SVEUČILIŠTE U SPLITU
FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Sustav za automatsku detekciju dima na slikama prirodnog krajolika

Jerko Ćurković

Split, Rujan 2023.

SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

Sveučilišni prijediplomski studij: **Računarstvo**

Oznaka programa: 120

Akademska godina: 2022/2023.

Ime i prezime: **Jerko Ćurković**

Broj indeksa: 401-2019

ZADATAK ZAVRŠNOG RADA

Naslov: **Sustav za detekcija dima na slikama prirodnog krajolika**

Zadatak: U završnom radu potrebno je izraditi sustav za automatsku detekciju dima. Sustav će biti temeljen na algoritmima dubokog učenja. Za implementaciju sustav za automatsku detekciju dima potrebno je koristiti MLID bazu podataka. Dobivene rezultate detaljno komentirati i obrazložiti.

Datum obrane: 14.09.2023.

Mentor:

doc. dr. sc. Dunja Božić-Štulić

IZJAVA

Ovom izjavom potvrđujem da sam završni rad s naslovom „**Sustav za automatsku detekciju dima na slikama prirodnog krajolika**“ pod mentorstvom doc. dr. sc. Dunje Božić-Štulić pisao samostalno, primijenivši znanja i vještine stečene tijekom studiranja na Fakultetu elektrotehnike, strojarstva i brodogradnje, kao i metodologiju znanstveno-istraživačkog rada te uz korištenje literature koja je navedena u radu. Spoznaje, stavove, zaključke, teorije i zakonitosti drugih autora koje sam izravno ili parafrazirajući naveo u završnom radu citirao sam i povezao s korištenim bibliografskim jedinicama.

Student

Jerko Ćurković

SADRŽAJ

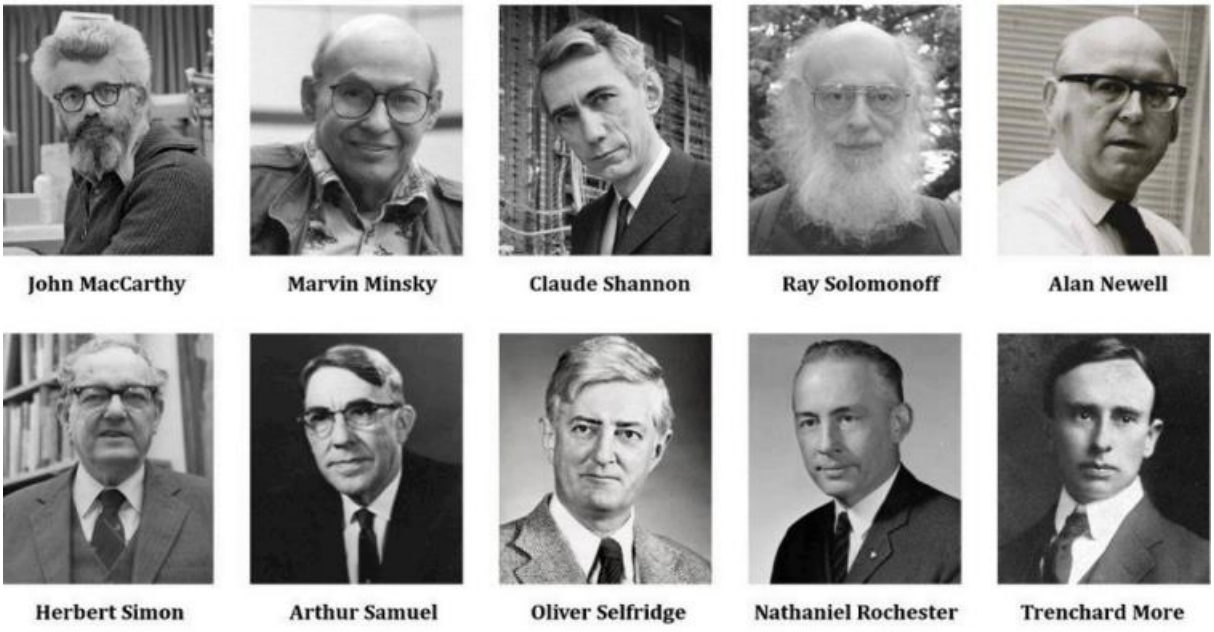
1 UVOD

Stara narodna poslovica glasi: „Gdje ima dima, ima i vatre“. Upravo je dim najraniji vizualni i mirisni pokazatelj koji ukazuje na vatru ili požar. Razlika između požara i vatre je u tome što je požar nekontrolirano gorenje koje nanosi materijalnu štetu ili ugrožava ljudske živote[1] dok je vatra gorenje koje može i ne može biti kontrolirano. Upravo zbog toga što je dim najraniji pokazatelj požara, važno ga je što prije uočiti kako bi se požar otkrio u začetku i minimizirala moguća materijalna šteta i ljudski gubitci. Prije su u šumama postojali čuvari čija je zadaća bila rano otkrivanje požara i upozoravanje na isti, no razvojem novih tehnologija, posebno u području računalnog vida i dubokog učenja, njihov je posao olakšan. Detekcija dima može biti kompleksna zbog varijacija u osvjetljenju, teksturama i pozadinskim objektima pa je daljnjim razvojem tehnologija vidljiv napredak u točnosti ovih algoritama za detekciju.

Ovaj rad usredotočuje se na istraživanje i implementaciju tehnika temeljenih na dubokom učenju kako bi se izradio jedan takav sustav sposoban za prepoznavanje prisutnosti dima na slikama. Rad je podijeljen u šest cjelina. U drugom je poglavlju dan uvid u pojam umjetne inteligencije, njen razvoj kroz povijest i podjela na neke osnovne cjeline. Iduće, treće poglavlje, pobliže objašnjuje što su to neuronske mreže, od kuda uopće taj naziv te kako se dijele. U istom je poglavlju fokus stavljen na konvolucijske neuronske mreže. Četvrto je poglavlje posvećeno programskom jeziku Python. U tom su poglavlju objašnjena osnovna pravila tog jezika te zašto je upravo taj jezik pogodan za razvoj umjetne inteligencije. Svo ovo akumulirano teorijsko znanje je u petom poglavlju pretvoreno u nešto praktično, odnosno u tom je poglavlju prikazana izrada jednog programa koji prepoznaje dim na slici te može imati praktičnu primjenu i pomoći u ranom detektiranju i upozoravanju na požar. U posljednjem je poglavlju nakon svega dan zaključak s kojim je ovaj rad zaokružena cjelina.

2 UMJETNA INTELIGENCIJA

Umjetna inteligencija (eng. artificial intelligence, AI) je područje istraživanja i stvaranja strojeva sposobnih za takvu vrstu aktivnosti, koja bi, da su je izveli ljudi, bila proglašena inteligentnom[2]. Ovo je samo jedna od mnogih definicija koje postoje za ovaj pojam, ali ona dobro ističe težnju povezivanja stroja i čovjeka tako da se ne zna razlika. Cilj umjetne inteligencije je stvoriti računalne programe i modele koji mogu razmišljati, učiti, rješavati probleme, donositi odluke i prilagođavati se promjenjivim okolnostima. Sam razvoj umjetne inteligencije je pokrenut s ciljem da se olakša ljudski posao i sam život, kao što između ostalog i svaki izum ima za cilj olakšati ljudima svakodnevnicu. U nekim hollywoodskim filmovima je umjetna inteligencija predstavljena kao nešto što je u početku bilo dobro i pomagalo ljudima, ali se s vremenom okrenula protiv ljudske vrste te ih željela porobiti i istrijebiti. Najpoznatiji primjer takvih filmova široj populaciji je filmski serijal „Terminator“ iz 1984. godine s Arnoldom Schwarzeneggerom u glavnoj ulozi. Iako su svi takvi filmovi samo fikcija i prikazuju negativnu krajnost, itekako su dobar podsjetnik svima koji se bave razvojem umjetne inteligencije da uvijek treba biti na oprezu. Sredinom dvadesetog stoljeća je započet razvoj umjetne inteligencije, paralelno sa samim razvojem računala, a posebno je važna 1956. godina. Tada je američki računalni znanstvenik John McCarthy skovao naziv „umjetna inteligencija“ i definirao ju kao znanost i tehnika izrade inteligentnih strojeva[3]. Tom se godinom smatra i rođenje ovog područja na radionici na Dartmouth Collegeu.



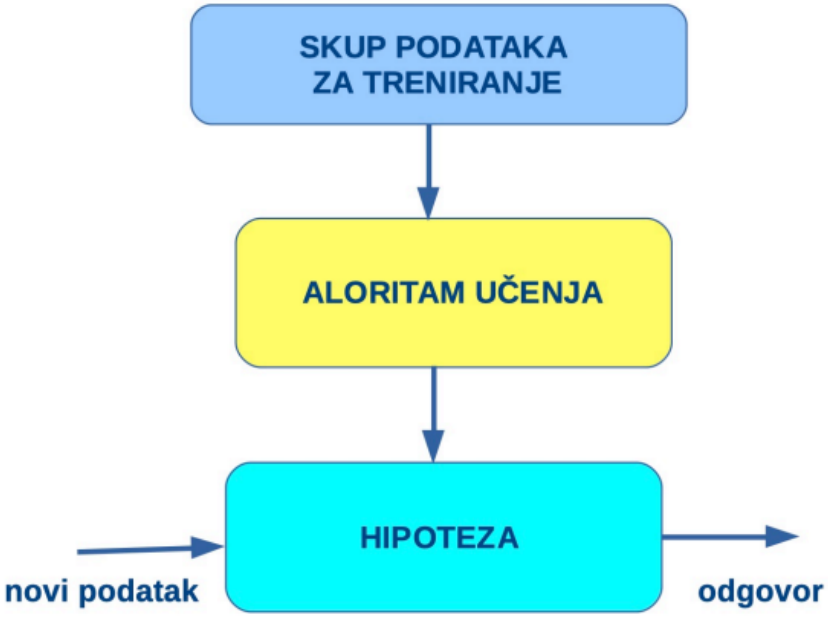
Slika 2.1 Deset sudionika radionice na Dartmouth Collegeu 1956. godine[4]

Sljedeća važna godina je 1961. kada su američki znanstvenici Allen Newell i Herbert A. Simon razvili program koji rješava probleme tako da simulira ljudske protokole koristeći se strategijom analize sredstva i cilja i nazvali ga „General Problem Solver (GPS)“[2]. Do početka primjene umjetne inteligencije u industriji trebalo je proći još dvadesetak godina. Godine 1985. više od 180 000 robota radi na proizvodnim trakama širom svijeta i više od 100 kompanija ugrađuje sustave s robotskim viđenjem[2]. Od tada pa do danas umjetna inteligencija postaje sveprisutna u ljudskom životu. Neki od najpoznatijih primjera su: Siri, programi za prepoznavanje slika, likovi u videoigrama, ChatGPT te autonomna vozila.

AI možemo podijeliti na slabu (usku) i jaku (opću)[5]. Slaba umjetna inteligencija je usmjerena na rješavanje specifičnih problema i zadataka. Takvi sustavi su specijalizirani za usko specifično područje pa nemaju sposobnost razumijevanja izvan tog područja. Upravo zato se i naziva uska. Neki od primjera su: Siri, Alexa, chatbotovi. Jaka umjetna inteligencija se odnosi na sustave koji bi bili sposobni primijeniti svoje znanje na različite domene, imali bi svijest o svojem okruženju te bi mogli donositi zaključke izvan onoga što su specifično naučili. Takva umjetna inteligencija još nije dostignuta. Umjetna inteligencija za podskup ima strojno učenje.

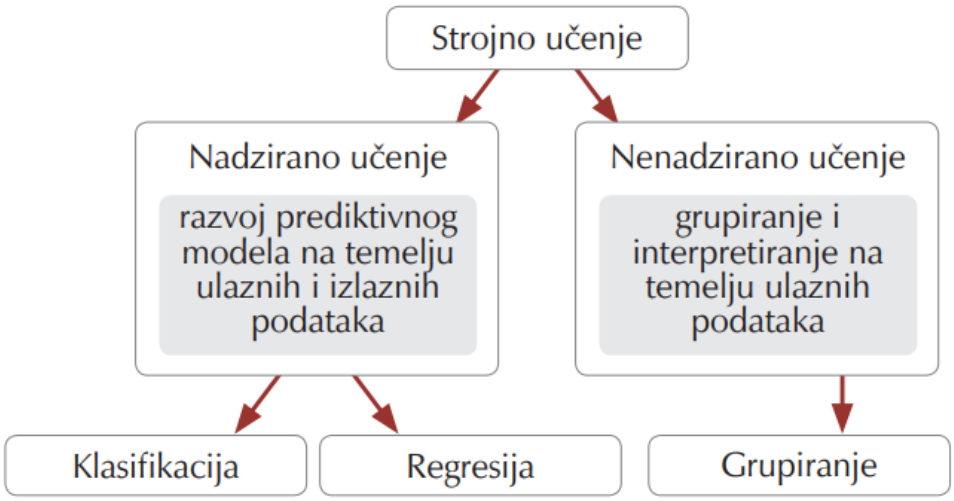
2.1 Strojno učenje

Strojno učenje (eng. Machine Learning) je termin kojeg je skovao Arthur Samuel, američki pionir računalnih igara i umjetne inteligencije, 1959. godine i iste godine ga definirao kao područje istraživanja u kojemu se računalu daje mogućnost učenja bez eksplicitnog programiranja[4]. Sam postupak strojnog učenja započinje tako da se algoritmu da set ulaznih podataka koje služe za trening algoritma. Potrebno je da taj set bude što veći kako bi algoritam prošao kroz velik broj primjera i samim tim imao što bolji ishod. Ovo je potpuno analogno kao i kada student uči za neki ispit. Ako student ima veliki broj primjera za učenje onda uočava sve više sličnosti među njima koje kasnije prepoznaje u konkretnim zadacima na ispitu. Tako dalje funkcionira i algoritam strojnog učenja koji iz seta podataka za trening izvlači pravilnosti i zakonitosti koje su učestale i na osnovu njih formira zaključak (hipotezu) koji mu služi da na osnovu toga za buduće ulazne podatke da odgovor. Na slici 2.2 je ilustriran postupak strojnog učenja.



Slika 2.2 Ilustracija postupka strojnog učenja[4]

Osnovna podjela strojnog učenja se može svesti na nadzirano i nenadzirano učenje. Razlika između ova dva tipa je ta što kod nadziranog učenja algoritmi rade s već poznatim setom ulazni i izlaznih podataka i ovi algoritmi se koriste za klasifikaciju i regresiju[6]. Nenadzirano učenje ne poznaje izlaze, nego na temelju ulaza pronalazi skrivene uzorke unutar ulaza i grupira podatke.

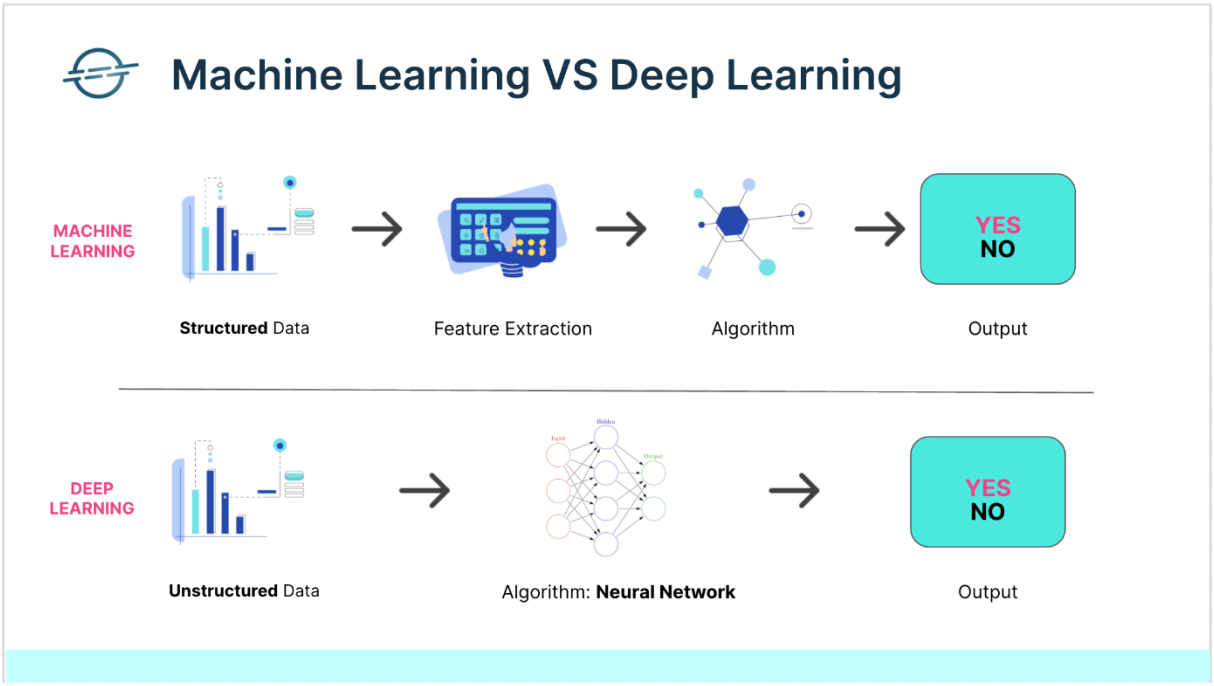


Slika 2.3 Osnovna podjela strojnog učenja[6]

Baš kao što i umjetna inteligencija ima podskup koji se zove strojno učenje, tako i strojno učenje ima duboko učenje za svoj podskup.

2.1.1 Duboko učenje

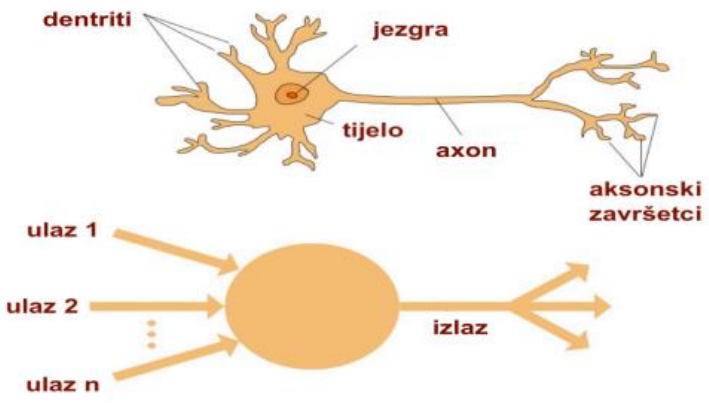
Duboko učenje (eng. Deep Learning) je grana strojnog učenja koja simulira način na koji ljudski mozak obrađuje informacije, koristeći neuronske mreže kako bi automatski naučila reprezentacije podataka. Ova tehnika omogućuje računalima da obavljaju složene zadatke koji zahtijevaju dublje razumijevanje i izvođenje različitih hijerarhijskih značajki iz podataka. Duboko učenje se ističe u usporedbi s tradicionalnim metodama strojnog učenja jer je učinkovitije u razumijevanju i prepoznavanju složenih oblika, uzoraka i značajki u podacima. Razlika u odnosu na strojno učenje je u tome što je za duboko učenje potreban veći skup podataka za treniranje, nije potrebna ljudska intervencija i za trening je potreban GPU. Ključna karakteristika dubokog učenja je korištenje višestrukih slojeva neurona, koji se nazivaju neuronske mreže. O njima je detaljnije riječ u poglavlju 3.



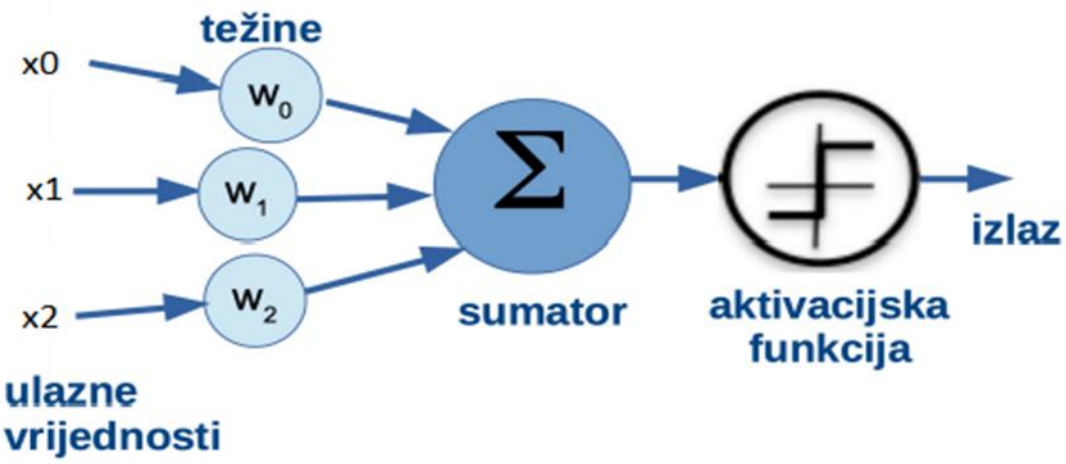
Slika 2.4 Razlika u obradi podataka kod strojnog i dubokog učenja

3 NEURONSKE MREŽE

Kako je cilj stvoriti program koji razmišlja kao čovjek, tako je pri dizajniranju takvih programa uzor ljudski mozak, alat s kojim ljudi razmišljaju. Mozak je središte živčanog sustava preko kojeg skuplja informacije od osjetila te ih procesira i šalje upute tijelu kako reagirate na podražaje. Osnovna gradbena jedinica živčanog sustava je živčana stanica (neuron). Neuron se sastoji od tijela (soma), ulaznih niti (dendrita) i izlaznih niti (aksona)[4]. Živčana stanica prenosi impuls tako da preko dendrita primi impuls, obradi ga u tijelu i onda preko aksona ga proslijedi idućem neuronu u mreži (sustavu). Budući da su za prihvat, obradu i prijenos informaciju u ljudskom tijelu zaslužni neuroni, tako je i za računala potrebno napraviti umjetne neurone. Zamisao da se imitira biološka funkcija mozga iznose McCulloch i Pitts sredinom dvadesetog stoljeća i predlažu uporabu logičkih sklopova za opisivanje neurona, ali je trebalo čekati za razvoj takvog modela zbog ograničenih računalnih resursa tog doba[7]. Američki psiholog Frank Rosenblatt 1958. godine predlaže model perceptrona i tako postavlja temelje modernih neuronskih mreža. Perceptron je računalna implementacija umjetnog neurona. Na slici 3.1 je prikazano kako su biološki neuroni poslužili za inspiraciju umjetnim neuronima, a na slici 3.2 je prikazan model perceptrona.

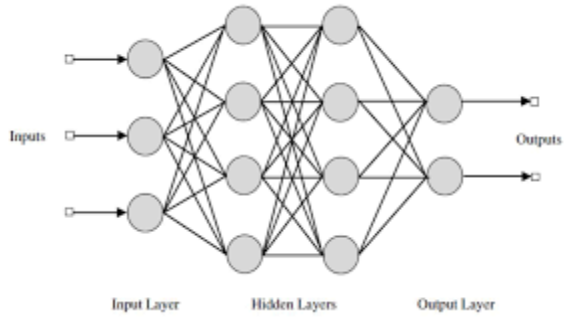


Slika 3.1 Prikaz biološkog i umjetnog neurona[4]



Slika 3.2 Model perceptrona[4]

Perceptron je u biti algoritam linearne regresije kojem je na izlazu dodana aktivacijska funkcija, funkcija jediničnog skoka koja za ulaznu vrijednost veću od 0 daje 1, a za ulaznu vrijednost manju od 0 daje -1[4]. Težine w_0, w_1 i w_2 su parametri koji reguliraju utjecaj ulaznih podataka na izlaz neurona. Svaki ulaz u neuron množi se sa svojom težinom, a težine se prilagođavaju tijekom procesa treniranja kako bi se optimizirala izlazna funkcija mreže. Aktivacijske funkcije primjenjuju se na težinsku sumu ulaza neurona kako bi se generirao konačni izlaz neurona. Ove funkcije omogućuju neuronskoj mreži da nauči i reagira na različite oblike i složenosti podataka. Umjetne neuronske mreže (engl. Artificial Neural Network, ANN) su mreže umjetnih neurona koji su najčešće poredani u slojeve[4]. Svim umjetnim neuronskim mrežama je zajedničko da se sastoje od više slojeva neurona: ulaznog, skrivenog (ili skrivenih) i izlaznog. Ulazni sloj je prvi sloj koji prima ulazne podatke te dimenzija ovog sloja odgovara dimenziji ulaznih podataka. Idući su skriveni slojevi koji obrađuju i transformiraju ulazne podatke kako bi naučili reprezentacije koje su korisne za rješavanje zadataka. Zadnji sloj neuronske mreže generira konačni izlaz ili predviđanja i zato se naziva izlazni sloj. Broj neurona u izlaznom sloju ovisi o vrsti problema koji mreža rješava. Na primjer, za klasifikaciju s više klasa, svaka klasa ima svoj neuron u izlaznom sloju. Na slici 3.3 je prikazana umjetna neuronska mreža s dva skrivena sloja.



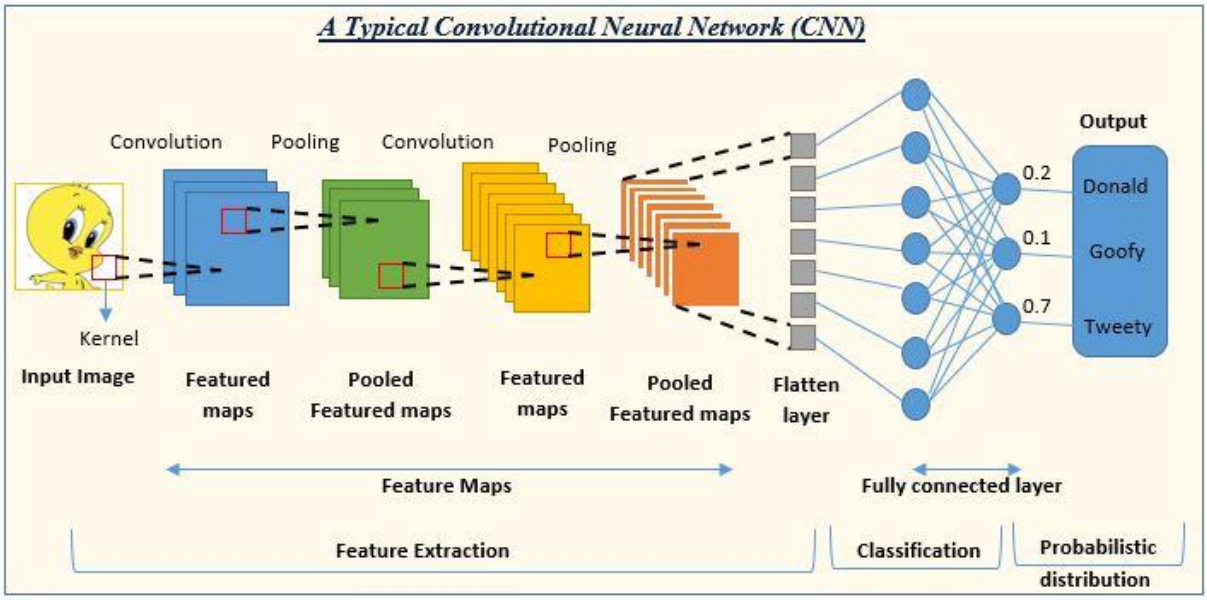
Slika 3.3 Umjetna neuronska mreža

Postoje različite vrste neuronskih mreža, svaka s posebnim arhitektonskim značajkama i namjenama. Višeslojni perceptroni su osnovne neuronske mreže s jednim ili više skrivenih slojeva između ulaznog i izlaznog sloja. Koriste se za širok spektar problema, uključujući klasifikaciju, regresiju i obradu podataka. Konvolucijske neuronske mreže su posebno prilagođene za obradu slika i drugih 2D struktura. Koriste konvoluciju za automatsko izlučivanje značajki iz slika, omogućavajući bolju lokalnu percepciju i smanjenje broja parametara. Njima je posvećen poseban odlomak 3.1 jer su bitne za praktični dio ovog rada. Rekurentne neuronske mreže su usmjerene na obradu sekvencijalnih podataka, kao što su nizovi riječi u rečenici ili vremenske serije. Ove mreže koriste povratne petlje koje omogućavaju prijenos informacija između vremenskih koraka, čime su posebno prikladni za analizu vremenski ovisnih podataka. Generativne suparničke mreže se koriste za generiranje novih podataka sličnih postojećim uzorcima. Ovi modeli uključuju dva dijela: generator koji stvara podatke i diskriminator koji pokušava razlikovati između generiranih i stvarnih podataka. Autoenkoderi su mreže koje se koriste za unos i rekonstrukciju podataka. Oni su korisni za smanjenje dimenzionalnosti, izdvajanje značajki i obnovu oštećenih podataka. Transformers su vrsta mreža koja je postala popularna za obradu sekvencijalnih podataka poput teksta. Ovi modeli koriste pažnju kako bi se fokusirali na relevantne dijelove sekvencije, omogućavajući bolje razumijevanje konteksta.

3.1 Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (eng. Convolutional Neural Networks, CNN) su specijalizirana vrsta neuronskih mreža za pretprocesiranje nestrukturiranih podataka, posebice vezanih za slike, tekst, zvuk i govor[4]. Već samo ime ove vrste neuronskih mreža ukazuje na to da se koristi operacija „konvolucija“. Konvolucija je matematička funkcija nastala integriranjem umnoška dviju funkcija po intervalu njihove definicije gdje su te funkcije

ravnopravne tako da svaka infinitezimalna promjena jedne funkcije utječe na drugu funkciju u cijelome intervalu definicije[8]. Prednosti konvolucijskih neuronskih mreža leže u njihovoj sposobnosti prepoznavanja lokalnih značajki, invarijantnosti na translaciju (pomak) i sposobnosti izvlačenja hijerarhijskih značajki. Ova mreža sastoji se od tri vrste slojeva, a to su: konvolucijski sloj, sloj sažimanja i potpuno povezani sloj[9]. Na slici 3.4 je prikazano kako radi konvolucijska neuronska mreža.



Slika 3.4 Konvolucijska neuronska mreža[10]

3.1.1 Konvolucijski sloj

Konvolucijski sloj (eng. Convolutional Layer) je osnovna komponenta konvolucijskih neuronskih mreža. Ovaj sloj igra ključnu ulogu u procesu izlučivanja značajki iz ulaznih slika ili drugih struktura s rešetkastom strukturom[10]. Konvolucijski se slojevi mogu dodavati jedan na drugoga te sa svakim novim slojem konvolucijska mreža postaje sve složenija, identificirajući veće dijelove slike. Prvi slojevi zaduženi su na jednostavne značajke poput boje ili rubova. Filter, mala matrica težina, se primjenjuje na određeni dio slike, a točkasti se proizvod izračunava između ulaznih piksela i filtera te se on unosi u izlazni niz. Filter obavlja jedan te isti posao sve dok ne prođe cijelu sliku[9].

3.1.2 Sloj sažimanja

Sloj sažimanja (eng. Pooling Layer), je važna komponenta konvolucijskih neuronskih mreža i igra ključnu ulogu u smanjenju dimenzionalnosti konvolucijskih mapa, što omogućava

ubrzanje računalnih operacija i smanjenje potencijalnog pretreniranja. Sloj sažimanja radi tako da izdvaja bitne informacije iz konvolucijske mape (sloja) tako da se smanji broj parametara u mreži i smanji računalna kompleksnost[10]. Najčešći tip sloja sažimanja je maksimalno grupiranje. Radi tako da ulaznu konvolucijsku mapu podijeli na manje podregije i za svaku se podregiju uzima najveća vrijednost i ta vrijednost postaje nova vrijednost u sažetom sloju. Tako smanjuje dimenzije ulazne konvolucijske mape, često na pola veličine u svakoj dimenziji.

3.1.3 Potpuno povezani sloj

Potpuno povezani sloj (eng. Fully Connected Layer) je sloj u konvolucijskim neuronskim mrežama koji se obično koristi na kraju mreže kako bi se donijele klasifikacijske ili regresijske odluke na temelju naučenih značajki iz prethodnih slojeva Svaki neuron u potpuno povezanom sloju povezan je sa svakim neuronima iz prethodnog sloja. Izlazi iz potpuno povezanog sloja obično se obrađuju pomoću aktivacijske funkcije kako bi se stvorile klasifikacijske ocjene ili numeričke izlazne vrijednosti, ovisno o vrsti zadatka[9]. Glavna uloga potpuno povezanog sloja u konvolucijskoj neuronskoj mreži je integracija naučenih značajki iz prethodnih slojeva kako bi se donijele konačne odluke o klasi objekta ili vrijednosti ciljne varijable. Uobičajeno je da se konvolucijska neuronska mreža sastoji od nekoliko konvolucijskih slojeva i slojeva sažimanja kako bi se izdvojile relevantne značajke iz ulaznih podataka, a potom se te značajke prenose u potpuno povezane slojeve za klasifikaciju, regresiju ili druge vrste analize.

4 PYTHON

Python je interpreterski, objektno-orijentirani, programski jezik visoke razine s dinamičkom semantikom[11]. Stvorio ga je nizozemski programer Guido van Rossum 1989. godine te ga je nazvao po britanskoj seriji „Monty Python's Flying Circus“, čiji je bio veliki obožavatelj. Koliko je osebujna ličnost govori i podatak da je imao titulu „dobronamjerni doživotni diktator“ (odnosi se na programski jezik Python).

Slika 4.1 Guido van Rossum, tvorac programskog jezika python

4.1 Značajke programskog jezika python

Interpreterski programski jezik znači da jezik koristi interpreter za izravno izvođenje napisanog koda. Interpreter je računalni program koji čita izvorni kod programa red po red i odmah ga pretvara u izlaz. Drugim riječima, interpretacija se događa u stvarnom vremenu, a kod se izvodi postupno, liniju po liniju. U tablici 4.1 je prikazana usporedba između interpreterskih i kompajlerskih programskih jezika.

	Interpreterski programski jezik	Kompajlerski programski jezik
Pretvorba koda	izvorni kod se red po red pretvara u izvršni oblik i odmah se izvodi	izvorni kod se pretvara u izvršni kod u jednom prolazu, stvarajući izvršnu datoteku koja se može izravno pokrenuti
Brzina izvođenja	sporije izvođenje od kompiliranih programa jer svaki redak mora biti interpretiran tijekom izvršavanja	brži jer se izvorni kod prevodi u optimizirani izvršni kod prije nego što se program izvede
Povratne informacije	brza povratna informaciju jer programer dobiva rezultate i poruke o pogreškama tijekom izvođenja	povratna informacija dolazi tek nakon što je program preveden i pokrenut
Portabilnost	često su više portabilni jer interpretacijski ovise o interpreteru, koji može biti dostupan na različitim platformama	mogu zahtijevati prilagodbe za svaku ciljanu platformu, što može otežati portabilnost

Tablica 4.1 Usporedba interpreterskih i kompajlerskih programskih jezika

Python se ističe svojom čitljivošću, jednostavnošću sintakse i bogatim ekosustavom biblioteka i alata. Za razliku od programskog jezika C ili C++ koji za odvajanje blokova koda koriste vitičaste zagrade, Python koristi tabulator koji olakšava svakome tko pogleda kod da odmah jasnije vidi odvojene blokove koda, a ne da se dodatno treba fokusirati gdje se koja zagrada otvara, a koja zatvara. Upravo zbog toga je omogućena bolje čitljivost koda pa je olakšano i kasnije održavanje koda. Ispod su dana dva primjera funkcije napisana u Pythonu (kod 4.1) i C-u (kod 4.2) koja vraća faktorijel broja 5 kako bi se pokazala jasnije čitljivost koda u Pythonu u odnosu na kod u C-u.

```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

result = factorial(5)
print(result)
```

Kod 4.1 Funkcija factorial u Pythonu

```c
#include <stdio.h>

int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int result = factorial(5);
    printf("%d\n", result);
    return 0;
}
```

Kod 4.2 Funkcija factorial u C-u

Bitno svojstvo Pythona je to što je „open-source" jezik s velikom zajednicom pa to omogućuje široku lepezu raznih materijala i foruma za pomoć pri savladavanju ovog jezika, a isto tako bogati broj biblioteka od kojih je biblioteci TensorFlow posvećen i poseban odlomak (odlomak 4.2) jer je ona korištena za praktični dio ovog završnog rada. Kroz sljedeće su odlomke detaljnije pojašnjene bitne programske strukture pythona.

4.1.1 Varijable

Varijabla je memorijska lokacija simboličnog imena u koju se sprema vrijednost nekog podatka. Jednostavnije, varijabla je „kutija" u koju se spremaju neke stvari i na kutiji se napiše smisleno ime kako bi se znalo što je u kutiji (npr. ako su u kutiji jabuke, onda je logičnije da na njoj piše „jabuke", a ne „kruške") kako bi se kasnije po potrebi moglo te stvari ponovo koristiti. Ista logika je u davanju imena varijablama. Preporučuje se davati smislena imena kako bi se odmah na prvu moglo prepoznati što koja varijabla predstavlja. Prilikom

davanja imena, potrebno je paziti na neka pravila pa tako ime ne smije započinjati s brojem i smiju se koristiti samo velika i mala slova engleske abecede i znak „ _ “ (donja crtica). Varijabla je jedinstveno određena imenom i memorijskom lokacijom te se varijabli pristupa preko njenog imena. Python je dinamički tipiziran jezik, što znači da varijable ne moraju biti eksplicitno deklarirane s tipom. U kodu 4.3 ispod su prikazani tipovi podataka u programskom jeziku python.

```
1
2  name = 'Jerko'      #string - niz znakova
3  age = 23            #integer - cjelobrojni tip podataka
4  height = 1.93       #float - realni tip podataka
5  isStudent = True    #boolean - logički tip podataka (ima vrijednost True ili False)
6
7
```

Kod 4.3 Tipovi podataka u programskom jeziku Python

Matematički znak jednakosti „ = “ se koristi za pridruživanje vrijednosti te treba paziti da ga se ne zamijeni sa znakom dvostruko jednako „ == “ koji predstavlja relacijski operator jednakosti. Neki od osnovnih operatora u pythonu su: aritmetički (prikazani u tablici 4.2), relacijski (prikazani u tablici 4.3) i logički (prikazani u tablici 4.4).

Operator	Namjena	Primjer	Rezultat x = 5; y = 2
+	Zbrajanje	rez = x + y	rez = 7
-	Oduzimanje	rez = x - y	rez = 3
*	Množenje	rez = x * y	rez = 10
/	Dijeljenje	rez = x / y	rez = 2.5
%	Ostatak dijeljenja	rez = x % y	rez = 1
**	Potenciranje	rez = x ** y	rez = 25
//	Cjelobrojno dijeljenje	rez = x // y	rez = 2

Tablica 4.2 Aritmetički operatori u Pythonu

Operator	Opis	Primjer	Rezultat x = 5; y = 2	Rezultat x = 5; y = 5
==	Usporedba jednakosti	x == y	False	True
!=	Usporedba nejednakosti	x != y	True	False
>	Strogo veće od	x > y	True	False
<	Strogo manje od	x < y	False	False
>=	Veće od ili jednako	x >= y	True	True
<=	Manje od ili jednako	x <= y	False	True

Tablica 4.3 Relacijski operatori u Pythonu

Operator	Opis
and	Konjunkcija, logički I
or	Disjunkcija, logički ILI
not	Negacija, logički NE

Tablica 4.4 Logički operatori u Pythonu

Operator *and* vraća vrijednost *True* samo ako oba operanda imaju vrijednost *True*, u suprotnom *False*. Operator *or* vraća vrijednost *False* samo ako su oba operanda *False*, u suprotnom *True*. Operator *not* negira operand. To znači ako je bio *True* onda vraća *False* i obrnuto.

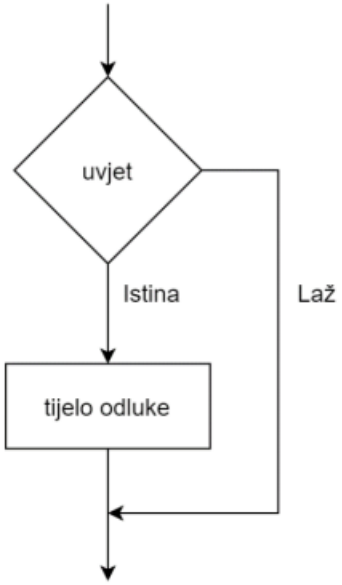
4.1.2 Upravljanje tokom programa

Kontrola toka programa omogućava da se odluči na koji način se program ponaša u zadanim okolnostima[12]. U Pythonu postoje dva osnovna načina kontrole toka, a to su odluke i petlje. Oduke pomažu odlučiti koji dio programskoga koda je potrebno izvršiti, a koji dio preskočiti.

Petlje olakšaju ponavljanje stvari kao na primjer ispisivanje svih parnih brojeva između 1 i 100.

Naredbe if, if-else, if-elif-else

Ove naredbe pripadaju kontroli toka za odluke. *If* naredba se sastoji od početnog uvjeta koji se ispituje te ako je uvjet ostvaren onda se ulazi u blok koda koji se izvršava i naziva se „tijelo odluke“. Ako uvjet nije zadovoljen onda se preskače taj blok koda. Na slici 4.2 je shematski prikazano kako izgleda *if* naredba, a kod 4.4 prikazuje jednostavan primjer *if* naredbe.

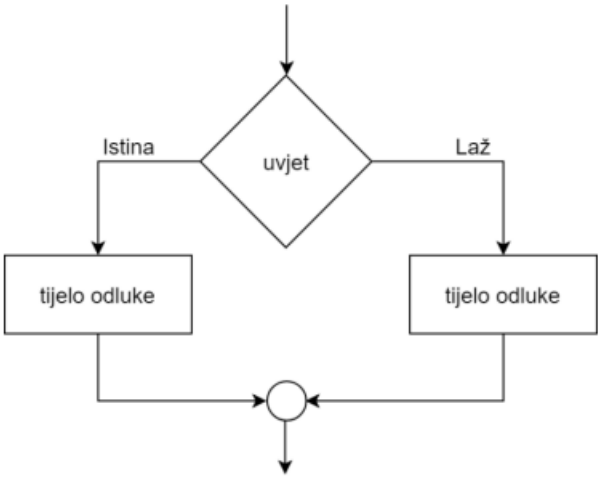


Slika 4.2 Shematski prikaz if naredbe

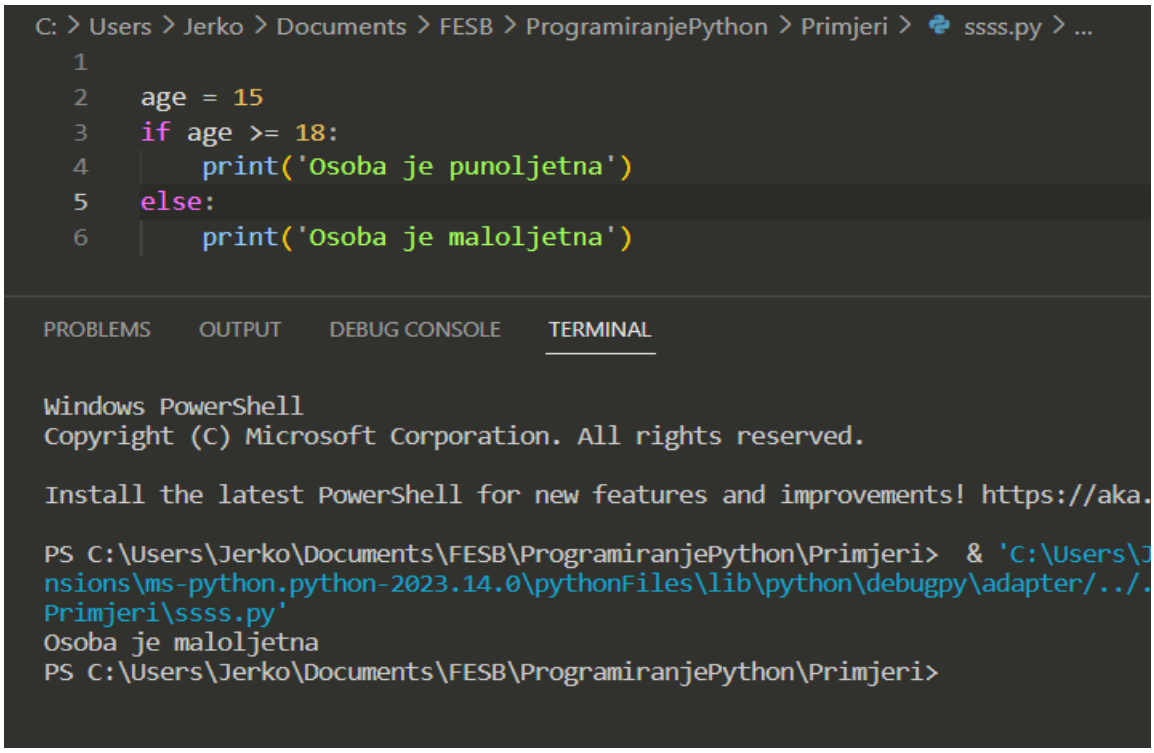
```
C: > Users > Jerko > Documents > FESB > ProgramiranjePython > Primjeri > ssss.py > ...
1
2   age = 23
3   if age >= 18:
4       print('Osoba je punoljetna')
5

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka

PS C:\Users\Jerko\Documents\FESB\ProgramiranjePython\Primjeri>  & 'C:\Users\
nsions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../
Primjeri\ssss.py'
Osoba je punoljetna
PS C:\Users\Jerko\Documents\FESB\ProgramiranjePython\Primjeri>
```

Kod 4.4 Primjer if naredbe

If-else naredba se koristi kada je na temelju uvjeta potrebno donijeti neku odluku. Ako je uvjet ispunjen onda se izvršava tijelo *if* odluke, a ako nije onda se izvršava tijelo *else* odluke. Na slici 4.3 je shematski prikazano kako izgleda *If-else* naredba, a kod 4.5 prikazuje jednostavan primjer *If-else* naredbe.

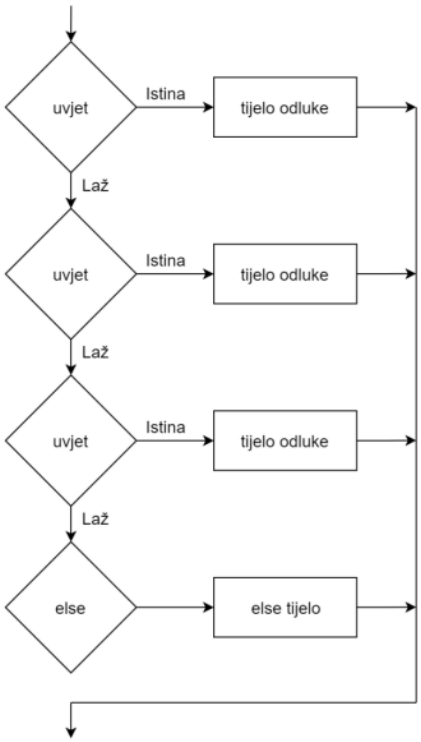


Slika 4.3 Shematski prikaz if-else naredbe



Kod 4.5 Primjer if-else naredbe

Kako je u dosta slučaja potrebno donositi još kompleksnije odluke, tako je uvedena i *if-elif-else* naredba koja izvršava samo tijelo odluke onog uvjeta koji je prvi zadovoljen, a ako nije zadovoljen niti jedan uvjet onda se izvršava tijelo *else* odluke. Na slici 4.4 je shematski prikazano kako izgleda *if-elif-else* naredba, a kod 4.6 prikazuje jednostavan primjer *if-elif-else* naredbe.



Slika 4.4 Shematski prikaz if-elif-else naredbe

C: > Users > Jerko > Documents > FESB > ProgramiranjePython > Primjeri > ssss.py > ...

```
1
2   age = 15
3   if age == 18:
4       print('Osoba ima 18 godina')
5   elif age == 19:
6       print('Osoba ima 19 godina')
7   else:
8       print('Osoba ima manje od 18 ili više od 19 godina')
```

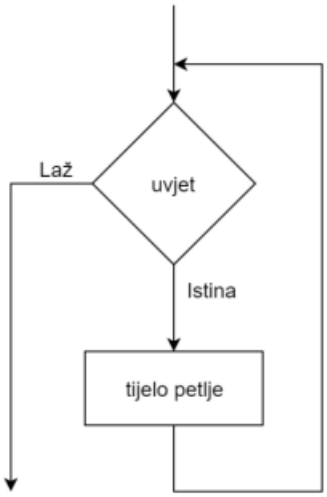
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PS

PS C:\Users\Jerko\Documents\FESB\ProgramiranjePython\Primjeri>  & 'C:\Users\Jerko\
nsions\ms-python.python-2023.14.0\pythonFiles\lib\python\debugpy\adapter/../..\deb
Primjeri\ssss.py'
Osoba ima manje od 18 ili više od 19 godina
PS C:\Users\Jerko\Documents\FESB\ProgramiranjePython\Primjeri>
```

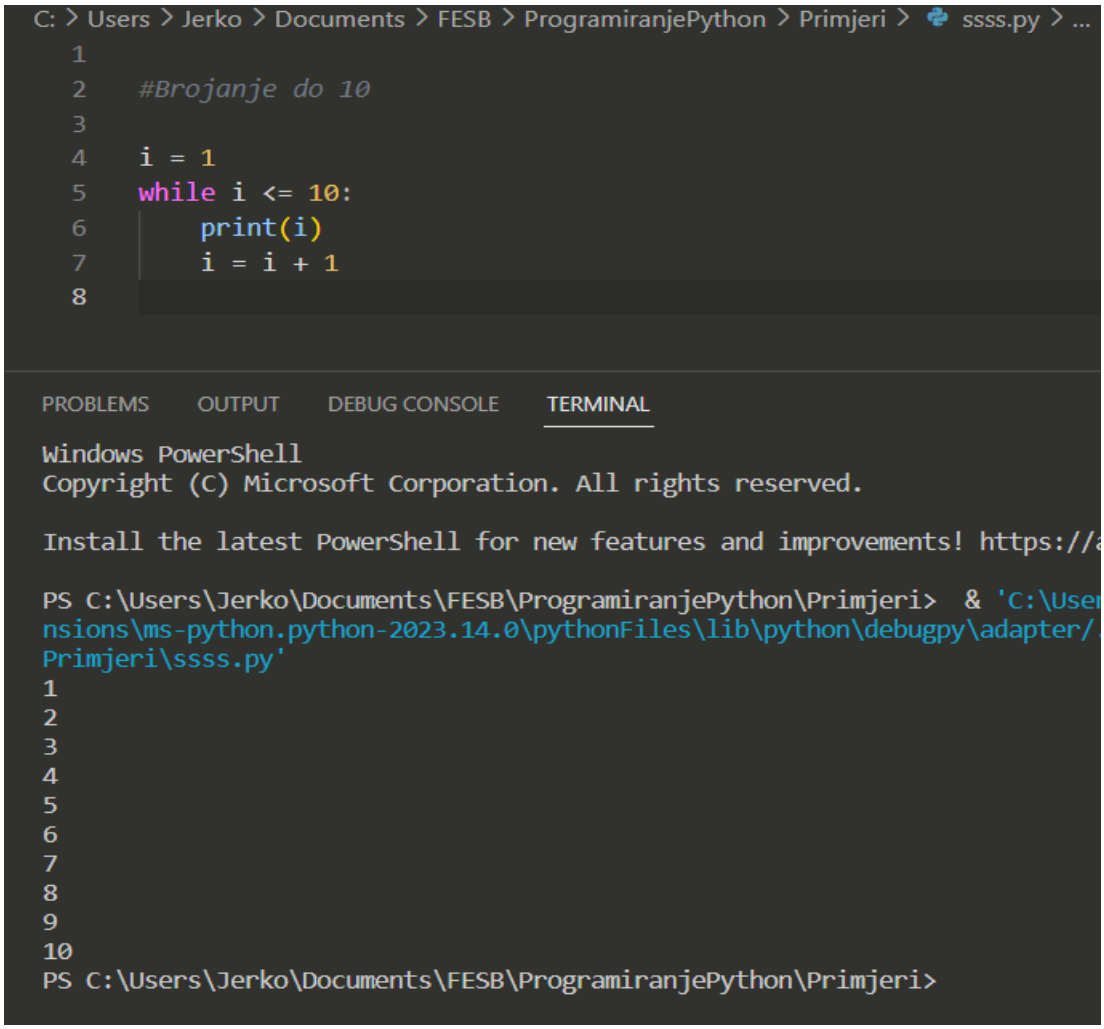
Kod 4.6 Primjer if-elif-else naredbe

While i for petlje

While petlja funkcionira tako da prvo ispituje uvjet koji dolazi iza ključne riječi *while*. Ako je uvjet ispunjen onda se ulazi u petlju i izvršava se blok koda unutar nje koji se naziva „tijelo petlje“. Ako uvjet nije zadovoljen onda se ne ulazi u petlju nego se program nastavi dalje izvršavati. Na slici 4.5 je shematski prikazano kako funkcionira *while* petlja, a kod 4.7 prikazuje jednostavan primjer *while* petlje.

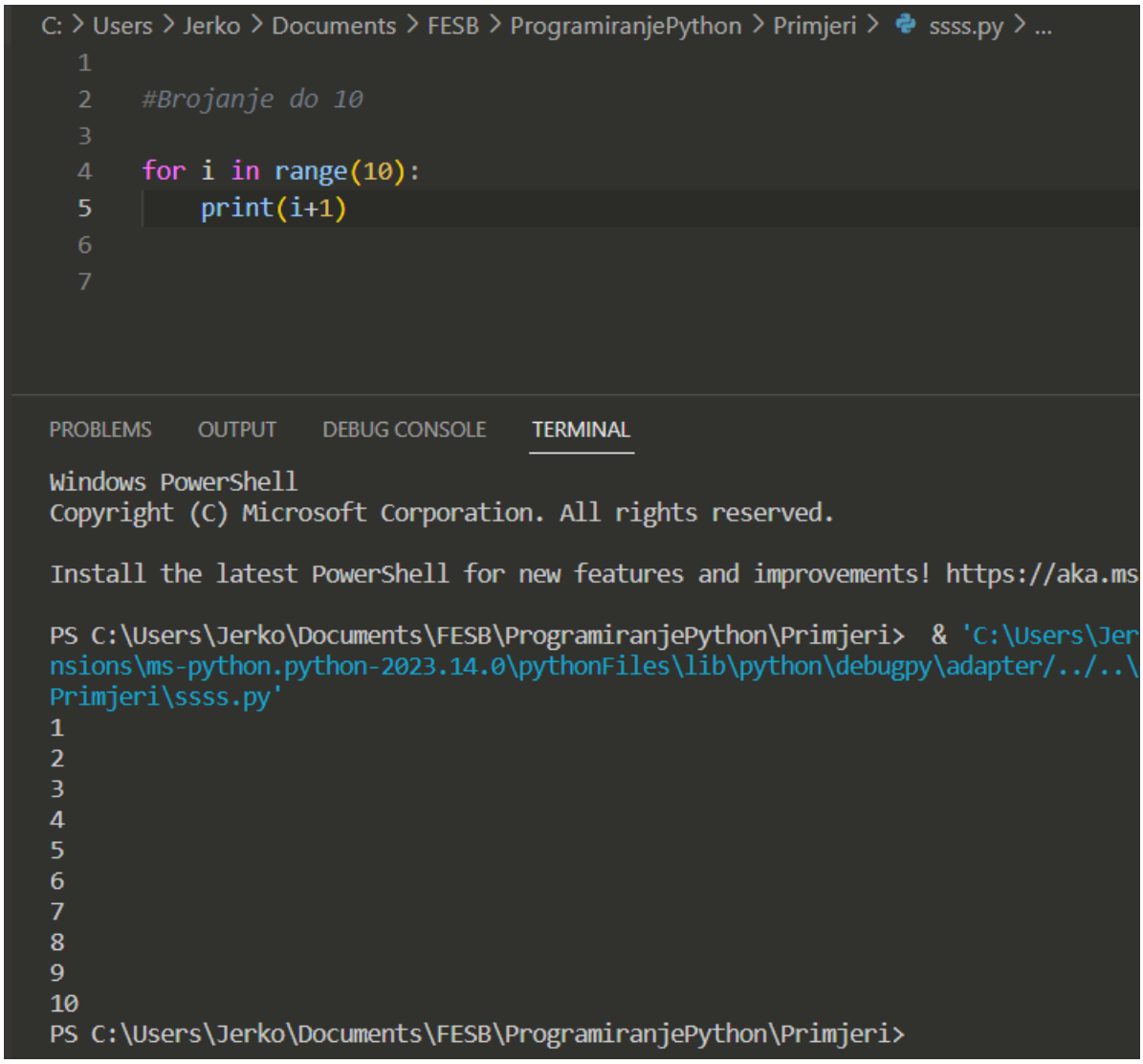


Slika 4.5 Shematski prikaz while petlje



Kod 4.7 Primjer while petlje

Kod Pythona petlja *for* nema početnu ni završnu vrijednost, a ni aritmetički faktor uvećanja te petlja *for* u Pythonu iterira kroz elemente zadane sekvence[12]. Ključne riječi su *for* i *in* te se upotrebljava funkcija *range()* koja ako prima jedan parametar onda kreira listu od onoliko elemenata koliki je parametar, a ako prima dva elementa onda napravi listu elemenata tako da je početni element liste prvi parametar, a posljednji element liste je onda drugi parametar. Ispod je dan kod 4.8 koji radi isto ka i primjer koda 4.7, ali ovaj put s *for* petljom.



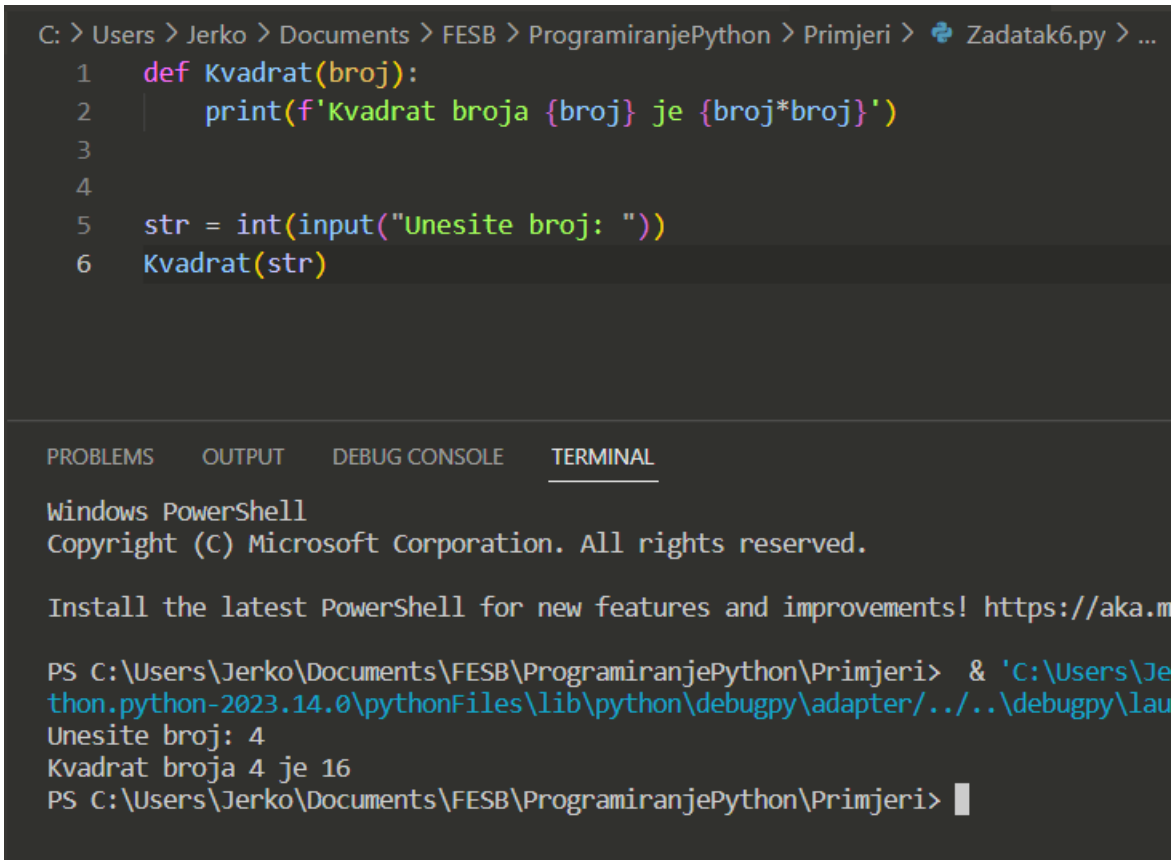
Kod 4.8 Primjer for petlje

4.1.3 Funkcije

Funkcije su programske strukture koje služe kako bi se jednom napisani odsječak koda mogao primijeniti nad više različitih operanda. Nastale su s ciljem da se smanji redundancija jer da nema njih onda bi se ponovo trebao kopirati isti kod za obavljanje neke radnje. Tako na primjer računanje kvadrata nekog broja. Kada bi bilo potrebno računati kvadrat nekog broja više puta u kodu onda bi se svaki put trebao pisati isti kod za računanje i tako se nepotrebno komplicira sami kod koji postaje nepregledniji pa se zato koriste funkcije. Funkcija za računanje kvadrata se definira kada prvi put zatreba i nakon toga ako opet bude potrebno računanje kvadrata onda se pozove ta funkcija. Na slici 4.6 je prikazano kako se definira funkcija.

Slika 4.6 Način definiranja funkcije[12]

Prvo je potrebno napisati ključnu riječ *def* te nakon toga napisati ime funkcije. Poželjno je koristiti smisleno ime koje opisuje što ta funkcija zapravo radi (npr. ako funkcija obavlja računsku operaciju zbrajanja onda je logičnije da se zove „zbroji“ nego „oduzmi“). Nakon imena funkcije dolaze zagrade u kojima se nalaze parametri koje funkcija prima. Tu je isto poželjno navesti samo one parametre koji su potrebni za rad funkcije, a ne navoditi ih „bez veze“. Na kraju te linije se stavlja dvotočka „ : “ koja označava da se ulazi u tijelo funkcije u kojem se nalazi blok naredbi. Na samom kraju funkcije se koristi ključna riječ *return* koja označava da funkcija nakon obavljene radnje vraća neku povratnu vrijednost te se nakon riječi navodi i vrijednost koja se vraća (ime varijable ili neki izraz). Ispod je primjer koda 4.9 koji prikazuje jednostavnu funkciju za kvadriranje nekog broja.



Kod 4.9 Primjer funkcije za kvadriranje

4.2 Biblioteka TensorFlow

TensorFlow je softverska biblioteka otvorenog koda za numeričko izračunavanje pomoću grafikona protoka podataka[13]. Može se koristiti u nizu zadataka, ali ima poseban fokus na obuku i zaključivanje dubokih neuronskih mreža. Googleov Brain tim je 2015. godine razvio TensorFlow za Googleovu internu upotrebu u istraživanju i proizvodnji[14]. Ova biblioteka omogućuje razvoj i implementaciju različitih modela strojnog učenja, posebno dubokih neuronskih mreža, za rješavanje različitih problema uključujući prepoznavanje uzoraka, analizu slika, obradu prirodnog jezika, generiranje sadržaja i još mnogo toga. Iako je TensorFlow osmišljen za upotrebu s različitim programskim jezicima, Python je najčešće korišten jezik za rad s TensorFlowom. Jedan od glavnih razloga zašto je to tako je zato što je sintaksa Pythona čitljiva i intuitivna, što olakšava rad s TensorFlowom, posebno za one koji nisu „stručnjaci“ za programiranje. Nadalje, Python omogućuje brz razvoj i testiranje modela. Ovo je osobito važno prilikom eksperimentiranja s različitim arhitekturama modela i hiperparametrima. U razvoju modernih programa vrlo je važna interoperabilnost, a Python ima dobru interoperabilnost s drugim jezicima kao što su C, C++ i Java. Ovo je korisno ako

treba optimizirati određene dijelove koda ili koristiti postojeće biblioteke napisane u drugim jezicima. Na kraju, Python je popularan jezik za razvoj AI aplikacija i analizu podataka, a TensorFlow je moćan alat za stvaranje modela koji rješavaju složene probleme u tim područjima. Kombinacija ovih dviju tehnologija omogućuje razvoj cjelovitih rješenja i zato su idealan spoj u području razvoja umjetne inteligencije. TensorFlow Lite je također softverska biblioteka otvorenog koda, međuplatformski okvir dubokog učenja koji pretvara unaprijed obučeni model u TensorFlowu u poseban format koji se može optimizirati za brzinu ili pohranu[14]. Tenzori su glavne komponente TensorFlowa definirani kao osnovne strukture podataka tj. višedimenzionalni niz ili popis. Spojni rubovi u bilo kojem dijagramu toka su tenzori. To su višelinearne karte koje mogu biti bilo što, od vektorskih prostora do realnih brojeva. Odnosno, tenzor može biti skalar, vektor ili matrica. Za izgradnju tenzora, potrebno je razmotriti izgradnju i pretvaranje n-dimenzionalnog niza. Moguće su dvije dimenzije, a to su sljedeće: jednodimenzionalni tenzor, normalna struktura niza koja uključuje jedan skup vrijednosti iste vrste podataka, te dvodimenzionalni tenzor za čije se stvaranje koristi niz nizova[15]. U idućem dijelu je dan primjer izrade jednostavne neuronske mreže korištenjem TensorFlow biblioteke.

4.2.1 Primjer izrade jednostavne neuronske mreže

Iako su današnje aplikacije vrlo složene i komplicirane, ovdje je prikazano da je rad s TensorFlow bibliotekom izrazito jednostavan. Neuronska mreža koja će biti izrađena prepoznavat će znakove uz pomoć MNIST baze podataka. MNIST je poznata baza podataka koja se često koristi za testiranje algoritama za prepoznavanje slika i strojnog učenja[16]. Ova baza sadrži skup slika rukom pisanih brojeva (0-9) napisanih na bijelom pozadinskom polju. Svaka slika je dimenzija 28x28 piksela i prikazuje crno-bijeli prikaz broja. Primjer je preuzet sa službene stranice TensorFlowa[17]. Na samom početku je prvo potrebno uključiti biblioteku TensorFlow, a to prikazuje kod 4.10.

```
[1]  import tensorflow as tf
     print("TensorFlow version:", tf.__version__)

     TensorFlow version: 2.12.0
```

Kod 4.10 Uključivanje biblioteke TensorFlow

U idućem je koraku potrebno uključiti MNIST bazu podataka te pripremiti podatke za obradu tako da ih se pretvori u brojeve s pomičnim zarezom, a to prikazuje kod 4.11.

```
[2]  mnist = tf.keras.datasets.mnist

     (x_train, y_train), (x_test, y_test) = mnist.load_data()
     x_train, x_test = x_train / 255.0, x_test / 255.0

     Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
     11490434/11490434 [==============================] - 0s 0us/step
```

Kod 4.11 Uključivanje MNIST baze podataka i priprema podataka za obradu

Sljedeći korak je izgradnja *tf.keras* sekvencijalnog modela slaganjem slojeva (kod 4.12) i model vraća vektor neobrađenih predviđanja za svaki primjer (kod 4.13).

```
[3]  model = tf.keras.models.Sequential([
       tf.keras.layers.Flatten(input_shape=(28, 28)),
       tf.keras.layers.Dense(128, activation='relu'),
       tf.keras.layers.Dropout(0.2),
       tf.keras.layers.Dense(10)
     ])
```

Kod 4.12 Izgradnja tf.keras sekvencijalnog modela

```
predictions = model(x_train[:1]).numpy()
predictions

array([[ 0.32473785,  0.21438828,  1.0525547 , -0.04642025,  0.0936158 ,
         0.1463013 , -0.25231546, -0.2886321 ,  0.39501536,  0.0639919 ]],
      dtype=float32)
```

Kod 4.13 Ispis vektora predviđanja

Zatim se koristi funkcija *tf.nn.softmax* koja pretvara te vektore neobrađenih predviđanja u vjerojatnosti za svaku klasu (kod 4.14).

```
[5]  tf.nn.softmax(predictions).numpy()

     array([[0.108312  , 0.09699567, 0.22426601, 0.07472823, 0.08596102,
             0.09061135, 0.06082268, 0.05865343, 0.11619776, 0.08345187]],
           dtype=float32)
```

Kod 4.14 Funkcija tf.nn.softmax

U idućem je koraku potrebno definirati funkciju gubitka za obuku koja uzima vektor neobrađenih predviđanja i *True* indeks, pa vraća skalarni gubitak za svaki primjer (kod 4.15).

```
[6]  loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

Kod 4.15 Funkcija gubitka

```
[7]  loss_fn(y_train[:1], predictions).numpy()

     2.401176
```

Kod 4.16 Podešavanje funkcije gubitka

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

Kod 4.17 Keras model.compile

Prije početka treninga, potrebno je konfigurirati i kompajlirati model koristeći Keras *model.compile* (kod 4.17). Klasa optimizatora se postavlja na *adam*, gubitak se postavlja na funkciji *loss_fn* koja je podešena ranije (kod 4.16). Na kraju se navode metrike koje će se procijeniti za model postavljanjem parametra metrike na *accuracy*. Zatim se koristi metoda *model.fit* kako bi se minimizirali gubitci i prilagodili parametri (kod 4.18).

```
model.fit(x_train, y_train, epochs=5)

Epoch 1/5
1875/1875 [==============================] - 9s 4ms/step - loss: 0.2981 - accuracy: 0.9130
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1442 - accuracy: 0.9570
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1099 - accuracy: 0.9667
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0857 - accuracy: 0.9739
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0747 - accuracy: 0.9762
<keras.callbacks.History at 0x7aa9cb0ff880>
```

Kod 4.18 Metoda model.fit

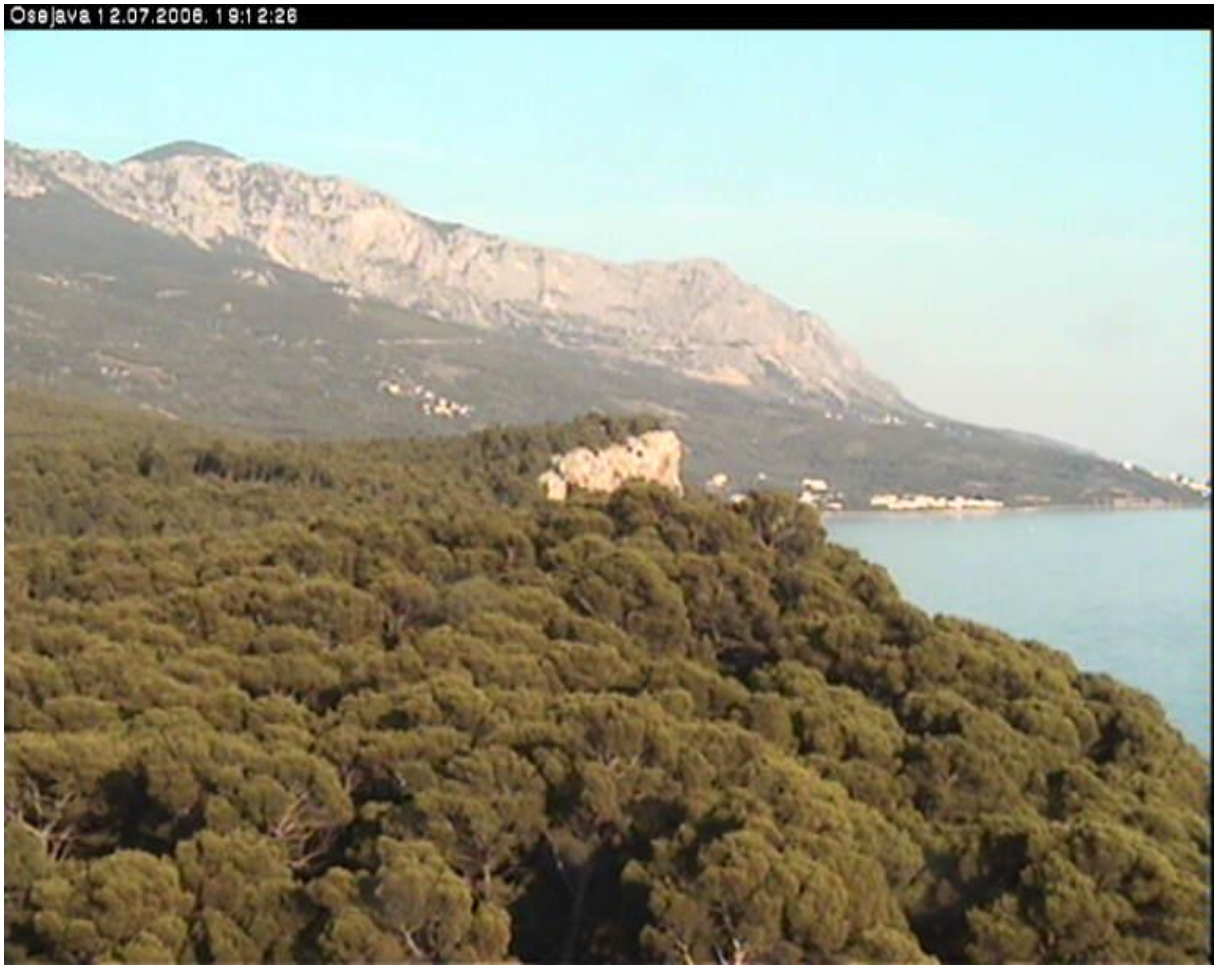
Za sam kraj se koristi metoda *model.evaluate* kako bi se provjerile performanse modela (kod 4.19).

```
[10]  model.evaluate(x_test,  y_test, verbose=2)

      313/313 - 1s - loss: 0.0746 - accuracy: 0.9768 - 715ms/epoch - 2ms/step
      [0.07464639842510223, 0.9768000245094299]
```

Kod 4.19 Metoda model.evaluate

5 PRAKTIČNI DIO

Nakon svih prethodnih odlomaka u kojima su stečeni teorijski preduvjeti za izradu sustava za detekciju požara na slici, ovaj dio završnog rada se sada fokusira na izradu konkretnog modela. Model je izrađen u Google Colabu, besplatnoj online platformi koja pruža pristup moćnim računalnim resursima za analizu podataka i istraživanje metoda dubokog učenja[18], dok je za ulazne podatke korištena FESB MLID baza podataka. FESB označava Fakultet elektrotehnike, strojarstva i brodogradnje, a MLID označava skup podataka koji se odnose na slike mediteranskog krajolika (eng. Mediterranean Landscape Image Dataset)[19]. Sastoji se od ukupno 400 slika prirode mediteranskog krajobraza te je na slici prikazan jedan primjer slike iz te baze.



Slika 5.1 Primjer slike iz FESB MLID baze podataka[19]

Od ukupno 400 slika, njih 190 je korišteno za treniranje modela za prepoznavanje požara dok je njih 30 nasumičnim odabirom korišteno za testiranje. Omjer slika za treniranje u odnosu na slike za testiranje je 84:16.

5.1 Pripremanje Colab bilježnice

Na samom je početku potrebno postaviti vrijeme izvođenja na GPU i pripremiti okruženje u kojem se razvija model. Potrebno je se pobrinuti da sve potrebne biblioteke budu instalirane i svi potrebni moduli budu uvezeni.

Kod 5.1 Pripremanje okruženja za razvoj modela

Iznad, u kodu 5.1, je prvo korištena naredba *!pip* za instalaciju TensorFlow biblioteke s podrškom za GPU i TensorFlow Hub biblioteke koja omogućava ponovno korištenje obučenih modela. U idućim se linijama uvoze potrebni moduli: *matlplotlib.pylab*, omogućuje crtanje grafova i vizualizaciju podataka, prethodno instalirane TensorFlow i TensorFlow Hub, NumPy pruža podršku za rad s višedimenzionalnim nizovima i matricama, te matematičke funkcije za njih, Pandas omogućava manipulaciju i analizu podataka kroz DataFrame strukturu. Linijom *pd.set_option("display.precision", 8)* se postavlja da će se brojevi s pomičnim zarezom u DataFrame-ima prikazivati s osam decimalnih mjesta.

5.2 Povezivanje s bazom podataka

U ovom je koraku prvo potrebno povezati Google Drive s Google Colab okruženjem kako bi se moglo pristupiti podacima za treniranje i testiranje koji se nalaze na disku i postaviti putanju do ciljanih podataka. Kod 5.2 prikazuje taj postupak.

```
[3]
from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[4]
data_root='/content/drive/My Drive/ZavrsniRad/smoke_images'
print(data_root)

/content/drive/My Drive/ZavrsniRad/smoke_images
```

Kod 5.2 Povezivanje s Google Drive-om i postavljanje putanje

Nakon linije *drive.mount('/content/drive')* dolazi upit za autorizaciju pristupa Google Drive-u. Nakon autorizacije, sadržaj Google Drive memorije bit će dostupan unutar direktorija */content/drive* u Google Colab okruženju. Zatim se u varijablu *data_root* sprema putanja do direktorija s podacima koja se kasnije i ispisuje linijom *print(data_root)*. U idućem je koraku potrebno pripremiti podatke za treniranje i validaciju modela, uključiti potrebne biblioteke te pripremiti generatore podataka za trening i validaciju koji će se koristiti za treniranje modela, a to prikazuje kod 5.3.

```
[5]
import tensorflow as tf
import keras

IMAGE_SHAPE = (224, 224)
TRAINING_DATA_DIR = str(data_root)
print(TRAINING_DATA_DIR);
datagen_kwargs = dict(rescale=1./255, validation_split=.20)
valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwargs)
valid_generator = valid_datagen.flow_from_directory(
TRAINING_DATA_DIR,
subset="validation",
shuffle=True,
target_size=IMAGE_SHAPE
)
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(**datagen_kwargs)
train_generator = train_datagen.flow_from_directory(
TRAINING_DATA_DIR,
subset="training",
shuffle=True,
target_size=IMAGE_SHAPE)

/content/drive/My Drive/ZavrsniRad/smoke_images
Found 38 images belonging to 2 classes.
Found 152 images belonging to 2 classes.
```

Kod 5.3 Pripremanje podataka za treniranje i validaciju

Biblioteka Keras se koristi za definiranje arhitekture neuronske mreže i postavljanje parametara modela za prepoznavanje dima na slikama. Varijabla *IMAGE_SHAPE* postavlja dimenzije slika koje će se koristiti za trening i validaciju. U ovom slučaju, slike će biti skalirane na veličinu 224x224 piksela. Putanja do direktorija s podacima, koja je prethodno već spremljena u varijablu *data_root*, se ovaj put sprema u varijablu *TRAINING_DATA_DIR* i ispisuje. Dalje se definira rječnik *datagen_kwargs* koji sadrži argumente za konfiguriranje

ImageDataGenerator-a. Argument *rescale=1./255* normalizira boje slika na raspon [0, 1], a *validation_split=.20* postavlja omjer podataka koji će se koristiti za validaciju. U idućoj se liniji kreira se objekt *valid_datagen* klase *ImageDataGenerator* za generiranje validacijskih podataka. U liniji ispod se koristi *flow_from_directory* metoda kako bi se generirali podaci iz direktorija za validaciju. Metodi se šalju sljedeći podaci: *TRAINIG_DATA_DIR*, *subset="validation*" koji označava da se generiraju podaci za validaciju, *shuffle=True* koji označava da će se podaci izmiješati pri generiranju te *target_size=IMAGE_SHAPE* koji postavlja dimenzije na koje će se slike skalirati. Idućom se linijom kreira objekt *train_datagen* klase *ImageDataGenerator* za generiranje trening podataka. U liniji ispod se opet koristi *flow_from_directory* metoda kako bi se generirali podaci iz direktorija za trening. Metodi se šalju isti podaci kao u prethodnom slučaju uz jedinu razliku što je *subset="training"* koji označava da se generiraju podaci za trening.

```
[7]
import numpy as np
```

```
[8]
image_batch_train, label_batch_train = next(iter(train_generator))
print("Image batch shape: ", image_batch_train.shape)
print("Label batch shape: ", label_batch_train.shape)
dataset_labels = sorted(train_generator.class_indices.items(), key=lambda pair:pair[1])
dataset_labels = np.array([key.title() for key, value in dataset_labels])
print(dataset_labels)
```

```
Image batch shape:  (32, 224, 224, 3)
Label batch shape:  (32, 2)
['No_Smoke' 'Smoke']
```

Kod 5.4 Generiranje serije trening slika

Kod 5.4 omogućava razumijevanje oblika i sadržaja generirane serije trening slika i oznaka, te da se stvori lista interpretacija za oznake koje model koristi. Linija *image_batch_train, label_batch_train = next(iter(train_generator))* koristi *next(iter(...))* za dohvaćanje sljedeće serije slika i njihovih odgovarajućih oznaka iz generatora trening podataka. Iduće dvije linije su zaslužne za ispis oblika (dimenzija) serije slika i oznaka koja je generirana. Iduća linija kreira sortiranu listu parova (oznaka, indeks) iz rječnika koji mapira nazive oznaka na njihove indekse, a linija ispod stvara niz naziva oznaka iz sortirane liste parova, gdje se prvo slovo svake oznake pretvara u veliko slovo. Na kraju se ispisuje lista naziva oznaka.

5.3 Treniranje modela

Nakon što su pripremljeni podaci i generatori za trening i validaciju, u ovoj fazi izrade se definira sekvencijalni model za duboko učenje koji koristi prethodno treniranu MobileNetV2

arhitekturu kao osnovu (kod 5.5). Korištenje prethodno trenirane arhitekture kao osnove može olakšati proces učenja modela.

```
[9]
import tensorflow_hub as hub
```

```
[10]
model = tf.keras.Sequential([
hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4",
output_shape=[1280],
trainable=False),
tf.keras.layers.Dropout(0.4),
tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
model.build([None, 224, 224, 3])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 2)                 2562

=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

Kod 5.5 Definiranje sekvencijalnog modela za duboko učenje

Linija *model = tf.keras.Sequential([...])* definira sekvencijalni model koristeći *tf.keras.Sequential*, što omogućava dodavanje slojeva jedan za drugim. Linija *hub.KerasLayer("https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4", output_shape=[1280], trainable=False)* dodaje sloj *Keras Layer* iz TensorFlow Hub-a koji koristi prethodno treniranu MobileNetV2 arhitekturu. Postavlja se da sloj nije treniran (*trainable=False*) i određuje se izlazni oblik (*output_shape=[1280]*). *tf.keras.layers.Dropout(0.4)* dodaje sloj odbacivanja s vjerojatnošću odbacivanja od 0.4. Ovo pomaže u sprječavanju prenaučenosti modela. Iduće se dodaje potpuno povezani sloj s brojem izlaznih neurona jednakim broju klasa (oznaka) u podacima. Aktivacijska funkcija sloja je *softmax*, što je tipično za višeklasne klasifikacijske modele. Metodom *model.build()* se radi model i postavlja ulazni oblik na *[None, 224, 224, 3]*, što odgovara dimenzijama slika koje će model primiti kao ulaz. Metodom *model.summary()* se ispisuje sažeti pregled strukture modela, uključujući broj parametara i veličine izlaza svakog sloja. Idući korak prije samog treninga je postavljanje ključnih parametara za trening modela, uključujući način optimizacije, funkciju gubitka i metriku evaluacije, a to prikazuje kod 5.6.

```
[11]
    model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['acc'])
```

Kod 5.6 Postavljanje ključnih parametara za trening

Prvo se postavlja optimizator za model. U ovom slučaju koristi se Adam optimizator, koji je popularan algoritam za optimizaciju gradijenta u dubokom učenju. Pozivom *tf.keras.optimizers.Adam()* se stvara instanca ovog optimizatora s postavkama po defaultu. Zatim se postavlja funkcija gubitka koja će se koristiti tijekom treniranja modela. U ovom slučaju koristi se kategorijalna križna entropija, što je često korištena funkcija gubitka za višeklasne klasifikacije. Na kraju se definira lista metrika koje će se koristiti za evaluaciju modela tijekom i nakon treniranja. U ovom slučaju, koristi se metrika točnosti koja mjeri postotak ispravno klasificiranih instanci. Model je sada spreman za treniranje. Idući dio koda 5.7 izvršava trening modela na pripremljenim podacima za trening i validaciju.

```
#Run model training

steps_per_epoch = np.ceil(train_generator.samples/train_generator.batch_size)
val_steps_per_epoch = np.ceil(valid_generator.samples/valid_generator.batch_size)
hist = model.fit(
train_generator,
epochs=10,
verbose=1,
steps_per_epoch=steps_per_epoch,
validation_data=valid_generator,
validation_steps=val_steps_per_epoch).history
```

```
Epoch 1/10
5/5 [==============================] - 45s 8s/step - loss: 0.8082 - acc: 0.5263 - val_loss: 0.6551 - val_acc: 0.6316
Epoch 2/10
5/5 [==============================] - 1s 234ms/step - loss: 0.6201 - acc: 0.6184 - val_loss: 0.5699 - val_acc: 0.7368
Epoch 3/10
5/5 [==============================] - 1s 211ms/step - loss: 0.4485 - acc: 0.8092 - val_loss: 0.5552 - val_acc: 0.6842
Epoch 4/10
5/5 [==============================] - 1s 308ms/step - loss: 0.5122 - acc: 0.7961 - val_loss: 0.5204 - val_acc: 0.7105
Epoch 5/10
5/5 [==============================] - 1s 251ms/step - loss: 0.4080 - acc: 0.8158 - val_loss: 0.4946 - val_acc: 0.7632
Epoch 6/10
5/5 [==============================] - 1s 229ms/step - loss: 0.3769 - acc: 0.8158 - val_loss: 0.5037 - val_acc: 0.7368
Epoch 7/10
5/5 [==============================] - 1s 220ms/step - loss: 0.3492 - acc: 0.8618 - val_loss: 0.5268 - val_acc: 0.7632
Epoch 8/10
5/5 [==============================] - 1s 227ms/step - loss: 0.2924 - acc: 0.8684 - val_loss: 0.5104 - val_acc: 0.7368
Epoch 9/10
5/5 [==============================] - 1s 214ms/step - loss: 0.2595 - acc: 0.8947 - val_loss: 0.4995 - val_acc: 0.7368
Epoch 10/10
5/5 [==============================] - 1s 213ms/step - loss: 0.2972 - acc: 0.8947 - val_loss: 0.5018 - val_acc: 0.7368
```

Kod 5.7 Postupak treniranja modela

Prve dvije linije računaju broj koraka (iteracija) po epohi za trening i validaciju. Linija *steps_per_epoch* računa koliko koraka je potrebno da se obradi sve trening primjere u jednoj epohi, a *val_steps_per_epoch* radi isto za validacijske primjere. Idući dio izvršava stvarni trening. Metoda *fit* prima sljedeće važne parametre: *train_generator* je generator podataka za

trening, *epochs=10* je broj epoha (iteracija kroz kompletan skup podataka) za treniranje modela, *verbose=1* se postavlja da ispisuje napredak treniranja tokom svake epohe, *steps_per_epoch=steps_per_epoch* je broj koraka po epohi, *validation_data=valid_generator* je generator podataka za validaciju, *validation_steps=val_steps_per_epoch* je broj koraka po epohi validacije. Nakon završetka treniranja, povijest treniranja (gubici i metrike) će biti pohranjena u varijablu *hist*. Također, ispisivat će se informacije o napretku treniranja tijekom svake epohe. Nakon završetka treninga je potrebno dobiti uvid u perfomanse modela.

```
[13]
# Measure accuracy and loss after training

final_loss, final_accuracy = model.evaluate(valid_generator, steps = val_steps_per_epoch)
```

```
2/2 [==============================] - 0s 20ms/step - loss: 0.5018 - acc: 0.7368
```

```
[14]
print("Final loss: {:.2f}".format(final_loss))
print("Final accuracy: {:.2f}%".format(final_accuracy * 100))
```

```
Final loss: 0.50
Final accuracy: 73.68%
```

Kod 5.8 Ispis konačnih performansi modela

Kod 5.8 mjeri točnost i gubitak modela nakon što je završeno treniranje i vidljivo je da točnost modela iznosi 73.68%, a gubitak je 0.5. Kako su ljudi vizualna bića i najbolje razumiju podatke kada su im prikazani vizualno, tako je i postupak treniranja prikazan grafički. Prvo je potrebno ponovo uključiti modul *matlplotlib.pylab*, kod 5.9, pa zatim napisati kod za crtanje grafova, kod 5.10.

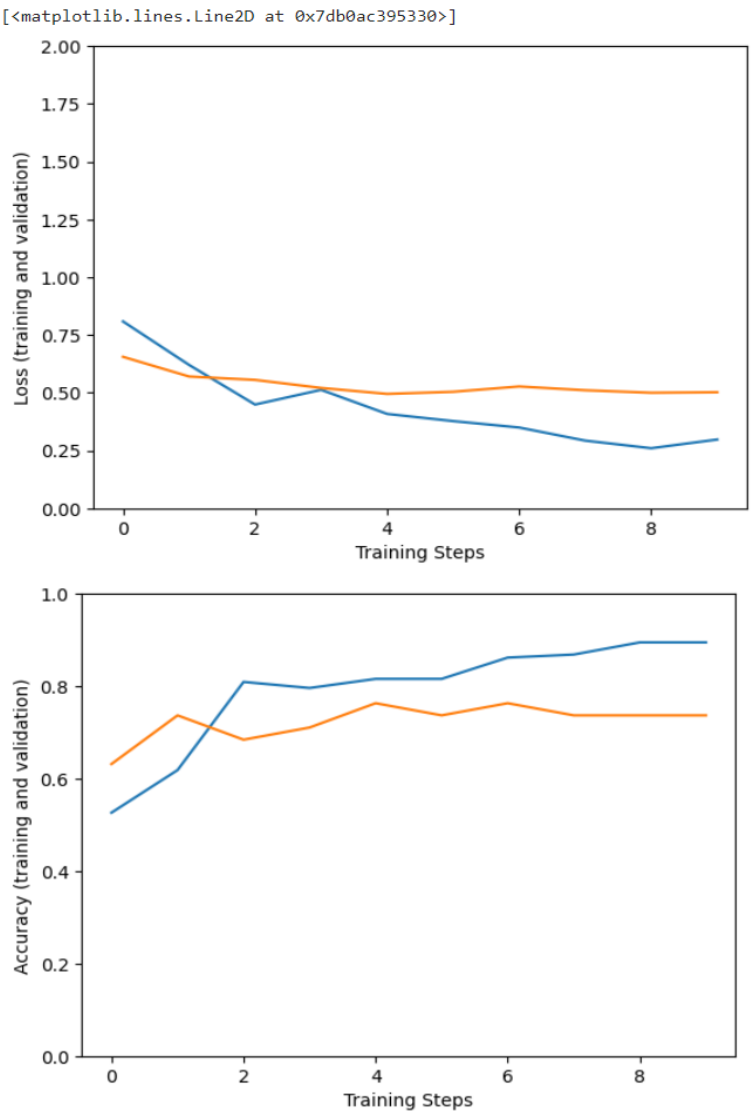
```
[15]
import matplotlib.pyplot as plt
```

Kod 5.9 Uključivanje modula matlplotlib.pylab

```
[16]
# Visualize training process

plt.figure()
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(hist["loss"])
plt.plot(hist["val_loss"])

plt.figure()
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(hist["acc"])
plt.plot(hist["val_acc"])
```

Kod 5.10 Crtanje grafova za vizualizaciju faze treniranja

Crtaju se dva grafa (jedan za vizualizaciju gubitka, a drugi za vizualizaciju točnosti). Rezultat koda 5.10 je slika 5.2 na kojoj je se plava krivulja na oba grafa odnosi na postupak treniranja, a narančasta se krivulja odnosi na validacijski set podataka.



Slika 5.2 Vizualizacija treninga

Vidljivo je da se iz koraka u korak smanjuje gubitak, a povećava točnost.

5.4 Spremanje modela

Nakon obavljenog treninga, u ovoj se fazi model sprema na disk i ponovo učitava. Prije samog spremanja provodi se novi trening, prije je potrebno uvesti potrebne module (kod 5.11), kako bi se usporedili rezultati iz ovog ciklusa i prethodnog, a to je dano u kodu 5.12.

```
[17]
      import tensorflow as tf
      import keras
      import tensorflow_hub as hub
```

Kod 5.11 Potrebni moduli za novo treniranje

```
[18]
steps_per_epoch = np.ceil(train_generator.samples/train_generator.batch_size)
val_steps_per_epoch = np.ceil(valid_generator.samples/valid_generator.batch_size)
hist = model.fit(
train_generator,
epochs=10,
verbose=1,
steps_per_epoch=steps_per_epoch,
validation_data=valid_generator,
validation_steps=val_steps_per_epoch).history
```

```
Epoch 1/10
5/5 [==============================] - 1s 255ms/step - loss: 0.2403 - acc: 0.9013 - val_loss: 0.5055 - val_acc: 0.7368
Epoch 2/10
5/5 [==============================] - 1s 232ms/step - loss: 0.2501 - acc: 0.9013 - val_loss: 0.5201 - val_acc: 0.7105
Epoch 3/10
5/5 [==============================] - 1s 221ms/step - loss: 0.2656 - acc: 0.8816 - val_loss: 0.5200 - val_acc: 0.7105
Epoch 4/10
5/5 [==============================] - 1s 221ms/step - loss: 0.1984 - acc: 0.9211 - val_loss: 0.5327 - val_acc: 0.7105
Epoch 5/10
5/5 [==============================] - 1s 239ms/step - loss: 0.2208 - acc: 0.9342 - val_loss: 0.5270 - val_acc: 0.7105
Epoch 6/10
5/5 [==============================] - 1s 223ms/step - loss: 0.1851 - acc: 0.9276 - val_loss: 0.5232 - val_acc: 0.7105
Epoch 7/10
5/5 [==============================] - 1s 233ms/step - loss: 0.1920 - acc: 0.9408 - val_loss: 0.5192 - val_acc: 0.7368
Epoch 8/10
5/5 [==============================] - 1s 306ms/step - loss: 0.1876 - acc: 0.9342 - val_loss: 0.5240 - val_acc: 0.7105
Epoch 9/10
5/5 [==============================] - 1s 229ms/step - loss: 0.1777 - acc: 0.9474 - val_loss: 0.5325 - val_acc: 0.7105
Epoch 10/10
5/5 [==============================] - 1s 224ms/step - loss: 0.1936 - acc: 0.9342 - val_loss: 0.5296 - val_acc: 0.7105
```

Kod 5.12 Ponovno treniranje modela

Kod Kod 5.12 je potpuno isti kao i kod 5.7. Jedina je razlika u ispisu gdje se vidi da je trening obavljen u kraćem vremenu u odnosu na prvi put i krajnji gubitak je manji, a točnost veća. Sada se dolazi do postupka spremanja modela (kod 5.13).

```
[19]
      from tensorflow import keras
```

```
[20]
      SMOKE_SAVED_MODEL = "/content/drive/My Drive/ZavrsniRad/saved_models/smoke"
      model.save(SMOKE_SAVED_MODEL)
      smoke_model = keras.models.load_model(SMOKE_SAVED_MODEL)
```

```
[21]
      val_image_batch, val_label_batch = next(iter(valid_generator))
      true_label_ids = np.argmax(val_label_batch, axis=-1)
      print("Validation batch shape:", val_image_batch.shape)
```

```
Validation batch shape: (32, 224, 224, 3)
```

Kod 5.13 Spremanje modela

Prvo je potrebno uključiti biblioteku koja će se koristiti. Varijabla *SHOE_SAVED_MODEL* definira putanju do direktorija gdje će se spremiti model i povezani podaci. Idućom linijom se model sprema u direktorij koji je određen putanjom, a linijom ispod se spremljeni model učitaje u varijablu *smoke_model*. Dalje se koristi *next(iter(...))* za dohvaćanje sljedeće serije slika i oznaka iz generatora podataka za validaciju. U liniji ispod se koristi funkcija *np.argmax* kako bi se izračunali stvarni indeksi najviših vrijednosti u oznakama. Ovo je korisno jer su oznake kodirane kao one-hot vektori, a *argmax* vraća indekse najvećih vrijednosti. Na kraju se ispisuje oblik (dimenzije) serije validacijskih slika koja je generirana.

5.5 Testiranje modela

Nakon što je utvrđeno da je model ispravno spremljen i ponovo učitan, vrijeme je da se ispitaju njegove stvarne performanse na konkretnom primjeru, kod 5.14. To se radi u ovoj fazi izrade modela za detekciju dima.

```
[23]
     import pandas as pd
```

```
[24]
     tf_model_predictions = smoke_model.predict(val_image_batch)
     tf_pred_dataframe = pd.DataFrame(tf_model_predictions)
     tf_pred_dataframe.columns = dataset_labels
     print("Prediction results for the first elements")
     tf_pred_dataframe.head()
```

```
1/1 [==============================] - 1s 545ms/step
Prediction results for the first elements
```

	No_Smoke	Smoke
0	0.80099499	0.19900499
1	0.37592763	0.62407243
2	0.50960195	0.49039805
3	0.96494830	0.03505175
4	0.83154076	0.16845930

Kod 5.14 Predviđanja modela za validacijski skup

Na ponovno učitanom modelu se koristi metoda *predict* kako bi izvršio predviđanja na validacijskom skupu slika. Rezultat su predviđene vjerojatnosti za svaku klasu. Iduća linija stvara DataFrame koristeći predviđene vjerojatnosti. Svaki redak u DataFrame-u predstavlja jednu sliku, a svaki stupac predstavlja vjerojatnosti za određenu klasu. Linija ispod postavlja nazive stupaca u DataFrame prema nazivima klasa iz *dataset_labels*. Na kraju se naslov koji označava prikaz predviđanja za prve elemente i prikazuje prvih nekoliko redaka DataFrame-a

s predviđenim vjerojatnostima za svaku klasu. Dvije linije koda 5.15 omogućuju da se dobiju stvarno predviđene oznake (nazive klasa) za svaku sliku u validacijskom skupu na temelju predviđenih vjerojatnosti modela.

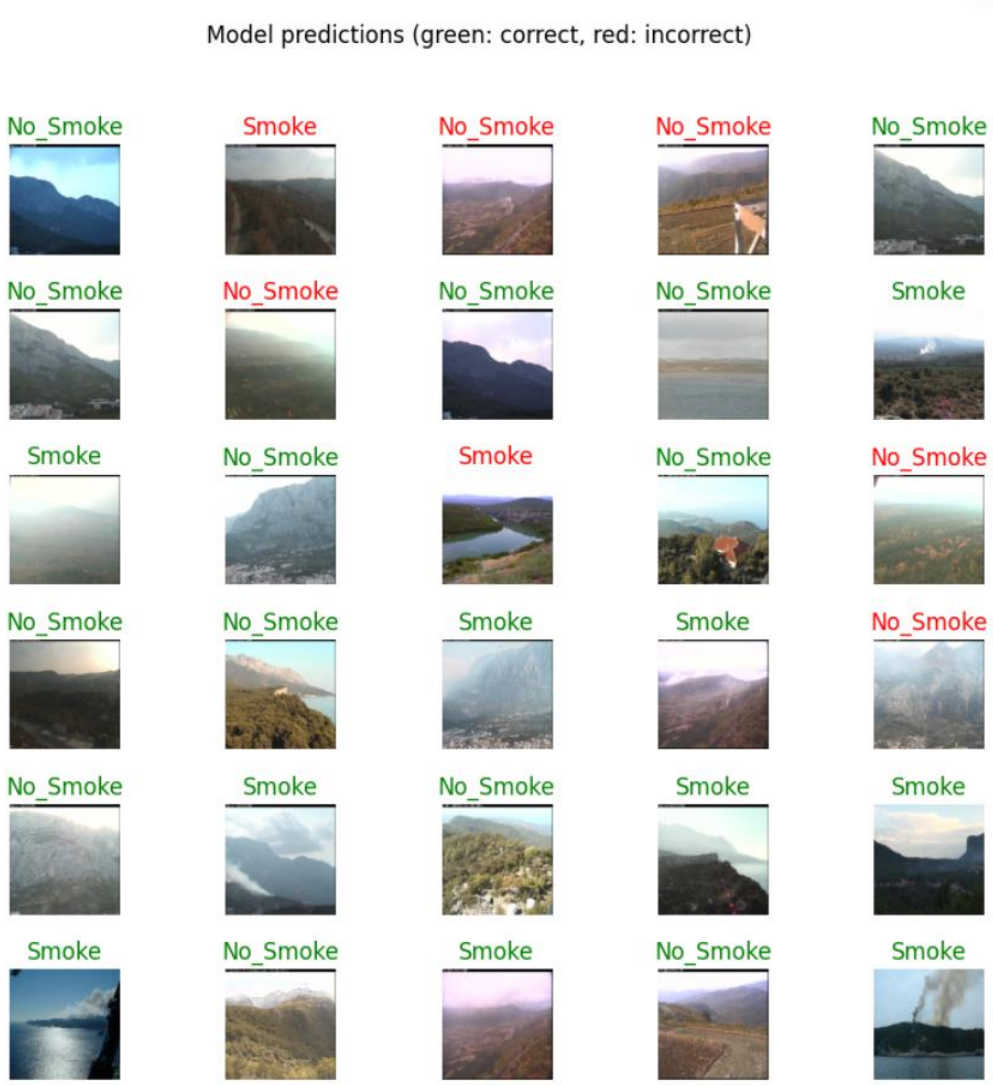
```
[25]
      predicted_ids = np.argmax(tf_model_predictions, axis=-1)
      predicted_labels = dataset_labels[predicted_ids]
```

Kod 5.15 Dobivanje stvarno predviđenih oznaka

Prvom se linijom izračunaju indeksi najviših vrijednosti u predviđenim vjerojatnostima. Ovo su indeksi koji odgovaraju klasama koje model najviše vjerojatno prepoznaje za svaku sliku. Iduća linija koristi izračunate indekse da pristupi nazivima klasa iz *dataset_labels* i tako dobije stvarno predviđene nazive klasa za svaku sliku. Kod 5.16 omogućava da se vizualno procijeni kako model radi na stvarnim slikama iz validacijskog skupa, prikazujući slike zajedno s predviđenim oznakama i bojama koje označavaju točnost predviđanja. Rezultat je slika 5.3. Zelenom bojom su napisane oznake koje je model ispravno klasificirao, a crvenom one koje je pogrešno klasificirao.

```
[26]
      # Print images batch and labels predictions

      plt.figure(figsize=(10,9))
      plt.subplots_adjust(hspace=0.5)
      for n in range(30):
        plt.subplot(6,5,n+1)
        plt.imshow(val_image_batch[n])
        color = "green" if predicted_ids[n] == true_label_ids[n] else "red"
        plt.title(predicted_labels[n].title(), color=color)
        plt.axis('off')
      _ = plt.suptitle("Model predictions (green: correct, red: incorrect)")
```

Kod 5.16 Vizualizacija procesa testiranja



Slika 5.3 Vizualni rezultati procesa testiranja

Na slici 5.3 je vidljivo da je model od 30 slika njih 7 pogrešno klasificirao, a ostale 23 ispravno. To daje postotak točnosti od 76.67%.

5.6 Konvertiranje modela u TFLite model

U ovoj se fazi trenirani model pretvara u TensorFlow Lite format, što je kompaktan format prikladan za ugradnju u mobilne uređaje i druge resursno ograničene okoline. Također, generira se kvantizirana verzija modela kako bi se smanjila njegova veličina na uštrb preciznosti. Spremljeni modeli bit će dostupni na putanjama *TFLITE_MODEL* i *TFLITE_QUANT_MODEL* (kod 5.17), a samu pretvorbu obavlja kod 5.18.

```
[27]
TFLITE_MODEL = "/content/drive/My Drive/ZavrsniRad/tflite_models/smoke.tflite"

TFLITE_QUANT_MODEL = "/content/drive/My Drive/ZavrsniRad/tflite_models/smoke_quant.tflite"
```

Kod 5.17 Putanje spremljenog modela u TFLite format

```
[28]
import tensorflow as tf
import keras


# Get the concrete function from the Keras model.
run_model = tf.function(lambda x : smoke_model(x))
# Save the concrete function.
concrete_func = run_model.get_concrete_function(
tf.TensorSpec(model.inputs[0].shape, model.inputs[0].dtype)
)
# Convert the model
converter = tf.lite.TFLiteConverter.from_concrete_functions([concrete_func])
converter.target_spec.supported_ops = [
  tf.lite.OpsSet.TFLITE_BUILTINS, # enable TensorFlow Lite ops.
  tf.lite.OpsSet.SELECT_TF_OPS # enable TensorFlow ops.
]
converted_tflite_model = converter.convert()
open(TFLITE_MODEL, "wb").write(converted_tflite_model)
# Convert the model to quantized version with post-training quantization
converter = tf.lite.TFLiteConverter.from_concrete_functions([concrete_func])
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
converter.target_spec.supported_ops = [
  tf.lite.OpsSet.TFLITE_BUILTINS, # enable TensorFlow Lite ops.
  tf.lite.OpsSet.SELECT_TF_OPS # enable TensorFlow ops.
]
tflite_quant_model = converter.convert()
open(TFLITE_QUANT_MODEL, "wb").write(tflite_quant_model)
print("TFLite models and their sizes:")
!ls "/content/drive/My Drive/ZavrsniRad/tflite_models" -lh
```

Kod 5.18 Konverzija modela u TFLite format

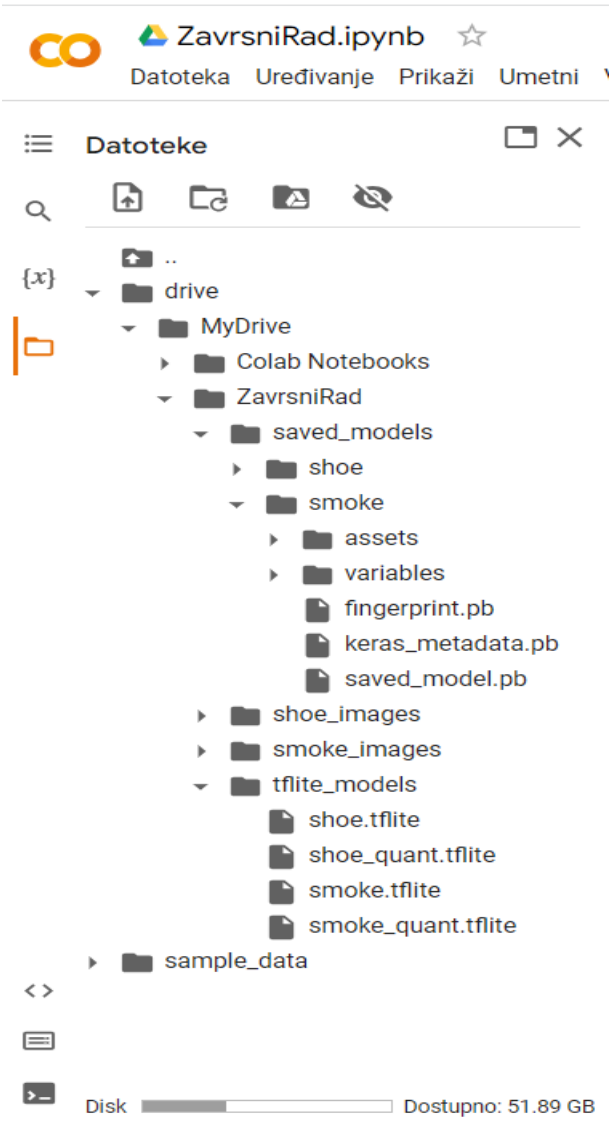
Kreira se TensorFlow funkcija *run_model* koja koristi *smoke_model* za izvođenje predviđanja. Ova funkcija će se koristiti za stvaranje konkretnih funkcija, optimizirane verzije TensorFlow grafa koje su spremna za izvođenje na određenim ulaznim podacima. Zatim se kreira konkretna funkcija iz *run_model* uz specificiranje oblika i tipa tenzora ulaza. Iduće se stvara TensorFlow Lite pretvarač iz konkretnih funkcija. Postavke omogućavaju određene skupove operacija koje će biti podržane u konvertiranom modelu. Nakon toga se model konvertira u TensorFlow Lite format i sprema na određenu putanju. Isto tako se kreira drugi pretvarač za kvantizirani model. Postavke za optimizaciju su postavljene na optimizaciju za veličinu. Nakon toga, kvantizirani model se također konvertira u TensorFlow Lite format i sprema na

drugu putanju. Na kraju se ispisuju veličine stvorenih TensorFlow Lite modela, a to je vidljivo na slici 5.4.

```
WARNING:absl:Please consider providing the trackable_obj argument in the from_concrete_functions. Providing without the trackable_obj argument is deprecated and it will use the deprecated conversion path.
WARNING:absl:Please consider providing the trackable_obj argument in the from_concrete_functions. Providing without the trackable_obj argument is deprecated and it will use the deprecated conversion path.
WARNING:absl:Optimization option OPTIMIZE_FOR_SIZE is deprecated, please use optimizations=[Optimize.DEFAULT] instead.
WARNING:absl:Optimization option OPTIMIZE_FOR_SIZE is deprecated, please use optimizations=[Optimize.DEFAULT] instead.
WARNING:absl:Optimization option OPTIMIZE_FOR_SIZE is deprecated, please use optimizations=[Optimize.DEFAULT] instead.
TFLite models and their sizes:
total 22M
-rw------- 1 root root 2.5M May 30 15:06 shoe_quant.tflite
-rw------- 1 root root 8.6M May 30 15:06 shoe.tflite
-rw------- 1 root root 2.5M Aug 30 16:31 smoke_quant.tflite
-rw------- 1 root root 8.6M Aug 30 16:31 smoke.tflite
```

Slika 5.4 Stvoreni TFLite modeli

Konačna struktura praktičnog dijela završnog rada je prikazana na slici 5.5.



Slika 5.5 Struktura praktičnog dijela završnog rada

6 ZAKLJUČAK

U današnjem svijetu gdje se teži da sve stvari u čovjekovoj okolini budu „pametne“ (od pametnih domova pa sve do pametnih automobila) umjetna je inteligencija ključna u ostvarivanju toga. Umjetna inteligencija je dinamično područje koje je donijelo revoluciju u načinu na koji ljudi razmišljaju i primjenjuju računalnu tehnologiju. Kroz razvoj sofisticiranih algoritama, dubokog učenja i neuronskih mreža, AI je postala ključna komponenta mnogih industrija. U ovom su radu spomenuti neki ključni povijesni događaji koji su doveli do današnje faze razvoja umjetne inteligencije i zašto je ona takva kakva je. To područje je preopširno i zahtjeva poseban rad kako bi se kvalitetno obradilo pa su u ovom radu obrađeni samo neki pojmovi polja umjetne inteligencije koji su važni za izradu sustava za prepoznavanje dima. Jedan od tih pojmova su neuronske mreže. Neuronske mreže simuliraju način na koji ljudski mozak obrađuje informacije i postigle su izvanredne uspjehe u zadacima kao što su prepoznavanje slika, obrada prirodnog jezika, autonomna vožnja i mnogi drugi. Kroz ovaj rad, istražene su osnove neuronskih mreža, uključujući arhitekture poput konvolucijskih neuronskih mreža. Neuronske mreže dubokog učenja predstavljaju temu od ključne važnosti u svijetu tehnologije i znanosti. Njihova sposobnost da obrade kompleksne obrasce i donesu informirane odluke otvara mnoge mogućnosti za rješavanje izazova i unaprjeđenje svakodnevice. Također je obrađen i programski jezik Python za kojeg su u radu izneseni osnovni koncepti, uključujući varijable, petlje i funkcije što čini temelj za izgradnju složenih programa. Python nudi izvanrednu ravnotežu između jednostavnosti i moći, čineći ga iznimno korisnim alatom za programiranje i rješavanje različitih problema. To je programski jezik koji kontinuirano evoluira i ostaje relevantan u brzom svijetu tehnologije. U praktičnom dijelu ovog rada je pokazano kako je korištenjem odgovarajućih alata i tehnika, kao što su TensorFlow i Keras, moguće znatno olakšati razvoj i treniranje modela. Uz to, vizualizacija rezultata treninga pomaže razumijevanju performansi modela i omogućava bolje donošenje odluka tijekom procesa razvoja i podešavanja modela. Za kraj treba razmotriti i etičke i društvene izazove koji prate brz razvoj umjetne inteligencije, uključujući pitanja privatnosti, sigurnosti i transparentnosti. AI nije samo tehnološki alat, već i duboki izazov za društvo. Stoga je ključno da svi zajedno rade na razvoju i primjeni umjetne inteligencije s pažnjom prema etičkim normama i zakonima kako bi osigurali korist za sve, bez diskriminacije ili štetnih posljedica.

LITERATURA

[1] „Hrvatska enciklopedija“, s Interneta, https://www.enciklopedija.hr/natuknica.aspx?id=49896, pristupljeno: 23.08.2023.

[2] „Povijest i perspektiva razvoja umjetne inteligencije u istraživanju uma“, s Interneta, https://www.pilar.hr/wp-content/images/stories/dokumenti/zbornici/mozak_i_um/mozak_i_um_105.pdf, pristupljeno: 25.08.2023.

[3] „Artificial Intelligence Definitions – Stanford HAI“, s Interneta, https://hai.stanford.edu/sites/default/files/2020-09/AI-Definitions-HAI.pdf, pristupljeno: 25.08.2023.

[4] „Uvod u umjetnu inteligenciju“, s Interneta, https://ai.fesb.hr/knjiga/AI-knjiga-FINAL.pdf, pristupljeno: 25.08.2023.

[5] „UMJETNA INTELIGENCIJA: DVOJBE SUVREMENOG RAZVOJA“, s Interneta, https://hrcak.srce.hr/file/320733, pristupljeno: 25.08.2023

[6] „Strojno učenje“, s Interneta, https://hrcak.srce.hr/file/382926, pristupljeno: 25.08.2023.

[7] „Neuronske mreže za početnike“, s Interneta, https://hrcak.srce.hr/file/425148, pristupljeno: 25.08.2023.

[8] „konvolucija | Struna | Hrvatsko strukovno nazivlje“, s Interneta, http://struna.ihjj.hr/naziv/konvolucija/19228/, pristupljeno: 27.08.2023.

[9] „A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way“, s Interneta, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, pristupljeno: 27.08.2023.

[10] „Convolutional Neural Network: An Overview“, s Interneta, https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/, pristupljeno: 27.08.2023.

[11] „What is Python? Executive Summary “, s Interneta, https://www.python.org/doc/essays/blurb/, pristupljeno: 27.08.2023.

[12] „UVOD U PROGRAMSKI JEZIK PYTHON“, s Interneta, https://bib.irb.hr/datoteka/914991.Uvod_u_programski_jezik_PYTHON.pdf, pristupljeno: 27.08.2023.

[13] „TensorFlow“, s Interneta, https://www.tensorflow.org/, pristupljeno: 28.08.2023.

[14] „What is TensorFlow? The machine learning library explained“, s Interneta, https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html, pristupljeno: 28.08.2023.

[15] „Introduction to Tensorflow“, s Interneta, https://www.educba.com/introduction-to-tensorflow/, pristupljeno: 28.08.2023.

[16] „MNIST skup podataka u Pythonu - osnovni uvoz i crtanje“, s Interneta, https://hr.linux-console.net/?p=5162#gsc.tab=0, pristupljeno: 28.08.2023.

[17] „TensorFlow Core | Machine Learning for Beginners and Experts“, s Interneta, https://www.tensorflow.org/overview, pristupljeno: 28.08.2023.

[18] „What is Google Colab and How Does it Work?“, s Interneta, https://saturncloud.io/blog/what-is-google-colab-and-how-does-it-work/, pristupljeno: 01.09.2023.

[19] „FESB MLID dataset“, s Interneta, http://wildfire.fesb.hr/index.php?option=com_content&view=article&id=66&Itemid=76, pristupljeno: 01.09.2023.

PRILOZI

Kazalo slika, tablica i kodova

Kazalo slika

Kazalo tablica

Kazalo kodova

Popis oznaka i kratica

AI Artificial Intelligence

GPS General Probem Solver

GPU Graphics Processing Unit

ANN Artificial Neural Network

CNN Convolutional Neural Networks

MNIST Modified National Institute of Standards and Technology

FESB Fakultet elektrotehnike, strojarstva i brodogradnje

MLID Mediterranean Landscape Image Dataset

TFLite TensorFlow Lite

SAŽETAK I KLJUČNE RIJEČI

Sažetak

U ovom je završnom radu dan uvid u područje umjetne inteligencije i programski jezik Python. Za programski jezik Python su izneseni osnovni koncepti koji su kasnije korišteni u radu. Objašnjeno je kako funkcioniraju neuronske mreže i njihov tip konvolucijske neuronske mreže. Svo stečeno teorijsko znanje i vještine je kulminiralo izradom sustava za prepoznavanje dima na slikama. U radu je prikazano i kako je jednostavno korištenjem TensorFlow biblioteke ubrzati proces razvoja modela i njegova treniranja. Primjenom teorijskog znanja i izradom praktičnog dijela ovog rada pokazano je koliko je velik i značajan utjecaj umjetne inteligencije za izradu modernih aplikacija i za osiguravanje sigurnosti i dobrobiti ljudske zajednice.

Ključne riječi

Požar, umjetna inteligencija, strojno učenje, duboko učenje, neuronske mreže, konvolucijske neuronske mreže, Python

SUMMARY AND KEYWORDS

Tittle

Software for automatic detection of smoke on images of natural landscape

Summary

In this final thesis, an insight into the field of artificial intelligence and the Python programming language is provided. The basic concepts of the Python programming language are outlined, which were later used in the thesis. An explanation of how neural networks work, specifically convolutional neural networks, is given. All the acquired theoretical knowledge and skills culminated in the development of a system for smoke detection in images. The thesis also demonstrates how the use of the TensorFlow library can expedite the model development and training process. Through the application of theoretical knowledge and the implementation of the practical part of this work, it is shown how significant and substantial the impact of artificial intelligence is in the development of modern applications and in ensuring the safety and well-being of the human community.

Keywords

Wildfire, Artificial Intelligence, Machine Learning, Deep Learning, Neural Networks, Convolutional Neural Networks, Python