Razpoznavanje objektov

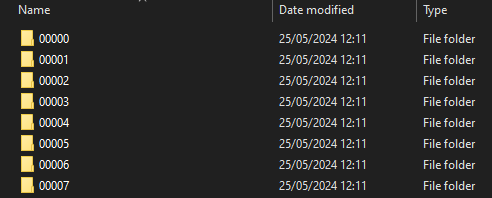
Osnove računalniškega vida

Anej Majnik

25.5.2024

# Izbira slik

Za izbor slik sem izbral *German Traffic Sign Recognition Benchmark GTSRB.* Ta dataset slik je razporejen v 43 razredov, vsak v svojem direktoriju označen od 00000 do 00042.



Slika 1: Direktoriji razredov

Slike, ki sem jih dobil so že bile augmentirane (30 augmentacij vsake slike). Označene so bile XXXXX\_YYYYY.ppm, kjer je XXXXX število originalne slike in YYYYY število augmentacije.



Slika 2: Primer slike 00000\_00000 iz razreda 00000



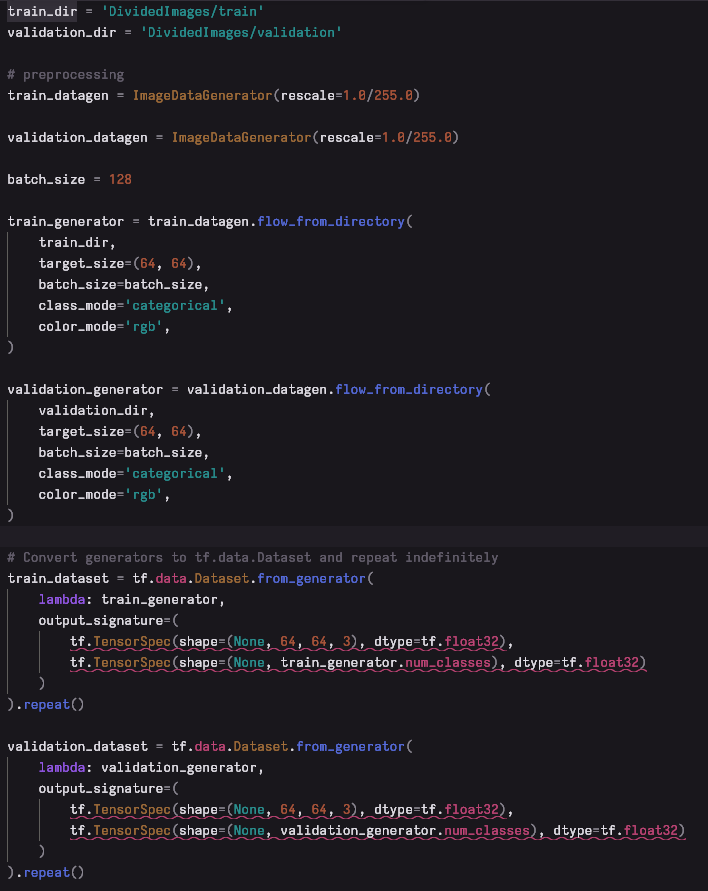
Slika 3: Primer slike 00000\_00013 iz razreda 00013



Slika 4: Primer slike 00000\_00000 iz razreda 00029

# Priprava in nalaganje slik

Slike sem najprej s python skripto razdelil na training in validation slike. Razdelil sem jih na 80% training in 20% validation. Slike med training in validation ne smejo biti iste in morajo biti porazdeljene enakomerno.



Slika 5: Koda za pripravo in nalaganje slik

Najprej sem definiral poti do training in validation direktorija. Uporabil sem ImageDataGenerator razred za generiranje in normalizacijo pikslov (od 0 do 1).

Nato uporabim funkcijo flow\_from\_directory funkcijo za generiranje batchov slik.

Nato pretvorim z tf.data.Dataset.from\_generator za boljši performance in uporabim .repeat(), da se ponavlja če zmanjka slik za batch.

# Augmentacija podatkov

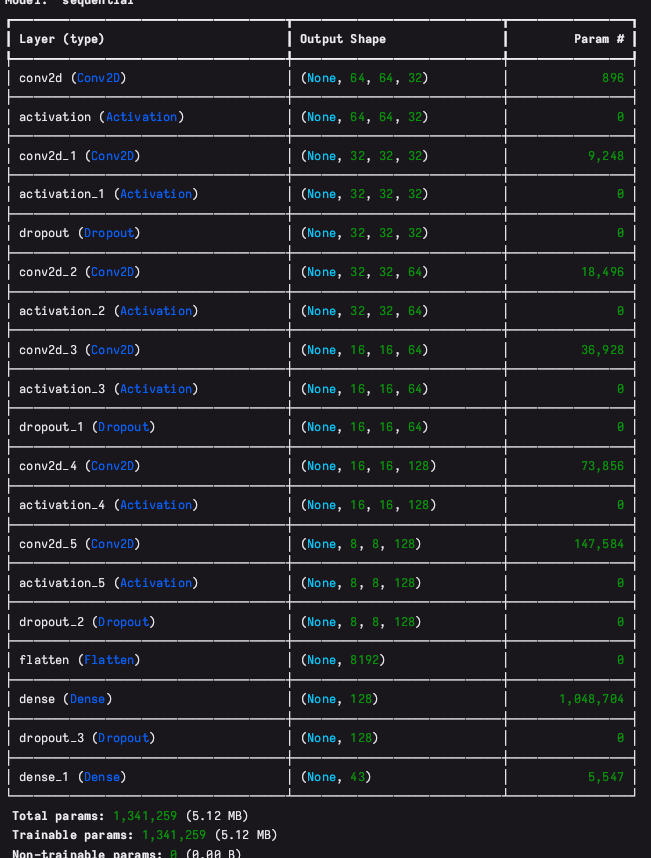
Podane slike so bile že augmentirane, zato jih nisem še posebej sam augmentiral. Podatkov je bilo dovolj, saj je bilo čez 40 000 slik in je bila natančnost zelo dobra. Augmentacije, ki so bile na slikah so rotacija, sprememba svetlosti, kontrasta itd.

# Sloji nevronske mreže



Slika 6: Odsek kode grajenja srednjega modela

Za manjši model sem opravil 2 ponovitvi, za srednji 3 ponovitve in za veliki 4 ponovitve



Slika 7: Izpis slojev pri srednjem modelu

# Podrobnosti izvedenih eksperimentov

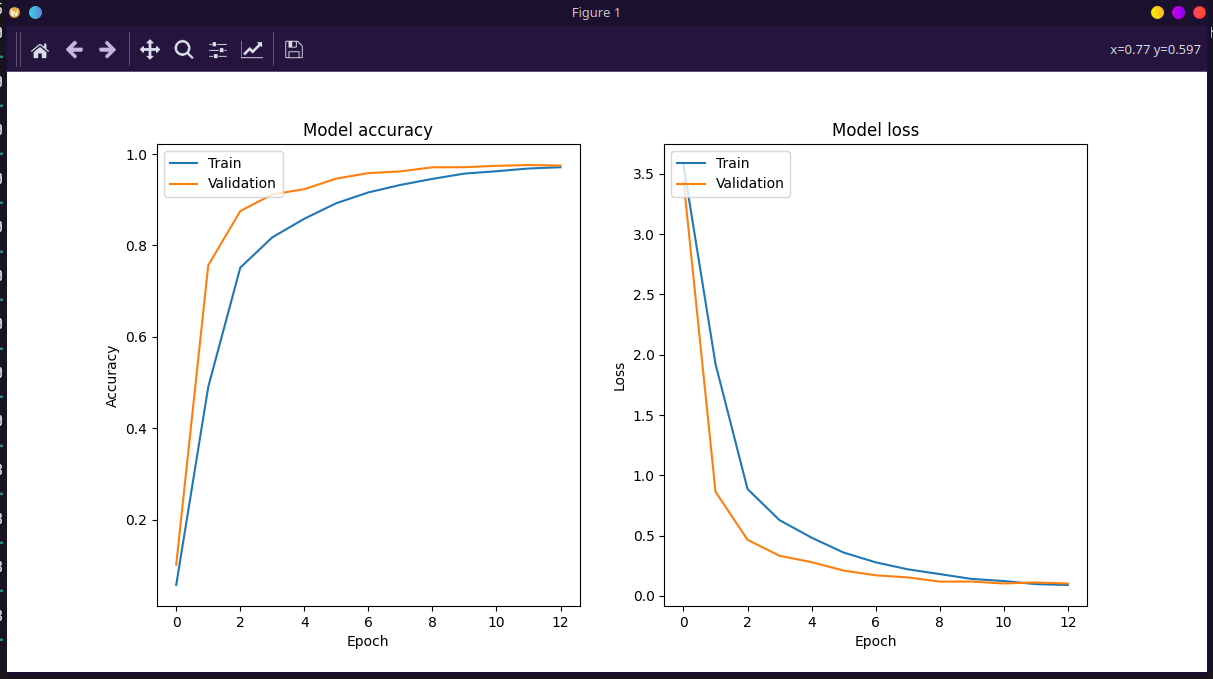
Na koncu optimizacije mi je uspelo vse tri modele optimizirati na približno isto raven.

Pri večjem modelu je bilo najprej potrebnih več epochov za dobro delovanje modela, ampak sem potem z optimizacijo parametrov (learning speed, batch size, število epochov, dropout itd) to zmanjšal.

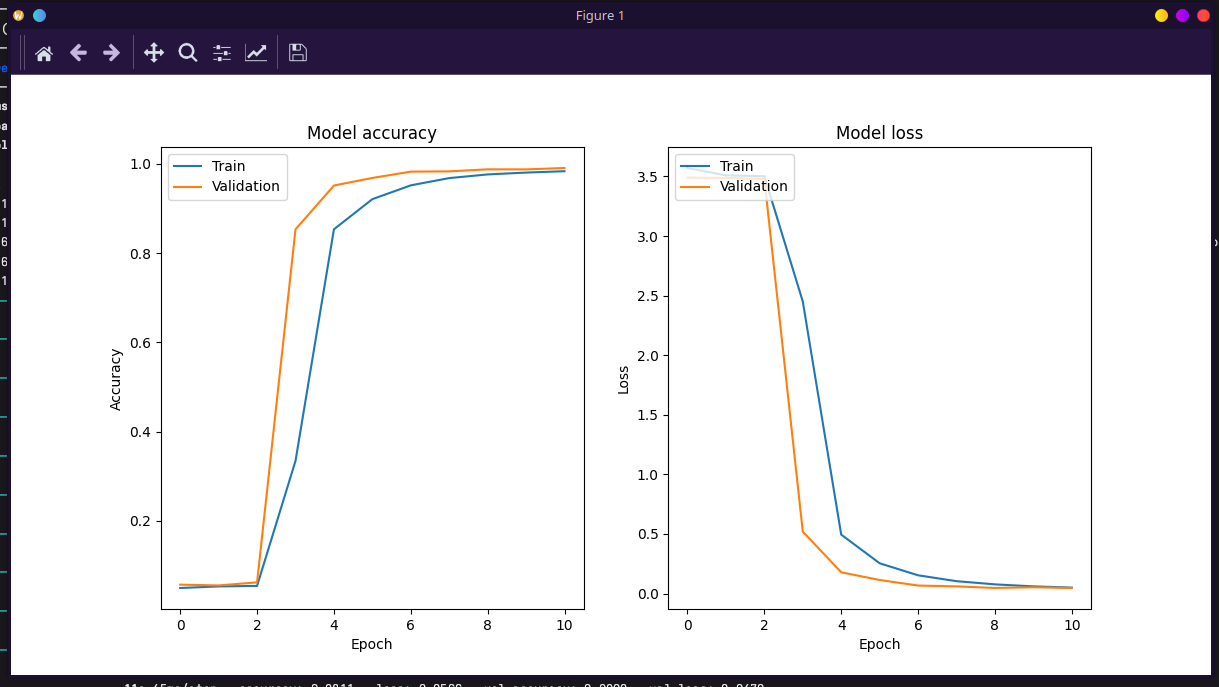
Dropout je bil pomemben del optimizacije, saj preprečuje overfitting in prisili mrežo, da generalizira podatke.

Pri manjšem modelu je bilo učenje bolj nihajoče, saj je včasih delal zelo dobro, včasih pa se validation accuracy skoraj nič ni povečal. Tudi to sem nato z optimizacijo parametrov večinoma odpravil.

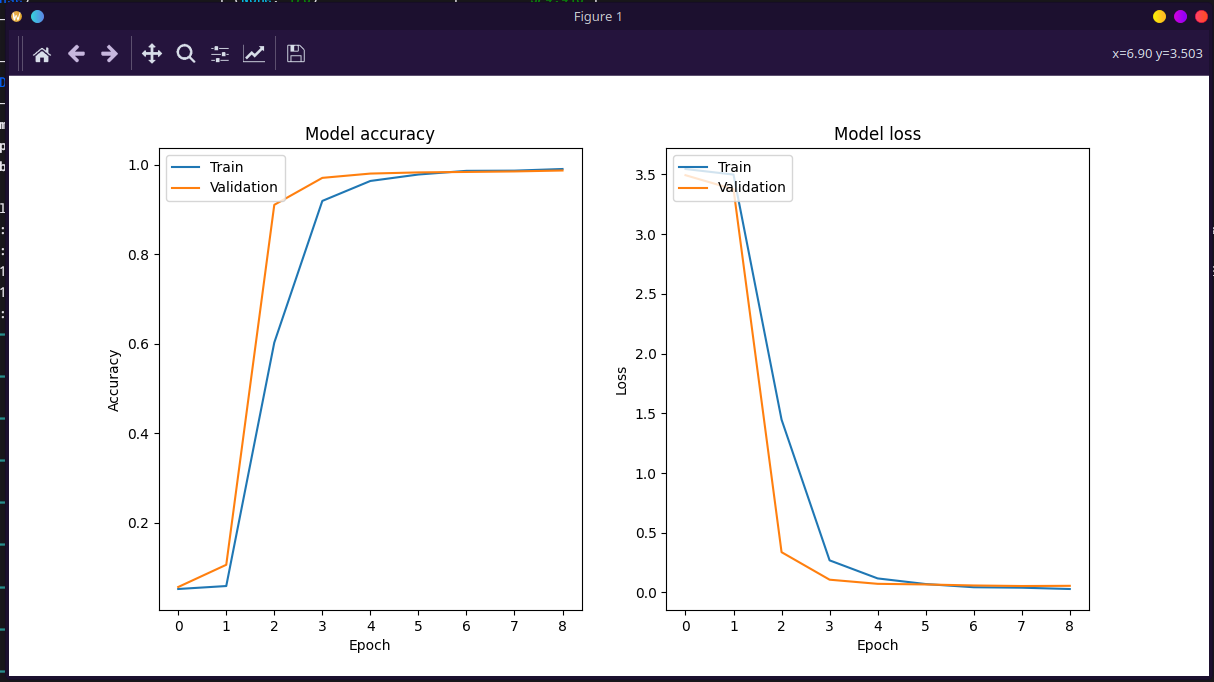
# Diagrami izgub med učenjem



Slika 8: Graf natančnosti in izgube pri majhnem modelu



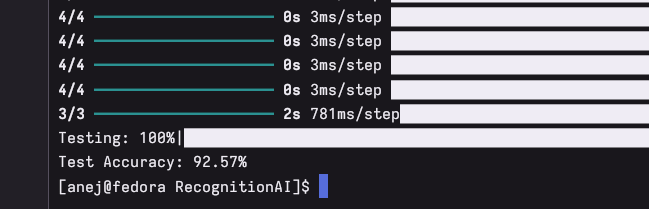
Slika 9: Graf natančnosti in izgube pri srednjem modelu



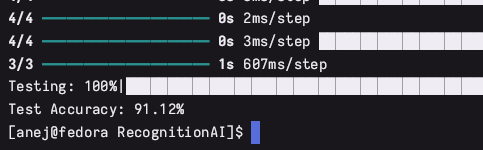
Slika 10: Graf natančnosti in izgube pri velikem modelu

# Rezultati učenja

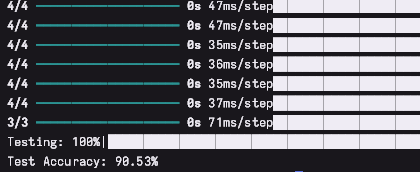
Po optimizaciji delujejo vsi modeli približno enako. Vsi dosežejo optimalno natančnost okoli epocha 5. Čeprav na grafu izgleda, kot da se dogaja overfitting, se uporabi stanje modela ob najboljši natančnosti. To sem dosegel z uporabo *restore\_best\_weights=True.*



Slika 11: Natančnost malega modela nad testnimi podatki



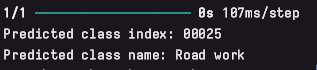
Slika 12: Natančnost srednjega modela nad testnimi podatki



Slika 13: Natančnost večjega modela nad testnimi podatki



Slika 14: Primer vhodne slike



Slika 15: Primer izhoda