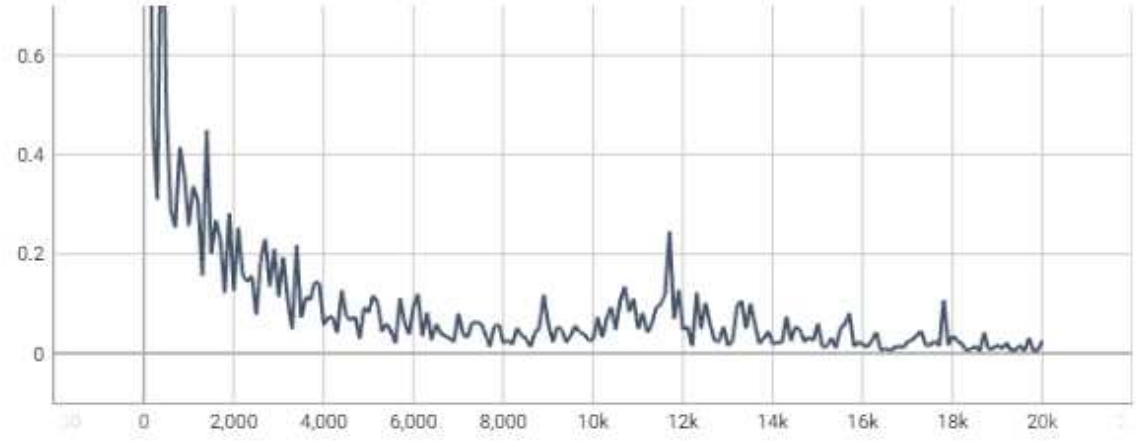
Pogledajmo najprije tijek treniranja Faster R-CNN ResNet modela.



Sl. 5.3. Faster R-CNN ResNet Classification loss



Sl. 5.4. Faster R-CNN ResNet Localization loss



Sl. 5.5. Faster R-CNN ResNet Total loss

Slike redom prikazuju klasifikacijski, lokalizacijski i ukupni gubitak mjeren tijekom treniranja.

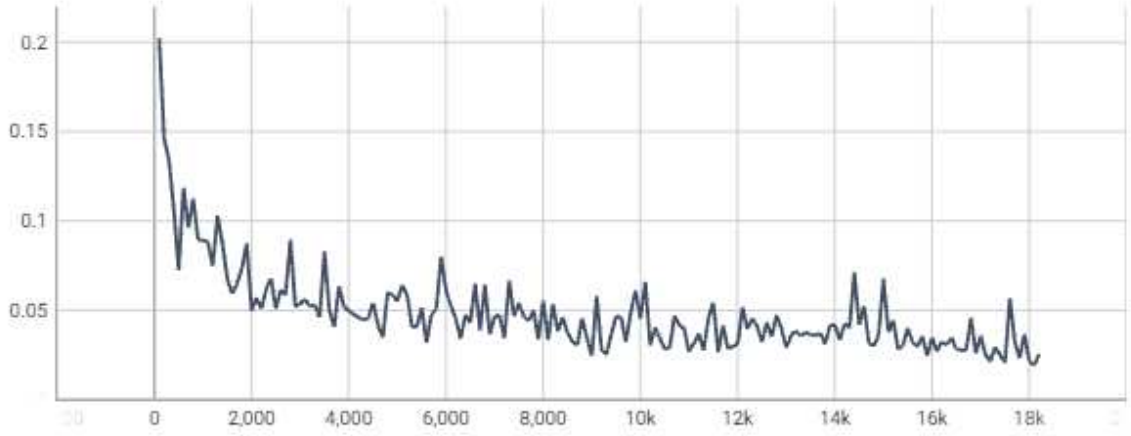
Prilikom izračuna metrika, računa se P i R za svaki IoU threshold krenuvši od 0.50 pa do 0.95 korakom 0.05 te je uzet prosjek tih rezultata i dobivena prosječna preciznost – AP, odnosno prosječni odziv - AR. Rezultati za Faster R-CNN ResNet su kako slijedi:

$$AP = 0.741$$

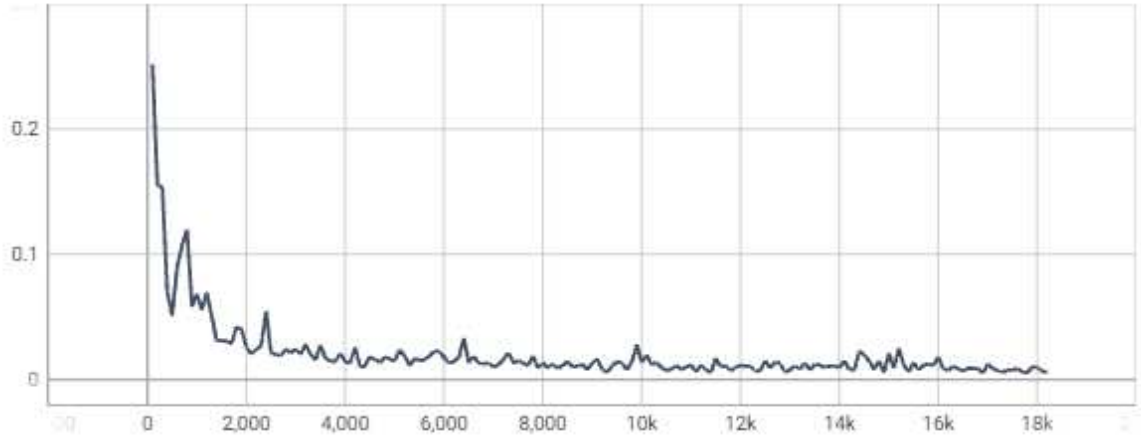
$$AR = 0.754$$

5.4. SSD MobileNet

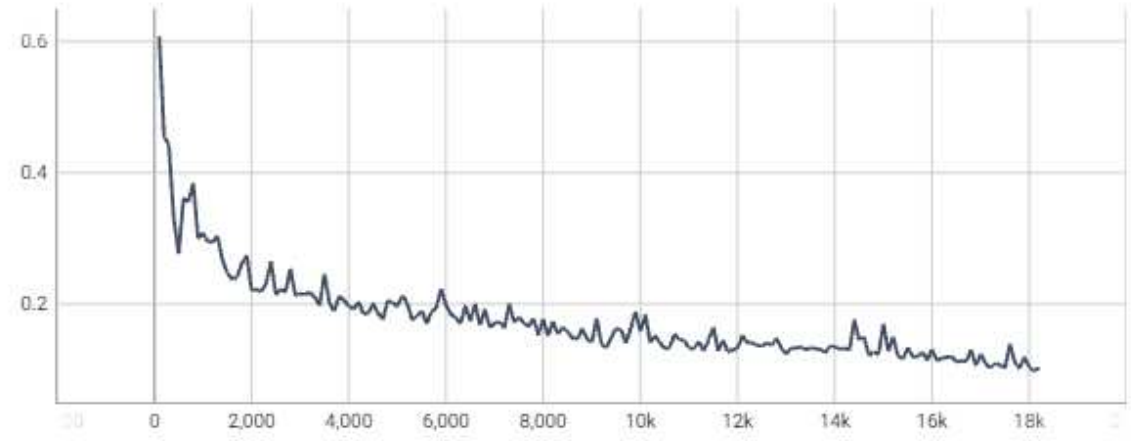
Pogledajmo tijek procesa treniranja SSD MobileNet modela, isto prikazan i kao kod Faster R-CNN modela.



Sl. 5.6. SSD MobileNet Classification loss



Sl. 5.7. SSD MobileNet Localization loss



Sl. 5.8. SSD MobileNet Total loss

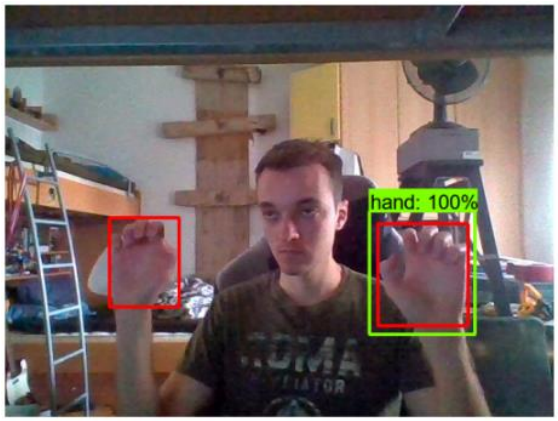
Rezultati za Faster R-CNN ResNet su kako slijedi:

$$AP = 0.716$$

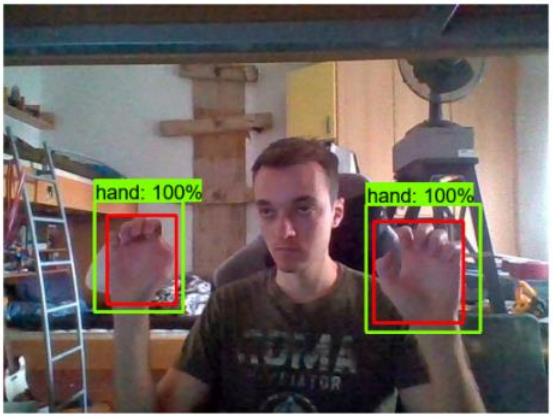
$$AR = 0.772.$$

5.5. Primjeri detekcija

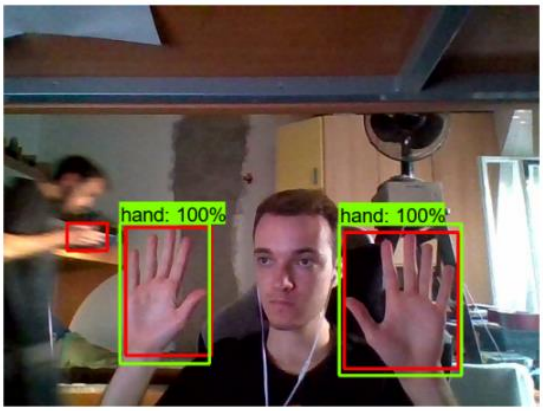
U nastavku slijede neki testni primjeri koji demonstriraju oba modela.



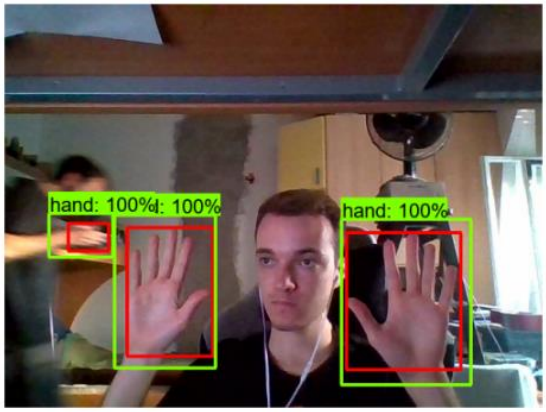
Sl. 5.9. SSD MobileNet neuspješna detekcija



Sl. 5.10. Faster R-CNN ResNet uspješna detekcija



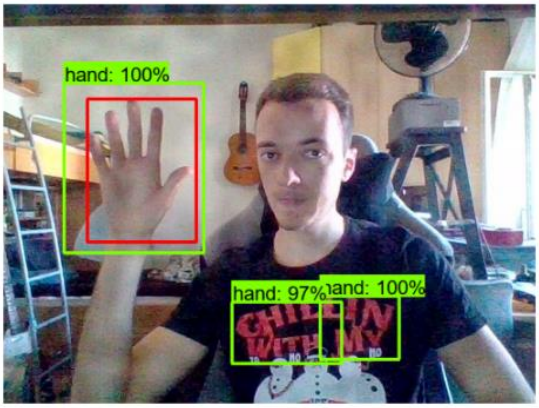
Sl. 5.11. SSD MobileNet neuspješna detekcija



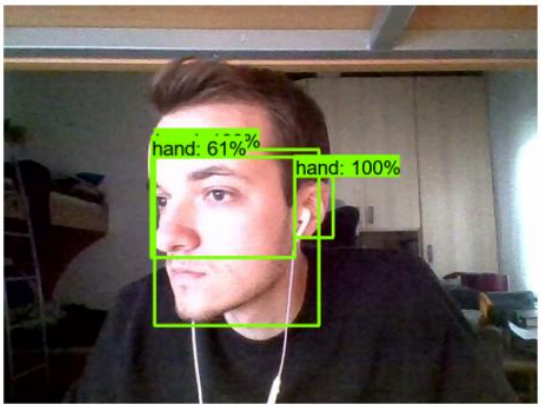
Sl. 5.12. Faster R-CNN ResNet uspješna detekcija

Slika 5.9 prikazuje primjenu SSD MobileNet modela na jedan od testnih primjera, dok slika 5.10 prikazuje primjenu Faster R-CNN ResNet modela na isti primjer. Vidimo da desni dlan SSD model ne prepoznaje dok ga R-CNN uredno prepozna. Slična situacija je na slici 5.11 i slici 5.12. Valja napomenuti i da su kod R-CNN modela predviđeni bounding-box-ovi inače veći nego kod SSD modela. To je i razlog zašto je prosječan odziv kod SSD modela malo bolji nego kod R-CNN modela – pri izračunu prosječnog odaziva, za veće IoU threshold

vrijednosti, R-CNN model propada jer je IoU vrijednost primjera manja od IoU threshold-a zbog veće površine bounding-box-a.



Sl. 5.13. Primjer krive detekcije



Sl. 5.14. Primjer krive detekcije

Slike 5.13 i 5.14 prikazuju primjere na kojima oba modela krivo detektiraju. Prikazani su rezultati R-CNN modela, dok su rezultati SSD modela vrlo slični.

Općenito, oba modela imaju slične performanse. Najčešće dlanove dovoljno blizu kamere detektira ispravno. Rijetko se pojavi dlan koji ne uspije detektirati. Problem je što u ne malom broju slučajeva detektira dlan tamo gdje u stvarnosti nije. Tako se zna dogoditi da lica i ruke zbog boje kože i neke druge predmete detektira kao dlan. Ove detekcije povećavaju FN što u konačnici daje lošiji odziv. Razlog ovim lažnim detekcijama je najvjerojatnije sam skup podataka koji bi trebalo kvalitetnije prikupiti i označiti.

Zaključak

U okviru rada uspješno su razvijena dva različita duboka modela za detekciju ljudskih dlanova na slikama. Uspješno je implementiran sustav za praćenje dlana kao uzastopno korištenje razvijenih modela za detekciju. Cjelokupni proces treniranja dubokih modela jako je olakšan uporabom prijenosa znanja i specijaliziranih biblioteka za detekciju objekata u slikama. Također , prijenosom znanja, pokazana je snaga razvijenih modela unatoč relativno malenome skupu podataka za treniranje i relativno kratkom vremenu treniranja dubokih modela.

Performanse modela su očekivane, Faster R-CNN model ima bolju preciznost, ali joj je nedostatak brzina i veličina. Iako su rezultati SSD modela, što se tiče preciznosti, malo lošiji, njegova brzina predviđanja i treniranja su znatna prednost nad Faster R-CNN modelom.

S time da su u radu testirani isključivo Faster R-CNN modeli i SSD modeli, bilo bi zanimljivo isprobati neke druge poznate arhitekture za detekciju objekata kao što su CenterNet, EfficientDet, Mask R-CNN i YOLO. Također, bolji rezultati bi možda bili ostvareni da je skup podatak pripremljen i sakupljen pažljivije i kvalitetnije.

U budućim radovima razvijeni modeli za detekciju bi mogli biti korišteni kao osnova za prijenos znanja za neke druge probleme, na primjer, za implementaciju sustava koji detektira i prepoznaje geste ruku te na temelju njih izvodi neku akciju.

Literatura

[1] Alpaydin, E., Introduction to Machine Learning. 3. izdanje. The MIT Press, 2014.

[2] Géron, A., Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. 2. izdanje. O'Reilly Media, Inc., 2019.

[3] Zhang, A., Lipton, Z.C., Li, M., Smola, A.J., Dive into Deep Learning. Poveznica: https://d2l.ai/

[4] Tensorflow Object Detection API Model Zoo. Poveznica: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md

[5] Tensorflow Object Detection API. Github poveznica: https://github.com/tensorflow/models/tree/master/research/object_detection

[6] Bambach, S. L., Stefan, Crandall, David J., Yu, C., Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions, 2015. Poveznica: http://vision.soic.indiana.edu/projects/egohands/

[7] Mittal, A., Zisserman, A., Torr, P., Hand Dataset. Poveznica: https://www.robots.ox.ac.uk/~vgg/data/hands/

[8] Tzutalin. Labellmg. Git code (2015). https://github.com/tzutalin/labellmg

[9] Vladimirov, Lj., Tensorflow Object Detection API tutorial, 2020., Poveznica: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#create-tensorflow-records

[10] Lin, T., Maire, M., Belongie, M., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., Dollár, P., Microsoft COCO: Common Objects In Context, 2017., Poveznica: https://cocodataset.org/#home

Sažetak

Naslov: Sustav za detekciju i praćenje dlana

Rad daje kratak uvod u strojno učenje, posebice umjetne neuronske mreže. Predstavlja duboko učenje i neke od dubokih modela korištenih za razvoj sustava za detekciju dlana. Opisan je postupak prikupljanja i pripreme skupa podataka korištenog za treniranje modela. Iskazan je postupak treniranja dubokih modela koji se služi Tensorflow Object Detection aplikacijskim programskim sučeljem te su predstavljeni rezultati svakog modela.

Ključne riječi: duboko učenje, računalni vid, konvolucijske neuronske mreže, detekcija objekata

Summary

Title: System for hand detection and tracking

This work gives a brief introduction into machine learning, especially artificial neural networks. It represents deep learning and some deep models used to develop the hand detection system. It describes the process of collecting and preparing the dataset used for model training. The procedure of deep model training using the Tensorflow Object Detection application programming interface is presented, and the results of each model are presented.

Key words: deep learning, computer vision, convolutional neural networks, object detection

Skraćenice

MSE	*Mean Square Error*	srednja kvadratna pogreška
ANN	*Artificial Neural Network*	umjetna neuronska mreža
CNN	*Convolutional neural networks*	konvolucijska neuronska mreža
R-CNN	*Region based CNN*	CNN bazirana na regijama