Klasifikacija terena na temelju računalnog vida i propriocepcijskih senzora mobilnog robota

Šošić, Diana

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:333698

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

KLASIFIKACIJA TERENA NA TEMELJU RAČUNALNOG VIDA I PROPRIOCEPCIJSKIH SENZORA MOBILNOG ROBOTA

Rijeka, rujan 2023.

Diana Šošić

0069081149

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Diplomski sveučilišni studij elektrotehnike

Diplomski rad

KLASIFIKACIJA TERENA NA TEMELJU RAČUNALNOG VIDA I PROPRIOCEPCIJSKIH SENZORA MOBILNOG ROBOTA

Mentor: Prof. dr. sc. Zlatan Car

Rijeka, rujan 2023.

Diana Šošić

0069081149

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 14. ožujka 2023.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Primjena umjetne inteligencije**
Grana: **2.03.06 automatizacija i robotika**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Diana Šošić (0069081149)**
Studij: Sveučilišni diplomski studij elektrotehnike
Modul: Automatika

Zadatak: **Klasifikacija terena na temelju računalnog vida i propriocepcijskih senzora mobilnog robota**

Opis zadatka:

Predstaviti problem i značaj prepoznavanja vrste terena za kretanje mobilnog robota, te dati pregled postojećih istraživanja. Opisati korišteni set podataka, uz fokus na korištene senzore, i metode strojnog učenja korištene u radu. Evaluirati modele na temelju ostvarenih rezultata, te komentirati učinkovitost algoritma.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Diana Šošić

Zadatak uručen pristupniku: 20. ožujka 2023.

Mentor:

Prof. dr. sc. Zlatan Car

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Dubravko Franković

IZJAVA

"Sukladno članku 9. Pravilnika o diplomskom radu, diplomskom ispitu i završetku diplomskih sveučilišnih studija Tehničkog fakulteta Sveučilišta u Rijeci izjavljujem da sam izradila ovaj diplomski rad samostalno, koristeći vlastito znanje i navedenu literaturu, u razdoblju od datuma zadavanja zadatka do datuma predaje."

Rijeka, rujan 2023.

Diana Šošić

Diana Šošić

ZAHVALA

Zahvaljujem se mentoru prof. dr. sc. Zlatanu Caru na pruženoj prilici izrade ovog diplomskog rada. Posebno se zahvaljujem asistentu mag. ing. comp. Sandiu Baressi Šegoti na izdvojenom vremenu, trudu i pomoći pri izradi diplomskog rada.

Također, zahvaljujem se svojoj obitelji i prijateljima na podršci i potpori tijekom svih godina studiranja.

Sadržaj

1. UVOD

Osnovna ideja ovog istraživanja je modelirati i usporediti različite performanse modela koje se temelje na postojećem setu podataka pomoću metoda računalnog vida. Koristit će se set podataka koji sadrži očitanja senzora i snimke različitih vrsta terena dobivene pomoću mobilnog robota. Općenito, klasifikacijom terena moguće je unaprijediti primjenu u različitim područjima poput agrikulture, geomorfologije, predviđanja i identificiranja opasnih terena.

Ovaj rad nadovezuje se na postojeće istraživanje u kojem se koristi mobilni robot za detektiranje terena [1]. Riječ je o mobilnom robotu koji je bespilotno zemaljsko vozilo tj. UGV (*eng. Unmanned Ground Vechile*) vozilo koje ima ugrađeno računalo. Ovakvom vrstom autonomnog vozila (robota) ne upravlja čovjek već se koriste metode umjetne inteligencije. Sastoji se od više senzora pomoću kojih se promatra i bilježi okolina te ima mogućnost autonomno donositi odluke na temelju prethodno sakupljenih informacija. Neke od ostalih funkcija su: detekcija objekata, ljudi i vozila koja se nalaze na putanji, sposobnost kretanja između različitih točaka putanje te obrada podataka u stvarnom vremenu.

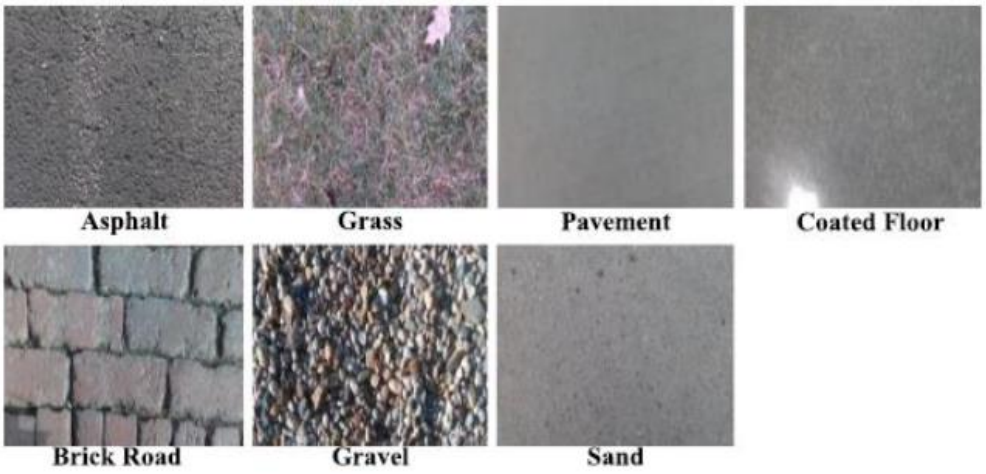
Tijekom procesa snimanja i prikupljanja podataka u obzir se uzima raznolikost i robusnost terena pod ograničavajućim i promjenjivim osvjetljenjima koja utječu na zabilježene podatke. Mobilni robot koristi propriocepcijske senzore koji mjere unutrašnjost stanja robotskog sustava poput brzine motora, opterećenja kotača te napona baterije. Primjeri ovakvih senzora mogu se pronaći u optičkim enkoderima, žiroskopu te akcelerometru.

Na temelju seta podataka potrebno je izraditi modele na temelju očitavanja senzora i snimaka. Izvršiti će se dvije klasifikacije od kojih je jedna klasifikacija slika, a druga klasifikacija numeričkih podataka. Za klasifikaciju slika koristit će se VGG16 model, dok će se za klasifikaciju podataka koristiti MLPClassifier metoda. Ovakve klasifikacije zahtijevaju treniranje neuronske mreže te implementaciju pojedinih metoda. Isto tako, na kraju će se prikazati dobiveni rezultati za različite parametre te će se usporediti performanse prethodno spomenutih modela kao i uvođenje dodatne metode s ciljem dobivanja najboljih rezultata.

2. PROCES PRIKUPLJANJA PODATAKA

Za prikupljanje podataka koristila se mobilna robotska platforma, a podaci su se obrađivali „offline“. U obzir se uzimaju signali lijevoga i desnoga kotača kao i propriocepcijski signali robota. Ovakav način omogućuje eliminiranje smetnje ovisno o kretanju robota te su propriocepcijski i vizualni signali komplementarni. Koriste se dva tipa klasifikatora; klasifikator koji je napravljen na temelju prethodno naučenog modula (*eng. Pre-trained module*), a baziran je na viziji te klasifikator koji se bazira na propriocepciji.

Mobilni robot je robot s kliznim upravljanjem i pogonom na četiri kotača, ima dva DC motora s enkoderima koji mjere kutne brzine kotača. Sadrži još i strujne senzore za mjerenje izlazne struje te ugrađeni IMU (*eng. Inertial measurement unit*). Na svakoj strani, prednji i stražnji kotač su spojeni s mjenjačem i tako se okreću skupa. IMU uređaj pruža mjerenje položaja vozila, pod koje spadaju Eulerovi kutevi, linearna akceleracija te kutna brzina. Koriste se dvije vrste kamera, od kojih se jedna koristi za praćenje kretnje robota, dok je druga kamera okrenuta pod kutem tako da prikuplja snimke na kojima se nazale različiti tipovi terena. Za svaku klasu trena, upotrijebila su se dva načina upravljanja: „straight driving“ i daljinsko upravljanje [2]. Na slici 2.1. nalazi se prikaz 7 vrsta terena koje se pojavljuju unutar ovog seta podatka, a to su: asfalt, trava, pločnik, obloženi pod, pod od cigle, šljunak i pijesak.

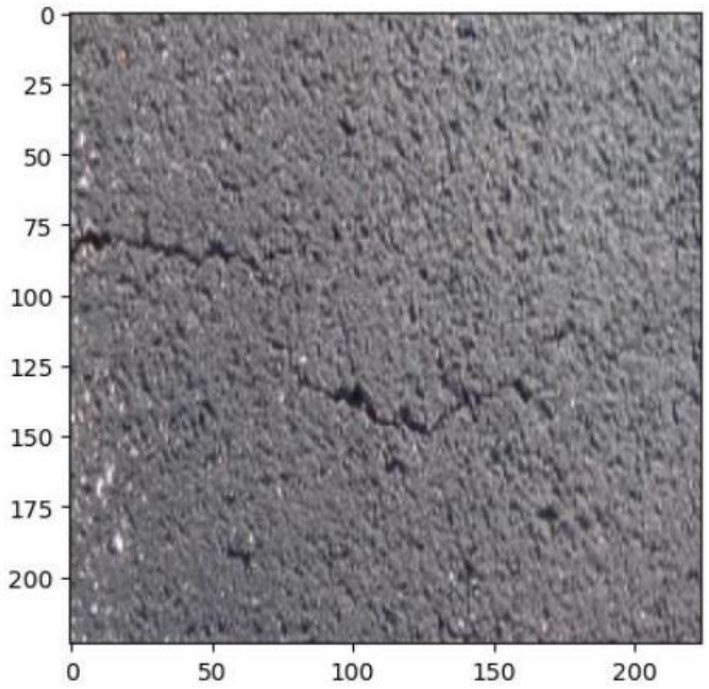


Slika 2.1. Snimljene vrste terena [3]

2.1. Materijali potrebni za izradu

Pri izradi programskog koda korišten je jedan od dostupnih seta podataka pod nazivom „testSet_c7_ver_2.hdf5“ veličine 2.30 GB [4]. Ovaj dataset nalazi se u sklopu drugih setova podataka istog izvora, a izabran je kao podskup zbog velike količine podataka i zbog toga što sadrži podjednak broj snimaka svih različitih klasa. Svaka HDF5 datoteka sadrži 4 tipa osnovnih unosa podataka: slike, oznake, signali i vremenske oznake. Dostupan je i primjer koda u programskom jeziku Python u obliku Python bilježnice (.ipynb) kako bi se demonstrirao način na koji se može pristupiti spremljenim podacima te kako ih formatirati. Ovaj set podataka korišten je za izradu modela umjetne inteligencije, koji su glavni dio ovog diplomskog rada.

Već spomenuta datoteka tipa HDF5 sadrži 4020 uzoraka, 7 klasa terena te ukupno vrijeme snimanja koje je spremljeno unutar same datoteke i iznosi 1.1 sat. Koriste se isključivo snimke prikupljene daljinskim upravljanjem. U ovom setu podataka nalaze se RGB slike čije su se dimenzije prilagodile na 224 x 224 piksela kako bi se dalje mogle obrađivati u treniranju mreže. Na slici 3.1. nalazi se prikaz jednog od selektiranih uzoraka, oznake asfalt. Nalazi se pod vremenskom oznakom „1612385348“.



Slika 2.2. Uzorak jednog tipa terena [5]

Prvi korak je izolacija slika i numeričkih podataka iz prethodno opisanog seta podataka. Izolirane podatke pohranjuje se u dvije odvojene numpy matrične datoteke. Za svaki izlaz potrebno je podatke podijeliti tako da, jedan set podataka sadrži set sa svim slikama i izlaz koji označava tip terena, bez numeričkih podataka dobivenih pomoću senzora. Drugi set podataka mora imati sve numeričke podatke sa senzora te odgovarajuće izlaze tj. vrstu terena. U nastavku je priložen primjer koda u kojemu su se izdvojili i spremili prethodno opisani tipovi podataka.

```
output_directory = r'C:\Users\diana\Desktop\MOJ PROJEKT_DIP'
np.save('dataset_slike.npy', images,labels)
np.save('dataset_numericki.npy',signals,labels,timeStamps)
```

Kada se ti podaci izoliraju, isti se koriste za treniranje dvije neuronske mreže. Jedna od njih je konvolucijska neuronska mreža, a druga direktna mreža. Metode koje će se primijeniti bit će detaljnije opisane u nastavku ovog rada.

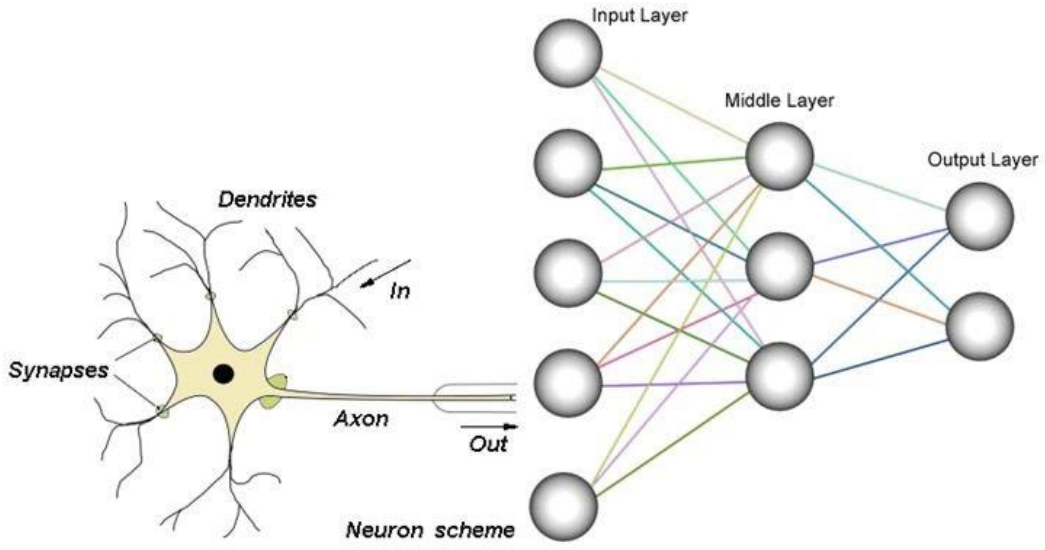
3. METODOLOGIJA

U ovom poglavlju bit će objašnjeno osnovno o umjetnim neuronskim mrežama i bit će detaljno opisana svaka metoda koja se koristila za potrebe dobivanja konačnih rezultata. Također, za svaku od pojedinih metoda, prikazat će se implementacija pomoću programskog koda.

3.1. Općenito o neuronskim mrežama

Neuronske mreže predstavljaju umjetne neronske mreže (ANN) koje spadaju u strojno učenje i dio su algoritama dubokog učenja. Njihov naziv i struktura pronalaze inspiraciju u ljudskom mozgu i oponašaju biološke neurone i njihovu međusobnu signalizaciju. Najjednostavniji primjer umjetne neuronske mreže čine slojevi čvorova, od kojih se razlikuju ulazni sloj, jedan ili više skrivenih slojeva te izlazni sloj. Svaki čvor predstavlja umjetni neuron i povezuje se s drugim čvorovima, s pridruženom težinom i pragom.

Na slici 3.1. se nalazi prikaz biološkog neurona i umjetne neuronske mreže. Biološki neuroni koriste elektrokemijski prijenos signala i komuniciraju pomoću sinapsi. Dendriti su ulazi koji primaju električne signale tj. impulse koji dalje preko sinaptičkih veza prenose informacije. Aksoni imaju funkciju izlaza jer provode informacije iz tijela neurona prema drugim ciljnim stanicama. S druge strane, umjetne neuronske mreže koriste matematičke funkcije za transformaciju ulaznih podataka i generiranja izlaza.



Slika 3.1. Biološki neuron i umjetna neuronska mreža [6]

Početak razvoja neuronskih mreža krenuo je od jednog perceptrona koji je dio plitkih neuronskih mreža. Takve mreže činile su jedan ulaz, najviše jedan skriveni sloj između te jedan izlaz. Ako se radi o dva ili više slojeva, mreža se naziva dubokom neuronskom mrežom. U mrežama dubokog učenja, svaki sloj čvorova trenira na posebnom skupu značajki u odnosu na izlaz prethodnog sloja. Ovakav pristup, dovodi do prepoznavanja sve složenijih značajki upravo zbog napredovanja u mreži. Duboke neuronske mreže oslanjaju se na mreže strojnog učenja koje se kontinuirano razvijaju tako što se uspoređuju procijenjeni rezultati sa stvarnim rezultatima.

Postoji više vrsta neuronskih mreža i mogu se klasificirati na principu: arhitekture, protočnosti podataka, načinu učenja, broju slojeva, aktivacijskih funkcija i slično. Modele neuronskih mreža mogu se podijeliti na:

- Perceptron – izvodi izračune za otkrivanje značajki ili uzoraka u ulaznim podacima, koristi se kao binarni klasifikator
- Feed-forward neuronske mreže – osnovni oblik neuronskih mreža gdje je protok informacija u samo jednom smjeru (ulaz-izlaz), nema povratnih veza, upotreba kod jednostavnijih klasifikacija, prepoznavanja lica i sl.

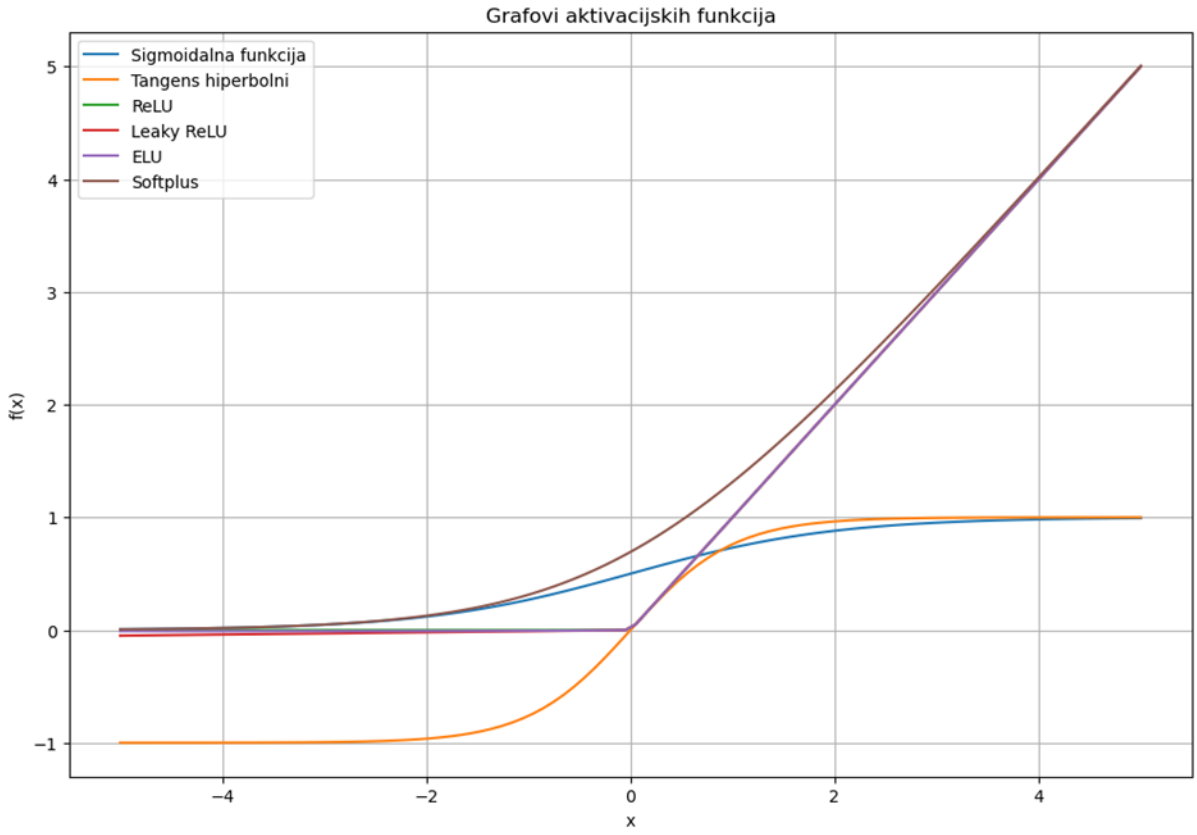
- Višeslojni perceptron (MLP) – svaki pojedinačni čvor povezan sa svim neuronima u sljedećem sloju (potpuno povezana neuronska mreža), dvosmjerno širenje, koristi se kod složenih klasifikacija, prepoznavanja govora
- Konvolucijske neuronske mreže (CNN) – sadrži trodimenzionalni raspored neurona,koriste konvolucijske slojeve za izvlačenje značajki iz ulaznih podataka tj. slika, upotreba kod obrade slika, prepoznavanja govora ili sl.
- Radial Basis Functional Neural Network – sastoji se od ulaznog vektora, sloja RBF neurona i izlaznog sloja s jednim čvorom po kategoriji, koristi se kod obnove napajanja
- Rekurentne neuronske mreže (RNN) – mreže s povratnom vezom, mogućnost obrađivanja sekvencijalnih podataka, koristi memorijske funkcije, npr. koristi se kod automatskog predlaganja teksta
- LSTM (*eng. Long Short-Term Memory*) – vrsta RNN mreža, koja koristi LSTM jedinice poput memorijske ćelije, postoje ulazna, izlazna i zaboravljena vrata za upravljanje informacijama u memoriji,
- Sequence to Sequence modeli – sastoji se od dvije RNN mreže, enkodera i dekodera, primjena ovakvog modela u chatbotovima i strojnim prijevodima
- Modularna neuronska mreža – ima niz različitih mreža koje funkcioniraju neovisno i obavljaju svoje pod-zadatke, primjena u sustavima za predviđanje dionica

3.2. Aktivacijske funkcije

Aktivacijska funkcija pretvara svoje ulaze u izlaze određenog raspona. Imaju ulogu da pomognu neuronskoj mreži da nauči složene obrasce u podacima. Aktivacijske funkcije uvode nelinearna složena svojstva stvarnog svijeta poput slika, videa, teksta ili zvuka u umjetne neuronske mreže.

Aktivacijske funkcije koje se najčešće koriste su: Sigmoid funkcija, hiperbolički tangens, ReLU (Rectified Linear Unit), Leaky ReLu, ELU (Exponential Linear Unit), Softplus. Na slici 3.2. nalazi

se prikaz mogućih aktivacijskih funkcija. Ovisno o arhitekturi mreže odabire se pogodna aktivacijska funkcija.



Slika 3.2. Graf aktivacijskih funkcija

U nastavku su opisane važnije funkcije i dani su njihovi matematički izrazi:

1. Sigmoidalna funkcija je matematička funkcija s karakterističnom krivuljom u obliku slova „S“ tj. sa sigmoidnom krivuljom. Transformira bilo koju vrijednost u domeni <-∞,∞> za vrijednost između 0 i 1. Označavamo je s grčkim slovom σ (sigma). Matematički opis definiran je izrazom (3.1):

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.1}$$

2. Tangens hiperbolni je matematička funkcija koja u domeni sadrži sve kompleksne brojeve u rasponu [-1,1]. Funkcija ima svojstvo srednje vrijednosti blizu 0 za razliku od Sigmoidalne funkcije koja ima srednju vrijednost oko 0.5. Upravo zbog toga se koristi tangens hiperbolni u smanjenju nestajućeg gradijenta (eng.vanishing gradient). Matematički opis definiran je izrazom (3.2):

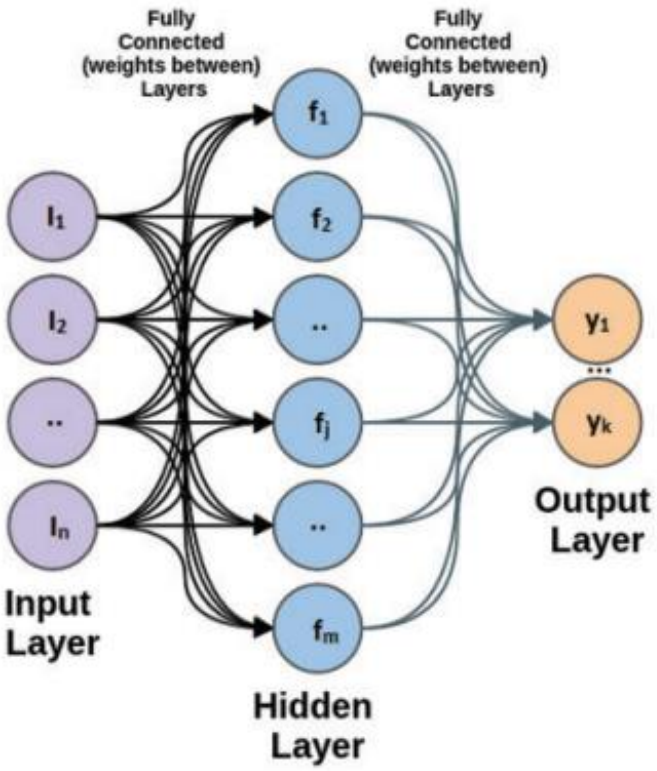
$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{3.2}$$

3. ReLU (*eng. Rectified Linear Unit*) je matematička funkcija koja je linearna za pozitivne vrijednosti ulaza i nula za negativne vrijednosti. Jednostavna i nelinearna, koristi se najčešće u dubokim neuronskim mrežama. Ovaj matematički opis dan je izrazom (3.3):

$$f(x) = x^{+} = \max(0, x) = \frac{x+|x|}{2} = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3.3}$$

3.3. Višeslojni perceptron (MLP)

Višeslojni perceptron (*eng. Multilayer perceptron)* je vrsta umjetne neuronske mreže koja se koristi u rješavanju složenih, linearnih neseparabilnih problematika. Spada u skupinu nadziranog strojnog učenja i dubokog učenja. MLP se sastoji od nekoliko slojeva čvorova koje čine neuroni koji su međusobno povezani, a svaki sloj ima svoju ulogu. Sastoji se od jednog ili više skrivenih slojeva koji se nalaze između ulaznih i izlaznih slojeva. Težine povezuju neurone u različitim slojevima, a svaka težina čini snagu veze između dva neurona. Važno je naglasiti da svaki sloj djeluje na izlazu svog prethodnog sloja te mreža nije povezana unutar petlje. Ovakva arhitektura naziva se eng. feed-forward mreža. Na slici 3.3. nalazi se prikaz prethodno opisane arhitekture.

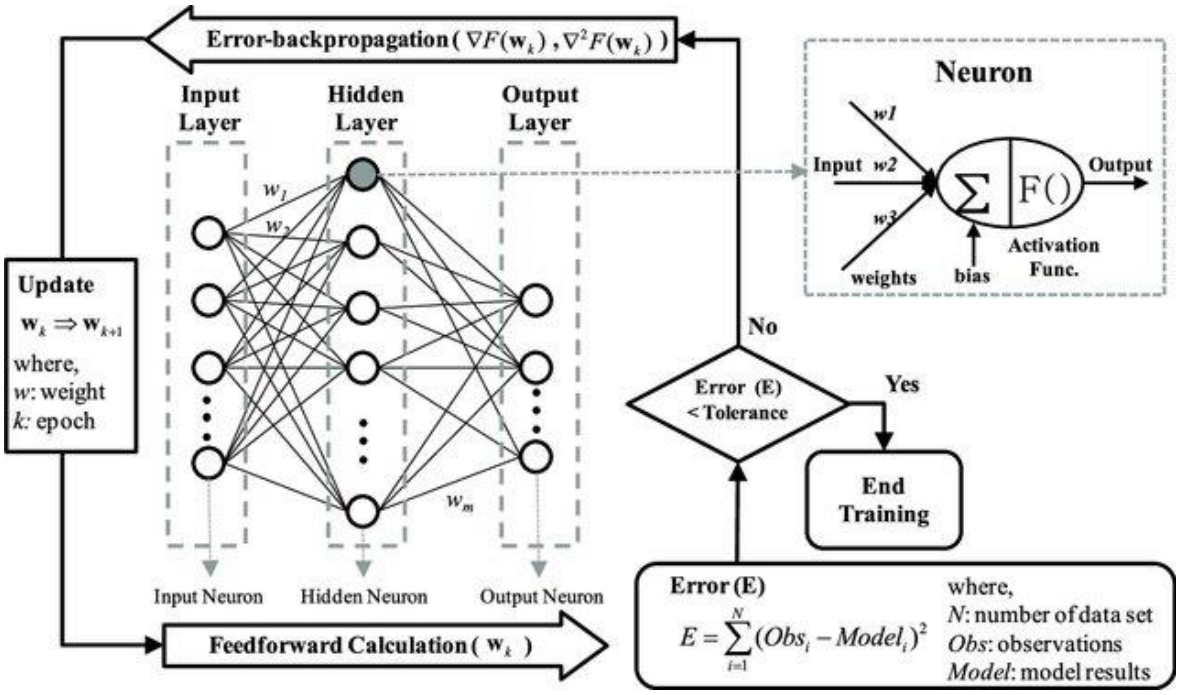


Slika 3.3. Neuronska mreža – MLP [7]

MLP se razlikuje od linearnog perceptrona u višestrukim slojevima kao i u nelinearnoj aktivacijskoj funkciji. Skriveni sloj kao i izlazni sloj koristi nelinearne aktivacijske funkcije kako bi se moglo riješiti složenije probleme. Najčešće se koristi ista aktivacijska funkcija za skriveni sloj, dok izlazni sloj koristi aktivacijsku funkciju različitu od skrivenog sloja i ovisi o vrsti predikcije nekog modela. Neuronska mreža ima sposobnost modelirati složenije i nestacionarne odnose između ulaza i izlaza, zahvaljujući nelinearnim aktivacijskim funkcijama. Za skriveni sloj kod MLP odabire se „ReLu“ aktivacijska funkcija. Za izlazni sloj, ovisno da li se radi o klasifikacijskom ili regresijskom tipu, odabire se najpogodnija funkcija.

MLP čini osnovu za učenje pod nadzorom (*eng. supervised learning*) kod kojeg model uči iz skupa ulaznih podataka i pripadajućih ciljnih vrijednosti ili oznaka. Koristi se algoritam propagacije unatrag (*eng. backpropagation)* pomoću kojega mreža prilagođava svoje težine da bi se minimalizirala greška

između stvarnih izlaza i predikcija mreže. Najčešće se koristi za klasifikaciju (dodjeljivanja ulaza unaprijed definiranim kategorijama) i regresiju koja služi za predviđanje kontinuiranih vrijednosti. Postupak treniranja čine algoritmi optimizacije, kao što je stohastički gradijentni spust koji ima sposobnost minimalizacije greške između stvarnih izlaza i predviđanja mreže.



Slika 3.4. Shematski dijagram backpropagation algoritma i model neurona [8]

Na slici 3.4. nalazi se dijagram backpropagation algoritma za treniranje mreže i model neurona. Backpropagation proces može se opisati u nekoliko koraka:

1. Inicijalizacija težina kao početak učenja,
2. propagacija unaprijed (*eng. forward pass*) gdje se ulazni podaci prosljeđuju, aktivacijske funkcije se primjenjuju na težinsku sumu za generiranje izlaznih podataka,
3. računanje greške gdje se izlaz mreže uspoređuje s očekivanim izlazom iz skupa podataka,
4. propagacija unatrag (*eng. backward pass*) gdje se vrši izračun gradijenta težina propagacijom greške unatrag kroz mrežu,

5. ažuriranje težina pomoću gradijenta i algoritama optimizacije, s ciljem minimalizacije greške, i
6. ponavljanje kroz više iteracija (epoha) propagacije unaprijed i unatrag.

MLP je glavni element dubokih arhitektura poput konvolucijskih neuronskih mreža (CNN) te rekurentnih neuronskih mreža (RNN). Neke od prednosti MLP algoritma su modeliranje kompleksnih nelinearnih odnosa između ulaza i ciljnih vrijednosti, moguća je aproksimacija bilo koje kontinuirane funkcije, imaju sposobnost automatskog učenja određenih značajka iz sirovih ulaznih podataka, te je moguće podesiti težine tijekom treniranja. Treniranje velikih seta podataka MLP-a dovodi do računalne složenosti što je jedan od nedostataka kao i potrebno podešavanje hiperparametra zbog postizanja optimalne kombinacije i overfittinga. Overfitting se pojavljuje kada se model u prevelikim količinama prilagodi podacima za treniranja, a kod novih podataka loše primjenjuje naučeno.

3.3.1. Implementacija MLPClassifier-a

MLPClassifier je algoritam klasifikacije koji se koristi unutar knjižnice strojnog učenja „Scikit-Learn-a“. Prvi korak je učitavanje potrebnih knjižnica za primjenu ove metode.

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
```

Nakon toga, učitavaju se numerički podaci i provjerava se oblik tih podataka.

```
# Load the data from file
data_num = np.load('dataset_numericki.npy')
print(data_num.shape)
```

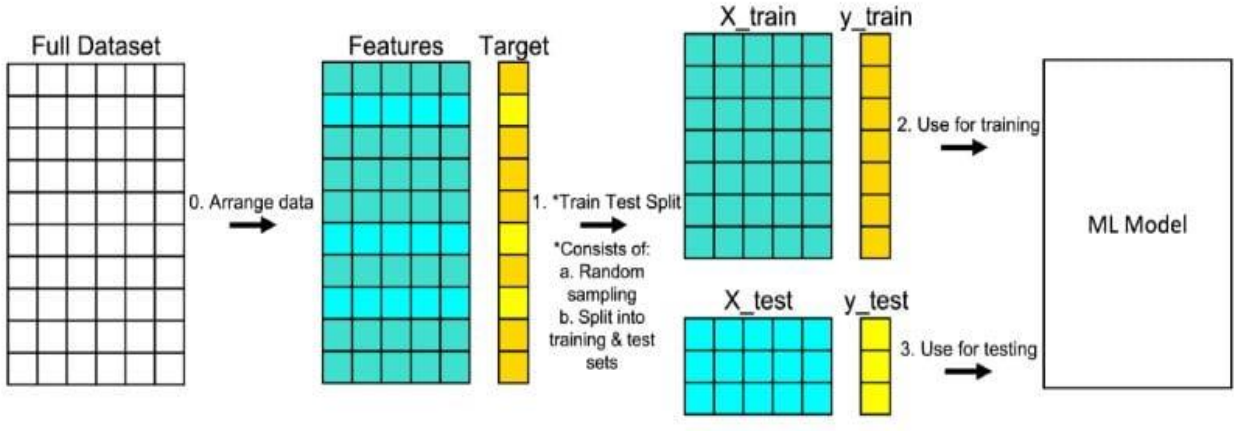
Podatke je potrebno preoblikovati i podijeliti na x i y kako bi se mogli dalje koristiti.

```
reshaped_data = data_num.reshape(data_num.shape[0], -1)
x = reshaped_data
y = labels
```

Ovdje se provjerava jesu li ispravnog oblika podaci. Dobiva se za x da su oblika (4020,900), a za y (4020,).

```
print('x shape:', x.shape)  # (4020, 900)
print('y shape:', y.shape)  # (4020,)
```

Podatke dijelimo na podatke za trening, testiranje i validaciju. To je proces kojim se simulira kakve će performanse model imati s novim podacima. Na slici 4.3. se nalazi prikaz općenitog postupka ovog procesa. Prvo se provjerava jesu li podaci pogodni za ovu metodu tako da se cijeli skup podataka razdvaja na značajke i cilj. Nakon toga se skup podataka dijeli na skup za obuku i skup za testiranje. Nasumično se vrši uzrokovanje bez zamjene, na npr. 75% skupa za treniranje i npr. 25% skupa za testiranje. Značajke i ciljevi pružaju informaciju gdje podaci idu: X_train, X_test, y_train i y_test. X_train i y_train se koriste za treniranje modela, a X_test i y_test za testiranje.



Slika 3.5. Train, test, split proces [9]

Podjela podataka prikazana je u nastavku koda.

```
# Podjela na trening, testiranje i validaciju
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=42)
```

Nakon toga kreira se „OneHotEncoder“ objekata i pretvara se podatke u ciljne podatke.

```
# Kreiranje OneHotEncoder objekta i pretvaranje ciljnih podataka

x_train_reshaped = x_train.reshape(x_train.shape[0], -1)
y_train_reshaped = y_train.reshape(y_train.shape[0], -1)
y_test_reshaped = y_test.reshape(y_test.shape[0], -1)
y_val_reshaped = y_val.reshape(y_val.shape[0], -1)
```

Ovim komandama, višedimenzionalno polje pretvaramo u jednodimenzionalno.

```
# Ravel ciljnih podataka
y_train_reshaped = np.ravel(y_train_reshaped)
y_test_reshaped = np.ravel(y_test_reshaped)
y_val_reshaped = np.ravel(y_val_reshaped)
```

Koristi se OneHotEncoder iz knjižnice „scikit-learn“koje kategoričke varijable pretvara u „OneHot“ varijable. Naredbom „fit_transform()“ primjenjuje se enkoder na „y_train_reshaped“ (ciljnu varijablu).

```
encoder = OneHotEncoder()
y_encoded = encoder.fit_transform(y_train_reshaped.reshape(-1, 1))
```

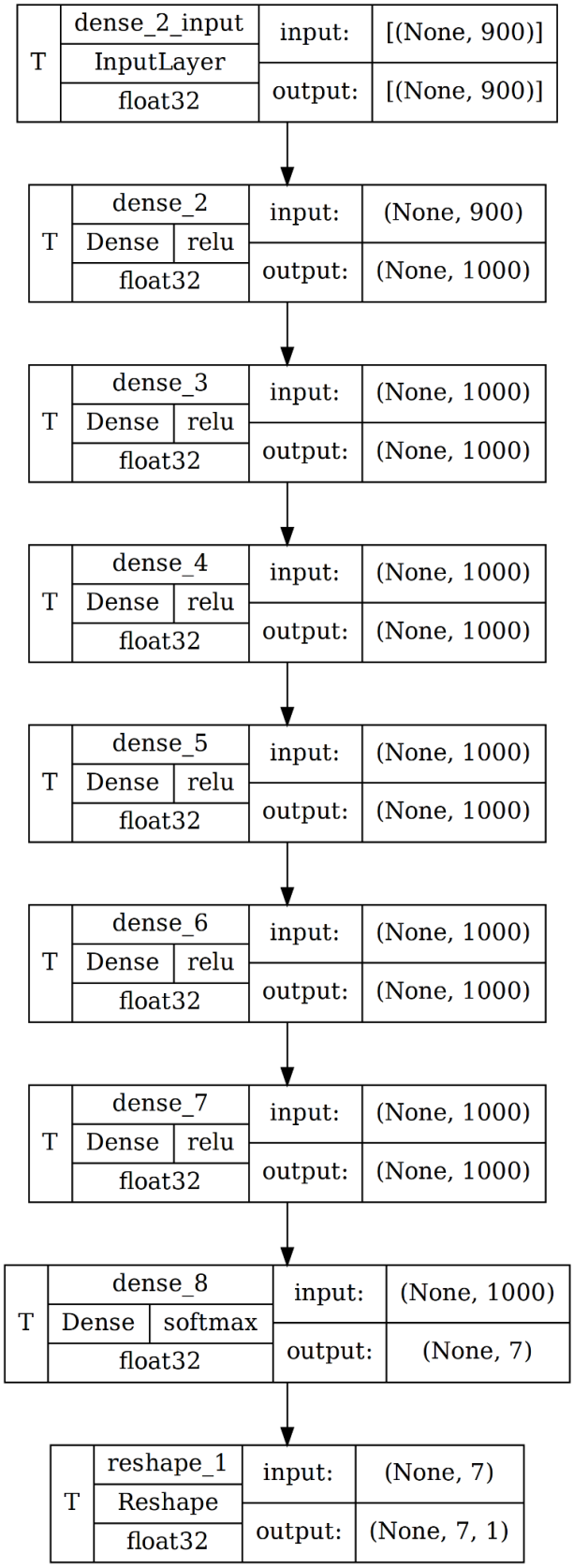
Kreira se model „MLPClassifier-a“ gdje se definira „max_iter“ tj. broj epoha. Nakon toga koristimo parameter: „hidden_layer_sizes”, „activation”, „solver“ i „learning rate“. „Hidden_layer_sizes“ parametrom specificiramo broj slojeva te broj čvorova u neuronskoj mreži, svaki element u tuple-u

predstavlja broj čvorova na i-toj poziciji gdje se nalazi i indeks tuple-a. Duljina tuple-a označava ukupan broj skrivenih slojeva u neuronskoj mreži. „Activation“ je parametar koji indicira aktivacijske funkcije za skrivene slojeve. „Solver-om“se specificiraju težine za optimizaciju preko čvorova. „Learing_rate“ (stopom učenja) hiperparametrom upravljamo koliko koraka model izvrši u smjeru optimizacije tijekom svake epohe. Postavljen je na listu 2 moguće vrijednosti: 'constant' koja označava korištenje iste stope učenja tijekom cijelog postupka učenja i 'adaptive' koji označava stopu učenja u kojoj će se modelu prilagoditi i mijenjati stopa učenja u svrhu optimizacije te ovakav uvjet može pomoći modelu da brže postigne bolje rezultate (tj. brza konvergencija). Ovi parametri će se mijenjati tijekom izvršavanja koda, s ciljem postizanja najboljih krajnjih rezultata.

```
mlp_gs = MLPClassifier(max_iter=5000000)

# Define hyperparameter grid
parameter_grid = {
    'hidden_layer_sizes': [(5,), (10,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
```

Na slici 3.6. se nalazi sekvencijalni model MLP-a dobiven pomoću Keras knjižnice. Definira se prvo ulazni sloj, a onda se lanac slojeva poveže na ulazni sloj. Sastoji se od 9 slojeva i svaki sloj je predstavljen s nazivom sloja i ulaznom i izlaznom dimenzijom. Slojevi su povezani međusobno strelicama. Prvi sloj je ulazni sloj s ulaznom dimenzijom None,900 i izlaznom dimenzijom None,900. To znači da ulaz u mrežu može imati bilo koji „batch size“ predstavljen sa „None“, a svaki primjer ulaza ima 900 značajki. Idućih 6 slojeva čine gusti slojevi i koriste ReLu aktivacijsku funkciju. Ovo su potpuno povezani slojevi gdje je svaki neuron u trenutnom sloju povezan sa svim neuronima prethodnog sloja. Osmi sloj je gusti sloj koji koristi Softmax aktivacijsku funkciju. Ovaj sloj ima ulaznu dimenziju None, 1000 i izlazni sloj None, 7. Posljednji sloj ima ulazne dimenzije None, 7 i izlazne dimenzije None,1 i taj sloj preoblikuje izlaz iz prethodnog sloja u jednu vrijednost. Moguće je zaključiti da je arhitektura dizajnirana za klasifikaciju 7 klasa.

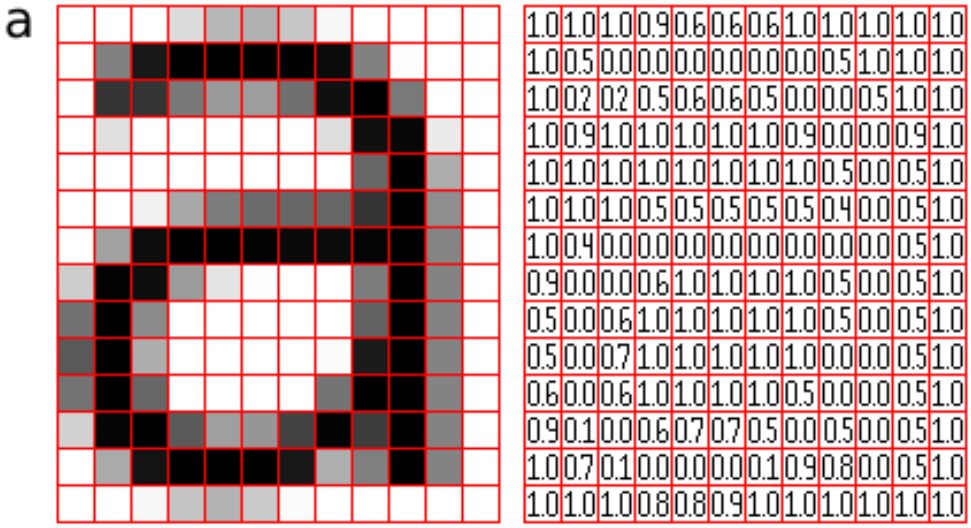


Slika 3.6. Arhitektura MLP modela

3.4. Konvolucijske neuronske mreže (CNN)

Konvolucijska neuronska mreža (*eng. Convolutional Neural Network*) poznata kao CNN ili ConvNet posebna je vrsta višeslojne neuronske mreže. Koristi se kod dubokog učenja u području računalnog vida. CNN je vrsta „feedforward“ neuronske mreže i koristi konvolucijsku strukturu za izvlačenje potrebnih značajki iz matričnog skupa podataka topologije koja izgleda kao mreža tj slika. Primjena CNN-a može se pronaći u nekoliko različitih područja poput: obrade slika, konkretno za klasifikaciju ili detekciju objekata, segmentaciju slika te prepoznavanje glasa ili otisaka prstiju. Digitalna slika se definira kao binarni prikaz vizualnih podataka, a sastoji se od niza piksela raspoređenih poput mreže.

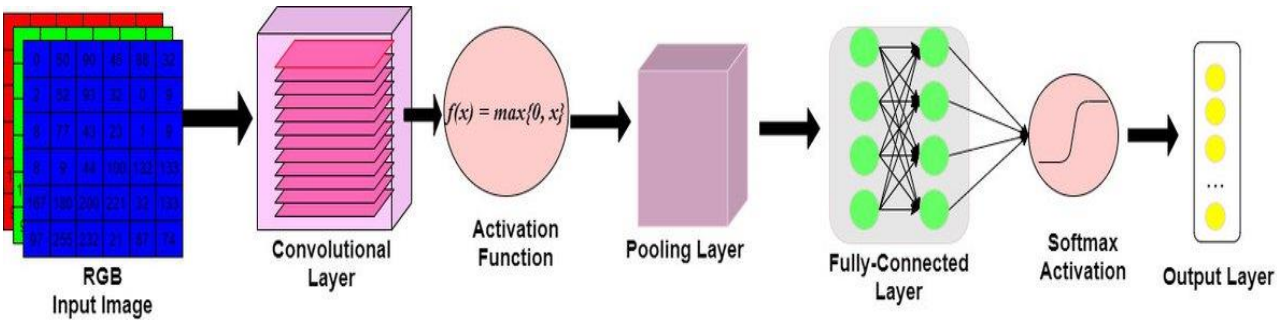
Na slici 3.7. nalazi se primjer jedne slike i mreže njezinih piksela. Važno je povezati doživljaj slike u ljudskom mozgu tj. reakciju neurona na podražaje u receptivnom polju biološki s CNN-om u svom receptivnom polju. Kod klasifikacije slika u prvim slojevima otkriva se rub, nakon toga jednostavniji oblici u drugom sloju, dok u višim slojevima se prepoznaju značajke.



Slika 3.7. Prikaz slike (slova) sa mrežom vrijednosti piksela [10]

Osnovne komponente CNN arhitekture su: ulazni sloj, konvolucijski sloj, potpuno povezani slojevi, sloj sažimanja i aktivacijska funkcija. Ulazna slika određenih dimenzija predstavlja ulazni sloj, točnije pretpostavka slike u boji. RGB slike čine tri dimenzije; visina, širina i dubina. Mreža ima sposobnost prepoznati sliku u dijelovima i može izvršiti operacije više puta kako bi dovršila potpunu obradu slike. Obrada slike uključuje konverziju slike iz RGB ili HSI skale u sivu skalu. Konvolucijski sloj je prvi sloj koji ima funkciju izdvajanja različitih značajki iz ulaznih slika, izvršava se matematička operacija konvolucije između ulazne slike i filtra određene veličine ili kernela. Filtar je najčešće matrica veličine 3 x 3, a prolazi kroz receptivna polja slike i detektira prisutne značajke. Detektor značajki je dvodimenzionalan (2-D) niz težina i dio je slike. Izlaz iz ovog sloja predstavlja mapu značajki koja sadrži informacije o slici poput rubova i kutova, te se nakon toga mapa značajki prenosi do drugih slojeva. Potrebno je transformirati linearne mape značajki u nelinearne.

Aktivacijska funkcija služi za učenje i aproksimaciju nekog kontinuiranog i složenog odnosa između varijabli mreže. Najčešće se koristi Softmax, a ostale aktivacijske funkcije koje se mogu koristi su ReLu, tanH i Sigmoidalna funkcija. Sloj sažimanja (*eng. pooling layer*) iz prethodnog sloja smanjuje prostornu dimenzionalnost značajki tako da smanjuje rezoluciju uz zadržavanje i izdvajanje dominantnih značajki. Max-pooling i average pooling se najčešće koriste. Max pooling uzima piksel s najvećom vrijednošću iz mape značajki dok average pooling izračunava prosjek elemenata unutar receptivnog polja za slanje u izlazni sloj. Potpuno povezani sloj se još naziva i gustim slojem (*eng. Dense Layer*) i ima sličnosti s višeslojnim perceptronom (MLP-om). U potpuno povezanom sloju svaki neuron je povezan sa svakim slojem u prethodnom sloju, te dodjeljuje jedinstvene težine za svaku ulaznu vrijednost. Na slici 3.8. nalazi se prikaz ovakve detaljno opisane arhitekture. Također, važno je još spomenuti ako dođe do overfittinga u procesu treniranja, postoji sloj ispadanja (*eng. Dropout Layer*) koji nasumično ispušta neurone iz neuronskih mreže. Ovakav pristup dovodi do smanjene veličine modela te poboljšanja performansi pojednostavljivanjem neuronske mreže.



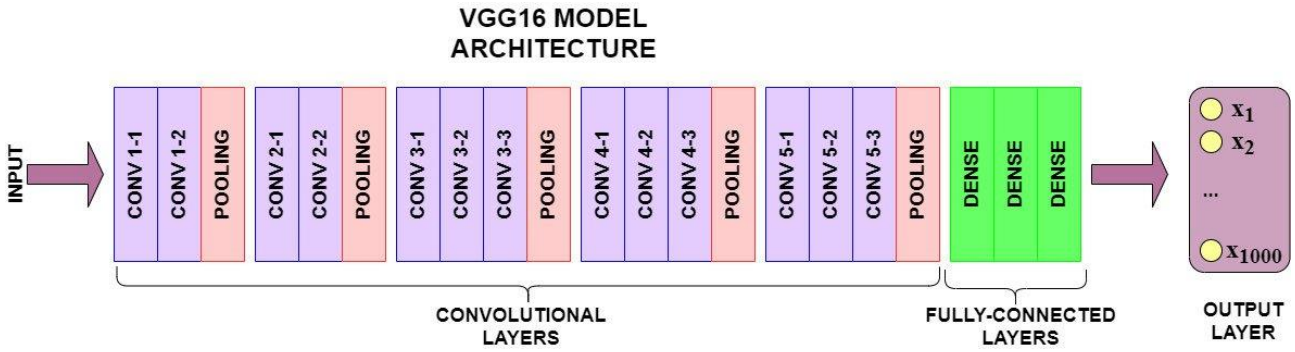
Slika 3.8. CNN arhitektura [11]

Neke od prednosti korištenja CNN-a su, manja opsežnost parametara dovodi do jednostavnijeg treniranja, efikasnost u obradi slike, prepoznavanje i klasifikacija objekata, automatizacija izvlačenja značajki te jednostavnija implementacija u velike mreže za razliku od modela umjetne neuronske mreže (ANN). S druge strane, potrebne su visoke performanse grafičkih kartica i ogromne količine memorije, duboke mreže mogu imati veliki broj parametara, poteškoće s malim setovima podataka mogu dovesti do pretreniranosti mreže. Postoje razne arhitekture poput: LeNet-5, AlexNet, VGGNet, GoogLeNet i ResNet.

3.5. VGG16 model

VGG16 (*eng. Visual Geometry Group 16*) jedan je od najpoznatijih CNN modela koji se koristi za klasifikaciju slika i detekciju objekata. Razvili su ga istraživači Karen Simonyan i Andrew Zisserman iz Visual Geometry grupe na Sveučilištu Oxford [12]. Ovaj model se smatra prekretnicom u području dubokog učenja i računalnog vida. Istraživači su došli do toga da se performanse modela mogu poboljšati ako se poveća dubina arhitekture. Upotrijebili su konvolucijske slojeve s jako malim filtrom tj. filtar veličine 3 x 3 umjesto velikog broja hiperparametara.

Na slici 3.9. nalazi se prikaz arhitekture VGG16 modela. VGG16 model sastoji se od 16 slojeva, od kojega 13 čine konvolucijski slojevi, a 3 su potpuno povezana sloja. Slika veličine 224 x 224 s tri kanala R,G,B predstavlja ulaz u mrežnu konfiguraciju. Predobradu podatka čini normalizacija RGB vrijednosti za svaki piksel. Unutar konvolucijskih slojeva, nalaze se i slojevi maksimalnog sažimanja, dok se kod potpuno povezanih slojeva može pronaći 3 gusta sloja. Svaki konvolucijski sloj koristi filtre kako bi iz slike izvukao različite detalje, a u slojevima sažimanja modelom se smanjuje broj parametara i vremenska složenost modela. Najčešće se koristi ReLU aktivacijska funkcija za postizanje nelinearnosti u izlaznim slojevima.



Slika 3.9. Općenita arhitektura VGG16 modela [13]

Ovakav model može klasificirati 1000 slika od 1000 različitih kategorija s čak 92.7% točnosti, a istreniran je na „ImageNet“ skupu podataka. ImageNet je skup podataka oko preko 15 milijuna slika visoke rezolucije koje pripadaju u otprilike 22 000 kategorija. Sadrži oko 1,2 milijuna slika za obuku, 50 000 slika za provjeru ispravnosti te 150 000 slika za testiranje [14].

Općenito, VGG16 arhitektura sadrži prvi i drugi konvolucijski sloj koji čine 64 filtera jezgri značajki, veličine 3 x 3. Kada ulazna slika prođe kroz prvi i drugi konvolucijski sloj, nove dimenzije su 224 x 224 x 64. Nakon toga se izlaz prosljeđuje sloju maksimalnog sažimanja s korakom od 2. Treći i četvrti konvolucijski sloj sastoje se od 124 kernel filtera značajki veličine 3 x 3. Iduća dva sloja čine slojevi maksimalnog sažimanja s korakom 2 koji smanjuju veličinu na 56 x 56 x 128. Peti, šesti i sedmi konvolucijski sloj koriste 256 mapi značajki, isto jezgre veličine 3 x 3. Slojevi maksimalnog sažimanja čine korak 2. Od osmog do trinaestog konvolucijskog sloja postoje dva seta jezgre veličine 3 x 3 te imaju 512 kernel filtera. Nakon toga slijedi sloj maksimalnog sažimanja s korakom 1. Četrnaesti i petnaesti sloj su potpuno povezani skriveni slojeva od 4096 jedinica. Šesnaesti sloj čini 1000 jedinica, Softmax izlazni sloj.

3.5.1. Implementacija VGG16 modela

Prvi korak je da učitamo potrebne knjižnice.

```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
```

U podatke učitavamo izolirani „numpy“ set podataka koji sadrži slike i provjeravamo njegov oblik.

```
data = np.load('dataset_slike.npy', allow_pickle=True)
print(data.shape)  # (4020, 224, 224, 3)
```

Iz slika izdvajamo sve elemente niza svih dimenzija, dok za oznake (*eng.labels*) uzimao vrijednosti iz prvog elementa po dimenzijama od 1-3 po 0-toj dimenziji.

```
images = data[:, :, :, :]
labels = data[:, 0, 0, 0]
```

U idućoj liniji koda, na niz slika primjenjuje se obrada ulaznih podataka tj. slika kako bi ih se pripremilo na daljnju obradu ili analizu. Vrijednosti se prilagođavaju formatu koji je prigodan za ovaj model.

```
images_edit = preprocess_input(images)
```

Varijablama x i y dodjeljujemo slike i oznake.

```
x = images_edit
y = labels
```

Nakon toga, izvršava se podjela podataka na treniranje, testiranje i validaciju kao i kod MLPClassifier modela.

```
# Train, test and validation data
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,
random_state=42)
train_x, val_x, train_y, val_y = train_test_split(train_x, train_y,
test_size=0.2, random_state=42)
```

U ovom koraku učitavamo već istrenirani VGG model koji sadrži težine iz “ImageNet” mreže.

```
# Load pre-trained VGG model
model = VGG16(weights='imagenet')
```

Pretvaramo varijable: „train_y“, „test_y“ i „val_y“ u „OneHot“ vektore što utječe na bolju prilagodbu podataka tijekom treniranja i evaluacije modela. Ovakav način omogućuje modelu da preciznije razlikuje klase i lakše obavlja proces klasifikacije.

```
# Pretvaranje oznaka u one-hot vektore
train_y = to_categorical(train_y, num_classes=1000)
test_y = to_categorical(test_y, num_classes=1000)
val_y = to_categorical(val_y, num_classes=1000)
```

Ispisujemo oblike podataka zbog provjere dimenzija.

```
# Print shape
print("Train images shape:", train_x.shape)
print("Train labels shape:", train_y.shape)
print("Test images shape:", test_x.shape)
print("Test labels shape:", test_y.shape)
print("Validation images shape:", val_x.shape)
print("Validation labels shape:", val_y.shape)
```

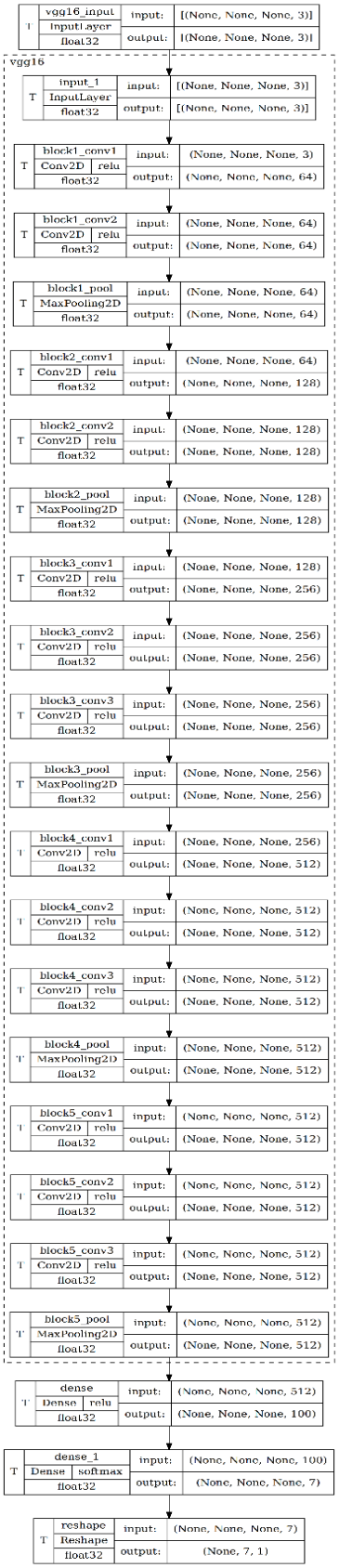
Model se prvo kompilira tako što se parametri postavljaju tako da odgovaraju procesu treniranja modela. Za parametar „optimizer“ odabire se 'adam' koji adaptivno kombinira različite optimizacijske tehnike. Kod funkcije gubitaka, odabire se 'categorical_crossentropy' zbog višeklasne klasifikacije, za metriku se odabire točnost ('accuracy').

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

Za treniranje modela, koristi se naredba „model.fit“ gdje se koriste ulazne varijable „train_x“ i „train_y“. Broj epoha označava koliko je cijeli skup za treniranje prošao kroz model, a „batch_size“ označava koliko primjera se koristi u svakom koraku optimizacije. Parametar „verbose“ određuje nakon svakih koliko epoha će se ispisivati informacije o procesu, a parametrom „validation_data“ se prate performanse modela tijekom treniranja. Ovi parametri će se mijenjati kako bi se prikazali najbolji rezultati.

```
model.fit(train_x, train_y, epochs=10, batch_size=32, verbose=10,
validation_data=(val_x, val_y))
```

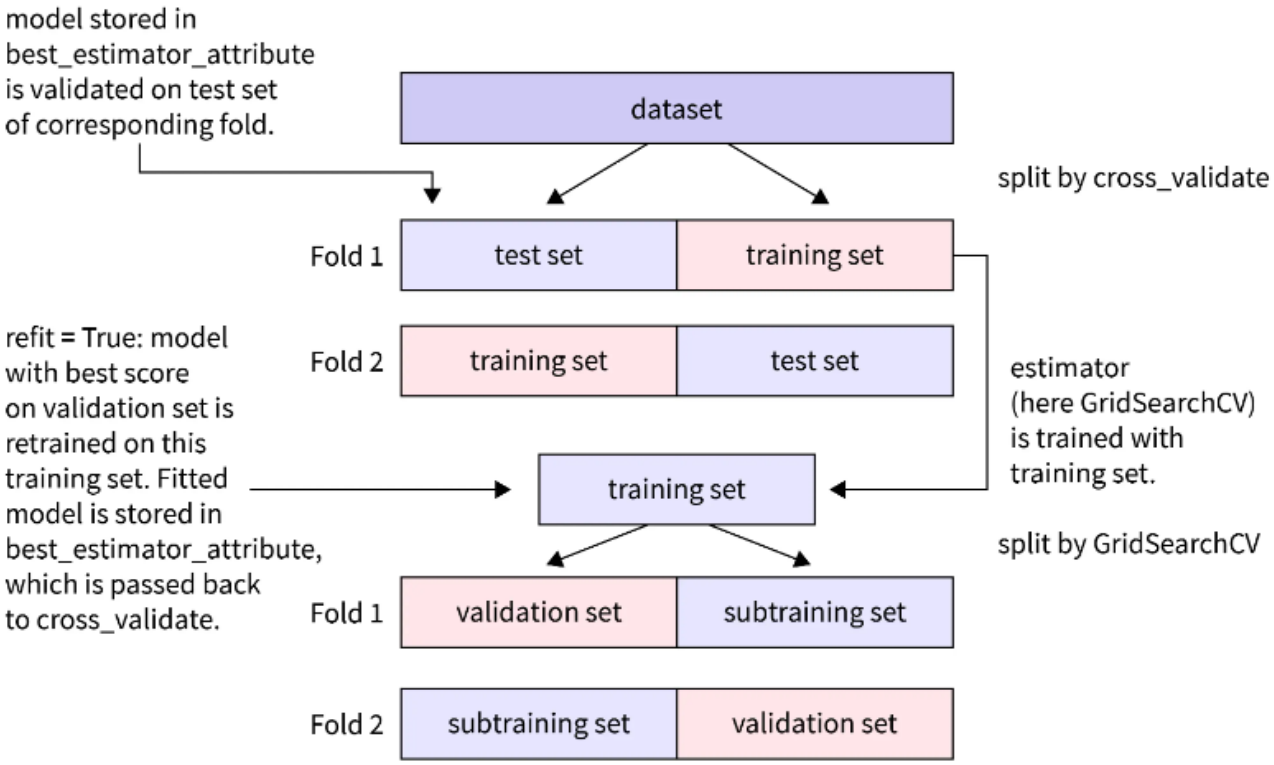
Na slici 3.10. nalazi se prikaz VGG16 arhitekture. Ovaj model sastoji se od 23 slojeva. Na ulazu se nalazi VGG16 ulazni sloj oblika None,None, None, 3. Nakon toga još jedan ulazni sloj s istom dimenzijom ulaza i izlaza kao u prethodnom sloju. 13 je konvolucijskih 2D slojeva koji izvršavaju operaciju konvolucije između ulaznog sloja i tenzora filtera sa ReLU aktivacijskom funkcijom. Općenito, tenzor je struktura koja se koristi za pohranu, prikaz i promjenu podataka. Njime se generalizira vektore i matrice te se prikazuje kao višedimenzionalno polje. Max Pooling slojeva je 5 te se pojavljuju 2 gusta sloja. Jedan gusti sloj koristi ReLU aktivacijsku funkciju dok drugi koristi Softmax aktivacijsku funkciju. Na kraju se nalazi sloj koji oblikuje izlaz u dimenzije None,7,1.



Slika 3.10. Arhitektura VGG16 modela

3.6. GridSearch CV

Hiperparametri se koriste za procjenu optimalnih parametara modela, a za razliku od parametara korisnik određuje vrijednosti hiperparametara. Pretraživanjem mreže (*eng. Grid Search*) koriste se različite kombinacije prethodno specificiranih vrijednosti hiperparametara. Moguće je sveobuhvatno pretraživanje nad određenim parametrima za estimaciju te izračun performanse svake kombinacije. Potrebno je i koristiti metrički sustav procjene za pretraživanje mreže na temelju kojeg će pretraživanje mreže generirati najbolje vrijednost. Pretraživanje mreže je upravljano metrikom izvedbe koja se mjeri unakrsnom validacijom (*eng. Cross-Validation*). Moguća je pojava overfittinga te je moguća loša izvedba na testiranim podacima iako je prethodno bila uspješna na treniranim podacima. Upravo zbog ovih razloga se koristi greška unakrsne provjere valjanosti.



Slika 3.11. Proces unakrsne validacije i GridSearchCV modela [15]

Na slici 3.11. nalazi se dijagram na kojem je prikazan proces izvršavanja unakrsne validacije. Od početnog seta podataka, podaci se dalje dijele na dva preklopa (*eng. fold*): set za testiranje i set podataka za treniranje. Model je treniran na podacima za treniranje i onda testiran na setu za testiranje. Proces se ponavlja za svakih preklop, a za procjenu izvedbe modela rezultati se računaju kao prosjek. Slijedi odabir najboljeg modela pomoću metode GridSearchCV-a koja pretražuje mrežu parametara i odabire najpogodniju kombinaciju hiperparametara za taj model.

Također, raspon vrijednosti s gornjom i donjom granicom se mora definirati. To se najčešće događa kada su vrijednosti hiperparametara kontinuirane. GridSearch automatski koristi unakrsnu validaciju za svaku kombinaciju parametara.

Najčešće korišteni parametri u GridsearchCV-u su:

- Estimator – stvara instancu modela kojeg se koristi,
- parm_grid – se koristi za specificiranje parametara za eksperimentiranje i vrijednosti za te parametre,
- scooring – strategija za procjenu performansi unakrsne - validacije modela,
- n_jobs – izvodi se paralelno, postavljanje n_jobs npr. na -1 omogućuje da model koristi sve procesore na računalu,
- cv – označava parametar unakrsne provjere valjanosti i odlučuje koliko preklopa model koristi za validaciju rezultata, i
- verbose – može se postaviti na 1 kako bi se dobili detaljan ispis dok se podaci setiraju u GridSearchCV.

Ova metoda, ima mogućnost paralelnog rada tj. može u isto vrijeme izvršavati različite kombinacije hiperparametara te se može koristiti kod raznih modela strojnog učenja. S druge strane, GridSearchCV ima nedostatke poput prevelike količine vremena za izračunavanje idealne kombinacije te se može dogoditi prestanak rada zbog nekompatibilnosti sa skupovima podataka koji imaju veliki broj dimenzija. Zbog toga se ova metoda većinom koristi kod manjih seta podataka.

3.6.1. Implementacija GridSearchCV-a

GridSearchCV koristit će se kod klasifikacije numeričkih podataka tako da se učita potrebna knjižnica iz „sklearn.model-a“.

```
from sklearn.model_selection import GridSearchCV
```

Nakon toga se kreira varijabla „cv“ (Cross-Validation) koja sadrži listu s jednim elementom u „x_train“ i „x_val“ podacima. Ova varijabla se kasnije koristi kao pomoć pri odabiru skupa podataka za unakrsnu validaciju tijekom optimizacije.

```
# Create GridSearchCV object
cv = [(x_train.shape[0], x_val.shape[0])]
```

Kreira se objekt „GridSearchCV“ koji se koristi za pretragu po skupu parametara definiranih u „parameter_grid“. Varijablu „n_jobs“ se postavlja na -1 s ciljem optimizacije paralelizirane na raspoloživim prostorima. Postavljanjem „cv“ na 3, koristit će se unakrsna validacija s 3 presjeka, a „refit“ je postavljen na 'True' tako da se najbolji model ponovno trenira na cijelom skupu za treniranje nakon završene optimizacije.

```
clf = GridSearchCV(mlp_gs, parameter_grid, n_jobs=-1, cv=3, refit=True)
```

Objektom „clf“ pretražuju se hiperparametri modela „mlp_gs“ pomoću podataka za treniranje, varijabla „x_train_reshaped“ i „y_train_reshaped“ za ciljnu varijablu. Varijablom „best_model“ referira se najbolji model koji se tijekom optimizacijskog procesa odabrao.

```
clf.fit(x_train_reshaped, y_train_reshaped)
best_model = clf.best_estimator_
```

3.7. Ulančani sloj (eng. Concatenate layer)

Ulančani sloj (Concatenate layer) je jedna od glavnih komponenti u dubokom učenju, a ima važnu ulogu u izgradnji kompleksnih arhitektura neuronskih mreža. Ulančani sloj je sloj koji povezuje više ulaznih tenzora kako bi se stvorio jedan izlazni tenzor. Ulaz čini popis tenzora istih oblika za razliku od osi ulančavanja koja je drugačijeg oblika. Izlaze nekoliko slojeva je potrebno kombinirati prije nego što se proslijede na idući sloj. Kod ovakve metode, uzimaju se najbolji parametri koji su pojedinačno dali najbolje rezultate i spajaju se upotrebom „Concatenate“ sloja. Cilj ove metode je omogućiti modelu da nauči kombinirati informacije iz različitih slojeva ili izvora podataka za dobivanje najpreciznijih i najtočnijih rezultata.

Postupak u programskom kodu je da se prvo učitaju potrebne knjižnice.

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import plot_model
import keras
from keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import activations
from keras.layers import Activation, Dense, concatenate, Reshape
```

Nakon toga, stvara se varijabla modela VGG16 gdje se definira broj klasa s indikacijom da posljednji sloj VGG16 modela ima 7 izlaznih čvorova. Varijablom „include_top“ postavljamo tako da se ne uključi posljednji sloj originalnog VGG16 modela.

```
model_vgg16 = VGG16(classes=7, include_top=False)
```

U nastavku je prikazana inicijalizacija modela te dodavanje različitih slojeva s aktivacijskim funkcijama.

```
model_1 = keras.Sequential()
model_1.add(model_vgg16)
model_1.add(Dense(100, activation='relu'))
model_1.add(Dense(7, activation='softmax'))
model_1.add(Reshape((7, 1)))
```

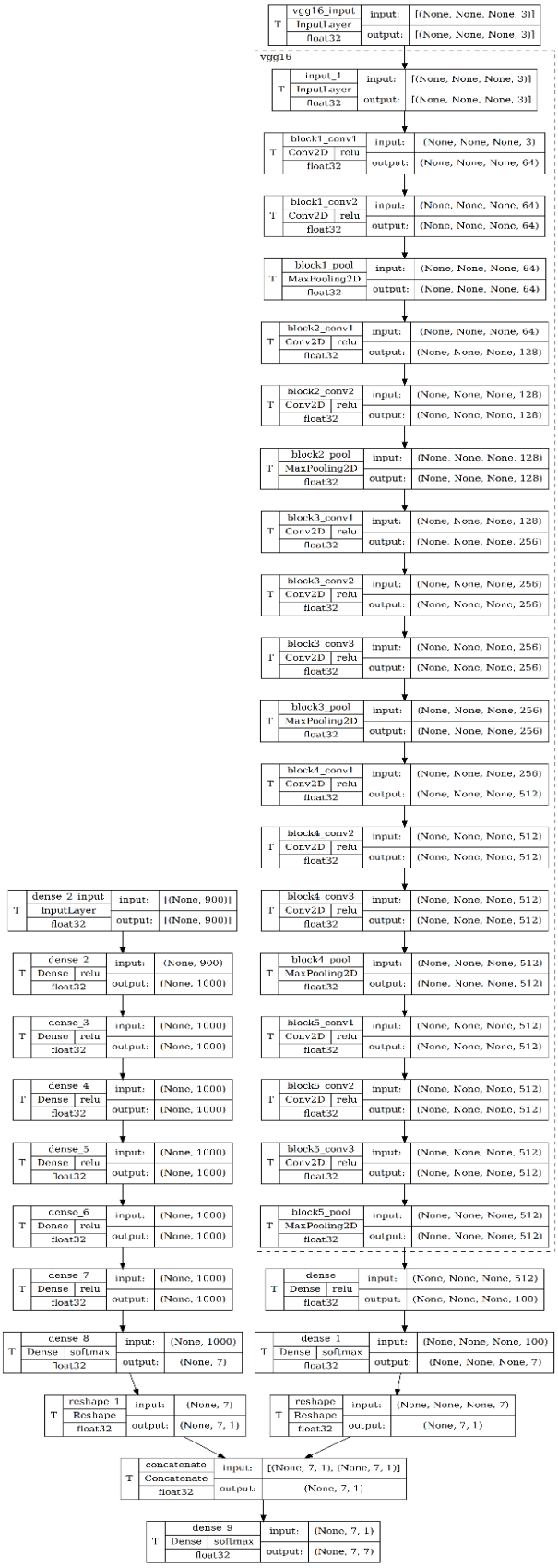
Kreira se još jedna prazna instanca sekvencijskog modela i dodjeljuju se slojevi, broj neurona i aktivacijska funkcija. Također, na kraju se vrše i operacije preoblikovanja.

```
model_2 = keras.Sequential()
model_2.add(Dense(1000, input_shape=(900,), activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(7, activation='softmax'))
model_2.add(Reshape((7, 1)))
```

Kreira se novi tenzor „combined“ kojim se ulančava izlaze iz modela jedan i modela dva. U idućim linijama stvaraju se novi potpuno povezani slojevi sa 7 izlaznih neurona, „softmax“ aktivacijskom funkcijom te izlaznim tenzorom „combined“. Za model tri, ulazi su modeli 1 i 2, a izlaz prethodno definiran model 3.

```
combined = concatenate([model_1.output, model_2.output], axis=0)
model_3 = Dense(7, activation="softmax")(combined)
model_3 = Model(inputs=[model_1.input, model_2.input], outputs=model_3)
```

Na slici 3.12. se nalazi prikaz kombinirane arhitekture VGG16 i MLP-a. Pomoću Concatenate sloja spojene su mreže tako da su uzeti najbolji hiperparametri iz svake pojedine mreže. Mreža je prethodno opisana sve do zadnja dva sloja. Predzadnji sloj čini Concatenate sloj koji uzima dva vektora dimenzija None,7,1 i spaja ga u jedan vektor dimenzija None,7,1. Nakon toga se povezuje na posljednji gusti sloj Softmax aktivacijske funkcije i izbacuje izlaz dimenzija None,7,7. Ovakav pristup doprinijeti će najboljim rezultatima upravo zbog kombinacije dvije različite metode.



Slika 3.12. Kombinacija VGG16 i MLP arhitekture (Concatenate layer)

4. REZULTATI

Kako bi se mogli prikazati rezultati potrebno je izraditi izvješće o klasifikaciji metrika (*eng. classification repor*t). Koristi se za prikaz 4 ključna parametra: accuracy, precision, recall i F1 Score. S ovim parametrima se iznosi procjena performansi određenog modela. Pomoću Pythona i knjižnice „sklearn“ mogu se ispisati rezultati za svaki parametar. Postoje četiri načina kako bi se provjerilo jesu li predviđanja točna ili pogrešna, a to su [16]:

1. TN / True Negative: slučaj je bio negativan i predviđeno negativno,
2. TP / True Positive: slučaj je bio pozitivan i predviđeno pozitivno,
3. FN / False Negative: slučaj je bio pozitivan, ali je predviđeno negativno, te
4. FP / False Positive: slučaj je bio negativan, ali se predviđao pozitivan.

Preciznost (*eng. Precision*) se može definirati kao omjer pravih pozitivnih rezultata i zbroja istinitih i lažnih pozitivnih rezultata.

$$Preciznost = \frac{Tp}{Tp+Fp} \tag{4.1.}$$

Točnost (*eng. Accuracy*) predstavlja zbroj istinitih pozitivnih i istinitih negativnih uzoraka podijeljen sa ukupnim brojem uzoraka. Ovakva formula vrijedi samo ako je model uravnotežen.

$$Točnost = \frac{Tp+Tn}{Tp+Tn+Fp+Fn} \tag{4.2.}$$

Odziv (*eng.Recall*) je sposobnost klasifikatora da pronađe sve pozitivne instance, definira se kao omjer istinitih pozitvnih rezultata i zbroja istinskih pozitivnih i lažno negativnih rezultata.

$$Odziv = \frac{Tp}{Tp+Fn} \tag{4.3}$$

F1 score predstavlja težinsku sredinu preciznosti i odziva-a tako da je najbolji rezultat 1.0, a najgori 0.0.

$$F1 = \left(\frac{2}{recall^{-1}+preciznost^{-1}}\right) = 2 \cdot \frac{preciznost \cdot recall}{preciznost+recall} \tag{4.4}$$

Kod numeričkih podataka koristio se „classification report“ iz knjižnice „sklearn.metrics“ i kod je prikazan u nastavku. Pomoću njega se evaluira performanse klasifikacijskih modela. Ispisuju se prethodno opisane metrike za svaku klasu. Varijabla „y_test_reshaped“ sadrži stvarne tj. ciljne vrijednosti (klase) za testni skup podataka, varijabla „y_pred“ sadrži niz predviđenih klasa koji se generirao pomoću klasifikacijskog modela za testni skup podataka.

```
from sklearn.metrics import classification_report

print(classification_report(y_test_reshaped, y_pred, zero_division=1.0))
```

Kod klasifikacije slika koristi se predikcija na određenom skupu podataka. Za svaku predikciju se izračunava indeks klase kojoj model pridaje najveću vrijednost. Koriti se naredba „argmax“. Za sva 4 parametra u nastavku: accuracy, recall, precision i F1 score uspoređuju se stvarne oznake („test_y“) s predviđenim oznakama („predicted_labels“) i izračunava se postotak kojim su točno predviđeni modeli.

```
# Convert predictions to discrete labels
predicted_labels = np.argmax(predicted_y, axis=1)

# Calculate accuracy
accuracy = accuracy_score(test_y, predicted_labels)
print("Accuracy:", accuracy)


# Pretvaranje predikcija u diskretne oznake
predicted_labels = np.argmax(predicted_y, axis=1)
test_labels = np.argmax(test_y, axis=1)

# Izračun preciznosti
```

```
accuracy = accuracy_score(test_labels, predicted_labels)
print("Preciznost (Accuracy):", accuracy)

# Izračun mjere odziva
recall = recall_score(test_labels, predicted_labels, average='macro')
print("Mjera odziva (Recall):", recall)

# Izračun mjere preciznosti
precision = precision_score(test_labels, predicted_labels,
average='macro')
print("Mjera preciznosti (Precision):", precision)

# Izračun F1-mjere
f1 = f1_score(test_labels, predicted_labels, average='macro')
print("F1-mjera:", f1)
```

Također, koristio se „zero_division“ kod odziva i preciznosti kako bi se odredilo što će se dogoditi ako ne bude stvarno pozitivnih (TN) i stvarno negativnih (TN) rezultata. Za određenu klasu, postavlja se „zero_division“ na 1 kako bi se izbjegla pojava 0 u brojniku ili nazivniku. Pretpostavka je da je model savrršen u predviđanju klase i to olakšava analizu i usporedbu performansi modela.

```
# Izračun mjere odziva s definiranim zero_division parametrom
recall = recall_score(test_labels, predicted_labels, average='macro',
zero_division=1)
print("Mjera odziva (Recall):", recall)

# Izračun mjere preciznosti s definiranim zero_division parametrom
precision = precision_score(test_labels, predicted_labels,
average='macro', zero_division=1)
print("Mjera preciznosti (Precision):", precision)
```

U tablici 4.1. nalaze se pojedini hiperparametri koji su korišteni u MLP metodi dobiveni pomoću pretraživanja mreže (GridSearchCV). Kod MLP metode hiperparametri koji se koriste su: broj neurona, broj slojeva, aktivacijska funkcija i stopa učenja. Za vrijednost neurona od 1000 i 6 slojeva postignuti su zadovoljavajući rezultati. Odabrane su tri aktivacijske funkcije, a to su: ReLU, Logistic i Identity. Najbolji rezultati dobiveni su pomoću ReLU aktivacijske funkcije i vrijednosti od 0.25 od odabranih za stopu učenja pri treniranju.

Tablica 4.1. Korišteni hiperparametri i prikaz najboljeg rezultata za MLP

Hiperparametri		**Najbolji rezultat**
Neurons	5, 10, 25, 50, 100, 250, 500, 1000, 2000	1000
Layers	1, 2, 3, 4, 5, 6	6
Activations	ReLU, Logistic, Identity	ReLU
Learning rate	0.5, 0.25, 0.1, 0.01, 0.0001, 0.00001	0.25

Izdvojili su se hiperparametri kod VGG16 metode pomoću pretraživanja mreže, a rezultati su prikazani u tablici 4.2. Hiperparametri koji se koriste u VGG16 modelu su: batch size, broj epoha, aktivacijske funkcije i stopa učenja. Najbolji rezultat hiperparametara batch size dobiven je za vrijednost 4, a za broj epoha kao najveća odabrana vrijednost od 50 pokazala se idealnom. Isto tako, u prethodno prikazanoj tablici, gdje se koristila MLP metoda, upotrijebile su se iste tri aktivacijske funkcije kao i kod VGG16 modela. Najbolji rezultati su i ovdje postignuti sa ReLU aktivacijskom funkcijom, dok je vrijednost od 0.01 odabrana kao idealnom za hiperparametar stope učenja.

Tablica 4.2. Korišteni hiperparametri i prikaz najboljeg rezultata za VGG16 model

Hiperparametri		**Najbolji rezultat**
Batch Size	1, 4, 8, 16, 32, 64	4
Epochs	10, 20, 30, 40, 50	50
Activations	ReLU, Logistic, Identity	ReLU
Learning rate	0.5, 0.25, 0.1, 0.01, 0.0001, 0.00001	0.01

Kod metode gdje se koristi Concatenate sloj nije napravljeno pretraživanje mreže, već su se uzeli i spojili najbolji rezultati dobivenim pojedinačnim pretraživanjem mreže za MLP i VGG16 model.

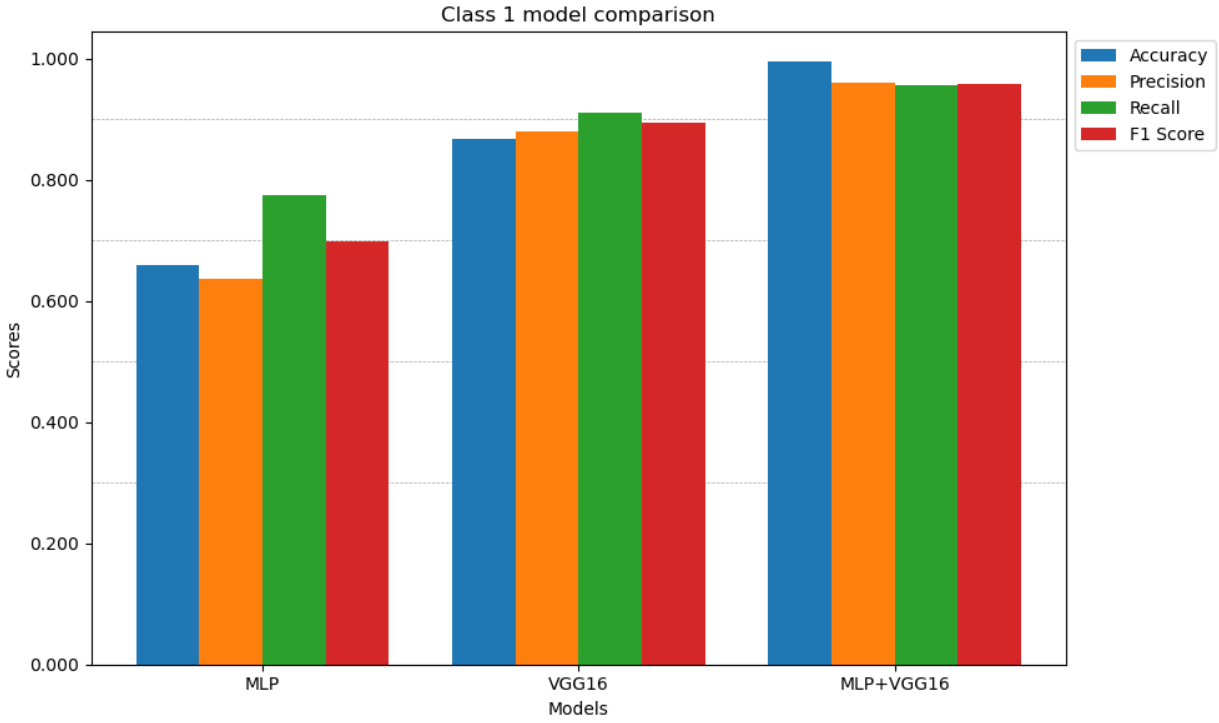
4.1. Klasa 1

U tablici 4.3. nalaze se vrijednosti parametara za sva tri modela korištena u ovom radu. Za točnost su najlošiji rezultati dobiveni MLP modelom, bolji su kod VGG16, a kombinacijom MLP-a i VGG16 modela dobiveni su najbolji rezultati. Kod preciznosti najlošiji su rezultati dobiveni s MLP-om, a najbolji s kombinacijom dva modela. Za odziv, vrijednost iznosi 0.7745 kod MLP-a, a najbolja vrijednost je postignuta kod kombinacije dva modela od 0.9599. F1 score je postignut s najvišom vrijednošću kod dva modela dobivena ulančanim slojem.

Tablica 4.3. Prikaz vrijednosti za klasu 1

	Accuracy	Precision	Recall	F1 score
MLP	0.65860	0.63625	0.77454	0.69861
VGG16	0.86837	0.87894	0.91022	0.89431
MLP+VGG16	0.99433	0.95989	0.95563	0.95775

Na slici 4.1. prikazan je stupičasti grafikon klase 1. Na x osi se nalaze nazivi modela, a na y osi postignute vrijednosti koje je moguće ostvariti. Vidljivo je da MLP metoda daje najlošije rezultate za sva 4 parametra, dok VGG16 metoda daje dosta dobre rezultate. Najbolji rezultati su postignuti kombinacijom VGG16 modela i MLP modela, za točnost je najviša vrijednost, dok su kod ostalih parametara vrijednosti slične.



Slika 4.1. Dijagram modela za klasu 1

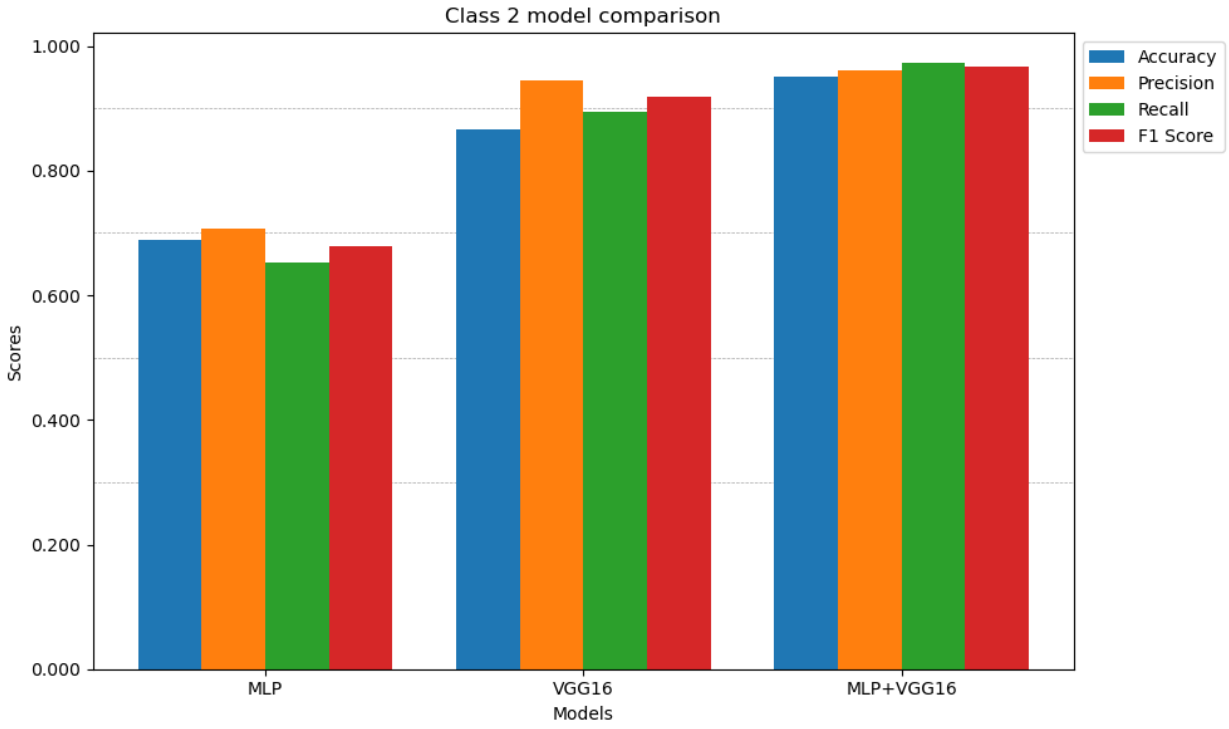
4.2. Klasa 2

Kod klase 2 za MLP model su dobiveni najlošiji rezultati, a kod VGG16 modela su bolji rezultati. Kod VGG16 modela za preciznost je postignuta vrijednost od čak 0.9432. Najbolje rezultate dobilo se kombinacijom dvaju metoda, od kojih je najbolja za parametar odziva od 0.9721. Prikaz se nalazi u tablici 4.4.

Tablica 4.4. Prikaz vrijednosti za klasu 2

	Accuracy	**Precision**	**Recall**	**F1 score**
MLP	0.68854	0.70688	0.65193	0.67829
VGG16	0.86532	0.94422	0.89462	0.91875
MLP+VGG16	0.95108	0.96057	0.97217	0.96634

Na slici 4.2. nalazi se usporedba izvedbe modela za klasu 2. U usporedbi s klasom 1, dobiveni su bolji rezultati kod VGG16 modela kao i kod spoja najboljih hiperparametara metoda (MLP+VGG16). Kod MLP modela nije vidljivo veliko poboljšanje u odnosu na klasu 1.



Slika 4.2. Dijagram modela za klasu 2

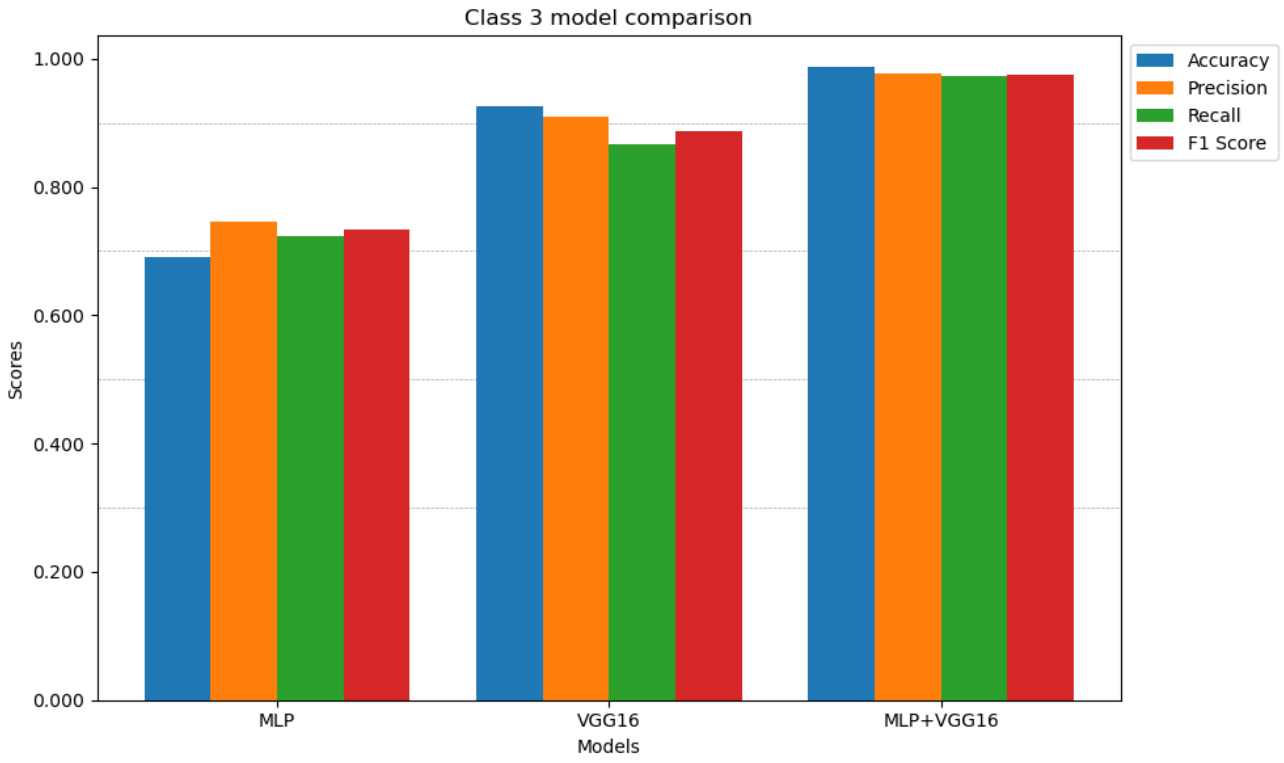
4.3. Klasa 3

Kod klase 3 vrijednosti MLP metode su bolje nego u prethodne dvije klase. Sa VGG16 modelom dobivene su bolje vrijednosti, iako je najlošija kod F1 score-a, a kod točnosti je postignuta vrijednost od 0.9257. Sa hiperparametrima uzetim iz obje metode vrijednosti su najbolje za svaki parametar u tablici 4.5.

Tablica 4.5. Prikaz vrijednosti za klasu 3

	Accuracy	Precision	Recall	F1 score
MLP	0.69054	0.74621	0.72266	0.73425
VGG16	0.92571	0.90891	0.86687	0.88739
MLP+VGG16	0.98633	0.97803	0.97380	0.97591

Na slici 4.3. nalazi se stupičasti dijagram za klasu 3 za svaki pojedini model. Kod klase 3 poboljšale su se vrijednosti parametara u odnosu na prethodno opisane dvije klase. MLP model ima slične vrijednosti kao prije.



Slika 4.3. Dijagram modela za klasu 3

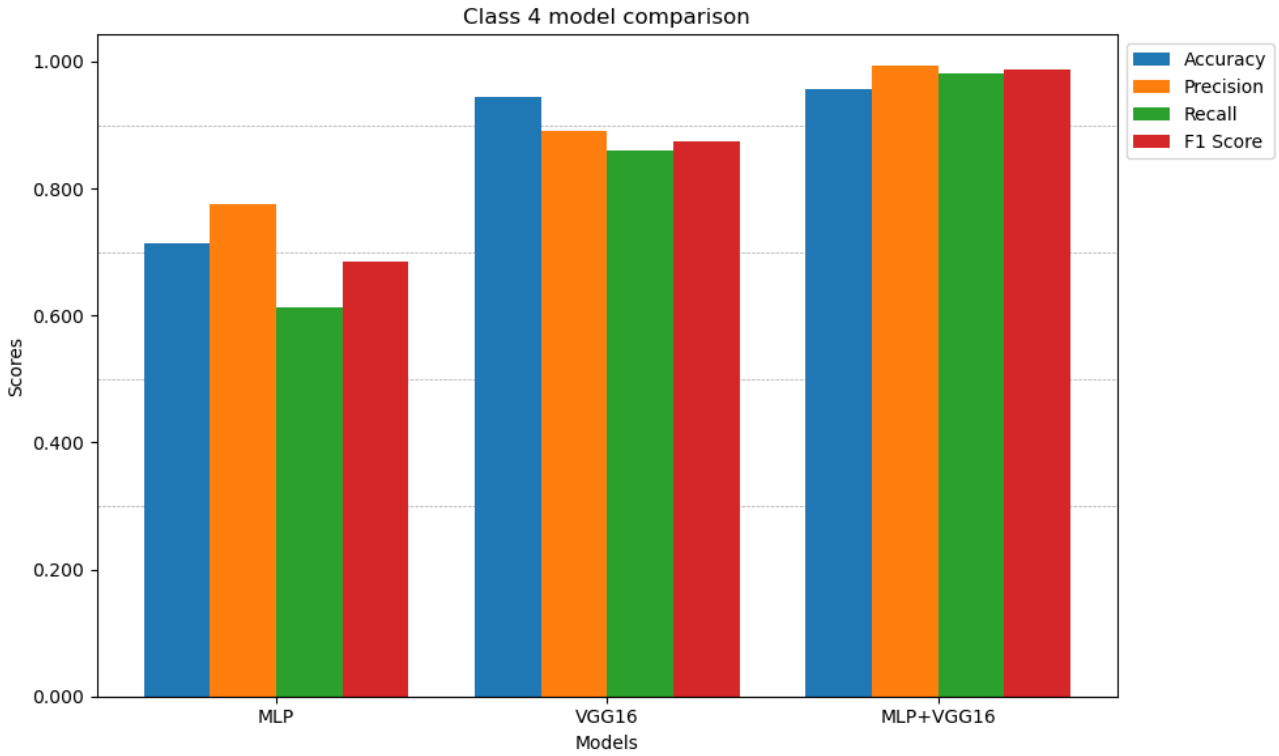
4.4. Klasa 4

Kod MLP metode, najbolji rezultat postignut je kod preciznosti, dok je najlošiji kod odziva. Sa VGG16 metodom najlošija je vrijednost kod odziva, a jako dobra postignuta je kod točnosti. Spajanjem MLP-a i VGG16 modela, dobivene su odlične vrijednosti gdje je najviša kod preciznosti od 0.9928.

Tablica 4.6. Prikaz vrijednosti za klasu 4

	Accuracy	**Precision**	**Recall**	**F1 score**
MLP	0.71373	0.77489	0.61247	0.68417
VGG16	0.94396	0.89023	0.86032	0.87502
MLP+VGG16	0.95710	0.99282	0.98145	0.98710

Grafičkim prikazom vidljiv je odnos svake metode za svaki parametar. VGG16 model ne zaostaje puno iza modela kombinacije najboljih hiperparametara. Najlošiji rezultati dobiveni su MLP metodom. Prikaz se nalazi na slici 4.4.



Slika 4.4. Dijagram modela za klasu 4

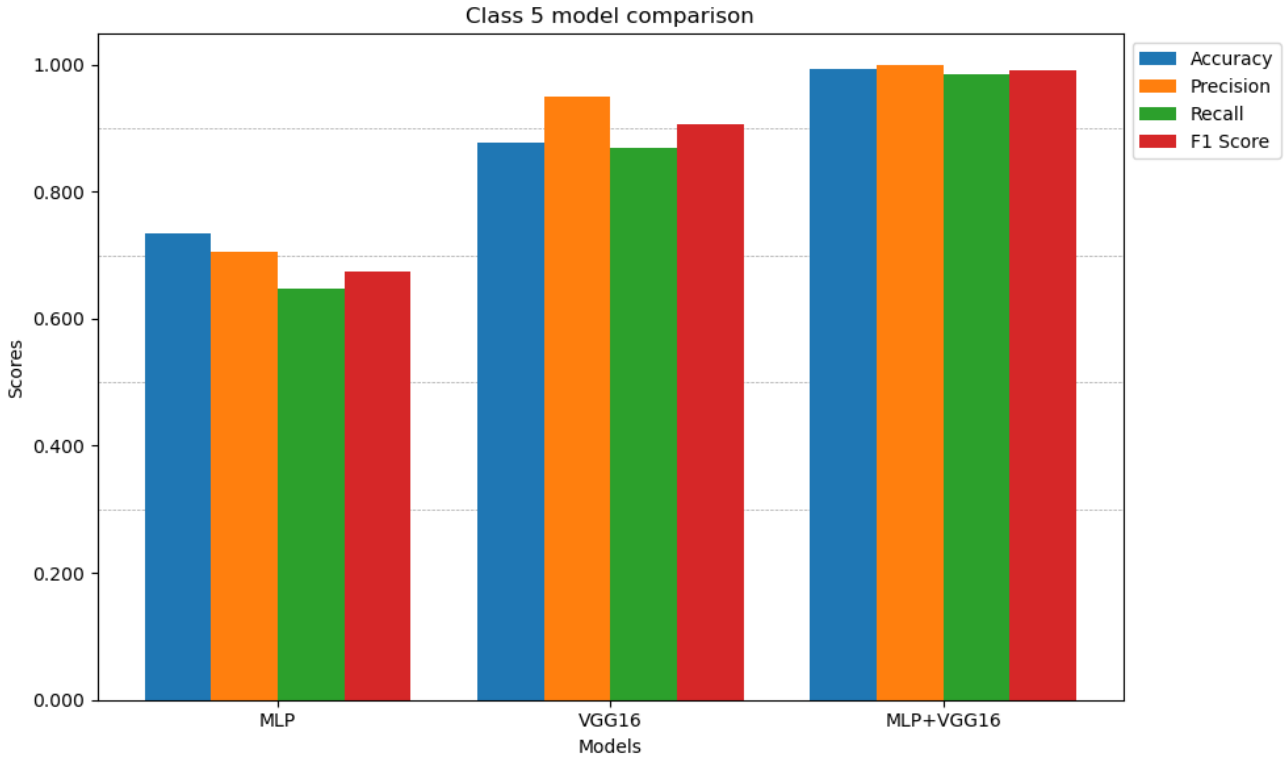
4.5. Klasa 5

Kod klase 5 za MLP metodu dobivene su vrijednosti koje nisu u potpunosti zadovoljavajuće. Najniža je vrijednost za preciznost kod MLP metode. Za VGG16 model vidljivo je poboljšanje u svim parametrima, a najviša vrijednost zabilježena je za preciznost. Korištenjem ulančanog spajanja, za čak 3 parametra dobivene su vrijednosti od 0.99.

Tablica 4.7. Prikaz vrijednosti za klasu 5

	Accuracy	Precision	Recall	F1 score
MLP	0.73475	0.70507	0.64702	0.67480
VGG16	0.87758	0.94888	0.86828	0.90679
MLP+VGG16	0.99354	0.99814	0.98367	0.99085

Stupičastim dijagramom na slici 4.5. može se vidjeti usporedba performansi svakog pojedinog modela. Očekivano najlošiji rezultati su za MLP, a najbolji za kombinaciju MLP-a i VGG16 modela. Kombinacijom su poprilično usklađene vrijednosti za svaki pojedini parametar.



Slika 4.5. Dijagram modela za klasu 5

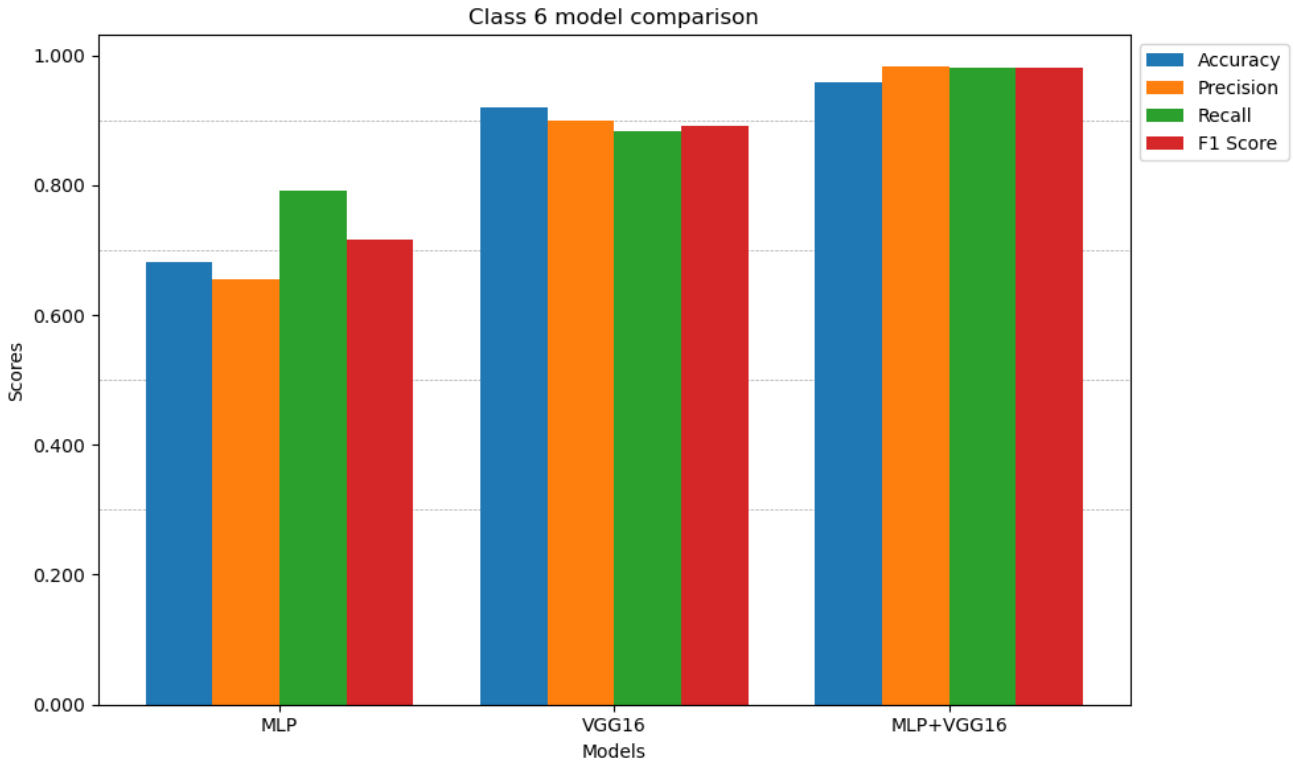
4.6. Klasa 6

U tablici 4.8. prikazane su vrijednosti 4 osnovna parametra za 3 različite metodologije. Najbolje vrijednosti dobivene su uzimanjem najboljih hiperparametara iz primjene MLP metode kao i VGG16 metode. Osrednji rezultati dobiveni su VGG16 modelom, a najniže vrijednosti zabilježene su kod MLP metode.

Tablica 4.8. Prikaz vrijednosti za klasu 6

	Accuracy	**Precision**	**Recall**	**F1 score**
MLP	0.68217	0.65438	0.79159	0.71647
VGG16	0.92050	0.89906	0.88282	0.89087
MLP+VGG16	0.95933	0.98195	0.98088	0.98142

Na slici 4.6. prikazane su vrijednosti u različitim bojama za svaki parametar i svaki pojedini model. Najbolje performanse postignute su kod kombinirane metode tj. spajanjem MLP-a i VGG16 modela. Vrijednosti VGG16 modela ne zaostaju puno iza kombinirane metode, dok su parametri dobiveni MLP metodom dosta lošiji.



Slika 4.6. Dijagram modela za klasu 6

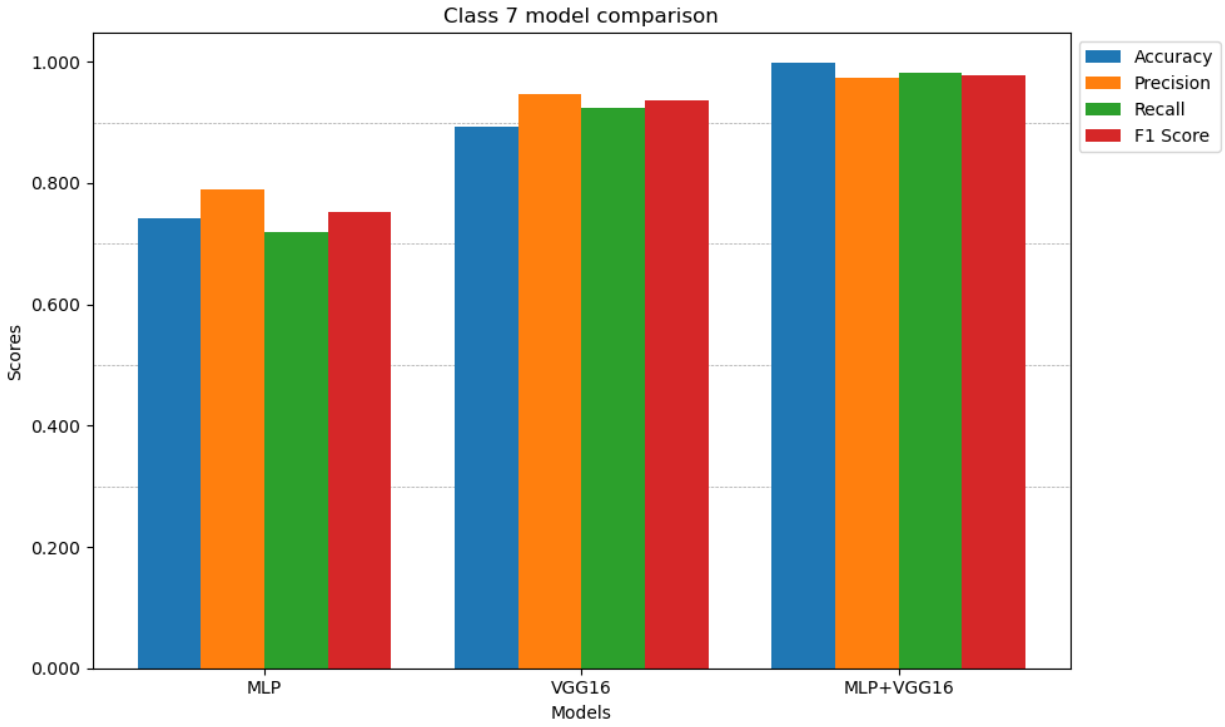
4.7. Klasa 7

U klasi 7 za MLP dobivene su najbolje vrijednosti od svih prethodnih 6 klasa, najniža je vrijednost za odziv od 0.71992, a najviša za preciznost od 0.7888. Dobri rezultati dobiveni su sa VGG16 metodom, a najbolji korištenjem „Concatenate“ sloja.

Tablica 4.9. Prikaz vrijednosti za klasu 7

	Accuracy	Precision	Recall	F1 score
MLP	0.74231	0.78888	0.71993	0.75283
VGG16	0.89370	0.94735	0.92474	0.93591
MLP+VGG16	0.99753	0.97396	0.98198	0.97795

Prikaz dijagrama klase 7 nalazi se na slici 4.7. Modelom VGG16 i Concatenate slojem dobivene su visoke vrijednosti blizu 1.000. MLP vrijednosti kao i do sada u ostalim klasama, zaostaju za ovim dvjema metodama. Kod „Concatenate“ spajanja, najviša vrijednost je kod točnosti, a najniža kod preciznosti.



Slika 4.7. Dijagram modela za klasu 7

4.8. Prosječni rezultati

U tablici 4.10. prikazane su prosječne vrijednosti izračunate za svih 7 klasa. Za parametar točnost, najniži je prosjek kod MLP metode sa vrijednošću od 0.7015. Najviša vrijednost postignuta je s kombiniranom metodom i iznosi 0.97703. Kod ostalih parametara najniža je vrijednost za MLP, a najviša za spoj hiperparametara MLP i VGG16 metode.

Tablica 4.10. Prikaz srednje prosječne vrijednosti za sve klase

	MLP	**VGG16**	**MLP+VGG16**
Accuracy	0.70152	0.89931	0.97703
Precision	0.71608	0.91680	0.97791
Recall	0.70288	0.88684	0.97565
F1 score	0.70563	0.90129	0.97676

5. ZAKLJUČAK

U istraživanju za testni set podataka korištene su metode: propriocepcijske mreže, vizionarske mreže, fuzijska mapa značajki i fuzija odluka. Najveći postotak dobiven je kod vizionarske mreže i iznosi 99.76%. U ovom radu korištene su metode: MLP, VGG16 te kombinacija MLP-a i VGG16 modela, a za svaku klasu napravljene su usporedbe četiri ključna parametara. Najveći postotak od 99.81% dobiven je u klasi 5 kombinacijom dvaju metoda za parametar preciznosti.

Metoda MLPClassifier pokazala se kroz svih sedam klasa kao najlošija metoda. Za razliku od VGG16 modela koji ima duboku strukturu mreže te ima sposobnost izdvajanja složenih značajki iz podataka, kod MLP-a broj neurona i slojeva može biti nedovoljan za učenje složenijih oblika u ovom setu podataka. Također, MLP može biti ograničen u prepoznavanju prostornih oblika i uzoraka te može imati nedovoljno podataka za obuku ili nema predtreniranje na sličnim podacima.

VGG16 model pokazao je dobre krajnje rezultate ali je i proces treniranja bio najduži upravo zbog složenosti mreže. Korištenjem ulančanog sloja kod kojeg su se spojili hiperparametri dvaju metoda dobivene su najbolje vrijednosti u svim klasama. Ovakvo spajanje omogućuje kombiniranje različitih vrsta informacija što utječe na sposobnost modela da nauči kompleksne odnose između značajki te da poboljša svoje performanse.

Općenito, alati dubokog učenja poput klasifikatora u primjeni, proizvođačima olakšavaju predvidjeti potencijalne probleme upravljanja nekim proizvodom. To u konačnici doprinosi smanjenju nedostataka gotovih proizvoda. Također, metrika pomaže kod podešavanja i poboljšanja novih aplikacija s procjenom preciznosti i uspješnosti neke aplikacije.

6. LITERATURA

[1],[2],[3],[4],[5] Yu Chen , Chirag Rastogi, and William R. Norris, Member, IEEE, „A CNN Based Vision-Proprioception Fusion Method for Robust UGV Terrain Classification“, IEEE ROBOTICS AND AUTOMATION LETTERS, VOL. 6, NO. 4, OCTOBER 2021

[6] Bhardwaj A., Kumar V.: „Prediction of Coefficient of Compression of Soil Using Artificial Neural Network, India, 2020.

[7]„Classify Sentences via a Multilayer Perceptron (MLP)“, s Interneta https://austingwalters.com/classify-sentences-via-a-multilayer-perceptron-mlp/,

pristupljeno 10.08.2023.

[8] Kim, Sung Eun & Seo, Il Won: „Artificial Neural Network ensemble modeling with conjunctive data clustering for water quality prediction in rivers“, Journal of Hydro-environment Research, 2015.

[9]„Understanding Train Test Split“, https://builtin.com/data-science/train-test-split, s Interneta, pristupljeno 10.08.2023.

[10] „Convolutional Neural Networks, Explained“, https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, s Interneta, pristupljeno 10.08.2023.

[11]„Convolutional Neural Networks — Image Classification w. Keras“, https://www.learndatasci.com/tutorials/convolutional-neural-networks-image-classification/, s Interneta, pristupljeno 12.08.2023.

[12] Karen Simonyan, Andrew Zisserman, „VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION“, Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015.

[13]„Hands-on Transfer Learning with Keras and the VGG16 Model“, https://www.learndatasci.com/tutorials/hands-on-transfer-learning-keras/, s Interneta, pristupljeno 12.08.2023.

[14]„VGG16 – Convolutional Network for Classification and Detection“, https://neurohive.io/en/popular-networks/vgg16/ , s Interneta, pristupljeno 12.08.2023.

[15]„Grid Search in Machine Learning“, https://www.scaler.com/topics/machine-learning/grid-search-in-machine-learning/ , s Interneta, pristupljeno 12.08.2023.

[16]„Understanding a Classification Report For Your Machine Learning Model“ , https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397, s Interneta, pristupljeno 12.08.2023.

7. POPIS SLIKA

8. POPIS TABLICA

9. SAŽETAK I KLJUČNE RIJEČI

U ovom radu istražuju se različite metode kojima se može klasificirati tip terena, temeljen na korištenju propriocepcijskih senzora mobilnog robota te računalnog vida. Koristi se set zabilježenih podataka na koji se nadograđuju metode za klasifikaciju terena. Podaci se prvo podijele na numeričke i podatke koji sadrže slike. Za klasifikaciju numeričkih podataka koristila se metoda MLPClassifier implementirana s metodom GridSearchCV-a, dok se je za podatke sa slikama koristio VGG16 model. Također, prikazana je dodatna metoda u svrhu dobivanja još boljih rezultata, a to je metoda ulančanog sloja (concatenate layer). Za prikaz rezultata koristila se klasifikacija metrika (classification report) pomoću koje su se usporedile performanse korištenih modela za svih sedam mogućih klasa terena.

Ključne riječi: klasifikacija terena, MLPClassifier, VGG16 model, concatenate layer, klasifikacija metrika, GridSearchCV

10. ABSTRACT AND KEYWORDS

In this thesis, we explore diverse methods for terrain classification using the proprioception sensors of mobile robots. We utilize a structured and labeled dataset to construct various classification techniques. Initially, the data is partitioned into numerical and visual segments. For the numerical segments, we employ the MLClassifier method integrated with GridSearchCV. In contrast, the VGG166 model is applied to the visual segments. We further introduce a concatenate layer method to enhance performance outcomes. To showcase these results, we employ a classification metrics report, facilitating a comprehensive comparison of the performance across all seven terrain classes.

Key words: terrain classification, MLPClassifier, VGG16 model, concatenate layer, classification metrics report, GridSearchCV

DODATAK A – KLASIFIKACIJA NUMERIČKIH PODATAKA

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import os
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.utils.validation import column_or_1d



# ## Enter the name of HDF5 file and an index (sample) of interest


filename =  "testSet_c7_ver_2.hdf5"
check_idx = 233 # check a specific index (sample sec)


# ## Visualize the data of selected sample

f = h5py.File(filename, 'r')
print(list(f.keys()))
labelp = ['asphalt','grass','gravel','pavement','sand','brick','coated
floor']
```

```
timeStamp = f['timeStamps']['timeStamps'][check_idx]
signal = f['signals']['signals'][check_idx]
#raw_signal = f['signals']['raw_signals'][check_idx]
image = f['images']['images'][check_idx]
label = f['labels']['labels'][check_idx]

f.close()

print("Timestamp is:",str(timeStamp))
print("Label is:",labelp[label])

Image = image.reshape(224,224,3)/255.0
imgplot = plt.imshow(Image)


# ## Read the entire HDF5 file (load the entire dataset) (memory
warning!!)


f = h5py.File(filename, 'r')
print(list(f.keys()))

timeStamps = f['timeStamps']['timeStamps'][:]
signals = f['signals']['signals'][:]
#raw_signals = f['signals']['raw_signals'][:]
images = f['images']['images'][:]
labels = f['labels']['labels'][:]

print(timeStamps.shape,"# of timeStamps")
print(signals.shape,"# of signals")
#print(raw_signals.shape,"# of raw signals")
print(images.shape,"# of images")
```

```
print(labels.shape,"# of labels")

f.close()


# ## Data reshape



images = images.reshape(images.shape[0],224,224,3)
#raw_signals = raw_signals.reshape(raw_signals.shape[0],100,11)
signals = signals.reshape(signals.shape[0],100,9)

print(timeStamps.shape,"# of timeStamps")
print(signals.shape,"# of signals")
#print(raw_signals.shape,"# of raw signals")
print(images.shape,"# of images")
print(labels.shape,"# of labels")



print("There are a total of",timeStamps.shape[0],"data entries in this
dataset. They are orgainzed by their unique timestamps. One second
segment of data is associated with one timestamp #.")
print("There are 9 channels of different signals: joint_states_left,
joint_states_right, lin_acc_x, lin_acc_y, lin_acc_z, %slip_left,
%slip_right, coeff_of motion_resistance_left, coeff_of
motion_resistance_right. As the sampling rate is 100 Hz, there are 100
timesteps in one second segment. The signal data are orgainzed
as",signals.shape[0],"timestamps,",signals.shape[1],"
timesteps,",signals.shape[2],"channels")
```

```
#print("Similarly, there are 11 channels for raw signals:
joint_states_left, joint_states_right, lin_acc_x, lin_acc_y, lin_acc_z,
T256_lin_x, T256_ang_z, curr_feedabck_left, curr_feedabck_right,
imu_roll, imu_pitch. The raw signal data are orgainzed
as",raw_signals.shape[0],"timestamps,",raw_signals.shape[1],"
timesteps,",raw_signals.shape[2],"channels")
print("The RGB images are resized
to",images.shape[1],"x",images.shape[2],"pixels.")
print("Finally, there are",labels.shape[0],"labels. asphalt: 0 | grass: 1
| gravel: 2 | pavement: 3 | sand: 4  | brick: 5  | coated floor: 6 ")



output_directory = r'C:\Users\diana\Desktop\MOJ PROJEKT_DIP'
np.save('dataset_slike.npy', images,labels)
np.save('dataset_numericki.npy',signals,labels,timeStamps)


# ## Treniranje mreže na podacima


# Load the data from file
data_num = np.load('dataset_numericki.npy')
print(data_num.shape)
reshaped_data = data_num.reshape(data_num.shape[0], -1)
x = reshaped_data
y = labels

# Provjera oblika podataka
print('x shape:', x.shape)  # (4020, 900)
print('y shape:', y.shape)  # (4020,)
```

```
# Podjela na trening, testiranje i validaciju
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=42)

# Provjera oblika podataka
print('x_train shape:', x_train.shape)
print('y_train shape:', y_train.shape)
print('x_test shape:', x_test.shape)
print('y_test shape:', y_test.shape)
print('x_val shape:', x_val.shape)
print('y_val shape:', y_val.shape)

# Kreiranje OneHotEncoder objekta i pretvaranje ciljnih podataka
x_train_reshaped = x_train.reshape(x_train.shape[0], -1)
y_train_reshaped = y_train.reshape(y_train.shape[0], -1)
y_test_reshaped = y_test.reshape(y_test.shape[0], -1)
y_val_reshaped = y_val.reshape(y_val.shape[0], -1)

# Ravel ciljnih podataka
y_train_reshaped = np.ravel(y_train_reshaped)
y_test_reshaped = np.ravel(y_test_reshaped)
y_val_reshaped = np.ravel(y_val_reshaped)

encoder = OneHotEncoder()
y_encoded = encoder.fit_transform(y_train_reshaped.reshape(-1, 1))

mlp_gs = MLPClassifier(max_iter=5000000)

# Define hyperparameter grid
parameter_grid = {
```

```
    'hidden_layer_sizes': [(5,), (10,)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}

# Create GridSearchCV object
cv = [(x_train.shape[0], x_val.shape[0])]
clf = GridSearchCV(mlp_gs, parameter_grid, n_jobs=-1, cv=3, refit=True)

clf.fit(x_train_reshaped, y_train_reshaped)
best_model = clf.best_estimator_


print(classification_report(y_test_reshaped, y_pred, zero_division=1.0))
```

DODATAK B – KLASIFIKACIJA SLIKA

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import os
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score



# ## Enter the name of HDF5 file and an index (sample) of interest



filename =  "testSet_c7_ver_2.hdf5"
check_idx = 233 # check a specific index (sample sec)

# ## Visualize the data of selected sample



f = h5py.File(filename, 'r')
```

```
print(list(f.keys()))
labelp = ['asphalt','grass','gravel','pavement','sand','brick','coated
floor']

timeStamp = f['timeStamps']['timeStamps'][check_idx]
signal = f['signals']['signals'][check_idx]
#raw_signal = f['signals']['raw_signals'][check_idx]
image = f['images']['images'][check_idx]
label = f['labels']['labels'][check_idx]

f.close()

print("Timestamp is:",str(timeStamp))
print("Label is:",labelp[label])

Image = image.reshape(224,224,3)/255.0
imgplot = plt.imshow(Image)


# ## Read the entire HDF5 file (load the entire dataset) (memory
warning!!)



f = h5py.File(filename, 'r')
print(list(f.keys()))

timeStamps = f['timeStamps']['timeStamps'][:]
signals = f['signals']['signals'][:]
#raw_signals = f['signals']['raw_signals'][:]
images = f['images']['images'][:]
labels = f['labels']['labels'][:]
```

```
print(timeStamps.shape,"# of timeStamps")
print(signals.shape,"# of signals")
#print(raw_signals.shape,"# of raw signals")
print(images.shape,"# of images")
print(labels.shape,"# of labels")

f.close()


# ## Data reshape


images = images.reshape(images.shape[0],224,224,3)
#raw_signals = raw_signals.reshape(raw_signals.shape[0],100,11)
signals = signals.reshape(signals.shape[0],100,9)

print(timeStamps.shape,"# of timeStamps")
print(signals.shape,"# of signals")
#print(raw_signals.shape,"# of raw signals")
print(images.shape,"# of images")
print(labels.shape,"# of labels")




print("There are a total of",timeStamps.shape[0],"data entries in this
dataset. They are orgainzed by their unique timestamps. One second
segment of data is associated with one timestamp #.")
print("There are 9 channels of different signals: joint_states_left,
joint_states_right, lin_acc_x, lin_acc_y, lin_acc_z, %slip_left,
%slip_right, coeff_of motion_resistance_left, coeff_of
motion_resistance_right. As the sampling rate is 100 Hz, there are 100
timesteps in one second segment. The signal data are orgainzed
```

```
as",signals.shape[0],"timestamps,",signals.shape[1],"
timesteps,",signals.shape[2],"channels")
#print("Similarly, there are 11 channels for raw signals:
joint_states_left, joint_states_right, lin_acc_x, lin_acc_y, lin_acc_z,
T256_lin_x, T256_ang_z, curr_feedabck_left, curr_feedabck_right,
imu_roll, imu_pitch. The raw signal data are orgainzed
as",raw_signals.shape[0],"timestamps,",raw_signals.shape[1],"
timesteps,",raw_signals.shape[2],"channels")
print("The RGB images are resized
to",images.shape[1],"x",images.shape[2],"pixels.")
print("Finally, there are",labels.shape[0],"labels. asphalt: 0 | grass: 1
| gravel: 2 | pavement: 3 | sand: 4  | brick: 5  | coated floor: 6 ")


# In[8]:


output_directory = r'C:\Users\diana\Desktop\MOJ PROJEKT_DIP'
np.save('dataset_slike.npy', images,labels)
np.save('dataset_numericki.npy',signals,labels,timeStamps)


# ## Klasifikacija slika i treniranje mreže



data = np.load('dataset_slike.npy', allow_pickle=True)
print(data.shape)  # (4020, 224, 224, 3)

images = data[:, :, :, :]
labels = data[:, 0, 0, 0]

print(images.shape)  # (4020, 224, 224, 3)
```

```
print(labels.shape)  # (4020,)

images_edit = preprocess_input(images)
x = images_edit
y = labels


# Train, test and validation data
train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.2,
random_state=42)
train_x, val_x, train_y, val_y = train_test_split(train_x, train_y,
test_size=0.2, random_state=42)

# Load pre-trained VGG model
model = VGG16(weights='imagenet')


# Pretvaranje oznaka u one-hot vektore
train_y = to_categorical(train_y, num_classes=1000)
test_y = to_categorical(test_y, num_classes=1000)
val_y = to_categorical(val_y, num_classes=1000)

# Print shape
print("Train images shape:", train_x.shape)
print("Train labels shape:", train_y.shape)
print("Test images shape:", test_x.shape)
print("Test labels shape:", test_y.shape)
print("Validation images shape:", val_x.shape)
print("Validation labels shape:", val_y.shape)

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
```

```
model.fit(train_x, train_y, epochs=10, batch_size=32, verbose=10,
validation_data=(val_x, val_y))

predicted_y = model.predict(test_x)

# Convert predictions to discrete labels
predicted_labels = np.argmax(predicted_y, axis=1)

# Calculate accuracy
accuracy = accuracy_score(test_y, predicted_labels)
print("Accuracy:", accuracy)


# Pretvaranje predikcija u diskretne oznake
predicted_labels = np.argmax(predicted_y, axis=1)
test_labels = np.argmax(test_y, axis=1)

# Izračun preciznosti
accuracy = accuracy_score(test_labels, predicted_labels)
print("Preciznost (Accuracy):", accuracy)

# Izračun mjere odziva
recall = recall_score(test_labels, predicted_labels, average='macro')
print("Mjera odziva (Recall):", recall)

# Izračun mjere preciznosti
precision = precision_score(test_labels, predicted_labels,
average='macro')
print("Mjera preciznosti (Precision):", precision)

# Izračun F1-mjere
f1 = f1_score(test_labels, predicted_labels, average='macro')
```

```
print("F1-mjera:", f1)



# Izračun mjere odziva s definiranim zero_division parametrom
recall = recall_score(test_labels, predicted_labels, average='macro',
zero_division=1)
print("Mjera odziva (Recall):", recall)

# Izračun mjere preciznosti s definiranim zero_division parametrom
precision = precision_score(test_labels, predicted_labels,
average='macro', zero_division=1)
print("Mjera preciznosti (Precision):", precision)
```

DODATAK C – COMBINED MODEL

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import plot_model
import keras
from keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras import activations
from keras.layers import Activation, Dense, concatenate, Reshape



model_vgg16 = VGG16(classes=7, include_top=False)
model_1 = keras.Sequential()
model_1.add(model_vgg16)
model_1.add(Dense(100, activation='relu'))
model_1.add(Dense(7, activation='softmax'))
model_1.add(Reshape((7, 1)))


model_2 = keras.Sequential()
model_2.add(Dense(1000, input_shape=(900,), activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(1000, activation='relu'))
model_2.add(Dense(7, activation='softmax'))
model_2.add(Reshape((7, 1)))
```

```
combined = concatenate([model_1.output, model_2.output], axis=0)
model_3 = Dense(7, activation="softmax")(combined)
model_3 = Model(inputs=[model_1.input, model_2.input], outputs=model_3)
```