UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET
KATEDRA ZA RAČUNARSTVO

DUBOKO UČENJE

Prepoznavanje gestova ruku uz pomoć konvolucione neuronske mreže u PyTorch-u

Studenti:
Maša Radenković, 1264
Jelena Tošić, 1116

Mentor:

dr Aleksandar Milosavljević

Niš, 2021. god.

Sadržaj

1. Uvod i definicija problema

Prepoznavanje getova ruku od izuzetne je važnosti, što se tiče naprednih korisničkih interfejsa, a pogotovu u oblasti pristupačnosti digitalnih uređaja osobama sa invaliditetom [1]. Kao reprezentativan primer u ovoj oblasti imamo prepoznavanje gestova ruku koji predstavljaju simbole znakovnog jezika, što gluvonemim osobama daje mogućnost lakše komunikacije sa onima koji ne poznaju zankovni jezik [2] i olakšava njihov svakodnevni život. Još jedna široko zastupljena primena je daljinska kontrola ugrađenih i mobilnih uređaja gestovima [3], poput okidanja kamere pametnog telefona kada se detektuje pesnica ili raširena šaka i slično. U skorije vreme, prepoznavanje gestova ruku dobija sve više primena što se tiče interakcije sa objektima u virtuelnim okruženjima i proširene stvarnosti [4].

Tradicionalno, za ovu svrhu su inicijalno korišćeni algoritmi iz oblasti obrade slike i računarskog vida, koji se zasnivaju na primeni geometrije [5] ili poklapanje maski [6]. Međutim, osim toga što su neki od njih vrlo zahtevni za procesiranje u realnom vremenu, problem su i niske performanse klasifikacije. Što se javno dostupnih otvorenih kodova koji koriste OpenCV i metode računarskog vide tiče, na većini probanih primera tačnost je bila ne više od 70% [5, 6]. Za eksperimentalne primere i prototipe ovo mogu biti prihvatljive performanse, ali ne i za kritične primene.

Međutim, popularnost i progres dubokih neuronskih i konvolucionih mreža u poslednjoj deceniji značajno je uticao na performanse algoritama za klasifikaciju slika [7], što se može primeniti i na gestove ruku [8]. Duboke konvolucione mreže izdvajaju svosjtva iz slika na način koji je inspirisan radom ljudskog mozga i daje izuzetne rezultate, ukoliko je skup za obuku neuronske mreže dovoljno veliki [7]. Prema tome, nedostatak ovog pristupa jeste potreba za velikim brojem uzoraka prilikom nadgledane obuke, čije kreiranje može zahtevati velike napore. Osim slikanja adekvatnih uzoraka, neophodno je i ručno označiti slike, što traži zahteva dodatno vreme. Međutim, kao glavna prednost jeste robusnost i visoke performanse koje se mogu ostvariti [7].

U ovom radu je prikazana primena dubokih konvolucionih mreža za klasifikaciju 10 vrsta gestova na osnovu slika sa kamere. Korišćen je javno dostupan skup podataka [9] konstruisan uz pomoć Leap Motion uređaja, a PyTorch [10] u programskom jeziku Python za implementaciju. Na slici 1, dat je pregled gestova koji su obuhvaćeni klasifikacijom.

Slika 1 Korišćeni tipovo gestova ruku iz skupa podataka

Ovaj framework za duboko učenje u Pyton-u je sve popoularniji, pogotovu u naučnim krugovima, a njegove prednosti u odnosu na ostale, po kojima se izdvaja su [11]: bolje iskustvo prilikom debagiranja, čvršća integracija sa Python-om, drži se objektno-orijentisanog pristupa.

2. Teorijska osnova

2. 1. ReLU aktivaciona funkcija

ReLU aktivaciona funkcija je najčešći izbor kod skrivenih slojeva dubokih neuronskih mreža. Široku primenu stekla je, sa jedne strane, zbog jednostavnosti implementacije, a sa drugi strane, zato što prevazilazi ograničenja ostalih popularnih izbora u ovoj oblasti – sigmoida i hiperboličkog tangensa. Manje je podložna nestajućim gradijentima[1], koji sprečavaju modele dubokog učenja da se efektivno obuče za željeni zadatak, ali ipak ne prevazilaze neke druge probleme, poput prezasićenih[2] ili "mrtvih"[3] jedinica.

ReLU funkcija se definiše na sledeći način:

$$ReLU(x) = \max(0.0, x) \qquad (2)$$

Ovo znači da u slučaju kada je ulazna promenljiva x negativna, onda je izlaz ove funkcije 0, dok se prosleđuje x, u suprotnom.

2. 2. Softmax aktivaciona funkcija

Softmax aktivaciona funkcija daje kao izlaz vektor vrednosti, čiji je ukupan zbir 1.0, a one se mogu interpretirati kao verovatnoće pripadanja nekoj klasi. Srodna je argmax funkciji koja daje 0 za sve vrednosti, osim za odabranu, kada vraća 1. Softmax je manje striktna verzija argmax funkcije, koja daje izlaz u formi niza pozitivnih verovatnoća. Ulaz ove funkcije je vektor realnih vrednosti, dok je izlaz takođe vektor, iste dužine, sa vrednostima koje u zbiru daju 1.0 poput verovatnoća.

Softmax se računa na sledeći način (*N* je ukupan broj elemenata niza):

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}} \qquad (2)$$

U suštini, softmax predstavlja primenu eksponencijalne funkcije nad svakim elementom vektora, ali se vrši normalizacija dobijenih vrednosti deljenjem sumom svih tih eksponenata, što osigurava da je suma komponenti izlaznog vektora jednaka 1.

[1] Nestajući gradijent je problem koji se javlja kod metoda zasnovanih na gradijentnom spustu, kada vrednosti gradijenta postanu toliko male da se mreža više ne može obučavati

[2] Prezasićenost u neuronskim mrežama je pojava da neuroni daju izlaze koji su bliski asimptotama odgovarajućih aktivacionih funkcija, što dovodi do sporog ažuriranja težina, jer je gradijent izuzetno mali

[3] Mrtve jedinice su one komponente u neuronskoj mreži koje se ne ažuriraju tokom obuke

2. 3. Konvolucioni sloj neuronske mreže

Konovolucioni sloj uči mapu svojstava (feature map) na osnovu sirovih piksela slike. Obično je prvi ovakav sloj odgovoran za učenje svojstava niskog nivoa, poput ivica, boja orijentacije i slično, dok ostali slojevi izvlače svojstva visokog nivoa. Na taj način, duboke neuronske mreže sa više konvolucionih slojeva imaju sposobnost da razumeju sadržaj slika. Ovaj sloj obezbeđuje prostorne relacije između piksela učenjem svojstava slike korišćenjem malih prozora ulaznih podataka.

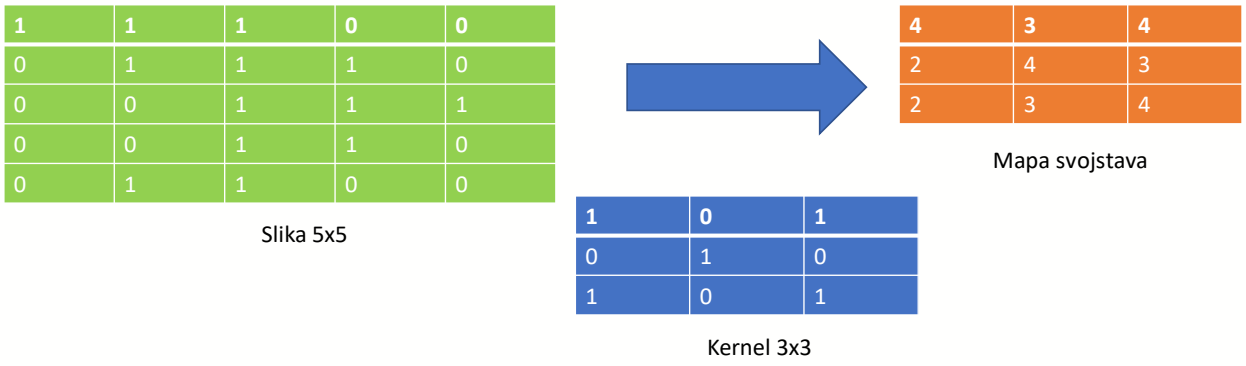
Veličinu mape svojstava određuju sledeći parametri:

1) dubina (depth) – broj filtara koji se koriste za operaciju konvolucije
2) korak (stride) – broj piksela za koliko se matrica filtra pomera kroz ulaznu matricu
3) dopuna (padding) – Popunjavanje ulazne matrice nulama oko granica
4) veličina filtra (kernel) – Dimenzije konvolucionog filtra koji se koristi

Formula za računanje dimenzija mape svojstava veličine (*input* x *input*) koja se dobija primenom konovolucionog filtra zadatih dimenzija (kernel), dopunom (padding) i korakom (stride), data je kao:

$$\frac{(input - kernel + 2 \cdot padding)}{stride} + 1 = input^{*} \qquad (3)$$

Pri tome, rezultat koji se dobija primenom konvolucionog filtra biće dimenzija ($input^{*}$ x $input^{*}$). Na sličan način, dimenzije se računaju i pri prolasku kroz pooling sloj. Na slici 2 je dat primer primene 3x3 konvolucionog filtra za dobijanje mape svojstava slike dimenzija 5x5.



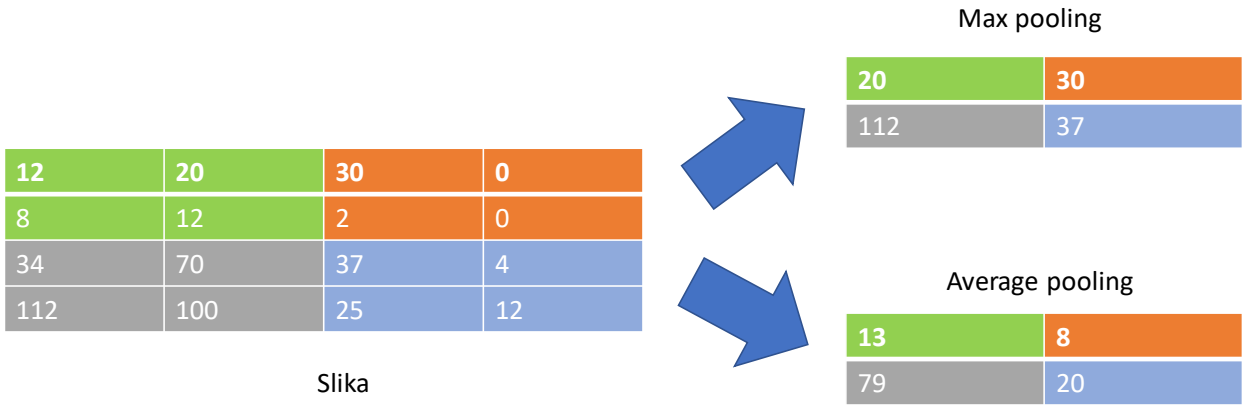
Slika 2 Primer dobijanja mape svojstava primenom konvolucionog filtra

2. 3. Pooling

Pooling, takođe poznat i kao subsampling ili downsampling. Ima ulogu da izvrši operaciju downsampling po prostornim dimenzijama (širina, visina), što smanjuje broj dimenzija mape svojstava, ali teži da sačuva najbitnije informacije. Na ovaj način se smanjuje i broj potrebnih izračunavanja u narednim slojevima, što dovodi do boljih performansi, pogotovu za slike većih dimenzija.

Najčešće su korišćena dva tipa pooling-a: 1) max pooling – vraća maksimalnu vrednost iz oblasti koja je pokrivena kernel-om 2) average pooling – vraća srednju vrednost. Jedna od prednosti max pooling-a jeste to što osim redukcije dimenzija feature mape, on vrši i uklanjanje šumova.

Pooling se obično primenjuje nakon ReLU sloja. Na slici 3, dat je primer primene ova dva različita tipa pooling-a nad istom slikom, sa kernelom dimenzija 2x2 i stride-om 2. Kao što se može videti, za ove vrednosti parametra se obe dimenzije ulazne slike prepolove, tako da od slike 4x4 dobijamo izlaz 2x2.



Slika 3 Ilustracija primene max i average pooling kernel-a dimenzija 2x2 i stride-om 2 nad ulazom dimenzija 4x4

2. 4. PyTorch

PyTorch[4] [8] je framework otvorenog koda, namenjen mašinskom i dubokom učenju, prvenstveno u oblasti računarskog vida i obradi govornog jezika. Nastala je 2016. godine kao rezultat rada istraživačke laboratorije za veštačku inteligenciju kompanije Facebook. Glavni fokus je na implementaciji za Python programski jezik, iako postoji C++ interfejs ka njegovim funkcionalnostima. U poslednje vreme stiče ogromnu popularnost, pa se na njega oslanjaju čak i neka velika komercijalna rešenja, poput Tesla Autopilota za autonomnu vožnju automobila.

PyTorch obuhvata sledeća dva aspekta: 1) rad sa tenzorskim podacima (poput NumPy), što omogućava veliko ubrzanje prilikom izvršenja na grafičkom procesoru (konkretno, NVIDIA CUDA) 2) skup metoda i klasa za rad sa neuronskim mrežama i duboko učenje, koji se oslanjaju na sistem za automatizovano diferenciranje 3) pojedini aspekti rada sa skupovima podataka.

Za razvoj korisničkih modula neuronskih mreža, neophodno je naslediti Module i implementirati dve njegove funkcije:

- `__init__` : konstruktor u okviru koga definišemo sve podkomponente (slojeve) neuronske mreže i opciono, oslanjajući se na postojeće module u PyTorch-u za različite tipove slojeva – linearni, konvolucioni, pooling i ostalo. Pre svega, na početku ovog konstruktura je potrebno pozovati konstruktor nadklase `super().__init__()`

[4] https://pytorch.org/

- `forward`: ova funkcija definiše šta se dešava prilikom faze prosleđivanja unapred, posebno se skreće pažnja na uspostavljanje veza između slojeva. To je zapravo funkcija gde pozivamo metode slojeva u željenjom redosledu.

U Listingu 1, dat je primer Python koda za definisanje korisničkog modula uz pomoć klase Module jednostovane neuronske mreže koja ima jedan linearni sloj.

```
class MojaDubokaMreza(torch.nn.Module):
    def __init__(self, broj_ulaza, broj_klasa):

        # konstruktor nadklase
        super(MojModul1, self).__init__()

        # definisanje podmodula
        self.linear = torch.nn.Linear(broj_ulaza, broj_klasa)

    def forward(self, x):
        # povezivanje slojeva i prosleđivanje rezultata
        izlaz = self.linear(x)
        return izlaz
```

Listing 1 Primer neuronske mreže sa jednim slojem definisane nasleđivanjem PyTorch klase Module

Postupak obuke modela neuronske mreže uz pomoć PyTorch framework-a se implementira ručno kroz petlju, koja ima sledeće korake:

1. Iteracija kroz podatke (u serijama)
2. Vršenje predikcije (*forward pass* – prosleđivanje unapred kroz slojeve)
3. Čišćenje prethodno izračunatih gradijenata
4. Izračunati gubitke (poziv odabrane funkcije greške)
5. Novo računanje gradijenata (*backward pass* – prosleđivanje unazad)
6. Ažuriranje parametara (uz pomoć optimizatora)

U Listingu 2 je data generalizacija petlje za obuku mreže upotrebom PyTorch-a.

```
for epoha in range(broj_epoha):
     for uzorci, oznake in loader:
          uzorci=uzorci.to(device)
          oznake=oznake.to(device)
          predikcije=model(uzorci)
          gubici=f_greske(predikcije, oznake)
          gubici.backward()
          optimizer.step()
          model.zero_grad()
```

Listing 2 Šablon petlje za obuku neuronske mreže u PyTorch-u

Što se rada sa skupovima podataka u PyTorch framework-u tiče, koriste se dve klase: Dataset i Dataloader.

Dataset predstavlja PyTorch klasu za predstavljanje skupa podataka, dok se pojedinačni uzorci iz skupa mogu pribaviti na osnovu indeksa. Neophodno je implementirati dve funkcije.

- `__len__`: vraća dužinu skupa podataka
- `__getitem__`: vraća pojedinačan uzorak iz skupa podataka

Listing 3 daje primer implementacije korisničke klase za rad sa skupom podataka.

```
class SkupPodataka(Dataset):

    def __init__(self):
        #Inicijalizacija podataka - preuzimanje i učitavanje fajlova
        #Koristimo numpy ili pandas
        xy = np.loadtxt('podaci.csv', delimiter=',', dtype=np.float32, skiprows=1)
        self.br_uzoraka = xy.shape[0]

        # Prva kolona je oznaka, ostalo su ulazne promenljive
        self.x_data = torch.from_numpy(xy[:, 1:]) # veličina [br_uzoraka, br_svojstava]
        self.y_data = torch.from_numpy(xy[:, [0]]) # size [br_uzoraka, 1]

    # dataset[i] vraća i-ti uzorak
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # len(dataset) vraća dužinu skupa podataka
    def __len__(self):
        return self.br_uzoraka
```

Listing 3 Primer definicije korisničkg skupa podataka nasleđivanjem Dataset-a u PyTorch-u

Sa druge strane, imamo klasu Dataloader, koja služi za iteriranje kroz skup podataka. Ona pruža mogušćnost višeprocesorskog učitavanja podataka, automatsko deljenje skupa podataka u serije, mešanje podataka i još dosta toga. U Listingu 4, dat je isečak koda koji pokazuje primenu ove klase za deljenje skupa podataka predstavljenog objektom klase Dataset u serije od po 10 uzoraka, mešanje, ali i obilazak uzoraka korišćenjem Dataloader-a kao iteratora.

```
from torch.utils.data import DataLoader

loader = DataLoader(dataset, batch_size=10, shuffle=True)
for uzorak, oznaka in loader:
    print(uzorak, oznaka, sep="\n")
    break
```

Listing 4 Isečak koda koji ilustruje primenu DataLoader klase

U Tabeli 2 je dat pregled ključnih klasa i funkcija PyTorch-a, relevantnih za konvolucione neuronske mreže koje se koriste u ovom radu sa njihovim parametrima.

Tabela 2 Pregled relevantnih klasa za rad sa konvolucionim mrežama u PyTorch-u

Klasa/funkcija	Objašnjenje
`torch.nn.Conv2d(`**`in_channels`**`= broj`	Klasa kojom se definiše 2D konvolucioni sloj sa

kanala na ulazu, **out_channels** = broj kanala na izlazu, **kernel_size** = veličina konvolucionog filtra, **stride** = razmak, **padding_mode [opciono]**= vrsta dopune sa strane – nule, cirkularno...)	svojim karakteristikama. U ovom radu se radi sa slikama pa je broj dimenzija 2, a recimo, što se vremenskih serija tiče, koriste se 1D.
torch.nn.Dropout2d(**p**=verovatnoća odbacivanja)	Sloj koji nasumično anulira ceo kanal (kanal je 2D mapa svojstava). Anuliranje kanala se vrši nezavisno jedan od drugog, na svakom prostiranju unapred kroz mrežu, sa verovatnoćom p korišćenjem uzoraka Bernulijeve distribucije. Obično ulaz u ovaj sloj dolazi iz Conv2d modula. Služi da poveća nezavisnost mapa svojstava, što dovodi do boljih rezultata, pa je preporučljivo da se uvek koristi.
nn.BatchNorm2d(**num_features**=broj svojstava)	Normalizacija se koristi da sva ulazna svojstva svedemo na istu skalu. Ovaj tip normalizacije radi nad batch-om, korišćenjem sredine (μ) i standardne devijacije (σ). $h_{ij}^{norm} = \frac{h_{ij} - \mu_j}{\sigma_j}$ Da bi se ispoljila moć skrivenih slojeva, dodaju se dva parametra u normalizaciji – gama i beta. Nakon normalizacije, vrši se još jedan korak. $h_{ij}^{final} = \gamma_j h_{ij}^{norm} + \beta_j$ Parametri gama i beta se uče zajedno sa drugim parametrima mreže. Ako je gama jednaka sredini a beta standardnoj devijaciji, onda je izlaz jednak h_norm.
*torch.nn.MaxPool2d(**kernel_size**=veličina filtra, **stride**=veličina razmaka)*	Modul za 2D max pooling. Od parametara, relevatni su veličina filtra i razmak.
torch.nn.Linear(**in_features**=broj ulaznih svojstava, **out_features**= broj izlaznih svojstava)	Modul za primenu linearne transformacije nad ulaznim podacima sa in_features svojstava, pri čemu se generiše out_features svojstava na izlazu.
torch.nn.ReLU(**inplace**=False)	Modul za definiciju sloja koji primenjuje ReLU aktivaciju nad ulazom. Operaciju je moguće opciono izvesti inplace, što zavisi od istoimenog parametra – da li je True ili False.

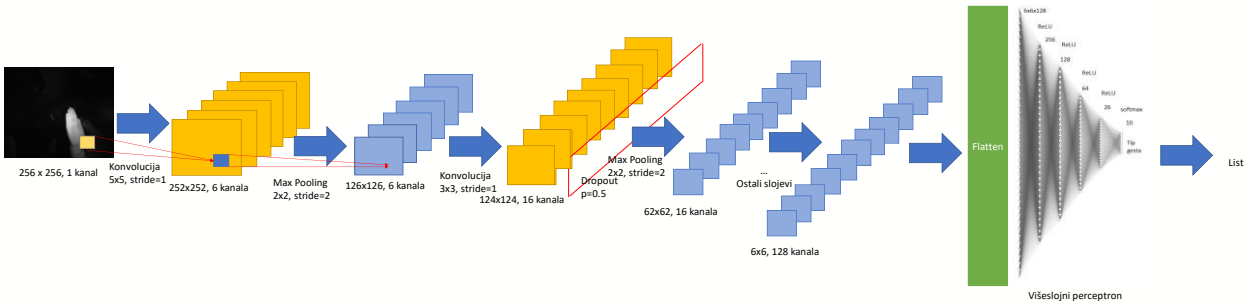
3. Arhitektura konvolucione mreže za klasifikaciju gestova ruku

Što se arhitekture mreže tiče, imamo ukupno 5 konvolucionih skrivenih slojeva, pri čemu svaki od njih prate batch normalizacija, ReLU sloj i max pooling. Osim toga, svi osim prvog skrivenog sloja pre

normalizacije imaju i dropout sloj, koji teži da promoviše nezavisnost mapa svojstava, tako što nasumično anulira neki od kanala sa datom verovatnoćom. Nakon konvolucionih slojeva, slika prolazi kroz multilayer perceptron (MLP) koji ima 4 skrivena sloja. Prilikom pripreme, slike se prvo smanjuju na dimenzije 256 x 256, da bi se izbeglo procesiranje velikog broja piksela, a time i povećala brzina treninga. Tabela 1 daje rezime arhitekture neuronske mreže, dok je njena islutracija prikazana na slici 4.

Tabela 1 Pregled arhitekture mreže

Tip sloja	Kernel	Stride	Padding	Ulaz	Izlaz	Broj filtera	Aktivacija
Konvolucioni 1	5x5	1	0	256x256	252x252	6	ReLU
Max pooling 1	2x2	2	0	252x252	126x126	6	
Konvolucioni 2	3x3	1	0	126x126	124x124	16	
Droput, p=0.5							
Max pooling 2	2x2	2	0	124x124	62x62	16	
Konvolucioni 3	3x3	1	0	62x62	60x60	32	
Dropout, p=0.7							
Max pooling 3	2x2	2	0	60x60	30x30	32	
Konvolucioni 4	3x3	1	0	30x30	28x28	64	
Dropout, p=0.6							
Max pooling 4	2x2	2	0	28x28	14x14	64	
Konvolucioni 5	3x3	1	0	14x14	12x12	128	
Dropout, p=0.5							
Max pooling 5	2x2	2	0	12x12	6x6	128	
Fully connected 1	-	-	-	6x6x128	256	-	
Fully connected 2	-	-	-	256	128	-	
Fully connected 3	-	-	-	128	64	-	
Fully connected 4	-	-	-	64	26	-	
Izlaz				26	10	-	Softmax



Slika 4 Ilustracija arhitekture konvolucione neuronske mreže za prepoznavanje gestova ruku

Prvi konvolucioni sloj: Konvolucioni sloj ima kernel dimenzija 5 x 5, korak 1, bez dopune, prima kao ulaz sliku dimenzija 256 x 256 sa jednim kanalom, a kao izlaz daje 6 mapa svojstava, pri čemu je njihova dimenzija izračunata po formuli (3):

$$(256-5+2*0)/1+1=251+1=252$$

Prema tome, svaka mapa svojstava imaće dimenzije 252 x 252.

Nakon toga, radi se batch normalizacija za 6 mapa svojstava, a zatim primenjuje ReLU aktivaciona funkcija. Na kraju prvog skrivenog sloja se primenjuje pooling, i to max pooling, sa kernel-om dimenzija 2x2 i korakom 2, što rezultuje downsampling-om slike po dužini i širini, tako da slika dimenzija n x n postaje (n/2) x (n/2). Na ovaj način se čuvaju samo najbitnija svojstva iz mape, smanjuju njene dimenzije, ali i vreme potrebno za obradu. Prema tome, kada za dimenzije max pooling sloja i stride primenimo formulu (3), dobijamo da se dobija izlaz dimenzija (252/2) x (252/2), što je 126 x 126.

Drugi konvolucioni sloj: Konvolucioni sloj ima kernel dimenzija 3 x 3, korak 1, prima kao ulaz feature mapu dimenzija 126x126 sa 6 kanala, pri čemu na izlazu generiše 16 mapa svojstava, a svaka od njih ima sledeće dimenzije, računato po formuli (3):

$$(126-3+2*0)/1+1=124$$

Nakon toga, za razliku od prethodnog sloja, primenjujemo droput sloj sa verovatnoćom 0.5. Dalje, slično kao i za prethodni sloj, primenjuje se batch normalizacija za 16 mapa svojstava, a zatim ReLU aktivaciona funkcija. Posle toga se primenjuje max pooling, sa istim karakteristikama kao i u prethodnom skrivenom sloju (ali i svim ostalim), tako da se dobija izlaz dimenzija (124/2) x (124/2), što je 62 x 62.

Treći konvolucioni sloj: Konvolucioni sloj sa kernel-om dimenzija 3 x 3, korak 1, bez padding-a, prima kao ulaz mapu svojstava dimenzija 62x62 sa 16 kanala, na izlazu generiše 32 mape svojstava, pri čemu svaka od njih ima dimenzije:

$$(62-3+2*0)/1+1=60$$

Dalje, imamo dropout sa verovatnoćom 0.7, a zatim i batch normalizaciju za 32 mape svojstava, pa ReLU aktivacionu funkciju, kao i u prethodnom sloju. Nakon toga, ide max pooling sloj na isti način kao i za sve slojeve, tako da je izlazna mapa svojstava nakon toga dimenzija 30 x 30.

Četvrti konvolucioni sloj: Isto kao i u prethodna dva sloja – konvoliconi kernel dimenzija 3 x 3, korak 1, bez dopune, a na ulazu dolazi feature mapa 30x30 sa 32 kanala. Pri tome, na izlazu generiše 64 mapa svosjtava sa sledećim dimenzijama:

$$(30-3+2*0)/1+1=28$$

Posle toga ide dorput verovatnoće 0.6, pa onda batch normalizacija i ReLU. Na izlazu iz skrivenog sloja imamo takođe max pooling, tako da je izlaz dimenzija 14 x 14.

Peti konvolucioni sloj: Slično kao i u prethodnim slojevima - 3 x 3 kernel , korak 1, bez dopune, a na ulazu feature mapa 14x14 sa 64 kanala. Konvolcioni sloj na izlazu generiše 128 mapa svosjtava sa dimenzijama:

$$(14-3+2*0)/1+1=12$$

Zatim, primenjuju se dropout sa verovatnoćom 0.5 i batch normalizacija, pa ReLU. Konačno, na izlazu se primenjuje identičan max pooling, tako da su dimenzije svake od dobijenih mapa svojstava 6 x 6, pri čemu ih ima 128.

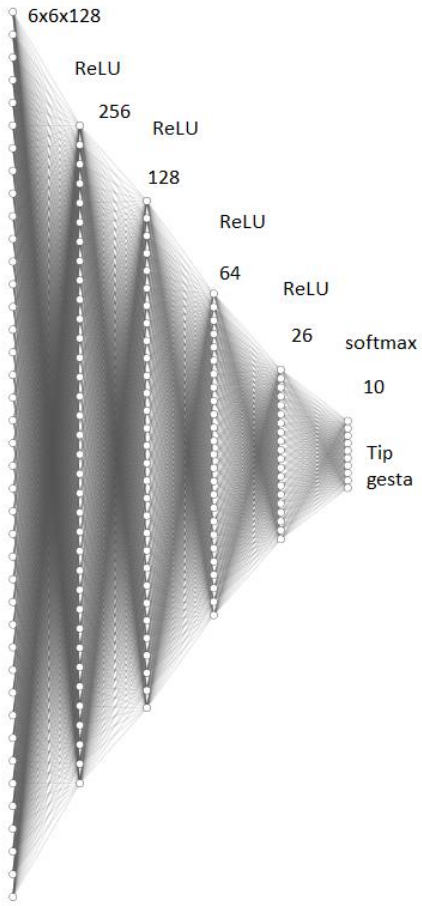
U Listingu 5 je prikazan deo konstruktora klase naše konovlucioneneuronske mreže kojim se definišu konovlucioni i prateći slojevi.

```
            nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5, stride = 1),
            nn.BatchNorm2d(6),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 3, stride = 1),
            nn.Dropout2d(p=0.5),
            nn.BatchNorm2d(16),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels = 16, out_channels = 32, kernel_size = 3, stride = 1),
            nn.Dropout2d(p=0.7),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 3, stride = 1),
            nn.Dropout2d(p=0.6),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),

            nn.Conv2d(in_channels = 64, out_channels = 128, kernel_size = 3, stride = 1),
            nn.Dropout2d(p=0.5),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
```

Listing 4 Isečak PyTorch koda za definiciju konvolucionih i pratećih slojeva

Višeslojni perceptron (MLP): Nakon prolaska kroz zadnji konvolucioni sloj, dobijamo adekvatnu formu slike u vidu 128 mapa svojstava dimenzija 6x6, koje se mogu dalje klasifikovati upotrebom višeslojne perceptronske mreže. Pre toga, potrebno je izravnati (operacija flatten) ove informacije u okviru jednog kolona-vektora. Zatim, ovako dobijeni kolona-vektor se dovodi na ulaz feed-forward neuronske mreže, koja vrši propagaciju unazad (backpropagation) u svakoj iteraciji treninga. Nakon serije epoha, model postaje sposoban da raspozna dominantna i pojedina svojstva niskog-nivoa, što se koristi za klasifikaciju slika upotrebom softmax-a. Koristimo ovu metodu, jer problem tretiramo kao višeklasno (isključivo, kategoričko) razvrstavanje slika, a korišćena je funkcija gubitaka unakrsne entropije (cross entropy).

Ulazni sloj ima ukupno 6x6x128 čvorova. Zatim, imamo 4 skrivena sloja sa ReLU aktivacionom funkcijom, pri čemu je broj čvorova redom - 256, 128, 64 i 26. Konačno, izlazni sloj ima onoliko čvorova koliko imamo različitih klasa slika, a to je 10 u našem slučaju. Ova sloj ima softmax aktivacionu funkciju.Konačno, izlaz iz ovog sloja pripadnost slike jednoj od 10 klasa gestova ruku. Na slici 5 je prikazana ilustracija mreže za klasifikaciju.



Slika 5 Potpuno povezani slojevi – višeslojna perceptronska mreža sa klasifikaciju

U Listingu 2 je dat PyTorch kod kojim je definisan fully connected deo duboke neuronske mreže za klasifikaciju gestova. Međutim, kao što se može videti, softmax se ne koristi eksplicitno prilikom definicije slojeva. Razlog za to jeste činjenica da torch.nn.CrossEntropyLoss() u PyTorch-u kombinuje nn.LogSoftmax() i nn.NLLLoss(), što znači da je softmax već implicitno sadržan i nema potrebe da se ponovo poziva [ref12].

```
            nn.Flatten(),
            nn.Linear(6*6*128, 256),
            nn.Dropout2d(p=0.4),
            nn.ReLU(inplace=True),
            nn.Linear(256, 128),
            nn.Dropout2d(p=0.3),
            nn.ReLU(inplace=True),
            nn.Linear(128, 64),
            nn.ReLU(inplace=True),
            nn.Linear(64, 26),
            nn.ReLU(inplace=True),
            nn.Linear(26, 10)
```

Listing 2 Isečak PyTorch koda kojim je implemntiran multilayer percpetron (MLP) deo duboke mreže

4. Opis eksperimenta

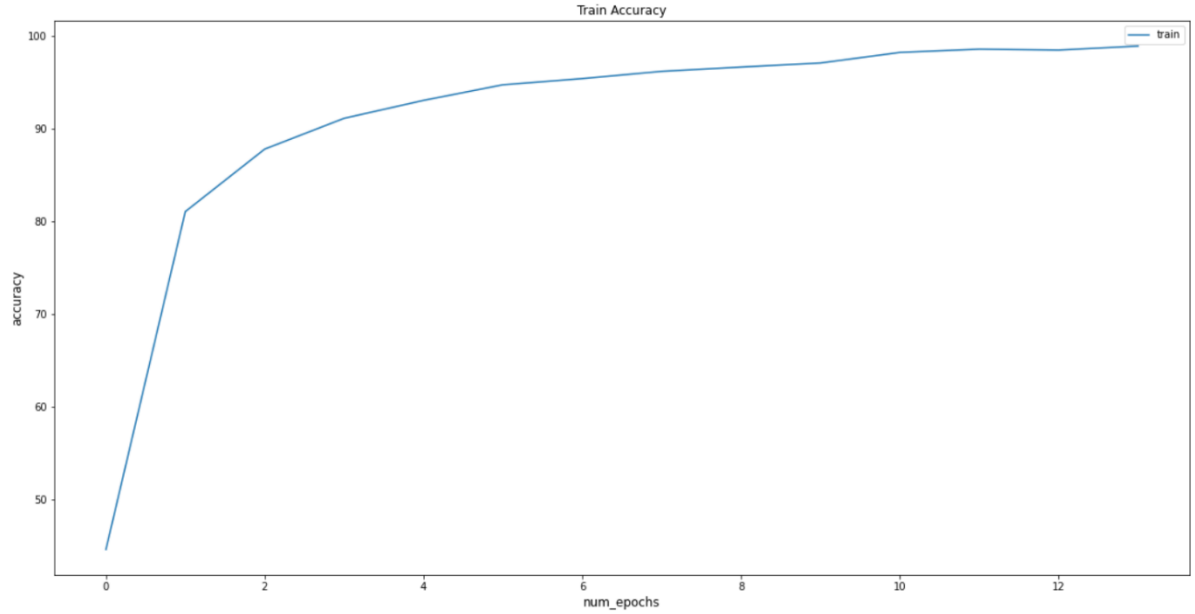
U ovom poglavlju je opisan proces obuke mreže, ali i njena evaluacija. Inicijalni skup podataka je imao 20 000 slika koje obuhvataju 10 različitih gestova. Ceo skup podataka smo podelili u trening i test skup, pri čemu je trening skup 80% svih slika (16 000), dok je evaluacija vršena nad test skupom od 4 000 slika (20% celog skupa).

Što se obuke tiče, veličina serija (batch_size) je bila 128 uzoraka. Dalje, uzet je parametar stope učenja α=0.001, a obuka je vršena u 14 epoha. Kao funkcija greške je korišćena unakrsna entropija (torch.nn.CrossEntropyLoss), a Adam optimizer. Mera perfromansi koju smo koristili je accuracy (tačnost), koja je definisana kao procenat tačno klasifikovanih uzoraka u odnosu na ukupan broj:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

TP – true positive
TN – true negative
FP - false positive
FN – false negative

Na test skupu je ostvaren accuracy od 98.9%, a na test setu oko 97.4%. Slika 6 prikazuje kako se accuracy povećava kroz epohe tokom obuke. I trening i evaluacija su izvršavani u Google Collaboratory[5] okruženju za Python, oslanjajući se na GPU. Razlog jeste to što ne zahteva trošenje dodatnog vremena oko konfiguracije i instalacije, može se pokrenuti online na bilo kom računaru, oslanja se na resurse u oblaku, a nudi besplatno i jak grafički hardver za upotrebu, što ubrzava procesiranje.Trening je trajao 9 minuta i 12 sekundi (preciznije - 552.226 sekundi), dok je za test bilo potrebno 10.319 sekundi.



Slika 6 Rast accuracy tokom treninga kroz epohe

[5] https://research.google.com/colaboratory/

```
net = ConvNet().to(device)
num_epochs = 14
learning_rate = 0.001
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=10, gamma=0.1)

train_loss = []
train_acc = []
total_step = len(train_loader)
net.train()
for epoch in range(1, num_epochs+1):

    running_loss = 0.0
    correct = 0
    total=0
    print(f'Epoch {epoch}\n')

    for i, batch in enumerate(train_loader): #gives us data and index

        # we have to reshape this samples of images
        data_ = batch["image"].to(device)
        target_ = batch["label"].to(device)
        optimizer.zero_grad()

        # TRAIN
        outputs = net(data_)
        loss = criterion(outputs, target_)
        loss.backward()
        optimizer.step()

        # PRINT LOSS
        running_loss += loss.item()
        _,pred = torch.max(outputs, dim=1)
        correct += torch.sum(pred==target_).item()
        total += target_.size(0)
        if (i) % 20 == 0:
            print(f'Epoch [{epoch}/{num_epochs}], Step [{i}/{total_step}], Loss:
{loss.item():.4f}')

    scheduler.step()
    train_acc.append(100 * correct / total)
    train_loss.append(running_loss/total_step)
    print(f'\ntrain-loss: {np.mean(train_loss):.4f}, train-acc: {(100 * correct/total):.4f}')
```

Listing 5 Trening petlja konvolucione mreže za prepoznavanje gestova

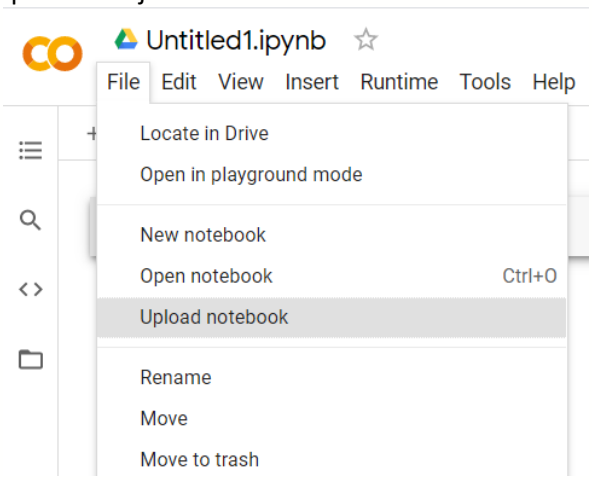
5. Zaključak

U ovom radu je prikazana primena duboke konvolucione mreže u oblasti prepoznavanja gestova ruku. Na osnovu dobijenih rezultata, može se zaključiti da ovakav pristup ima izuzetne performanse – čak oko 97.4% na test setu.

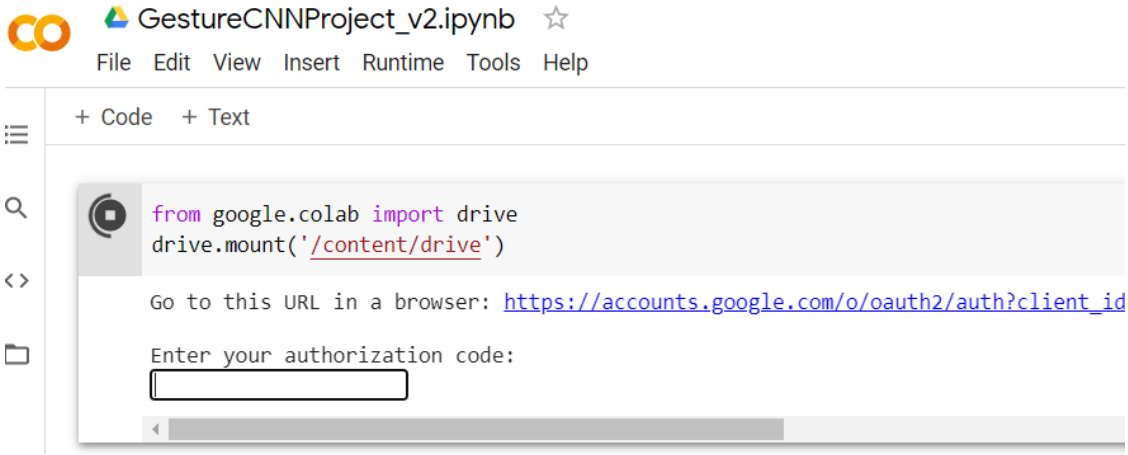
U budućnosti, planira se deployment modela uz pomoć Flask-a, što omogućava pristup funkcionalnostima za predikcije preko REST API servisa. Na taj način, omogućena je njegova integracija sa drugim aplikacijama. Primer upotrebe bi bila video igra u proširenoh realnosti gde se interaguje sa objektima ili 3D junacima upravlja gestovima. Osim toga, kontrola muzičkih instrumenata u proširenoj realnosti je još jedan mogući, inovativni scenarijo primene.

Dodatak – upustvo za pokretanje

1. Preuzeti arhivu sa skupom podata archive.zip, sa sledeće adrese: https://drive.google.com/file/d/1SLUpSyKp01i_4RqAohRVH3NJznJC-3dU/view?usp=sharing
2. Upload-ovati ovu arhivu na Google Drive sopstvenog Google naloga unutar foldera data2 i to na sledećoj putanji: MyDrive/data2/archive.zip
3. Skinuti Python Jupyter skriptu GestureCNNProject_v2.ipynb sa adrese: https://drive.google.com/file/d/14TOQIzlaxuuAlzRRm7BBXjMkVLdZ6kPW/view?usp=sharing
4. Otvoriti Google Collaboratory okruženje https://colab.research.google.com/
5. Klik na opciju New Notebook
6. Postaviti prethodno preuzeti fajl iz koraka 3.



7. Skripta je sada otvorena i može se pokrenuti. Kliknite na play za prvi korak. Prikazaće se konzolna poruka da je neophodno izvršiti autentikacije, jer se traži pristup vašem Google Drive-u. Prema tome, neophdono je kliknuti na link, a zatim prekopirati kod za auotrizaciju. Tek nakon unosa koda za autorizaciju u odgovarajuće polje je moguće pristupiti iz skripte Vašem disku.



Reference

[1] R. Ma, Z. Zhang and E. Chen, "Cognitive Computing Solutions for Complexity Problems in Computational Social Systems" Complexity 2021(1):1-11, 2021. https://doi.org/10.1155/2021/6679746
[2] B. Fei et al., “Gesture recognition for sign language Video Stream Translation”, 2020 5th International Conference on Mechanical, Control and Computer Engineering, pp. 1315-1319, 2020. https://doi.org/10.1109/ICMCCE51767.2020.00288
[3] S. J. Horng, X. Z. Hu, W. J. Cao, "Gesture Recognition Applications Developed in Embedded Systems", 2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP), pp. 157-162, 2018. https://doi.org/10.1109/PAAP.2018.00035
[4] S. Rani, K. J. Dhrisya, M. Ahalydas, "Hand gesture control of virtual object in augmented reality", 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1500-1505, 2017. https://doi.org/10.1109/ICACCI.2017.8126053
[5] Hand Gesture Recognition using Python and OpenCV - Part 2 [online], dostupno na: https://gogul.dev/software/hand-gesture-recognition-p2 , poslednji pristup: 01/07/2021.
[6] Hand Gesture Recognition [online], odstupno na: https://github.com/adilsofficial/HandGesture , poslednji pristup: 01/07/2021.
[7] Y. Bengio, “Learning Deep Architectures for AI”, *Foundations and Trends in Machine Learning*, Vol. 2, No. 1, November 2009, pp. 1-127.
[8] E. Stevens, L. Antiga, T. Viehmann, *Deep Learning with PyTorch*, Manning Publications, 2020.
[9] T. Mantecón, C.R. del Blanco, F. Jaureguizar, N. García, “Hand Gesture Recognition using Infrared Imagery Provided by Leap Motion Controller”, Int. Conf. on Advanced Concepts for Intelligent Vision Systems, ACIVS 2016, Lecce, Italy, pp. 47-57, 24-27 Oct. 2016. https://doi.org/10.1007/978-3-319-48680-2_5
[10] Hand Gesture Recognition Dataset [online], dostupno na: https://www.kaggle.com/gti-upm/leapgestrecog , poslednji pristup: 01/07/2021.
[11] PyTorch vs. TensorFlow: An Overview [online] , dostupno na: https://phoenixnap.com/blog/pytorch-vs-tensorflow , poslednji pristup: 23/04/2021.
[12] Should I use softmax as output when using cross entropy loss in pytorch? [online], dostupno na : https://stackoverflow.com/questions/55675345/should-i-use-softmax-as-output-when-using-cross-entropy-loss-in-pytorch , poslednji pristup: 01/07/2021.