UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET
ODSJEK ZA RAČUNARSTVO I INFORMATIKU

Prepoznavanje font-a na osnovu oblika slova sa slike korištenjem dubokih konvolucijskih neuronskih mreža

PROJEKTNI ZADATAK

- DRUGI CIKLUS STUDIJA -

Mentor:

Vanr. prof. dr Amila Akagić

Studenti:

Halilović Kemal
Mehmedović Faris
Memić Miralem
Omić Kenan

Sarajevo,
januar 2023.

Sadržaj

Popis slika

Popis tabela

Uvod u problematiku

1. Zašto je izabrani problem interesantan u kontekstu praktične primjene?

    - Font igra važnu ulogu u digitalnom dizajnu, jer često dizajneri žele prepoznati i koristiti fontove za svoje aplikacije iz nekih referentnih izvora [1]. Prepoznavanje fonta sa slike ili digitalnog dizajna poznato je kao vizuelni problem prepoznavanja fonta (eng. *visual font recognition problem, VFR*). VFR ima veliku primjenu u uređivanju dizajna, slika, automatski prevod teksta scene, prenos stila teksta itd. Stoga, VFR ima veliku komercijalnu vrijednost.

2. Zašto do sada nije riješen?

    - Prikupljanje podataka iz stvarnog svijeta za veliki skup klasa fontova pokazalo se izuzetno teškim. Većina dostupnih tekstualnih slika iz stvarnog svijeta nema informaciju izlaza, pa je zadatak označavanja fonta sklon greškama i zahtijeva stručnost, koju većina ljudi ne posjeduje [2]. Ručno prepoznavanje fontova iz slika je vrlo neprecizno zbog sličnih značajki različitih vrsta fontova iz iste familije fontova. Postoji nekoliko popularnih online platformi, kao što su Fontspring, WhatTheFont itd., koje pokušavaju predvidjeti font iz učitane slike. Međutim, ove platforme zahtijevaju visoku rezoluciju učitanih slika i trebaju imati odgovarajuću pretprocesiranje [1]. Predloženi rezultati za te platforme su u većini slučajeva nezadovoljavajući. To se dešava za slučajeve prepoznavanje slike dokumenta, jer znak u svakom dokumentu obično ima malu razlučivost i dokument može sadržavati različite degradacije koje čine zadatak prepoznavanja teškim.

3. Zašto je izazovan za rješavanje?

    - Otkrivanje fonta sa slika i dokumenata je izazovan zadatak iz više razloga. Jedan od glavnih razloga je veliki broj vrsta fontova koje se obično koriste u različitim dokumentima i dizajnu [1]. Također, dinamičke osobine klase fontova i ovisnost o znakovima razlika između fontova (završeci slova, težina, nagibi itd.) utiču na značaj ove problematike, npr. u slučaju aplikacija, gdje je potrebno prepoznati font za digitalnu restauraciju arhivskih dokumenata [2].

4. Kako mislite da metode vještačke inteligencije mogu doprinijeti rješavanju problema?

    - Nekoliko pristupa baziranih na statistici i Gaborovom filteru [3] korištenih za analize dokumenata, su bili fokusirani samo na mali broj klasa fontova, pri čemu su pokazivali visoku osjetljivost na šum, zamućenje, izobličenja perspektive i složenost pozadine. Jedna od tehnika vještačke inteligencije korištene u prepoznavanju engleskog fonta na osnovu globalne analize teksta je vektorska mašina podrške (SVM) [4], koja je u kategoriji klasičnog mašinskog učenja. Mašinsko učenje prvo izdvaja osobine iz seta podataka „ručno", pa tek onda vrši konstrukciju modela za učenje. S druge strane, duboko učenje kombinuje učenje osobina i konstrukciju modela, te može učiti direktno iz slika, teksta ili zvuka. Inspirisani velikim uspjehom koje su metode dubokog učenja postigle u nekim drugim zadacima kompjuterske vizije, realizirani su određeni pristupi u prepoznavanju fontova [1][2], uz korištenje konvolucijskih neuronskih mreža.

5. Koji su očekivani rezultati?

    - Izdvajanje značajki fonta uz velik broj klasa je iznimno zahtjevan zadatak. Modeli dubokog učenja zasnovani na CNN-u pokazuju značajno poboljšane rezultate u usporedbi sa drugim tehnikama vještačke inteligencije (Support vector machine, Random forest, Recurrent neural network) za testirani ograničen broj klasa [5].

1. Pregled stanja u oblasti

U literaturi su dostupni brojni radovi gdje možemo pronaći različite metode za klasifikaciju i tehnike za ekstrakciju značajki, kao i nekoliko standardnih skupova podataka, usvojenih za postizanje bolje tačnosti prilikom prepoznavnja font-a na osnovu oblika slike. Metode vještačke inteligencije, koje se koriste u prepoznavanju fonta na slici su SVM, Random forest, rekurentna neuronska mreža (RNN) [5] i konvoluciona neuronska mreža (CNN) [1][2][5][6]. Skupovi podataka imaju različite forme, kao npr. slike engleskog teksta [7], slike latinskih fontova [8] ili mješanih fontova [9][10].U narednim sekcijama pristupiti ćemo detaljnoj analizi naučnih radova, skupova podataka koji su korišteni, tehnika predprocesiranja, pristupima učenja na osnovu podataka i analizi eksperimenata.

1.1 DeepFont: Identify Your Font from An Image

U načnom radu "DeepFont: Identify Your Font from An Image" [2], autora Zhangyang Wang, Jianchao Yang, Hailin Jin, Eli Shechtman, Aseem Agarwala, Jonathan Brandt i Thomas S. Huang proučava se problem vizualnog prepoznavanja fontova (VFR) i vrši se unapređenje stanja razvijanjem DeepFont sistema. Izvršeno je učenje iz skupa podataka AdobeVFR syn, na podskupu AdobeVFR (9.20 GB) [7]. Skup podataka sadrži slike koje prikazuju engleski tekst i sastoji se od 1000 sintetičkih slika za treniranje i 100 za testiranje, za svaku od 2383 klase fontova. Skupovi za treniranje i testiranje se nazivaju VFR_syn_train (2 383 000 podataka) i VFR_syn_val (238 300), respektivno.

Uveden je pristup Convolutional Neural Network (CNN), koristeći tehniku prilagođavanja domena zasnovan na Stacked Convolutional Auto-Encoder (SCAE), koji koristi veliki broj neoznačenih tekstualnih slika iz stvarnog svijeta u kombinaciji sa sintetičkim podacima prethodno obrađenih na specifičan način.

Obrađen je novi pristup kompresije modela zasnovanog na učenju, kako bi se smanjila veličina modela DeepFont bez žrtvovanja performansi. DeepFont sistem postiže tačnost veću od 80% (top-5) na promatranom skupu podataka, a također proizvodi dobru mjeru sličnosti fontova za font izbor i sugestija. Također postignuta je oko 6 puta tačnija kompresija modela bez vidljivog gubitka prepoznavanja.

Prije unosa sintetičkih podataka u trening model, popularno je vještačko povećanje podataka za trening pomoću oznaka za očuvanje transformacija u cilju smanjenje prekomjernog trošenja. Autori naučnog rada su dodali sljedeće distorzije i greške sintetičkim slikama teksta:

1. **Šum**: mali Gausov šum sa nultom srednjom vrijedosti i standardnom devijacijom vrijednosti 3 se dodaje na ulaz
2. **Zamućenje**: nasumično Gausovo zamućenje sa standardnim devijacom od $2,5$ do $3,5$ se dodaje na ulaz
3. **Rotacija**: nasumično parametrizirana affina transformacija se dodaje na ulaz
4. **Sjenčanje**: pozadina ulaza je popunjena sa gradijentom u iluminaciji

5. **Varijabilni razmak između znakova**: prilikom renderovanja svake sintetičke slike postavljen je razmak između znakova (po piksel) da bude Gausova slučajna varijabla sa srednjom vrijednosti 10 i standardnom devijacijom 40, ograničenje $[0, 50]$

6. **Varijabilni omjer**: prije rezanja svake slika na ulazu, s označenom visinom, je komprimirana u širinu slučajnim omjerom, izvučenu iz uniformne raspodjele između $\frac{5}{6}$ i $\frac{7}{6}$.

Kao model učenja korištena je CNN sa više slojeva i SCAE bazirana adaptacija domene. Pristup u naučnom radu je proširen da izvlači značajke niskog nivoa, koje predstavljaju sintetičke i stvarne podatke. CNN arhitektura je slična kao ImageNet struktura. Korišten ulaz je dimenzija 105x105, koji je izveden iz normalizovane slike. Uvedena je squeeze operacija, koja skalira širinu.

Detalji testiranja: Izvršeno je treniranje SCAE na sintetičkim i stvarnim podatcima bez supervizije, uz stopu učenja od $0,01$. Srednja kvadratna greška (MSE) se koristi kao funkcija gubitka. Nakon treniranja SCAE, konvolucijski slojevi 1 i 2 su ubačeni u CNN. C_s podmreža, zasnovana na izlazu od C_u, se zatim obučava pod nadzorom. Početo je sa stopom učenja $0,01$ i slijedjeno uobičajenom heuristikom za ručno podešenje stope učenja za 10. Tehnika koja se koristi za dropout je primijenjena na fc6 i fc7 slojeve tokom treninga. Mrežni trening je implementiran korištenjem CUDA ConvNet paketa, i radi na radnoj stanici sa 12 Intel Xeon 2.67 GHz CPU-a i 1 GTX 680 GPU.

Detalji testiranja: Za svaku testnu sliku, najprije je izvršena normalizacija na 105 piksela u visinu, ali je urađena squeeze funkcija u širinu sa tri različita omjera, od kojih su svi izvedeni iz uniformne distribucije između 1.5 i 3.5. Ispod svake skale nad kojom je izvršena funkcija squeeze, pet uzoraka 105x105 su semplirani na nasumične lokacije. Dakle, ukupno 15 uzoraka za pojedinačnu sliku je semplirano. Pošto svaki pojedinači uzorak bi mogao proizvesti softmax vektor obučen CNN-om, u prosjeku imamo svih petnaest softmax vektora koji će odrediti konačni rezultat klasifikacije test slike.

Uvođenje SCAE-bazirane adaptacije domene pomaže treniranom modelu da postigne više od 80% tačnosti naspram top 5 tačnosti prepoznavanja font-ova. Novi model bez gubitaka se dalje primjenjuje kako bi se promoviralo skladištenje efikasnog modela. Sistem DeepFont ne samo da je efikasan za font prepoznavanje, već može proizvesti i mjeru sličnosti fontova za izbor i prijedlog fonta.

1.2 Large-Scale Font Identification from Document Images

Fontovi su bitan element digitalnog dizajna, i dizajneri često žele identificirati i koristiti različite fontove za dizajn. Prepoznavanje fontova iz različitih dokumenata je često vrlo neprecizno, shodno tome fokus je postavljen na unapređenje navedene problematike. U radu „Large-Scale Font Identification from Document Images" [1], autori Subhankar Ghosh, Prasun Roy i Saumik Bhattacharya testiraju preciznost algoritma za prepoznavanje fontova iz velikog skupa podataka.

Koristi se MC-GAN skup podataka za prepoznavanje fontova, koji sadrži 10000 latinskih fontova (verzija 1 sadrži 1,56 miliona slika sa skoro 2,5 GB) [8]. Da bi se generisale riječi iz karaktera, svi karakteri se standariziraju na ulaz dimenzija 64x64 pixela. Nakon toga, generisane su riječi sa 3 do 8 karaktera sa nasumično odrabranim fontovima iz baze podataka. Da bi baza bila još realističnija i bliža stvarnim problemima, nasumično se primjenjuje skaliranje generisanih riječi sa faktorom između 0.75 i 1.75. Također se primjenjuju operacije rotacija i

razmaka između karaktera u riječi. Završni skup podataka sadrži 200.000 slika za treniranje i 40.000 slika za testiranje.

Za analizu i zadatak klasifikacije, koristi se Convolutional Neural Network (CNN) sa tri bloka konvolucije (prateći VGG arhitekturu). Svaki blok konvolucije osim posljednjeg sadrži dva sloja konvolucije, popraćen sa max-pooling slojem. Slojevi u prvom bloku sadržavali su 64 filtera, u drugom 128 i u trećem 256 filtera. Svi slojevi su koristili filter veličine 3x3 sa ReLU aktivacijskom funkcijom. Nakon posljednjeg bloka konvolucije rađene su operacije max-pooling i izravnjavanja (eng. *flatten*), i dodaju se dva dense sloja sa aktivacijskim funkcijama ReLU i softmax zasebno.

Nasumično su odabrani karakteri iz skupa od 200.000 slika za trening, te je dobijena 73.57% tačnost validacije bez uvođenja promjena i 63.45% sa operacijama rotacije i skaliranja. Nakon toga je rađena procjena generalizacije značajki od fontova iz kojih CNN uči. Sa nasumično odabranim slovima za obuku i validaciju, dobijeno je 43.23% tačnost validacije bez promjena i 38.53% sa promjenom podataka. Performansa CNN-a je testirana preko čitavog dataseta, gdje se ispostavilo da za $k = 3$, model je dostignuo 56.68% top-3 tačnost.

1.3 Offline Computer-Synthesized Font Character Recognition Using Machine Learning Approaches

Tehnika za prepoznavanje teksta sa slika inače poznata pod pojmom optičko prepoznavanje znakova (eng. *Optical Character Recognition - OCR*) može se grupisati u dva skupa na osnovu ulaznih podataka na online i offline. U radu "Offline Computer-Synthesized Font Character Recognition Using Machine Learning Approaches" [5] autora Raghunath Dey i Chandra Rakesh je obrađen upravo offline OCR sa različitim tehnikama mašinskog učenja. Autori ističu važnost prepoznavanja kompjuterski sintetiziranih fontova, probleme pri prevođenju u mašinski razumljive forme, te mali broj drugih radova na ovu temu uprkos navedenom.

Koristi se podskup podataka chars74kFnt iz Chars74K dataseta (51.1 MB) [9]. Ovaj podskup sastoji se od crno-bijelih znakova rezolucije 128×128, sa 4 varijacije. To su kombinacije kurziva, podebljano, pravilno i podebljano sa kurzivom. Svaki karakter ima 1016 verzija. Prepoznavanje se vrši u odnosu 75:25 podataka kako bi se sastavili trening i testni skup podataka. Ukupan broj uzoraka je 62 992 od čega su 10 160 uzoraka cifara i po 26 416 uzoraka za velika i mala slova.

Za predprocesiranje su korišeni binarizacija, istanjivanje i detekcija ivica. Izdvajanje značajki urađeno je pomoću tri specifične metode: AMSF, DCTF i DCEF. Za klasifikaciju i prepoznavanje implementirana su četiri pristupa mašinskog učenja. To su SVM, Random forest, rekurentna neuronska mreža (RNN) i konvoluciona neuronska mreža (CNN). SVM i Random forest su iz Scikit-learn biblioteke, dok RNN i CNN priadaju Keras Tensor biblioteci.

Spomenute AMSF, DCTF i DCEF metode daju matrice karakteristika sa dimenzijama 256, 8 i 8 respektivno. Konačno, za procjene su korištene kombinacije sve ove tri karakteristike. Dimenzija ove matrice karakteristika je 272. U testiranjima je korištena k-fold unakrsna validacija sa vrijednošću k=4. Klasificiranje je vršeno sa velikim slovima, malim slovima, brojevima i sva tri skupa zajedno u 26, 26, 10 i 62 klase respektivno.

Rezultati testiranja pokazuju preciznost prepoznavanja od 94.89%, 92.45%, 97.86% i 90.93% za velika slova, mala slova, brojeve i skup sačinjen od sve 3 navedene kategorije zajedno. Vrijedi istaknuti da je postignuti rezultat nadmašio sve dosadašnje rezultate u postojećoj literaturi.

Sve kombinacije navedenih metoda i pristupa dale su zadovoljavajuće rezultate. Eksperimentalni rezultati ukazuju da se najveća tačnost postiže pomoću dubokih konovlucijskih neuronskih mreža u skoro svim slučajevima, što predstavlja dobar indikator da se radi o ispravnom pravcu unutar našeg istraživanja.

1.4 Character-independent font identification

Postoji malobrojan broj fontova čije su razlike jasno uočljive, zbog toga identificiranje fontova je izuzetno težak zadatak. U ovom naučnom radu [6], objavljenom od strane japanskog tima naučnika, Daichi Haraguchi, Shota Harada, Brian Kenji Iwana, Shinahara, i Seiichi Uchida su pokušali primijeniti metodu koja će za bilo koja dva karaktera odrediti font kojem pripadaju. Ovo je izuzetno zahtjevan zadatak s obzirom da su razlike između fontova manje od razlika između svih alfabeta koji postoje.

Pristup naučnika je bio da treniraju model tako da koriste slike na kojima će se nalaziti isti fontovi ali različite slova. Osobitost ovog problema identifikovanja fonta je što se svodi na problem pitanja da li dva karaktera na slici pripadaju istom fontu ili različitom. Prednost ovakvog sistema je što je on fleksibilniji od tradicionalnih sistema za identifikaciju fontova jer tradicionalni sistem samo prepoznaje one fontove koji su uneseni u sistem. Problem je u tome što se novi fontovi svakodnevno kreiraju, samim tim se postavlja ograničenje na sposobnosti tradicionalnih sistema za identifikaciju fontova, s obzirom da će im ogroman broj fontova biti nepoznat.

Također, ovako kreirani sistem za identifikaciju može da identificira veliki broj manjih slova sa slike i da ih međusobno grupiše u klase. Ovo može biti korisno kod historijskih dokumenata ili recimo u forenzici, gdje je rukopis od krucijalnog značaja. Dakle, zadatak ovog modela je da za dva para slika karaktera X_a i X_b koji pripadaju fontovima a i b, odredi da li su fontovi jednaki, odnosno da li je $a = b$. Ukoliko su jednaki, klasifikator će dodijeliti pozitivnu labelu, u suprotnom negativnu.

Za realizaciju identifikacije fonta, koristit će se CNN bazirani model sa dva ulazna toka, velikim brojem povezanih slojeva i binarnim klasifikatorom. Ulazni tokovi su identični. Svaki tok se sastoji iz četiri konvolucijska sloja i dva max pooling sloja. Veličina kernela konvolucije je 3x3, veličina koraka je 1 dok je veličina kernela pooling slojeva 2x2, sa korakom 2. Značajke iz konvolucijskih slojeva su spojene i spremljene u tri full-povezana sloja. ReLU aktivacija se koristi za skrivene slojeve i softmax funkcija se koristi za izlazni sloj. Tokom treninga, koristi se dropout u vrijednosti od 0.5 poslije prvog pooling sloja i između full-povezanih slojeva.

Dataset korišten za eksperiment čini 6 628 fontova skinutih sa *Ultimate Font Download*. Za kreiranje dataseta, izvršena je rasterizacija 26 velikih slova alfabeta u binarne slike dimenzija 100x100. Zbog eksperimentisanja i jednostavnosti, korištena su samo velika slova. Pomenuti fontovi su dalje podijeljeni u tri nezavisna font-a, 5000 iskorišteno za trening, 1000 za validaciju i 628 za testiranje.

Testiranje sistema je dovelo do zaključka da su razlike između karaktera generalno veće od razlika između fontova. Zbog toga je predložen model koji koristi dvotočni CNN - bazirani metod. Kao rezultat, predloženi model je uspio da identificira fontove sa 92.75% preciznosti koristeći 6-fold cross validaciju. Kvalitativna i kvantitavna analiza rezultata je dovela do zaključka da specifičnosti karaktera doprinose u preciznosti identifikacije. Npr. , slova "R" i "U" imaju veliku preciznost klasifikacije, dok recimo slova "I" i "Z" imaju manju.

1.5 Odabir skupa podataka na osnovu obrađenih radova

Nakon opsežne analize postojećih dataset-ova odlučeno je će tim koristiti generator za kreiranje slova različitih font-ova "TextRecognitionDataGenerator"[10]. Za početne obrade podataka generisano je 100 slika po jednom tipu fonta (3 klase). Dataset potreban za implementaciju ovog projekta je skup slika različitih slova abecede pisanih različitim font-ovima, (početni) skup podataka se sastoji od ukupno 300 slika font-ova slova. Navedene slike u grupisane po vrstama font-ova. Dimenzije pojedinačnih slika iznose 866x64 piksela, te svaka slika zauzima 17 kB. U poređenju sa skupovima podataka koji su obrađeni u prethodnim poglavljima, ovakav pristup omogućava generisanje proizvoljnog broja slika, shodno potrebama treniranja ili testiranja. Pored navedenog, omogućeno je kreiranje slika sa određenom distorzijom.

2. Korisnički definirane metode za obradu slike

U nastavku će biti opisane korisnički definirane metode za obradu slike:

2.1 Metoda pil_image

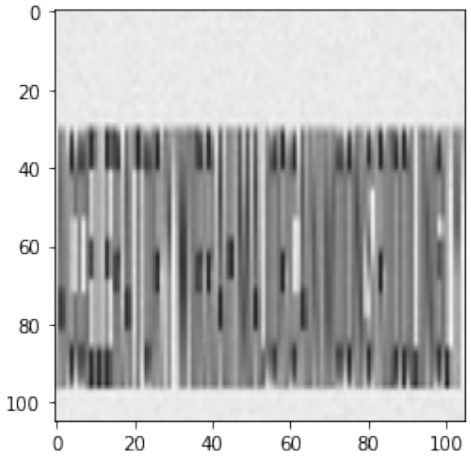
PIL (Python Imaging Library) je biblioteka u Pythonu koja omogućava manipulisanje slikama. Pruža niz funkcija za učitavanje, spremanje i manipulaciju slikama u različitim formatima datoteka, kao i crtanje teksta, linija i drugih oblika na slikama [11]. Ulazni parametar metode pil_image je putanja slike, najprije učitavamo sliku, a zatim pristupamo izmjeni veličine slike. Slika se učitava i identifikuje pomoću metode *PIL.Image.open()*. Pored navedene metode korištena je metoda *resize()* u svrhu promjne veličine slike. Ovoj metodi se prosljeđuje putanja slike. Na slici ispod je prikazana originalna slika, koja je učitana pozivom ***pil_image***.

Slika 2.1: Orginalna slika

A zatim je izvršena transformacija nad slikom pomoću sljedećeg koda.

Program 2.1: Prikaz *pil_image* metode

```
1 def pil_image(img_path):
2     pil_im =PIL.Image.open(img_path).convert('L')
3     pil_im=pil_im.resize((105,105))
4     imshow(np.asarray(pil_im))
5     return pil_im
```



Slika 2.2: Slika nakon poziva metode pil_image

2.2 Metoda noise_image

Šum (eng. *noise*) slike je neka nasumična varijacija svjetline ili nekih informacija o boji na slikama. To je degradacija signala slike uzrokovana vanjskim izvorima [12]. Na primjer, na crno bijelim slikama, šum predstavlja prisustvo crnih ili bijelih piksela tamo gdje ne bi trebali biti. Ljudsko oko bez većih poteškoća prepoznaje sliku sa šumom, dok modeli učenja konvolucionih neuronskih mreža će potencijalno pogrešno klasificirati, tako da i male distorzije u podacima mogu dovesti do pogrešnog zaključivanja. Postoji nekoliko metoda za dodavanje šuma na sliku:

- Gausov šum je vrsta šuma koja prati Gausovu distribuciju, a to je kriva u obliku zvona. Može se dodati slici generiranjem nasumičnih vrijednosti piksela koje slijede Gaussovu distribuciju originalnim vrijednostima piksela.
- Salt and pepper šum je vrsta šuma koja se sastoji od nasumičnih crno-bijelih piksela rasutih po cijeloj slici. Može se dodati slici zamjenom slučajnog odabira piksela crnim ili bijelim pikselima.
- Poissonov šum je vrsta šuma koja se javlja kada su vrijednosti piksela slike podložne nasumičnim fluktuacijama zbog diskretne prirode svjetlosti. Može se dodati slici generiranjem nasumičnih vrijednosti piksela na osnovu Poissonove distribucije originalnim vrijednostima piksela.

S obzirom da se u stvarnosti pojavljuju slike sa šumom i ostalim nepravilnostima, potrebno je napraviti model koji će izvršiti ispravnu klasifikaciju slike koja sadrži nepravilnosti. Zbog toga će metoda ***noise_image*** dodati šum na proslijeđenu sliku i na taj način će biti omogućeno učenje iz podataka koji sadrže nepravilnosti, a vjerovatnoća ispravnog predviđanja za slike sa nepravilnostima će biti veća. Kod metode *noise_image* je prikazan u nastavku.

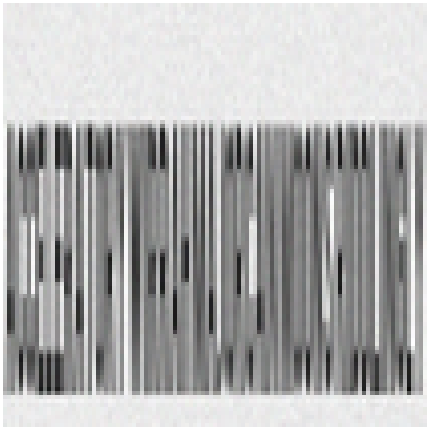
Program 2.2: Prikaz *noise_image* metode

```
1 def noise_image(pil_im):
2     img_array = np.asarray(pil_im)
3     mean = 0.0
4     std = 5
5     noisy_img = img_array + np.random.normal(mean, std, img_array.
          shape)
6     noisy_img_clipped = np.clip(noisy_img, 0, 255)
7     noise_img = PIL.Image.fromarray(np.uint8(noisy_img_clipped))
8     noise_img=noise_img.resize((105,105))
9     return noise_img
```

Navedena metoda će vratiti sliku nad kojom se izvršena distorzija sa Gaussovim šumom sa mean-om 0, standardnom devijacijom 5 i salt-and-pepper impulsnim šumom. Slika, kao proslijeđeni parametar najprije se konvertuje u niz pomoću ***np.asarray(pil_im)*** metode, a zatim se dodaje Gaussova raspodjela, koja ima mean 0, standardnu devijaciju 5 i dimenziju slike.

Salt-and-pepper šum dodajemo pomoću metode *clip()*, odnosno dodajemo nasumične svijetle boje (sa vrijednošću od 255 piksela) i nasumične tamne boje (sa vrijednošću 0 piksela) po cijeloj slici. Salt and pepper šum može biti uzrokovan iz nekoliko razloga kao što su mrtvi pikseli, greška analogno-digitalne konverzije, greška prijenosa bitova itd.

Nakon toga, pristupamo konvertovanju slike iz niza u sliku nekg formata, na primjer .JPG, .PNG i slično. To je urađeno pomoću *PIL.Image.fromarray* metode. Na slici 2.3 prikazana je slika koju je vratila korisnički kreirana *pil_image* metoda.



Slika 2.3: Slika nakon poziva metode noise_image

2.3 Metoda blur_image

Postoji nekoliko metoda za zamućenje slike u računarskoj grafici i obradi slike. Zamagljivanje je tehnika koja se koristi za izglađivanje ili prikrivanje detalja na slici, a često se koristi za uklanjanje šuma. Neke od uobičajenih metoda za zamućenje slike:

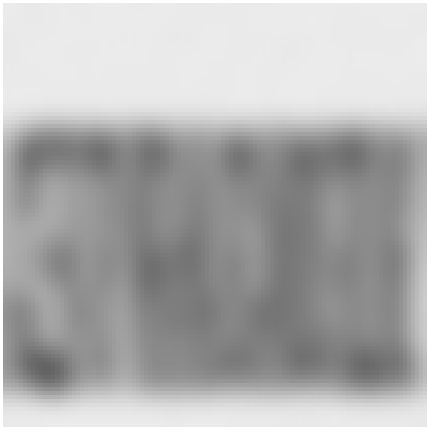
- Prosječna zamagljenost: metoda zamagljenosti koja funkcioniše tako što se svaki piksel na slici zamenjuje prosečnom vrednošću piksela u okolnom prozoru. Ovo se može implementirati korištenjem konvolucionog kernela koji ima sve težine postavljene na 1.
- Gausova zamagljenost: metoda zamagljenosti koja radi primjenom Gaussove funkcije na vrijednosti piksela na slici. Navedena metoda kreira zamućenost koja izgleda prirodnije od prosječnog zamućenja
- Median zamagljenost: metoda zamagljenosti koja radi tako što se svaki piksel na slici zamjenjuje srednjom vrijednošću piksela u okolnom prozoru. Ovo može biti korisno za uklanjanje salt and pepper šuma sa slike.

Da bi model što bolje mogao učiti iz podataka koji sadrže nepravilnosti, definisana je i metoda ***blur_image*** zasnovana na Gausovoj zamagljenosti.. Ova metoda vrši zamagljenje proslijeđene slike.

Program 2.3: Prikaz blur_image metode

```
1 def blur_image(pil_im):
2     blur_img = pil_im.filter(ImageFilter.GaussianBlur(radius=3))
3     blur_img = blur_img.resize((105, 105))
4     return blur_img
```

U ovoj metodi je korišten *ImageFilter* modul iz *PIL* biblioteke, koji sadrži definicije za predefinisani set filtera koji se koriste sa *Image.filter()* metodom. U liniji 2 pozvana metoda kreira zamagljenu sliku, gdje parametar *radius* specificira intenzitet zamagljenosti slike. Prikaz uticaja metode je prikazan na slici 2.4.



Slika 2.4: Slika nakon poziva metode blur_image

2.4 Metoda affine_rotation

Afina transformacija pomaže da se modifikuje geometrijska struktura slike, čuvajući paralelizam linija, ali ne i dužine i uglove. Sa navedenom transformacijom sačuvana je kolinearnost i omjer udaljenosti. Afina transformacija se bazira na matrici za upravljanje rotacijom, smicanjem, translacijom i skaliranjem [13]. Matrica afine rotacije je matrica 3x3 koja opisuje transformaciju rotacije. Ima sljedeći oblik [14]:

$$\begin{bmatrix} cos(\theta) & -sin(\theta) & tx \\ sin(\theta) & cos(\theta) & ty \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

gdje je θ ugao rotacije u radijanima, tx i ty su translacijski faktori u smjeru x i y, respektivno. Za transformaciju tačke (x, y) pomoću matrice afine rotacije, koriste se sljedeće formule:

$$[x'] = \begin{bmatrix} cos(\theta) & -sin(\theta) & tx \end{bmatrix} \cdot [x] \tag{2.2}$$

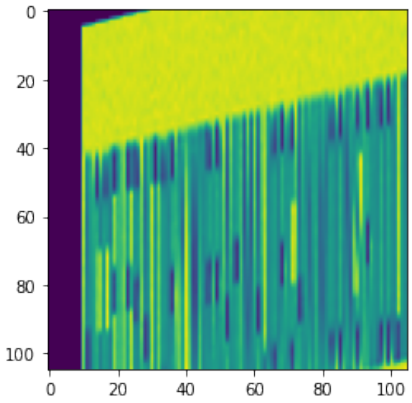
$$[y'] = \begin{bmatrix} sint(\theta) & cos(\theta) & ty \end{bmatrix} \cdot [y] \tag{2.3}$$

Rezultirajuća tačka (x', y') je rotirana verzija originalne tačke (x, y). U nastavku je prikazana metoda koja implementira afinu transformaciju.

Program 2.4: Prikaz affine_rotation metode

```
1  def affine_rotation(img):
2      rows, columns = img.shape
3      point1 = np.float32([[10, 10], [30, 10], [10, 30]])
4      point2 = np.float32([[20, 15], [40, 10], [20, 40]])
5      A = cv2.getAffineTransform(point1, point2)
6
7      output = cv2.warpAffine(img, A, (columns, rows))
8      affine_img = PIL.Image.fromarray(np.uint8(output))
9      affine_img=affine_img.resize((105,105))
10     return affine_img
```

Ovoj metodi se, umjesto pil slike, prosljeđuje pil slika konvertovana u niz. OpenCV (CV2) biblioteka pruža funkciju *cv2.getAffineTransform()* koja uzima kao ulaz tri para odgovarajućih tačaka i daje matricu transformacije.



Slika 2.5: Slika nakon poziva metode affine_rotation

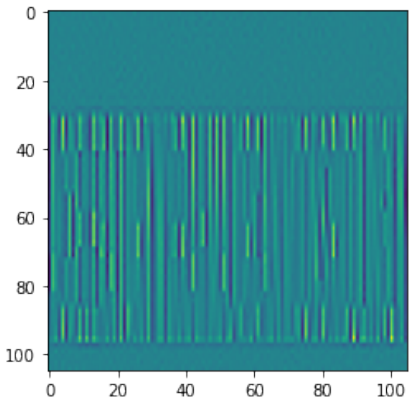
2.5 Metoda gradient_fill

Još jedna metoda koja se koristi u obradi slike je detekcija ivica, a u ovom slučaju je detekcija ivica izvršena pomoću korisnički definisane metode ***gradient_fill*** čija je implementacija prikazana ispod.

Program 2.5: Prikaz gradient_fill metode

```
1 def gradient_fill(image):
2     laplacian = cv2.Laplacian(image,cv2.CV_64F)
3     laplacian = cv2.resize(laplacian, (105, 105))
4     return laplacian
```

Detekcija ivica je tehnika obrade slike koja se koristi za identifikaciju tačaka na digitalnoj slici sa oštrim promjenama u osvjetljenosti slike. Tačke u kojima svjetlina slike oštro varira nazivaju se ivicama slike. Detekcija ivica smanjuje količinu podataka na slici i čuva strukturna svojstva slike. Ivice je moguće detektovati preko drugog izvoda. S obzirom da je slika 2D, potrebno je naći drugi izvod u obje dimenzije, a za to se koristi Laplacian operator. Ovaj operator je implementiran u biblioteci CV2, preko *Laplacian()* metode.



Slika 2.6: Slika nakon poziva metode gradient_fill

2.6 Poziv metoda

U kodu ispod, program prolazi petljom kroz listu putanja, te pomoću metode ***pil_image*** učitava, konvertuje i mijenja veličinu slike. Navedenu sliku konvertujemo u niz i ažuriramo varijablu *data*. Varijabla ***augument*** predstavlja nazive mogućih transformacija slike.

Pomoću ***itertools.combinations(augument, l+1)*** pravimo sve moguće kombinacije iz liste transformacija. Primjeri kombinacija ['noise', 'blur'], ['blur'], ['noise', 'blur', 'affine', 'gradient']. Na ovaj način primjenjujemo različite kombinacije transformacija (obrade slike) nad svakom slikom. Transformisana slika se smješta u ***temp_img*** varijablu, nakon čega se konvertuje u niz i ažurira se *data* varijabla.

Program 2.6: Poziv metoda

```
for imagePath in imagePaths:
    label = imagePath.split(os.path.sep)[-2]
    label = conv_label(label)
    pil_img = pil_image(imagePath)

    # Adding original image
    org_img = img_to_array(pil_img)
    data.append(org_img)
    labels.append(label)

    augument=["noise","blur","affine","gradient"]
    for l in range(0,len(augument)):

        a=itertools.combinations(augument, l+1)

        for i in list(a):
            combinations=list(i)
            temp_img = pil_img
            for j in combinations:

                if j == 'noise':
                    # Adding Noise image
                    temp_img = noise_image(temp_img)

                elif j == 'blur':
                    # Adding Blur image
                    temp_img = blur_image(temp_img)

                elif j == 'affine':
                    open_cv_affine = np.array(pil_img)
                    # Adding affine rotation image
                    temp_img = affine_rotation(open_cv_affine)

                elif j == 'gradient':
                    open_cv_gradient = np.array(pil_img)
                    # Adding gradient image
                    temp_img = gradient_fill(open_cv_gradient)

            temp_img = img_to_array(temp_img)
            data.append(temp_img)
            labels.append(label)
```

3. Osnovna metodologija

3.1 Duboke neuronske mreže

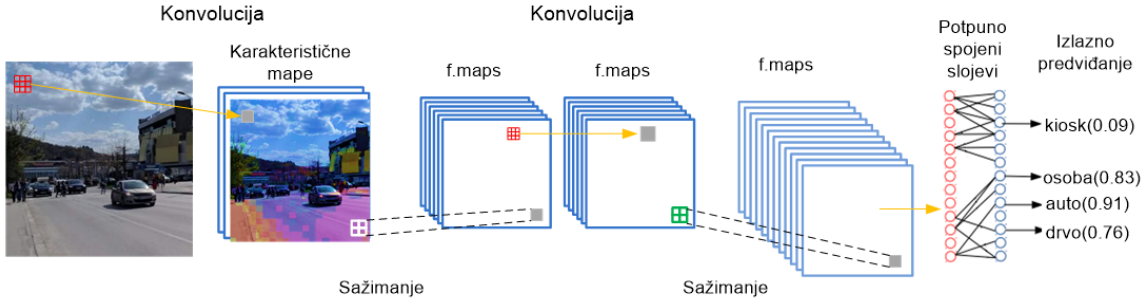
Svaka neuronska mreža je u osnovi kolekcija neurona i njihovih međusobnih konekcija odnosno sinapsi. Neuron (čvor) je funkcija sa skupom ulaza i jednim izlazom, sa zadatkom prikupljanja sadržaja iz ulaza, primjene funkcije na njih te slanja rezultata izlazu. Sinapse među neuronima povezuju izlaz neurona sa ulazom drugog. Svaka sinapsa ima jedan parametar – težinu, koji se može predstaviti kao jačina signala konekcije. Ove težine ukazuju neuronu kojem ulazu više da odgovara, gdje se pri treniranju ove težine prilagođavaju, što i jeste način na koji neuronske mreže uče [15].

Arhitekture dubokog učenja, kao što su duboka neuronska mreža ili rekurentna neuronska mreža primjenjene su na poljima računarske vizije, prepoznavanja i sinteze govora, mašinskog prevođenja, obrade prirodnih jezika, prepoznavanja zvuka i postižu rezultate jednake, ako ne i bolje od rezultata stručnjaka [16]. Tako se konvolucijske mreže najčešće koriste pri prepoznavanju slika dok su rekurentne i LSTM mreže više zastupljene u jezičkim aplikacijama [17].

3.1.1 Konvolucione neuronske mreže

Duboke konvolucione neuronske mreže (eng. *Deep learning convolution neural networks*) predstavljaju jednu od najvećih inovacija u računarskoj viziji. Mogu postići ljudsku performansu u nekim problemima klasifikacije slika i problemima detekcije objekata u slikama. U poređenju sa uobičajenim sistemom za klasifikaciju koji ima dvije faze, tj. fazu „manuelne“ ekstrakcije značajki slike (ivice, krive, regioni), te fazu učenja (treniranja) nekog klasifikatora (npr. SVM), DL CNN integrišu i uče end-to-end mapiranje između slike i njene klase, tj. uče hijerarhiju značajki i klasifikaciju istih. Za uspješnost DL CNN potrebne su velike količine podataka i velika računarska snaga kako za treniranje tako i za izvršavanje zadaće klasifikacije [18].

Konvolucijske neuronske mreže (eng. *Convolutional Neural Network, CNN*) su specijalizirana vrsta mreža za procesiranje podataka sa topologijom rešetke. Neki od takvih podataka su vremenska serija (1D rešetka sa uzorcima uzetim u regularnim vrementskim intervalima) te slika (2D rešetka piksela). Ime ove mreže polazi od matematičke operacije konvolucije koju mreža koristi. Primjer jedne konvolucijske mreže dat je na slici 3.1.



Slika 3.1: Primjer konvolucijske neuronske mreže

3.2 Treniranje konvolucione neuronske mreže

3.2.1 Određivanje stope učenja

U ovoj sekciji ćemo analizirati uticaj hiperparametara na proces treniranja. Hiperparametri se mogu klasificirati kao hiperparametri modela, koji upravljaju procesom treniranja vještačke neuronske mreže, te utiču na brzinu i kvalitet procesa učenja. Primjeri hiperparametara kod vještačkih neuronskih mreža su learning rate, batch size, broj slojeva, broj neurona u sloju itd.

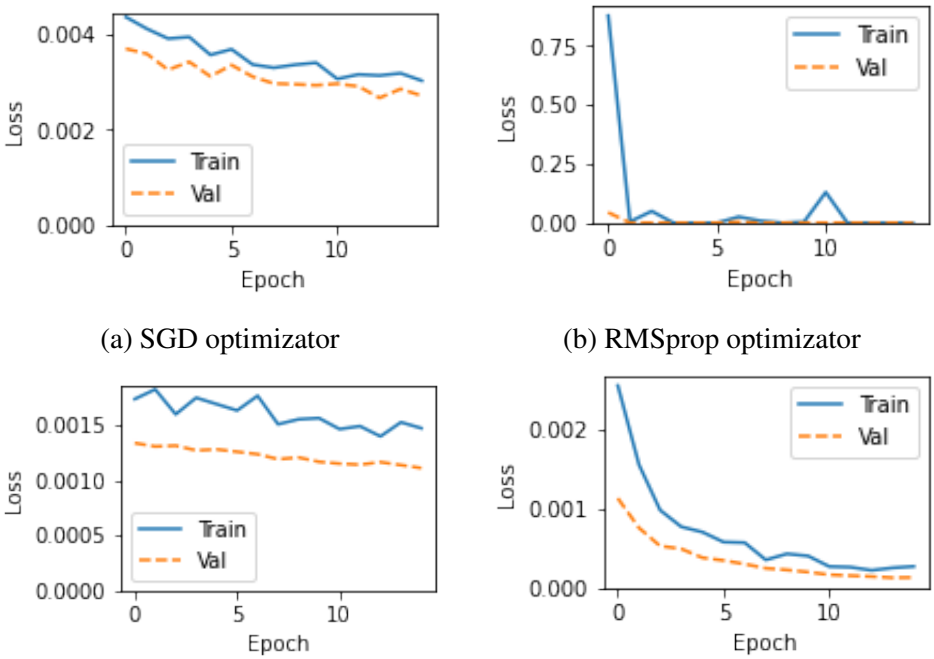
Pristup koji je korišten za optimizaciju hiperparametara je preko biblioteke KerasTuner. KerasTuner dolazi sa ugrađenim algoritmima Bayesian optimizacije, hyperband i slučajnog pretraživanja, a također je dizajniran tako da ga istraživači lako prošire kako bi eksperimentirali s novim algoritmima pretraživanja [19]. U nastavku je prikazano korištenje Hyperband algoritma, na različitim optimizatorima.

Koristeći KerasTuner podesili smo optimalne hiperparametre treniranja. Preporučeni broj epoha je išao u rasponu $2-14$. Za arhitekturu koju sami dizajniramo i treniramo potreban je znatno veći broj epoha da bi došla do značajne preciznosti. Preporučeni koeficijent brzine učenja je u domenu e-2 do e-3, a dropout u domenu $0.1-0.5$ za različite forme metoda optimizacije koje su ispitivane: Adagrad, Adam, RMSprop i SGD.

Tabela 3.1: Pregled hiperparametara za najbolje modele za pojedinačne optmizatore.

Optimizator	**Adagrad**	**Adam**	**RMSprop**	**SGD**
Broj podataka:	300	300	300	300
Broj filtera 1:	16	16	64	16
Kernel 1:	4	4	2	4
Stride 1:	1	2	2	1
Broj filtera 2:	32	166	64	32
Kernel 2:	4	2	3	4
Stride 2:	1	1	1	1
Dropout 1:	0.1	0.3	0.2	0.1
Dense vrijednosti:	192	96	352	192
Dense aktivacija:	relu	swish	softplus	relu
Dropout 2:	0.2	0.2	0.1	0.2
Stopa učenja:	0.00563	0.00213	0.00536	0.00563
Tačnost:	1.00	1.00	1.00	1.00

Optimizatori Adagrad, Adam, RMSprop i SGD koriste stopu učenja za ažuriranje koeficijenta. U nastavku ćemo prikazati grafike za funkciju gubitka za navedene optmizatore. Analizom grafika vidimo da će Adam optimizator imati dobru stopu učenja i vidimo da i sa povećanjem broja epoha neće doći do preporučene promjene koja podrazumijeva visoku stopu učenja. Kada razmatramo SGD i Adagrad optmizator, vidimo da postoji mogućnost pojave visoke stope učenja, ukoliko bismo povećali broj epoha. Pošto SGD optmizator ima manje vrijednosti funckije gubitke, prednost ćemo dati istom. RMSprop optimizator nećemo obrađivati zbog jako loših vrijednosti funkcije gubitka prikazanih na grafiku.



(a) SGD optimizator

(b) RMSprop optimizator

(c) Adagrad optimizator

(d) Adam optimizator

Slika 3.2: Grafovi za funkciju gubitka za različite optimizatore na skupu podataka od 300 slika.

Bitno je napomenuti da prilikom odabira strategije kod kreiranja vlastite CNN arhitekture, zbog moguće pojave overfittinga, u polaznu arhitekturu možemo dodati dropout sloj, L2 regularizaciju te batch normalizaciju. Svaki od navedenih slojeva ima efekat na funkciju gubitaka i preciznost. Npr. dodavanjem velikog broja droput slojeva dolazi do usporavanja samog učenja. Također velika vrijednost dropouta znatno usporava brzinu učenja. Mi smo koristili 2 dropout sloja sa vrijednošću 0.5. (preporuka je da se smanji). U sekciji 3.2.2 ćemo obraditi model učenja na kojem su primjenjena dosadašnja opažanja.

Raspoređivač stope učenja (eng. *learning rate scheduler)*

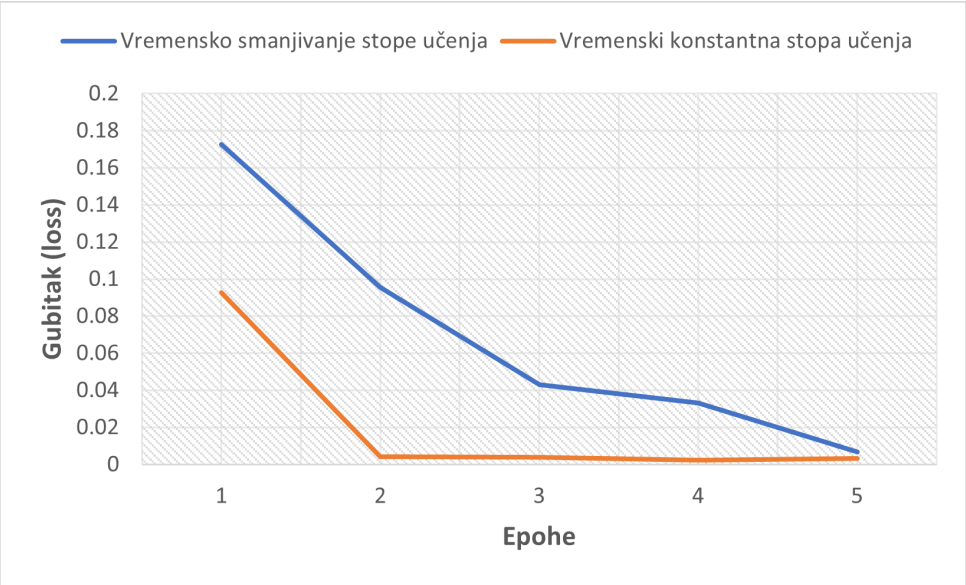
Korištena je tehnika vremenskog smanjivanja stope učenja, koja se ostvaruje kroz povećavanje broja iteracija. Navedena tehnika je implementira na sljedeći način:

Program 3.1: Prikaz tehnike vremenskog smanjivanja stope učenja.

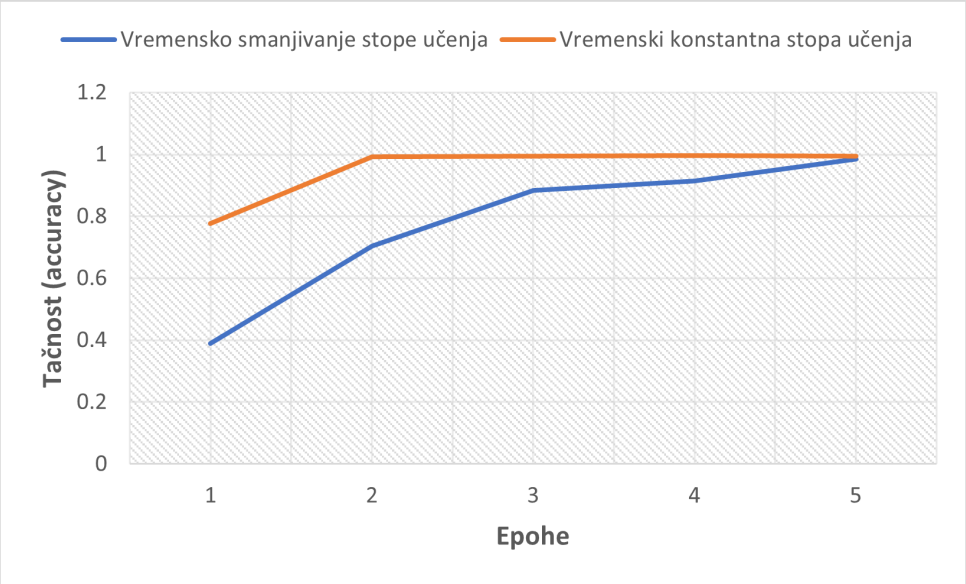
```
lr = 0.01
decay_rate = lr / epochs
mom = 0.9

sgd = optimizers.SGD(learning_rate=lr, decay=decay_rate, momentum=
    mom, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['
    accuracy'])
```

Uz navedenu tehniku, postignuti su sljedeći rezultati:



Slika 3.3: Grafik prikaza funkcije gubitka kroz epohe.



Slika 3.4: Grafik prikaza tačnosti kroz epohe.

Prilikom testiranja utvrđeno je da metoda, koja podrazumijeva vremenski konstantnu stopu učenja u toku procesa treniranja ima veću tačnost i manji gubitak u odnosnu na obrađenu metodu vremenskog smanjivanja stope učenja. Metoda poput eksponencijalog smanjanivanja stope učenja nije razmatrana, shodno tome da broj epoha nad kojim se izvršava model nije velik, najprije zbog manje kompleknosti slika, koje se obrađuju, ali i zbog hardverskiih ograničenja mašina, na kojima se trenira model. Bitno je napomenuti da izabrani broj epoha je dao očekivane rezultate prilikom prepoznavnja, te se nije išlo u daljnja testiranja.

3.2.2 Dizajniranje modela učenja

Došli smo do faze u kojoj dizajniramo CNN model. Korišten je Sequential format, koji je ujedno najčešće korišten format u Keras biblioteci [20]. Prvi sloj našeg modela je konvolucijski sloj. On će uzeti ulaze i pokrenuti konvolucione filtere na njima. Kada implementiramo konvolucijske slojeve u Kerasu, moramo odrediti broj kanala/filtera (to je 64 kanala/filtera koji su prikazani u programu ispod), veličinu kernela (48x48 u našem slučaju), veličinu ulaza i aktivaciju. Korištena je ReLu aktivacija, koja mijenja zadane parametre na način da se koriste pragovi koji nisu nula, te da se promijeni maksimalna vrijednost aktivacije i da se koristi višestruki unos (koji nije nula za vrijednost ispod praga).

Također, dodana je Batch normalizacija, koja normalizira ulaze koji idu u sljedeći sloj, osiguravajući da konvolucijska neuralna mreža uvijek kreira aktivacije sa istom distribucijom koju želimo. Važno je da nemamo previše pooling slojeva, jer svaki pooling odbacuje neke podatke smanjivanjem dimenzija ulaza sa datim faktorom.

Program 3.2: Kreiranje CNN modela

```
model=Sequential()

model.add(Conv2D(64, kernel_size=(48, 48), activation='relu',
    input_shape=(105,105,1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(24, 24), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Dodajemo Conv2DTranspose sloj s ciljem izvršenja inverzne operacije konvolucije. Sa UpSampling2D slojem radi se skaliranje slika korištenjem algoritma najbližeg susjeda ili bilinearnog upsamplinga.

Program 3.3: Povećanje uzorkovanja

```
model.add(Conv2DTranspose(128, (24,24), strides = (2,2), activation
    = 'relu', padding='same', kernel_initializer='uniform'))
model.add(UpSampling2D(size=(2, 2)))

model.add(Conv2DTranspose(64, (12,12), strides = (2,2), activation =
     'relu', padding='same', kernel_initializer='uniform'))
model.add(UpSampling2D(size=(2, 2)))
```

Nakon što završimo s konvolucijskim slojevima, moramo izravnati podatke, te zbog toga koristimo Flatten funkciju. Izravnavanje se koristi za smanjenje dimenzionalnosti ulaza na Dense sloj, koji očekuje jednodimenzionalni vektor (koji je opet matematički još uvijek višedimenzionalni objekt), gdje svaka kolona odgovara ulazu karakteristike Dense sloja.

Program 3.4: Ravnanje podataka

```
model.add(Conv2D(256, kernel_size=(12, 12), activation='relu'))
model.add(Conv2D(256, kernel_size=(12, 12), activation='relu'))
model.add(Conv2D(256, kernel_size=(12, 12), activation='relu'))
model.add(Flatten())
```

Ovdje možemo imati više Dense slojeva, a ti slojevi izvlače informacije iz mapa karakteristika kako bi naučili klasificirati slike na osnovu mapa karakteristika. Sloj Dropout nasumično postavlja ulaze na 0 sa frekvencijom stope na svakom koraku tokom vremena treninga, što pomaže u sprečavanju overfittinga. Konačno, Softmax funkcija aktivacije odabire neuron s najvećom vjerovatnoćom kao svoj izlaz, zaključujući da slika pripada toj klasi. Ovim je osnovna arhitektura konvolucione neuronske mreže završena.

Program 3.5: Obrada mapa karakteristika i zaključivanje

```
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(2383,activation='relu'))
model.add(Dense(3, activation='softmax'))
```

U narednom koraku definiramo veličinu batch-a (definira broj uzoraka za rad prije ažuriranja internih parametara modela) i broj epoha (definira koliko puta će algoritam učenja raditi kroz cijeli skup podataka za treniranje), kao i callback funkciju, koja će spriječiti pojavu overfittinga.

Program 3.6: Definiranje veličine batch-a i broja epoha

```
batch_size = 128
epochs = 25
model= create_model()
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=
    True)
model.compile(loss='mean_squared_error', optimizer=sgd, metrics=['
    accuracy'])

early_stopping=callbacks.EarlyStopping(monitor ="val_loss",
                                       mode ="min", patience = 5,
                                       restore_best_weights = True)

filepath="C:/Users/USER/Desktop/MPVI_projekat_model.h5"
checkpoint = callbacks.ModelCheckpoint(filepath, monitor='val_loss',
     verbose=1, save_best_only=True, mode='min')
callbacks_list = [early_stopping,checkpoint]

if os.path.exists('C:/Users/USER/Desktop/MPVI/MPVI_projekat_model.h5
    '):
  os.remove('C:/Users/USER/Desktop/MPVI/MPVI_projekat_model.h5')

model.save('MPVI_projekat_model.h5')
filepath="C:/Users/USER/Desktop/MPVI/MPVI_projekat_model.h5"

checkpoint = callbacks.ModelCheckpoint(filepath, monitor='val_loss',
     verbose=1, save_best_only=True, mode='min')

callbacks_list = [early_stopping,checkpoint]

################ fitting model into filepath
model.fit(trainX, trainY,shuffle=True,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(testX, testY),callbacks=callbacks_list)
```

4. Rezultati

Nakon dizajniranja modela izvršili smo kreiranje modela sa različitim hiperparametrima uz selektirani SGD optimizator, koji je na osnovu prethodne analize pokazivao najveći potencijal za obrađeni skup podataka. U Tabeli 4.1. prikazana je usporedna analiza korištenih hiperparametara i analiza tačnosti i funkcije gubitaka. Najbolji rezultati su prikazani za izbor hiperparametara u kolonama dva i pet. Istrenirani model ćemo koristiti da provjerimo prepoznavanje fonta nad određenim testnim podacima.

Tabela 4.1: Pregled hiperparametara za dizajnirani model sa SGD optmizatorom.

Optimizator:	**SGD**				
Dense vrijednosti 1:	4096	4096	4096	4096	4096
Dense aktivacija 1:	swish	elu	relu	swish	relu
Dropout 1:	0.1	0.5	0.5	0.5	0.5
Dense vrijednosti 2:	4096	4096	4096	4096	4096
Dense aktivacija 2:	swish	elu	relu	swish	relu
Dropout 2:	0.2	0.5	0.5	0.5	0.5
Dense vrijednosti 3:	4096	4096	4096	4096	4096
Dense aktivacija 3:	swish	elu	relu	swish	relu
Stopa učenja:	0.005	0.005	0.005	0.01	0.01
Tačnost:	0.329	1.000	0.995	0.659	1.000
Fukcija gubitka:	0.4476	1.72E-06	0.0031	0.2276	8.26E-06

Naredna etapa je uvoz slike na kojoj ćemo testirati naš model učenja. Najprije učitavamo sliku, radimo izmjenu njenih dimenzija i zamagljenje, zatim pretvaramo uvezenu sliku u niz podataka, čije vrijednosti će biti u rasponu 0 do 1. Procesiranu sliku šaljemo na evaluaciju našem modelu i pomoću operacije argmax pronalazimo argument koji daje maksimalnu vrijednost iz ciljne funkcije (ona klasa formata koja ima najveću vjerovatnoću da se podudara sa ulaznom slikom). Uspješno su izvršena testiranja za tri različite klase: DroidSens, Roboto-Regular i Pacifico.

4.1 Primjer prepoznavnja za tekst fonta DroidSans

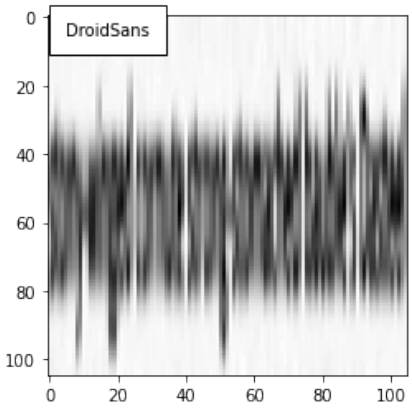
Program 4.1: Uvoz slike i pozivanje metoda za predprocesiranje nad slikom

```
1 img_path="/content/drive/MyDrive/droidsans_sample.jpg"
2 pil_im =PIL.Image.open(img_path).convert('L')
3 pil_im=blur_image(pil_im)
4 org_img = img_to_array(pil_im)
```

Program 4.2: Slanje slike na evaluciju modelu

```
data=[]
data.append(org_img)
data = np.asarray(data, dtype="float") / 255.0
y = np.argmax(model.predict(data), axis=-1)

label = rev_conv_label(int(y[0]))
fig, ax = plt.subplots(1)
ax.imshow(pil_im, interpolation='nearest', cmap=cm.gray)
ax.text(5, 5, label , bbox={'facecolor': 'white', 'pad': 10})
plt.show()
```

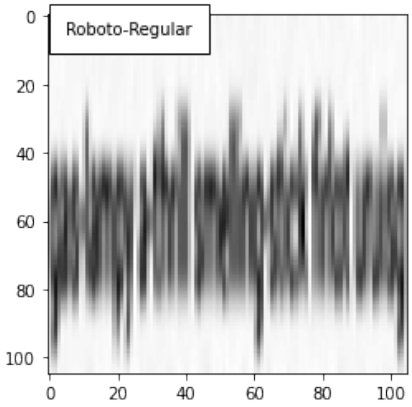


Slika 4.1: Rezultat prepoznavanja i obrade ulazne slike teksta fonta DroidSans.

4.2 Primjer prepoznavnja za tekst fonta Roboto-Regular

Program 4.3: Uvoz slike i pozivanje metoda za predprocesiranje nad slikom

```
img_path="/content/drive/MyDrive/roboto-regular_sample.jpg"
pil_im =PIL.Image.open(img_path).convert('L')
pil_im=blur_image(pil_im)
org_img = img_to_array(pil_im)
```

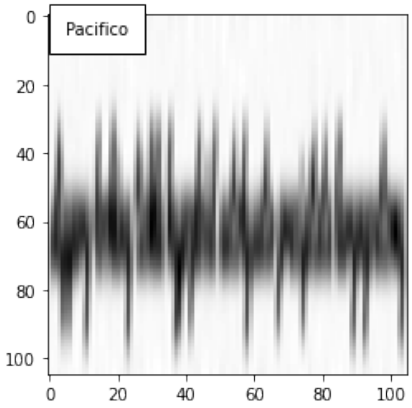


Slika 4.2: Rezultat prepoznavanja i obrade ulazne slike teksta fonta Roboto-Regular.

4.3 Primjer prepoznavnja za tekst fonta Pacifico

Program 4.4: Uvoz slike i pozivanje metoda za predprocesiranje nad slikom

```
1 img_path="/content/drive/MyDrive/pacifico_sample.jpg"
2 pil_im =PIL.Image.open(img_path).convert('L')
3 pil_im=blur_image(pil_im)
4 org_img = img_to_array(pil_im)
```



Slika 4.3: Rezultat prepoznavanja i obrade ulazne slike teksta fonta Pacifico.

5. Zaključak

U radu je prezentirana problematika vizuelnog prepoznavanja fontova sa slike. Ova zadaća je izuzetno složena, jer postoji veliki broj fontova, koje se koriste u različitim dokumentima i dizajnu, kao i značajan broj dinamičkih osobina klasa fontova i ovisnost o znakovima razlika između fontova (završeci slova, težina, nagibi). Izvršili smo usporedbu postavki i perfomansi istražene literature i našeg istraživanja, što je prikazano u Tabeli 5.1.

Tabela 5.1: Pregled postignutih rezultata u odnosu na SOTA.

	Članak [2]	**Članak [1]**	**Članak [5]**	**Članak [6]**	**Projekat**
Baza podataka	AdobeVFR (9.20 GB)	MC-GAN (2,5 GB)	chars74kFnt iz Chars74K (51.1 MB)	Ultimate Font Download (private)	TRDG
Skup za treniranje i broj klasa	engleski tekst, 2383	Latinski fontovi	Cifre, mala i velika slova, 62	različita slova, 6 628	Latinski fontovi, abeceda, 3
Broj slika	2 383 000 slika za treniranje i 238 300 za testiranje	200 000 slika za treniranje i 40 000 za testiranje	62.992, 75:25 omjer podataka za trening i testni skup	5000 za trening, 1000 za validaciju i 628 za testiranje	300 slika, 75:25 omjer podataka za trening i testni skup
Predpro-cesiranje	Šum, zamućenje, rotacija, sjenčenje, varijabilni razmak i omjer	Rotacija i skaliranje	Binarizacija, istanjivanje i detekcija ivica	/	Šum, zamućenje, rotacija, gradijent, detekcija ivica
Arhitektura DNN	CNN	VGG	CNN	dvotočni CNN	CNN
Tačnost	80%	56.68%	90.93%	92.75%	99.5-100%

U našem radu, prepoznavanja fontova je izvršeno na skupu podataka od 300 slika slova latinske abecede i 3 različite vrste fontova. S obzirom da se u stvarnosti pojavljuju slike sa različitim nepravilnostima, potrebno je napraviti model koji će izvršiti ispravnu klasifikaciju slike koja sadrži distorzije. Za predprocesiranje su korišene metode dodavanja šuma na sliku, zamagljenosti slike, modifikacije geometrijske strukture slike i detekcija ivica. Za klasifikaciju i prepoznavanje implementirana je konvoluciona neuronska mreža (CNN), gdje je izbor hiperparametara urađen korištenjem Keras Tuner-a (Tensor biblioteka). Proveli smo treniranje konvolucijske neuronske mreže korištenjem selektiranih parametara koeficijenta brzine učenja, broja epoha, izborom optimizatora učenja, uz kreiranje vlastite arhitekture CNN neuronske

mreže. Uz određene kombinacije parametara, dobivena je tačnost prepoznavanja fonta preko 99%. Interesantno je napomenuti da su najbolji rezultati postignuti korištenjem elu i relu aktivacijskih funkcija, stope učenja u opsegu $0.005 - 0.01$, SGD optimizacisjke funkcije, većeg broja Dropout slojeva, pri čemu je tačnost iznosila 0.095, a iznos funkcije gubitka do reda 10 na minus šestu. Ovi rezultati su dobijeni koristeći 3 izlazne klase, pri istraživanju prepoznavanja većeg broja klasa (fontova), trebalo bi koristiti veću bazu podataka.

Buduća istraživanja mogla bi ići u sljedećem pravcu:

- interesantno bi bilo izvršiti analizu nad mnogo većem broju slika, npr. 5000, koji bi uključivao i različite font-ove koji ne pripadaju porodici latinskih font-ova, čime bi imali značajno veći broj izlaznih klasa.
- moguće je nad tako proširenim skupom ulaza, koristiti različite vrste javnih CNN mreža (ResNet, VGGnet i sl.), koje mogu dati visoki nivo tačnosti prepoznavanja fontova.
- stečena znanja u kreiranju vlastite arhitekture konvolucijske neuronske mreže proširiti kroz eksperimentalni rad, nad većim skupom podataka.

Istraživanje u ovom pravcu bi trebalo nadograditi adekvatnom hardverksom podrškom računara, koji obezbjeđuje visok nivo akceleracije pri realizacije zadatataka mašinskog učenja.

Literatura

[1] Ghosh, S., Roy, P., Bhattacharya, S., Pal, U., "Large-scale font identification from document images", in Asian Conference on Pattern Recognition. Springer, 2019, str. 594–600.

[2] Wang, Z., Yang, J., Jin, H., Shechtman, E., Agarwala, A., Brandt, J., Huang, T. S., "Deepfont: Identify your font from an image", in Proceedings of the 23rd ACM international conference on Multimedia, 2015, str. 451–459.

[3] Avilés-Cruz, C., Rangel-Kuoppa, R., Reyes-Ayala, M., Andrade-Gonzalez, A., Escarela-Perez, R., "High-order statistical texture analysis—-font recognition applied", Pattern Recognition Letters, Vol. 26, No. 2, 2005, str. 135–145.

[4] Ramanathan, R., Soman, K., Thaneshwaran, L., Viknesh, V., Arunkumar, T., Yuvaraj, P., "A novel technique for english font recognition using support vector machines", in 2009 international conference on advances in recent technologies in communication and computing. IEEE, 2009, str. 766–769.

[5] Dey, R., Balabantaray, R. C., "Offline computer-synthesized font character recognition using machine learning approaches", in Proceedings of International Conference on Machine Intelligence and Data Science Applications. Springer, 2021, str. 635–647.

[6] Haraguchi, D., Harada, S., Iwana, B. K., Shinahara, Y., Uchida, S., "Character-independent font identification", in International Workshop on Document Analysis Systems. Springer, 2020, str. 497–511.

[7] Wang, Z., Yang, J., Jin, H., Shechtman, E., Agarwala, A., Brandt, J., Huang, T. S., "Adobevfr syn (adobe visual font recognition synthetic dataset)", 2015, dostupno na: https://paperswithcode.com/dataset/adobevfr

[8] Ge, Y., Abu-El-Haija, S., Xin, G., Itti, L., "Fonts dataset". University of Southern California, 2021, dostupno na: http://ilab.usc.edu/datasets/fonts

[9] de Campos, T., "The chars74k dataset". Microsoft Research India, dostupno na: http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

[10] Belval, E. (2019) Textrecognitiondatagenerator, dostupno na: https://github.com/Belval/TextRecognitionDataGenerator#textrecognitiondatagenerator----

[11] Patidar, P., "Python imaging library (pil) tutorial", 2021, dostupno na: https://thecleverprogrammer.com/2021/08/21/python-imaging-library-pil-tutorial/

[12] Swain, A., "Noise in digital image processing". Image Vision, 2018, dostupno na: https://www.tensorflow.org/tutorials/keras/keras_tuner

[13] Patidar, P., "Affine transformation- image processing in tensorflow-part 1". MLAIT, 2020, dostupno na: https://medium.com/mlait/affine-transformation-image-processing-in-tensorflow-part-1-df96256928a

[14] "Linear mapping method using affine transformation". The MathWorks, dostupno na: https://www.mathworks.com/discovery/affine-transformation.html

[15] Russell, S., Norvig, P., "Artificial intelligence: A modern approach, global edition 4th", Foundations, Vol. 19, 2021, str. 23.

[16] Ciregan, D., Meier, U., Schmidhuber, J., "Multi-column deep neural networks for image classification", in 2012 IEEE conference on computer vision and pattern recognition. IEEE, 2012, str. 3642–3649.

[17] Schmidhuber, J., "Deep learning in neural networks: An overview", Neural networks, Vol. 61, 2015, str. 85–117.

[18] Zhang, A., Lipton, Z. C., Li, M., Smola, A. J., "Dive into deep learning", arXiv preprint arXiv:2106.11342, 2021.

[19] "Introduction to the keras tuner". TensorFlow, dostupno na: https://www.tensorflow.org/tutorials/keras/keras_tuner

[20] Nelson, D., "Image Recognition and Classification in Python with TensorFlow and Keras", November 2021, dostupno na: https://stackabuse.com/image-recognition-in-python-with-tensorflow-and-keras/

Indeks pojmova