



Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Tedensko poročilo

pri predmetu Umetna inteligenca

Člani skupine:

Nik Terglav

Luka Lamprečnik

Simona Zhirova

Marija Jovanova

Teden 1

1. Tabela zadolžitve

Član	Zadolžitve
Simona Zhirova	Model 1 -Raziskava RNN (LSTM) arhitekture - Priprava in organizacija podatkov (koordinate gest iz Mediapipe) - Razdelitev v učne/testne/validacijske množice
Marija Jovanova	Model 1 - Implementacija osnovne strukture LSTM modela (TensorFlow/Keras) - Dokumentacija, zakaj je LSTM ustrezen za ta tip podatkov
Luka Lamprečnik	Model 2 - Raziskava najbolj primerne arhitekture nevronske mreže za moj primer - Sprememba posnetkov za učenje - izbira knjižnice
Nik Terglav	Model 3 -Raziskava arhitektur nevronske mreže in odločitev, katera je primerna za moj model -Ustvariti učne, testne in validacijske množice - Seznaniti se z ogroženo/knjižnico, ki bom uporabil za moj model - Osnovna implementacija

2. Opis dela

V prvem sklopu smo se člani skupine seznanili z osnovnimi arhitekturami nevronske mreže in metodami strojnega učenja, ki jih bomo uporabili pri našem projektu. Preučili smo delovanje **konvolucijskih (CNN)**, **rekurentnih (LSTM)** in **gostih (Dense)** nevronske mreže, saj vsak od treh načrtovanih modelov zahteva nekoliko drugačen pristop. Modele smo porazdelili na naslednji način:

- **Model 1 – Gestna kontrola (Marija Jovanova in Simona Zhirova)**

Za prepoznavo gest rok bomo uporabili **rekurentno nevronske mreže LSTM**, saj želimo obdelovati zaporedje gibanja (časovno odvisne podatke). LSTM je posebej primeren za sledenje zaporedju koordinat, ki jih pridobimo s pomočjo Mediapipe knjižnice. Geste bodo razvrščene najprej po ciljni napravi (npr. radio, klima), nato pa po konkretni akciji (npr. vklop, izklop).

- **Model 2 – Detekcija drže voznika (Luka Lamprečnik)**

Za zaznavo telesne drže bomo uporabili kombinacijo **konvolucijske (CNN)** in **goste mreže (Dense)**. CNN bo iz slike zaznal ključne točke telesa, nato pa bomo s pomočjo Dense plasti razvrstili držo v različne kategorije (npr. pogled naprej, pogled v stran, sključena drža itd.).

- **Model 3 – Razvoj modela za analizo okolje-specifičnih podatkov (Nik Tergrav)**

Za zaznavo okolja bomo uporabili **preprosto nevronske mreže z CNN arhitekturo**, ki bo analizirala povzetke iz vhodnih podatkov, kot je npr. čas v dnevu. Pri nadgradnji modela pa želim razviti v hibridno arhitekturo in dodati še prepoznavo tipa vremena.

2.1. Simona Zhirova

Podatke sem organizirala v mape glede na napravo (npr. radio, ogledala, okna) in vrsto geste (npr. vklop, izklop), kar je omogočilo boljšo preglednost in enostavnejše razdeljevanje v učne, testne in validacijske množice. Pri razdeljevanju sem pazila na enakomerno zastopanost razredov in preprečila podvajanje vzorcev v različnih množicah.

Za potrebe prvega modela, namenjenega prepoznavanju gest rok, sem raziskala rekurentne nevronske mreže, s poudarkom na arhitekturi LSTM (Long Short-Term Memory). Ta vrsta mreže je posebej primerna za obdelavo zaporednih podatkov, saj omogoča učenje dolgoročnih odvisnosti v časovni vrsti. Pri preučevanju sem se osredotočila na razumevanje, kako LSTM celice shranjujejo in posodablajo informacije prek vhodnih, izhodnih in pozabnih vrat, ter kako to vpliva na prepoznavo gibanja rok skozi čas.

Za porazdelitev podatke v mape za učne, testne in validacijske množice najprej sem naložila podatke iz različnih map, ki vsebujejo .npy datoteke, predstavljajoče različne geste in ukaze. Za vsak niz podatkov sem izvedla potrebno obdelavo, vključno z usklajevanjem dimenzij (npr. z zapolnjevanjem, da so vsi podatki imeli enako obliko). Nato sem iz podatkov izluščila vhodne podatke in pripadajoče oznake (X in y). Vhodni podatki (X) so predstavljali koordinate, pridobljene iz Mediapipe, medtem ko so oznake (y) vsebovale razrede, povezane z vsakim vzorcem (npr. različne geste ali ukazi). Te podatke sem shranila v datoteke `X.npy` in `y.npy`, ki sem jih nato uporabila za nadaljnje korake.

Potem, sem podatke razdelila na učne, testne in validacijske množice z uporabo funkcionalnosti `train_test_split` iz knjižnice `scikit-learn`, kar je omogočilo uravnoteženo zastopnost različnih razredov v vseh množicah. Na koncu sem podatke organizirala v tri mape (`train`, `val`, `test`), kjer sem jih pripravila za nadaljnje usposabljanje modela in oceno njegovega delovanja.

2.2. Marija Jovanova

Pri zasnovi modela za prepoznavo gest rok smo se odločili za uporabo rekurentne nevronske mreže, natančneje arhitekture LSTM (Long Short-Term Memory). Geste rok so zaporedni pojav – sestavljene so iz niza premikov, ki se dogajajo skozi čas. Vsaka gesta predstavlja zaporedje koordinat, ki jih pridobimo s pomočjo knjižnice Mediapipe, in te koordinate moramo obdelati tako, da ohranim njihovo časovno povezanost. Izbrali smo LSTM, saj omogoča modelu, da "si zapomni" pomembne informacije iz prejšnjih časovnih korakov in jih uporabi pri interpretaciji trenutnega stanja. Za razliko od navadnih nevronskih mrež, ki vsako sliko ali podatkovno točko obravnavajo ločeno, LSTM upošteva zaporedje in tako bolje razume kontekst gesta. Ta sposobnost zaznavanja dolgoročnih odvisnosti je ključna za prepoznavo zapletenejših gest, kjer pomen izhaja iz celotnega gibanja, ne le posamezne pozicije. S to arhitekturo tako dosegamo boljše razumevanje dinamičnih vzorcev v podatkih in s tem tudi natančnejšo klasifikacijo uporabniških gest.

Za prvi preizkus delovanja sem implementirala osnovni LSTM model, ki služi kot začetna različica sistema za razpoznavanje gest na podlagi časovno zaporednih podatkov. Model sem zasnovala tako, da prejme vhodne podatke in oznake in skozi zaporedni arhitekturi LSTM plasti poskuša prepoznati razred geste, ki ji posamezen vzorec pripada. Model vsebuje dve LSTM plasti z 64 nevroni in aktivacijsko funkcijo ReLU, med katerima sta dodani Dropout plasti (z verjetnostjo 0.2), ki pomagata zmanjšati prenaučенost. Izhodna plast je gosto povezana (Dense) s številom nevronov, ki ustreza številu razredov, in uporablja softmax aktivacijsko funkcijo za napoved verjetnosti vsakega razreda. Ta osnovna implementacija je bila izvedena z namenom, da preverim ali je možno s podatki doseči smiselno klasifikacijo ter da ocenim, ali se lahko na tej osnovi model še nadgrajuje. Trenutni model torej predstavlja prvi poskus, ki mi bo služil kot izhodišče za nadaljnje izboljšave arhitekture in podatkovne obdelave.

2.3. Nik Terglav

Za moj model prepoznavanja okolja sem se odločil za uporabo arhitekture konvolucijske nevronske mreže CNN za prepoznavanje okoljskih podatkov. Ker so posnetki zaporedje slik, mi omogoča CNN ekstrakcijo prostorskih značilnosti iz posameznih okvirjev, to mi omogoča prepoznavo določenih vzorcev. Začetni model bo zgrajen na tej arhitekturi, namreč želim prepoznati ali je na posnetku dan ali noč, CNN mi omogoča obravnavo svetlobnih pogojev, značilnosti slike, kontrast... Ker želim omogočiti mojemu modelu možnost nadaljnjega razvoja sem prav tako izbral arhitekturo, ki jo je možno nadgraditi v hibridni model, npr. Dodatek LSTM ali GRU, kar omogoča obdelavo časovne dimenzije in povezanosti med okvirji posnetkov. Tako bom lahko implementiral tudi kompleksnejše funkcionalnosti. Torej trenutno sem za model zasnoval CNN arhitekturo s pomočjo katere bom rešil osnovne funkcionalnosti s sprotnim učenjem pa bom model tudi nadgradil do končnih zahtev. Model bom učil na podlagi podatkov, ki sem jih razdelil s pomočjo skripte glede na delitev: trening(70% posnetkov), validacija(15%), testiranje(15%).

2.4. Luka Lamprečnik

Cilj tega modela (detekcija drže voznika) je razvrstiti telesno držo voznika med vožnjo in tako zagotoviti, da sistem za upravljanje gest deluje le, kadar je voznik v ustreznem položaju za varno uporabo. Z modelom želimo zaznati, ali ima voznik roke na volanu, ali gleda naprej ali vstran, in ali je njegova drža vzravnana ali sključena.

Podatke za učenje modela smo zajeli z lastnimi video posnetki, kjer vsak član ekipe izvaja različne geste pred kamero. Ti posnetki ne prikazujejo resničnega okolja v avtomobilu, temveč simulirano okolje doma. S pomočjo knjižnice MediaPipe smo iz videoposnetkov pridobili koordinate ključnih točk telesa (posebno rok, ramen in glave). Te točke smo shranili v .npy datoteke in jih uporabili kot vhodne podatke za učenje.

Za razpoznavo drže uporabljamo kombinacijo konvolucijske nevronske mreže (CNN), ki prepozna vzorce v položaju telesa, in goste plasti (Dense), ki iz teh značilnic izpelje končno klasifikacijo. Model predvidi, v kateri kategoriji se nahaja drža: npr. "roki na volanu in pogled naprej" ali "eno roko spodaj in pogled v stran". Ti razredi služijo kot dodatni filter za sprejemanje ali zavrnitev geste.

Model bo treniran s pomočjo knjižnic TensorFlow/Keras, podatki pa bodo razdeljeni na učne, validacijske in testne množice, da preverimo njegovo zanesljivost.

3. Težave, rešitve in ideje za nadaljne delo

Model 1 - Ena izmed glavnih težav je bila, da so nekatere kategorije v podatkovnem naboru vsebovale premalo primerov (manj kot dva). Zaradi tega je prišlo do napake pri razdeljevanju podatkov na učni, validacijski in testni del z uporabo metode `train_test_split` in argumenta `stratify`. Funkcija zahteva, da ima vsaka razred najmanj dva primera, kar pri redkejših razredih ni bilo zagotovljeno. **Rešitev** je bila odstranitev vseh primerov, ki so pripadali razredom s premalo podatki, preden sva izvedla delitev. Tako sva zagotovila, da ima vsaka kategorija dovolj primerov za pravilno deljenje in treniranje modela.

Ideje za nadaljnje delo: Razširitev podatkovnega nabora, da bi vsi razredi imeli uravnoteženo število primerov, lahko uporabljamo naprednejših arhitektur, kot so **Bidirectional LSTM** ali **CNN-LSTM** za boljše razpoznavanje, dodajanje augmentacije podatkov (npr. hrup, razteg) za izboljšanje generalizacije modela in na koncu izdelava uporabniškega vmesnika za prepoznavanje gest v realnem času.

Model 2 - Največja težava je, da posnetki niso narejeni v realnem avtomobilu, temveč doma, pogosto brez celotnega telesa v kadru. Zaradi tega bo model težko generaliziral na prave razmere. Rešitev je zajem novih posnetkov, kjer bo telo voznika bolj vidno in bo kamera postavljena podobno kot v avtomobilu. V prihodnje razmišljamo tudi o razširitvi modela z LSTM ali 3D CNN arhitekturo, ki bi zaznala spremembe drže skozi čas, ne le v eni sličici.

Teden 2

1. Tabela zadolžitvev

Član	Zadolžitve
Simona Zhirova	Model 1 - Gestna kontrola (za radio in klima) <ul style="list-style-type: none">- Porazdelitev posamezne podatke za model za radio in model za klimatskih naprav- Treniranje modela z LSTM arhitekturo- Testiranje modela
Marija Jovanova	Model 1 - Gestna kontrola (za okna in vzvratna ogledala) <ul style="list-style-type: none">- Porazdelitev posamezne podatke za model za okna in model za vzvratnih ogledal- Treniranje modela z LSTM arhitekturo- Testiranje modela
Luka Lamprečnik	
Nik Terglav	Model 3 - Model za analizo okolje-specifičnih podatkov <ul style="list-style-type: none">- Dodati novo zbirko podatkov- Posodobiti skripto za razdelitev podatkov v množice, zaradi menjave zbirke- Dokončati implementacijo modela in razširiti delovanje- Treniranje modela- Validacija modela- Testiranje modela

2. Opis dela

2.1. Simona Zhirova

Ta teden sem se osredotočila na implementacijo in izboljšavo modelov. Najprej sem znova razdelila podatke na učne, testne in validacijske sete, da sem zagotovila ustrezno podlago za trening in evaluacijo. Implementirala sem dva modela: enega za upravljanje radia, ki vključuje geste, kot so `turn_on_radio`, `turn_off_radio`, `volume_up`, `volume_down`, `next_station` in `previous_station`, ter drugega za upravljanje klimatske naprave, ki zajema geste za vklop, izklop in prilagajanje temperature. Modele sem trenirala z uporabo LSTM arhitekture in k-fold navzkrižne validacije, pri čemer sem dosegla povprečno natančnost okoli 90 %. Nato sem modele testirala, analizirala rezultate, kot so matrike zmede in poročila o klasifikaciji, ter izvedla popravke kode, kjer sem dodala augmentacijo podatkov (naključni šum, premiki, skaliranje).

Med razvojem sem naletela na več izzivov, ki sem jih postopoma reševala. Največjo težavo je predstavljala zmeda med gestama `turn_off_radio` in `turn_on_radio`, saj so bili vzorci gibanja v podatkih preveč podobni, kar je razvidno iz matrike zmede, kjer je bilo 5 napačnih klasifikacij. Prav tako sem odpravila težavo z nepravilnim preslikavanjem oznak, saj sem ugotovila, da vrstni red map (npr. `next_station`, `previous_station` itd.) določa pravilne oznake, in ustrezno popravila skripte. Poleg tega sem zaznala, da trenutni podatki ne zajemajo dovolj dinamike odpiranja in zapiranja dlani, kar sem poskušala izboljšati z dodajanjem funkcij, kot je izračun razdalj med prsti, ter s priporočili za zbiranje novih podatkov z bolj izrazitimi gibi. S temi popravki sem pripravila trdnejšo osnovo za nadaljnje testiranje v realnem času in morebitno združevanje modelov.

2.2. Marija Jovanova

Ta teden sem nadgradila osnovno implementacijo modela, ki sem ga prejšni teden naredila. Sem se odločila delati 2 ločena modela za okna in vzvratnih ogledal. Dodala sem še nekaj primerov, da bo več in s tem da bom lažje in natančneje trenirala model, ki sem jih posnemala in zajela z MediaPipe v .npy in sem znova porazdelila podatke na učne, testne in validacijske.

Z skripto `data_widows.py` in `data_mirrors.py` sem .npy obdelala podatke (z dopolnjevanjem dolžine zaporedij) in sem jih shranila v .npy formatu za nadaljno uporabo. Obe skripti služita nalaganju in pripravi podatkov za učenje iz .npy datotek, ki so organizirane po mapah glede na labele. Skripti sta podobni, razlikujeta se v mapi s podatki (`data_windows` ali `data_mirrors`). Z skripto se preberejo vse .npy datoteke iz podmap (vsaka podmapa je ena labela), vsakemu razredu dodeli številčno oznako, naloži zaporedja in jih shrani v seznam. Potem prilagodi dolžino vseh zaporedij z padding, da bodo enako dolga in shrani pripravljene podatke in oznake v .npy datoteke v obliki `x_*.npy` in `y_*.npy`. 3D podatki (zaporedja) so shranjeni v

x_windows.npy in x_mirrors.npy, oznake razredov - labela so shranjene v y_windows.npy in y_mirrors.npy.

Skripte porazdelitev_data_okna.py in porazdelitev_data_ogledal.py naložijo vhodne podatke iz oblike x_*.npy in oznake iz oblike y_*.npy, prikaže število primerkov na razred in porazdeli podatke na učne, validacijske in testne množice. Na koncu jih shrani v x_*)_train.npy/y_*)_train.npy, x_*)_val.npy/y_*)_val.npy, x_*)_test.npy/y_*)_test.npy.

Sem naredila tudi dve skripti za implementacijo modela za okna in vzvratnih ogledal. Skripte so namenjene za učenju nevronske mreže za klasifikacijo gibov na podlagi časovnih zaporedij. Na začetku naloži podatki x_*.npy in y_*.npy. Vsi primeri se skrajšajo na maksimalno dolžino 80 okvirjev, da imajo enotno dolžino. Dodala sem tudi augmentacijo, ki vsakemu vzorcu doda naključen šum, raztegne ali premakne točke in s tem se pridobijo še 2 dodatna niza umetno spremenjenih podatkov, kar poveča učno množico trikrat. Potem se podatki normalizirajo (odšteje se povprečje in deli s standardnim odklonom). Izračunajo se uteži razredov, tako da se bolj redki razredi pri učenju ne zapostavijo. Model temelji na: 1D konvoluciji za obdelavo časovnih značilnosti, LSTM plasti za zaznavanje zaporednih vzorcev, Dropout za zaščito pred preprileganjem, Dense slojih za končno klasifikacijo med 6 razredi z softmax. Izvede se 5-kratna delitev podatkov na učne in validacijske dele. Za vsak preizkus se izpiše: natančnost (pove splošno pravilnost modela), konfuzijska matrika (kje se model moti) in poročilo klasifikacije (preciznost - od vseh napovedi, da je primer razreda X, koliko jih je bilo dejansko pravih, priklic - od vseh primerov razreda X, koliko jih je model pravilno prepoznal, F1-mera - kombinacija preciznosti in priklica, pove kako uravnotežen je model pri natančnih in popolnih napovedih). Na koncu se model ponovno nauči na celotni množici (z delom na validacijo), da dobi čim boljše končne rezultate in se model shrani v .keras obliki.

2.3. Nik Terglav

Zaradi kratko-časovnih sprememb pri dodeljevanju opravil sem moral ta teden izbrati novo zbirko podatkov na podlagi katerih bom treniral model. Na spletu sem našel zbirko, ki je vsebovala točno takšne slike kot sem jih potreboval. Te zbirke sem prenesel in ustrezno citiral avtorje, ki so odgovorili za njen obstoj. Zaradi novih podatkov sem spremenil skripto, ki mi osnovne podatke razdeli na podlagi določenega razmerja v "train", "validate" in "test" množice.

Za tem sem pričel s razširjevanjem modela, najprej sem pripravil novo anaconda okolje, da sem razrešil "dependency" težave, ker sem se odločil za uporabo drugih knjižnic (tensorflow -> pyTorch). Zaradi boljše učinkovitosti sem se odločil, da bom serijsko nalagal slike, to sem implementiral s DataLoader funkcijo iz knjižnice. Vse podatke sem naložil in transformiral za boljše delovanje, to vsebuje resize, menjava v tensor format in normalizacija pikslov.

Nato je nastopila implementacija, inicijalizacija in treniranje modela. Pri tem sem si močno pomagal z pytorch dokumentacijo. Moj model vsebuje dve konvolucijske plasti, eno popolnoma

povezano plast in pooling. Ko sem razrešil težave pri implementaciji modela sem dodal še izgubno (loss) funkcijo in optimizer, ti dve sta odgovorni za evaluacijo in sprotno optimiziranje modela. Nato sem lahko pričel z treniranjem in validacijo, ustvaril sem train zanko, ki traja za določeno število “epoch” in trenira model na podlagi naloženih podatkov. Za tem sem še definiriral validate funkcijo, ki validira natančnost modela. Funkcijo kličem znotraj train zanke in rezultate tudi izpisujem.

Na koncu sem potreboval še zagnati mojo skripto in shraniti model. Ko sem dobil ustvarjen model je ostalo samo še njegovo testiranje. Z kratko podobno zanko kot prejšnji dve sem se sprehodil zdaj čez testne podatke in izpisal končno natančnost modela.

2.4. Luka Lamprečnik

Kaggle - videji za učenje

3. Težave, rešitve in ideje za nadaljne delo

Model 3 - Že na začetku sem se soočil s težavo nepravilnih “dependency-jev”, ker sem se odločil za uporabo druge knjižnice za razvoj modela. To sem rešil tako, da sem ustvaril novo anaconda okolje in ga ustrezno konfiguriral ter namestil ustrezne knjižnice. Naslednja težava, ki je nastopila je bila nepravilna oblika vhodnih/izhodnih podatkov. Definiral sem preprosto funkcijo, ki vhodne podatke poda čez model in za vsak korak izpiše obliko. Ko sem popravil napačne spremenljivke sem dobil ustrezen izhod [8, 2], kar pomeni 8 slik, vsaka 2 oznaki. Zadnja težava je bila sistemska/strojna napaka, namreč se mi je zaradi prevelikega števila “batch” in števila “epoch” prenosni računalnik ugašal. To sem rešil, da sem prilagodil spremenljivke in treniral model ne v jupyter lab-u ampak sem shranil notebook kot skripto in jo pognal v terminalu.

Ideje za nadaljno delo - Ker s trenutno arhitekturo nisem uspel implementirati še prepoznavanje tipa vremena, želim nadgraditi model v hibridnega, ki bo omogočal več izhodov in prepoznavo oznak.

4. Rezultati

4.1. Model 1

4.2. Model 2

4.3. Model 3

```
(model-okolje-env) C:\Users\tergl\Desktop\Git repos\Upravljanje_z_gestami\UI\Model 3 - Razvoj modela za analizo okolje-specifičnih podatkov>python Model3.py
Oznake/Labels: ['day', 'night']
Število slik v učnem sklopu: 6430
Oblika vhodnih slik: torch.Size([8, 3, 224, 224])
Oznake: tensor([1, 0, 1, 1, 0, 1, 0, 0])
Oblika izhoda modela: torch.Size([8, 2])
Epoch 1/5: 100%|██████████| 804/804 [03:12<00:00, 4.18it/s, Loss=0.104]
Epoch 1/5 - Training Loss: 0.1042, Validation Loss: 0.0796, Validation Accuracy: 97.68%
Epoch 2/5: 100%|██████████| 804/804 [03:11<00:00, 4.19it/s, Loss=0.0601]
Epoch 2/5 - Training Loss: 0.0601, Validation Loss: 0.0829, Validation Accuracy: 98.40%
Epoch 3/5: 100%|██████████| 804/804 [03:10<00:00, 4.21it/s, Loss=0.0441]
Epoch 3/5 - Training Loss: 0.0441, Validation Loss: 0.0674, Validation Accuracy: 98.55%
Epoch 4/5: 100%|██████████| 804/804 [03:11<00:00, 4.21it/s, Loss=0.0203]
Epoch 4/5 - Training Loss: 0.0203, Validation Loss: 0.1428, Validation Accuracy: 96.66%
Epoch 5/5: 100%|██████████| 804/804 [03:11<00:00, 4.20it/s, Loss=0.0109]
Epoch 5/5 - Training Loss: 0.0109, Validation Loss: 0.1305, Validation Accuracy: 98.48%
```

```
model = TimeOfDayCNN()
model.load_state_dict(torch.load("model.pth", weights_only=True))
model.eval()
```

```
test_model(model, device, test_loader)
```

Test Accuracy: 98.11%