Univerza v Mariboru

FERI

UMETNA INTELIGENCA

SEMINARSKA NALOGA

Prepoznava glasovnih ukazov

Maj Fontana Korošec

3. letnik, Telekomunikacije (UN), UM FERI

Asistent:

Mladen Borovič, mag. inž. rač. in inf. tehnol.

Poljčane, 9. 6. 2023

1. Kazalo vsebine

2. Kazalo slik

3. Zasnova aplikacije

Pri tej seminarski nalogi sem se odločil zasnovati enostavno aplikacijo, ki na glasovni ukaz odpre eno izmed štirih pogosto uporabljenih aplikacij na mojem računalniku. Celotna rešitev je napisana v Pythonu. Za aplikacije, ki jih moja aplikacija odpira sem izbral terminal, beležnico, Python interpreter, ter Windows Explorer na lokaciji, kjer se nahajajo dokumenti. Glasovni ukazi so tako: »terminal«, »notepad«, »Python« ter »documents«.

3.1. Zajem glasu

Uporabnik zajem ukaza prične tako, da pritisne kombinacijo tipk na tipkovnici. Zaznava kombinacije je rešena s pomočjo knjižnice global-hotkeys.

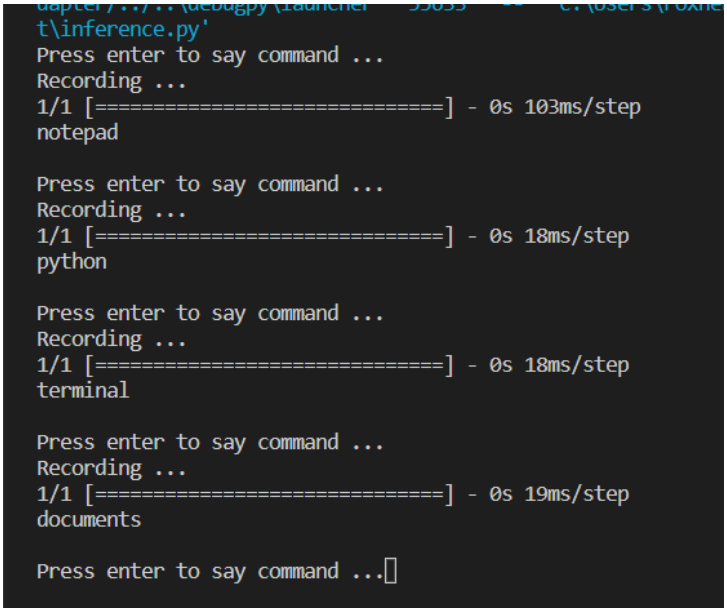
Zajem zvoka je implementiran v lastni Python knjižnici, namenjeni procesiranju signalov. Ta zvočne podatke iz mikrofona neprestano bere s pomočjo knjižnice PyAudio, ter jih pertvarja v normalizirana Numpy polja, ob zahtevi za ukaz pa jih prične zapisovati v polje za zajem.

3.2. Prepoznava ukaza

Ko je zajeto zadostno število otipkov, se le-ti pošljejo v model za klasifikacijo, ta pa vrne izhodni vektor verjetnosti da gre za posamezen ukaz. Z argmax funkcijo se izloči ukaz z najvišjo verjetnostjo. Model za klasifikacijo je konvolucijska nevronska mreža, izdelana s knjižnico TensorFlow.

3.3. Zagon aplikacij

Zagon aplikacij je rešen z ukazom lupini preko funkcije os.system.



Slika 1: Prikaz delovanja aplikacije, pred implementacijo zaznavanja kombinacij tipk

4. Zasnova modela za klasifikacijo

4.1. Vhodni podatki in preprocesiranje

Vhod je enokanalni posnetek zvoka predstavljen z enodimenzionalnimi vzorci amplitude v časovni domeni, preslikanimi med -1 in 1. Preden gredo v mrežo, se ti podatki transformirajo s kratkočasovno Fourierjevo transformacijo, ter tako pretvorijo v dvodimenzionalne otipke frekvence in amplitude, na katerih se lahko kasneje izvaja dvodimenzionalna konvolucija.

4.2. Izhodni podatki

Izhod je enodimenzionalni vektor, ki predstavlja ocenjene verjetnosti, da se bil posnet posamezen ukaz. Za izračun izgube je uporabljen keras.losses.SparseCategoricalCrossEntropy. Za optimizator učenja je bil uporabljen optimizator Adam.

4.3. Struktura modela

Model je ustvarjen z TensorFlow konstruktorjem models.Sequential. Podatki gredo pri procesiranju v modelu skozi naslednje sloje:

1. zmajšanje velikosti slikovnega vhoda (layers.Resizing),
2. normaliziranje vrednosti slikovnega vhoda (layers.Normalization),
3. več slojev 2D konvolucije (layers.Conv2D),
4. Združevanje z maksimumom (layers.MaxPooling2D),
5. Več gosto povezanih nevronskih slojev (layers.Dense).

Natančno število slojev je bilo določeno v procesu optimizacije hiperparametrov.

Layer (type)	Output Shape	Param #
resizing_2 (Resizing)	(None, 32, 32, 1)	0
normalization (Normalization)	(None, 32, 32, 1)	3
conv2d_6 (Conv2D)	(None, 30, 30, 80)	800
conv2d_7 (Conv2D)	(None, 26, 26, 112)	224112
conv2d_8 (Conv2D)	(None, 22, 22, 48)	134448
max_pooling2d_2 (MaxPooling2D)	(None, 5, 11, 48)	0
dropout_6 (Dropout)	(None, 5, 11, 48)	0
flatten_2 (Flatten)	(None, 2640)	0
dense_6 (Dense)	(None, 32)	84512
dropout_7 (Dropout)	(None, 32)	0
dense_7 (Dense)	(None, 80)	2640
dropout_8 (Dropout)	(None, 80)	0
dense_8 (Dense)	(None, 4)	324

Total params: 446,839
Trainable params: 446,836
Non-trainable params: 3

Slika 2: Končna arhitektura modela po končani optimizaciji hiperparametrov

5. Učenje modela

5.1. Zajem učnih podatkov

Da bi lahko učil klasifikacijski model, sem moral ustvariti učno množico. Ustvaril sem enostavno orodje, ki ob vnosu imena ukaza posname zvok iz mikrofona znotraj določenega časovnega okna, ter ga samodejno shrani v mapo z vzorci. Za vsakega od štirih ukazov sem posnel dvajset učnih vzorcev, torej skupaj 80 vzorcev. Vzorce sem shranjeval v wav formatu, s šestnajst-bitnim zapisom, zato, da jih je bilo mogoče predvajati tudi v privzetem predvajalniku operacijskega sistema Windows.

5.2. Avgmentacija učnih podatkov

Da bi v učno množico prinesel več variacije, ter izboljšal število učnih vzorcev, sem učne podatke pred učenjem avgmentiral. Podatke, prebrane iz mape, sem prvo večkrat ponovil, da sem zvišal število vzorcev, nato pa sem vzorcem dodal sledeče izgubne in neizgubne spremembe: krožni zamik podatkov (za emulacijo različnih časovnih zamikov pred pričetkom govora), sprememba amplitude ter beli šum.

5.3. Predprocesiranje podatkov

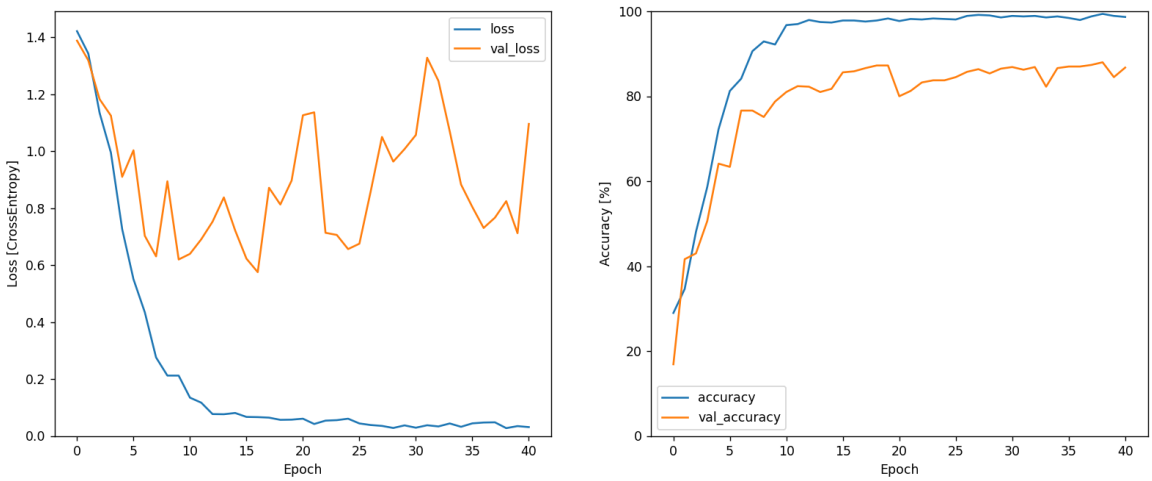
Zaradi oblike, v kateri orodje za branje zvočnih datotek TensorFlow knjižnice vrne prebrane vzorce, je bilo le-tem potrebno odstraniti dimenzijo, ki predstavlja zvočni kanal, saj imajo vhodni podatki zgolj en kanal, in dodatna dimenzija ni potrebna. Za kratkočasovno Fourierjevo transformacijo sem uporabil funkcijo, na voljo v TensorFlow knjižnici. Dvodimenzionalnemu spektogramu se pred konvolucijo še zmanjša velikost, njegove vrednosti pa se normalizirajo.

5.4. Učenje klasifikacijskega modela

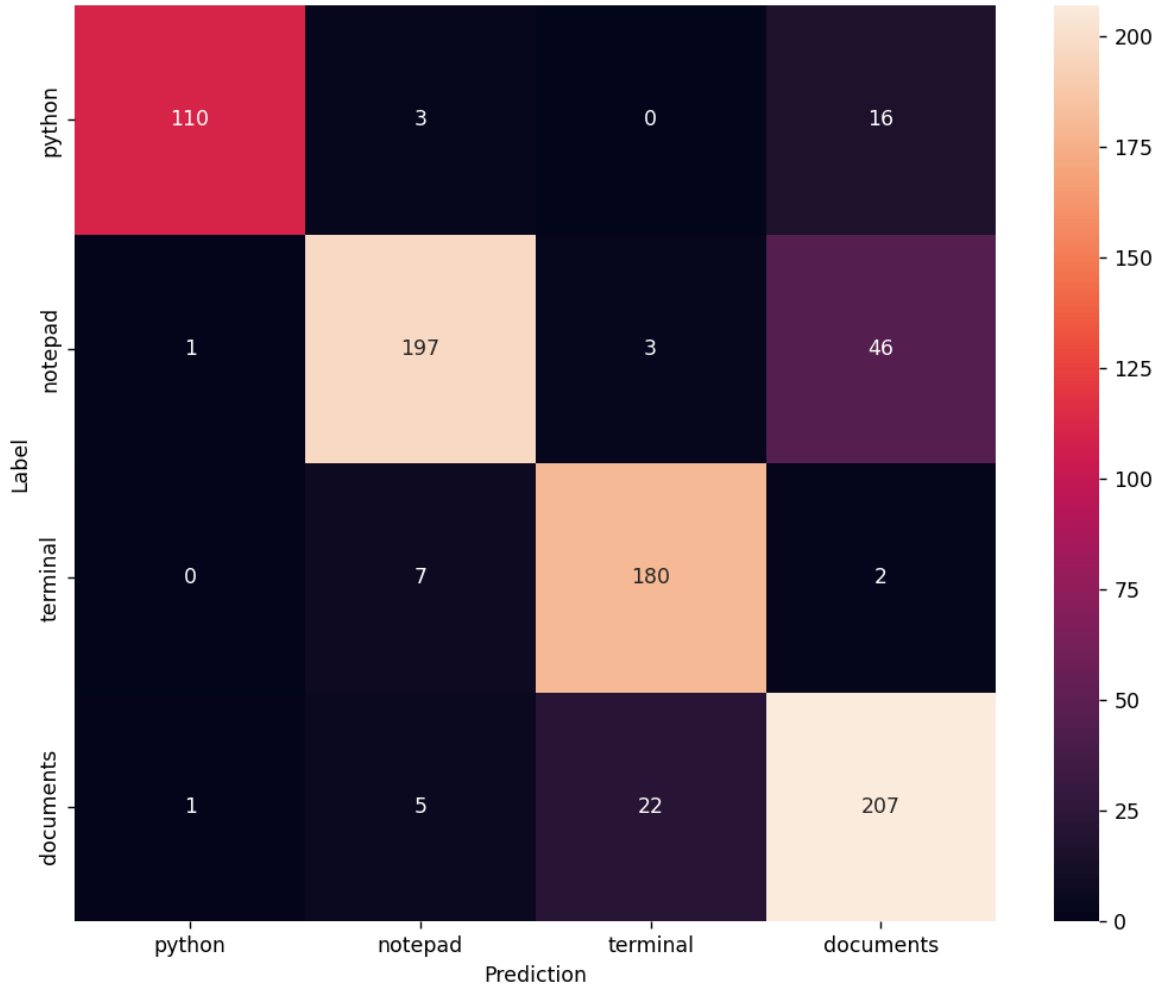
Model se je učil 40 epohov, kar je kot optimalno trajanje učenja določil optimizator hiperparametrov. Prav tako je optimizator določil velikosti posameznih slojev in določene druge parametre, kot na primer velikost konvolucijskega jedra. Za optimizacijo hiperparametrov je bil uporabljen keras_tuner.Hyperband. Po učenju se model shrani na disk, od koder ga lahko naloži aplikacija za poslušanje ukazov.

5.5. Uspešnost učenja

Natančnost validacije je po približno 20 epohih dosegla svojo končno vrednost 80 %. Natančnost pri treniranju pa je v bližino 100 % prišla že po 13 epohih, zato mislim, da je prihajalo do prekomerne prilagoditve znotraj posameznih učnih epoh.



Slika 3: Potek učenja na končni optimizirani konfiguraciji hiperparametrov



Slika 4: Matrika ujemanja pričakovanih in dejanskih rezultatov pri končnem testiranju mreže