Vizijski sustav prepoznavanja nota s notnog zapisa

Pongrac, Patrik

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Patrik Pongrac

Zagreb, 2020

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

prof. dr. sc. Dubravko Majetić Patrik Pongrac

Zagreb, 2020.

Izjavljujem da sam ovaj rad izradio sam, koristeći dosad stečena znanja i iskustva te navedenu literaturu.

Zahvaljujem se svoj mentoru, dekanu prof. dr. sc. Dubravku Majetiću na suradnji i susretljivosti prilikom pisanja ovog rada.

Posebna zahvala prijatelju na pomoći prilikom izrade ovog rada te djevojci i roditeljima na podršci tokom studija.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **Patrik Pongrac** Mat. br.: **0035205017**

Naslov rada na hrvatskom jeziku: **Vizijski sustav prepoznavanja nota s notnog zapisa**

Naslov rada na engleskom jeziku: **Vision system for music notation recognition**

Opis zadatka:

Prepoznavanje nota s notnog zapisa posebno je interesantan muzičarima i umjetnicima koji svakodnevno koriste notni zapis, a koji često puta nije dovoljno čitak i razumljiv. Predlaže se vizijski sustav koji pomaže u svim takvim situacijama i olakšava rad korisnicima.

U radu treba načiniti sljedeće:

1. Opisati problem prepoznavanja nota sa notnog zapisa.
2. Pokazati stanje primjene vizijskog prepoznavanja nota.
3. Predložiti strukturu vizijskog sustava za rješavanje problema prepoznavanja nota.
4. Potrebnu programsku podršku pisati u programskom paketu Python i pripadajućim otvorenim bibliotekama.
5. Izvesti zaključke rada.

Zadatak zadan:
28. studenog 2019.

Datum predaje rada:
1. rok: 21. veljače 2020.
2. rok (izvanredni): 1. srpnja 2020.
3. rok: 17. rujna 2020.

Predviđeni datumi obrane:
1. rok: 24.2. – 28.2.2020.
2. rok (izvanredni): 3.7.2020.
3. rok: 21.9. - 25.9.2020.

Zadatak zadao:

Prof. dr. sc. Dubravko Majetić

Predsjednik Povjerenstva:

Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom završnom radu razmatran je prijedlog vizijskog sustava za prepoznavanje nota iz notnog zapisa baziran na računalnom vidu te primjeni umjetnih neuronskih mreža. Rad započinje osnovama glazbene teorije radi boljeg razumijevanja ovog rada, nakon koje slijedi konceptualna razrada problema. Opisan je i razrađen princip rada modela računalnog vida te obrade slika kreiran za potrebe ovog rada. Objašnjen je princip rada osnovnih umjetnih neuronskih mreža kao te je detaljnije razrađena struktura te primjena konvolucijskih neuronskih mreža.

Modeli spomenutih modela kreirani su u programskom paketu Python koristeći biblioteku OpenCV.

Ključne riječi: računalni vid, obrada slika, umjetne neuronske mreže, konvolucijske neuronske mreže, glazba

SUMMARY

This final thesis considers the use of artificial neural networks in visual recognition systems.
The paper begins with the basics of music theory for a better understanding of this work, followed by a conceptual elaboration of the problem. The principle of computer vision and image processing model created for the purpose of this paper is described and elaborated. The principle of operation of basic artificial neural networks is explained and the structure and application of convolutional neural networks are elaborated in more detail.

The models mentioned above were created in Python programming language, using the OpenCV library of programming functions.

Key terms: computer vision, image processing, artificial neural networks, convolution neural networks,music

1 UVOD

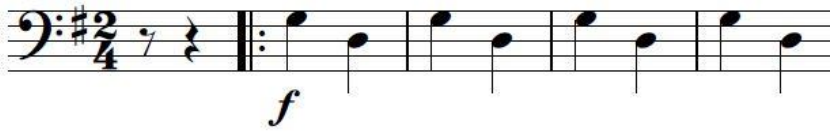
Umjetne neuronske mreže sve češće pronalazimo u svakodnevnom životu. Zbog svoje modularnosti, fleksibilnosti i mogućnosti prilagođavanja u odnosu na prethodna rješenja predstavljaju vrlo prigodan pristup za mnoge domene primjena. To je prikazano i dokazano u primjeru koji je razmotren u ovom radu.

Inicijalni problem svakog hobističkog, školovanog, samoukog ili profesionalnog glazbenika uvijek je razumijevanje notnog zapisa. Predloženo je rješenje koje omogućuje razumijevanje istog bilo kome, bez obzira na razinu znanja.
Ideja je izraditi vizijski sustav prepoznavanja i interpretiranja nota u svrhu olakšanog pristupa učenju i razumijevanju notnog zapisa uporabom umjetnih neuronskih mreža.

Prvi problem na koji se nailazi na prvi je pogled banalan – kako prepoznati notu?
To je glavna značajka koju očekujemo od vizijskog sustava, te je potrebno znati njega „naučiti“ isto.
U nastavku slijedi kratka razrada informacija iz područja teorije glazbe, potrebnih za bolje shvaćanje ovog rada.

2 OSNOVE TEORIJE GLAZBE

Na glazbu se strogo može gledati kao na zvukove u točno određenom vremenu. Vrijeme kada je potrebno proizvesti neki zvuk strogo je određeno i jednoznačno definirano u svakom notnom zapisu.
Sama brzina cjelokupne skladbe često je varijabilna, no zajedno s njom proporcionalno se mijenja i ritam skladbe.
Ritam skladbe može biti paran (2/4, 4/4) pa se broji 1, 2, 3, 4, 1, 2, 3, 4, 1, 2… ili neparan (3/4, 7/8), gdje brojanje ide 1, 2, 3, 1, 2, 3, 1, 2…
Oznaka takta uvijek se nalazi na samom početku notnog zapisa, kao što je prikazano na slici 2.1.



Slika 2.1. Početak notnog zapisa

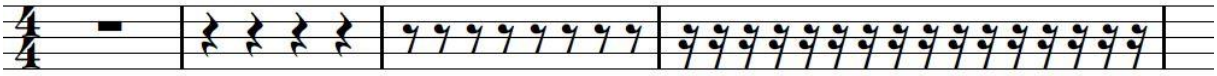
Jedna od upečatljivijih i globalno shvaćenih karakteristika notnog zapisa su horizontalne linije koje se protežu duž cijelog retka zapisa. One su smjernice za organizaciju i pozicioniranje samih nota.

Nadalje, svaka skladba segmentirana je na taktove. Takt je definiran kao područje između dvije vertikalne linije, zvane taktnim linijama te je njegovo trajanje relativno s obzirom na ritam i brzinu skladbe.

Slika 2.2. Trajanje nota

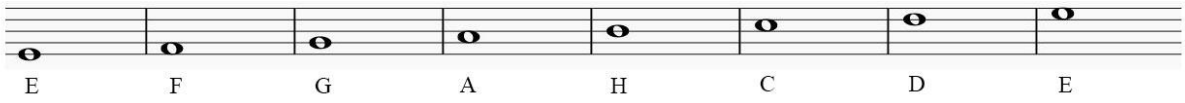
Na Slici 2.2. prikazano je pet taktova, od kojih svi, dogovorno, uvijek traju jednako. U svakom taktu nalazi se jedna vrsta nota. Svaka od prikazanih vrsta specifična je isključivo po vremenu trajanja.
Na slici se nalaze redom: puna nota, polovinka, četvrtinka, osminka te šesnaestinka. Naravno, postoji i više razina segmentiranja taktova na note sa još kraćim trajanjem ali njihovo korištenje je podosta rjeđe.
Note različitih vrsta smiju se proizvoljno kombinirati. Jedino je pravilo da ukupno trajanje nota unutar jednog takta traje točno onoliko koliko traje taj takt.
Jednako kao što note signaliziraju početak i trajanje zvuka, često je potrebno upravo suprotno – tišina. Svaka tišina, tj. period u kojem nema zvukova u skladbi, definirana je pauzom.

Jednaka pravila o vremenu trajanja nota, kao što je prethodno spomenuto, vrijede i za pauze.
Također, i pauze su podložne segmentiranju na dionice kraćeg trajanja (Slika 2.3.) te ih se koristi u kombinaciji s notama kraćeg trajanja kako bi se postigao željeni efekt.



Slika 2.3. Trajanje pauza

Sljedeća varijabla nota prikazana je na Slici 2.4. Ono što možemo određivati osim vremena trajanja pojedine note jest visina njezinog tona.



Slika 2.4. Oktava

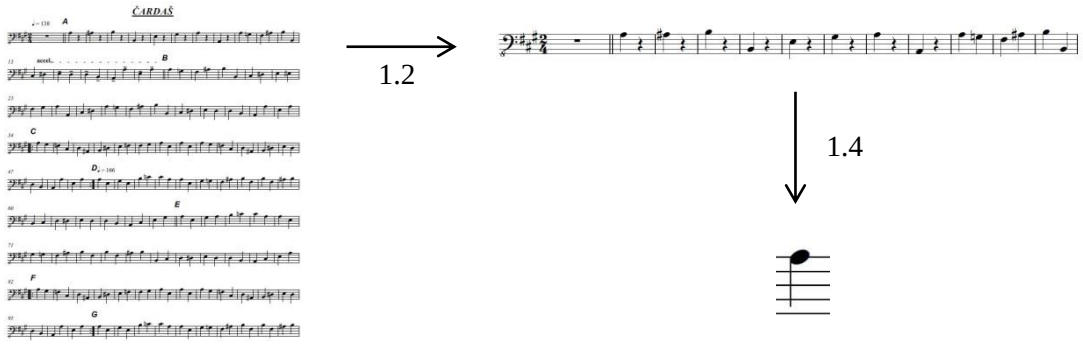
Na Slici 2.4. prikazano je osam nota sekvencijalno uzlazne visine tona koje čine jednu oktavu. Visina nota može nadilaziti ove prikazane na primjeru slike 4, kako s gornje, tako i s donje strane, što je često i slučaj.

3 IZRADA SUSTAVA

3.1 KONCEPTUALNA RAZRADA

Od sustava se očekuje da učita odabranu sliku notnog zapisa, pronađe note te prepozna o kojim se notama radi. Taj proces zamišljen je na sljedeći način:

1. Inicijalna obrada slike računalnim vidom
    1.1 Pronalaženje notnih redova
    1.2 Vertikalna segmentacija, odbacivanje neiskoristivih dijelova slike
    1.3 Pronalaženje simbola unutar notnog reda
    1.4 Horizontalna segmentacija, sortiranje korisnih dijelova slike, odbacivanje neiskoristivih



Slika 3.1. Konceptualna obrada slike

2. Prepoznavanje segmentiranih simbola iz dijelova slike s obzirom na poznate simbole primjenom neuronskih mreža

3.2 INICIJALNA OBRADA ULAZNE SLIKE

3.2.1 PRIPREMA ULAZNE SLIKE ZA OBRADU

Izrada kompletnog sustava realizirana je programskim paketom Python 3.7, uključujući obradu slika, konvolucijske umjetne neuronske mreže te grafičko sučelje.
Svaka manipulacija slikama izvršena je koristeći biblioteku programskih funkcija OpenCV unutar programskog paketa Python.

Prije nego počnemo razmatrati samu obradu ulaznih slika moramo se uvjeriti da će rezultati te obrade biti iskoristivi. Uzmimo u obzir dva najčešća izvora slikovnog zapisa nota: fotografije i pdf dokumenti. Oba predstavljaju problem na svoj način.

Razmotrimo li prvo varijantu ulaza pdf dokumenta, možemo uočiti kako je ona jednostavnije rješiva. Naime, u Python-u imamo mogućnost konverzije tipova dokumenata samo jednom linijom naredbi.
Iz pdf dokumenta u .jpg oblik, nama pogodan za manipulaciju, to se izvodi:

```
slika = pdf2image.convert_from_path('pdf_file.pdf', br_str)
```

Radi li se o fotografiji notnog zapisa, gotovo je nemoguće da ona ispadne savršena, tj. slikana okomito na sam notni zapis, bez deformacija, pomaka, smetnji, šuma…

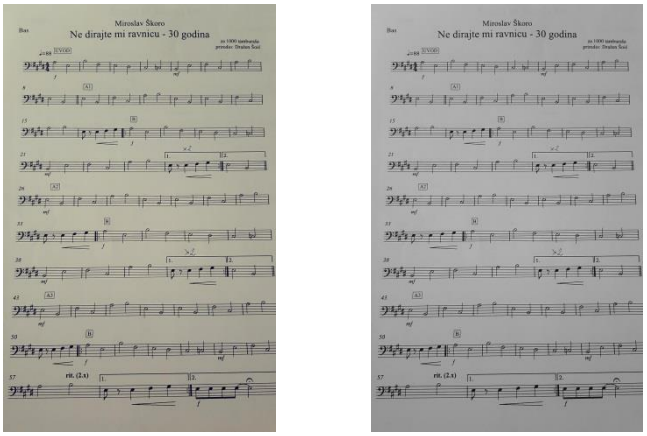
U tim slučajevima, moguće je manipulirati slikom na način da ju se dovede u stanje pogodnije za čitanje i daljnje obrađivanje. Idealna slika za obradu savršeno je binarna (crno – bijela), u a4 je formatu, redovi su horizontalni, stupci vertikalni, nema šuma ni nečistoća te je dovoljno visoke rezolucije.

Vizualno prepoznavanje objekata u ovom radu izvedeno je gledajući same piksele. Kako bi obrada slika bila što jednostavnija, razmatrat će se samo dvije vrste piksela – crni i bijeli. To podrazumijeva uklanjanje svih boja iz slike, kao i svih gradijenata sive boje, upravo svega onoga što nije potpuno crno ili potpuno bijelo.

Prvi korak koji se poduzima uklanjanje je zasićenja boja iz slike:

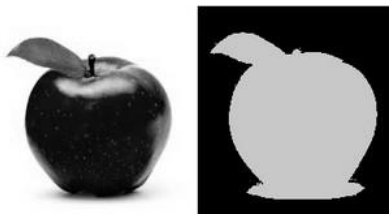
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY),
```

što rezultira efektom prikazanim na slici 3.2.



Slika 3.2. Uklanjanje zasićenja bojom [1]

Sljedeće što je potrebno napraviti jest izdvojiti dijelove slike koje zadrže objekte koje želimo analizirati. Ta separacija je bazirana na razlici razina svjetlosti piskela između objekata koji su nam od interesa i pozadine. Kako bismo odredili koji su nam pikseli bitni, a koji ne, provodimo usporedbu vrijednosti intenziteta svakog piksela u odnosu na prag (eng. *threshold*). Jednom kad smo separirali važne piksele, određujemo im fiksne vrijednosti po volji (vrijednost 0 za crnu boju, 255 za bijelu).



Slika 3.3. Separacija dijelova slike [2]

Paket OpenCV nudi na raspolaganje 5 vrsta praga: binarni, invertirani binarni, skraćeni, *threshold to zero* te invertirani *threshold to zero.* [2]

Najpogodniji za slučaj ovog rada, te ujedno i onaj koji je primijenjen je prag tip binarni invertirani (eng. *binary inverted*). Razlog tomu je isključivo kompatibilnost sa mnogim prethodno definiranim funkcijama programskog paketa python koje je lakše primijeniti u slučaju kada je vrijednost „nebitnih“ pikseli jednaka nuli. Detaljnije na stranici 18.

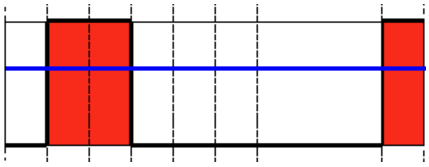
Invertirani binarni prag

Ova operacija matematički se može izraziti kao

$$dst(x,y) = \begin{cases} 0 & if\ src(x,y) > thresh \\ maxVal & else \end{cases}$$

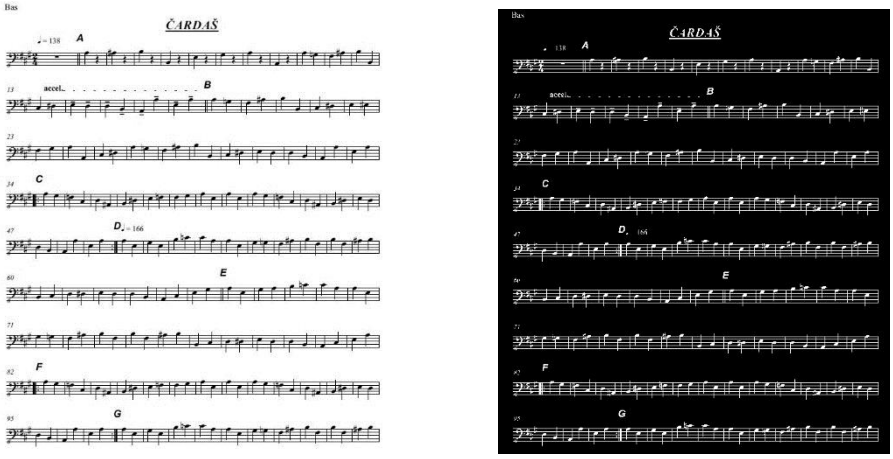
što znači da ako je vrijednost razine piksela veća od vrijednosti definirane varijablom `th`, nova razina postavlja se u vrijednost '0', a sve ostale postavljaju se u maksimalnu vrijednost, što pak znači da pozadina notnog zapisa postaje crna, a crtovlje zajedno sa svim simbolima postaje bijelo.

Vizualna prezentacija invertiranog binarnog praga prikazana je na slici 3.4.



Slika 3.4. Invertirani binarni prag [2]

Konkretno, primijenjeno na notni zapis prikazano je na slici 3.5.



Slika 3.5. Segmentirani notni zapis

Za postizanje tog efekta korištena je sljedeća naredba:

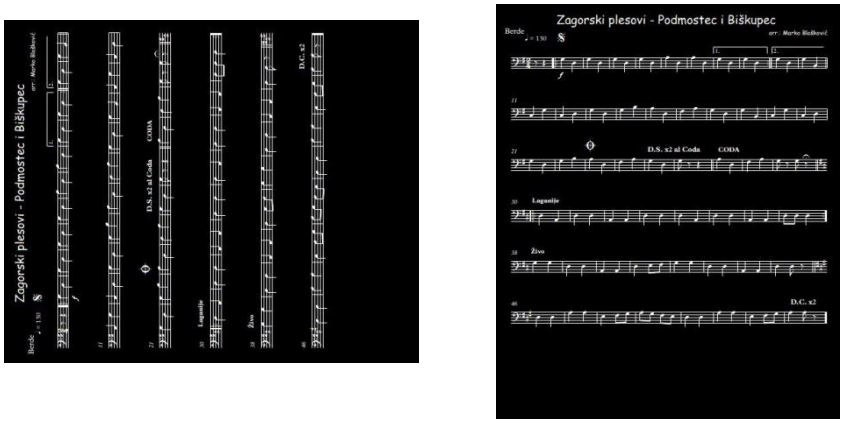
```
# threshold
th, threshed = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)
```

Prilikom fotografiranja slika postoji mogućnost da je fotografija spremljena u horizontalnoj orijentaciji (eng. *landscape*). Uzimajući u obzir segmentaciju slike koju je potrebno provesti (vidi poglavlje 3.2.2), neispravno spremljenu sliku potrebno je ispraviti u kompatibilnu vertikalnu orijentaciju.

Operacija `findNonZero pretražuje` vrijednosti piksela slike te odabire sve što nije '0'. Kako smo prethodno primijenili operaciju `threshold`, sve što na slici nije nula jest maksimalna vrijednost, odnosno svi bijeli pikseli.
Funkcija `minAreaRect` ima broje primjene, ali u ovom slučaju korištena je za opisivanje svih prethodno 'pronađenih' bijelih piksela te opisivanja pravokutnika minimalne moguće površine oko njih.
Naredba nastavlja izvlačenjem podataka iz dostupnih informacija o slici kao što su početna koordinata pronađenog pravokutnika, dijagonalno suprotna koordinata istog te kut zakreta u odnosu na x-os.

```
pts = cv2.findNonZero(threshed)              # pronalaženje bijelih piksela
ret = cv2.minAreaRect(pts)                   # stvaranje pravokutnika min pov

(cx, cy), (w, h), ang = ret                  # definiranje podataka pravokut
if w > h:                                    # usporedba visine i širine
    w, h = h, w                              # zamjena stupaca i redaka
    ang += 90                                # rotiranje za 90 stupnjeva
```

Vizualni prikaz te naredbe naizgled je jednostavna operacija, kao što je prikazano na slici 3.6.



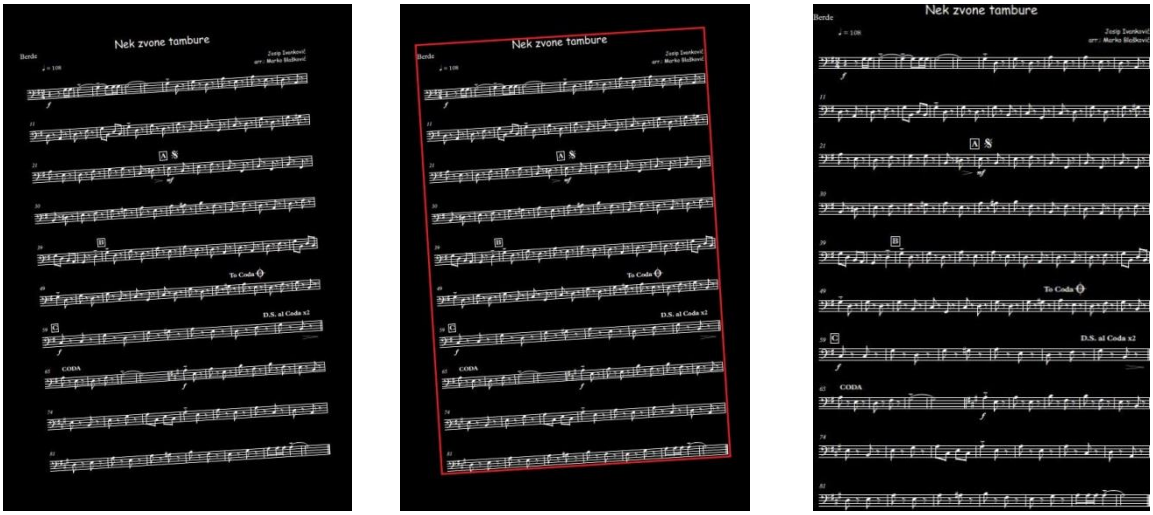
Slika 3.6. Ispravljanje orijentacije slike

Srodno prethodnoj operaciji, na sličan način moguće je eliminirati rotacijske greške nastale nepažljivim fotografiranjem.

Nastavno na sve prethodno definirane varijable nastavlja se sljedećom operacijom:

```
M = cv2.getRotationMatrix2D((cx, cy), ang, 1.0)
rotated = cv2.warpAffine(threshed, M, (img.shape[1], img.shape[0]))
```

Funkcija `getRotationMatrix2D` u dvodimenzionalnom koordinatnom sustavu definira matricu rotacije, uzimajući u obzir ulaze parametre fiksne točke rotacije definirane koordinatama na slici te kut zakreta iste. Odmah iza nje nalazi se funkcija `warpAffine` koja generira novu mrežu piksela u ovisnosti o prethodno definiranim uvjetima pozicije i rotacije iste te ju ispunjava odgovarajućim pikselima iz prethodne, originalne mreže piksela, te na taj način omogućujući transformaciju slike.
Operacija transformacije slike notnog zapisa rotacijom prikazana je na slici 3.7.

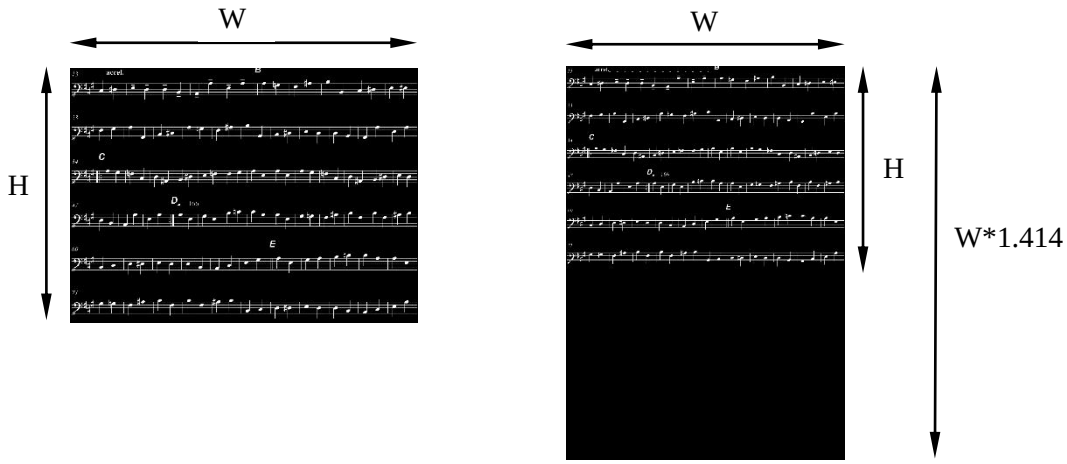


Slika 3.7. Ispravljanje rotacije slike

Moguć je slučaj da fotografije ili slike budu raznih formata, ovisno o rezoluciji i omjeru dimenzija slike objektiva kamere. Poželjno za ovaj slučaj jest da sve slike budu jednakog formata, odnosno jednakih omjera stranica. Razlog tomu detaljnije je objašnjen na stranici 20.
Najjednostavniji način za rješavanje ovog problema proširivanje je jedne dimenzije dosad obrađene slike. Uzimajući u obzir kako je prethodno slika notnog zapisa obrađena na način da je ispravno orijentirana, tj. da joj je vertikalna dimenzija veća od horizontalne, možemo primijeniti sljedeću operaciju:

```
bordered = cv2.copyMakeBorder(rotated,0,int((W*1.414)-H),0,0,
cv2.BORDER_CONSTANT,value=[0, 0, 0])
```

Njezina uloga je na postojeću dosad obrađenu sliku nadodati proširenje s donje strane slike kako bi joj omjeri stranica bili 1 x 1.414. To je postignuto na način da se nadoda proširenje dimenzije `(W*1.414)-H,` gdje je `W` širina, a `H` visina slike. Grafički prikaz ove operacije vidljiv je sa slike 3.8.



Slika 3.8. Formatiranje slike

3.2.2 VERTIKALNO SEGMENTIRANJE

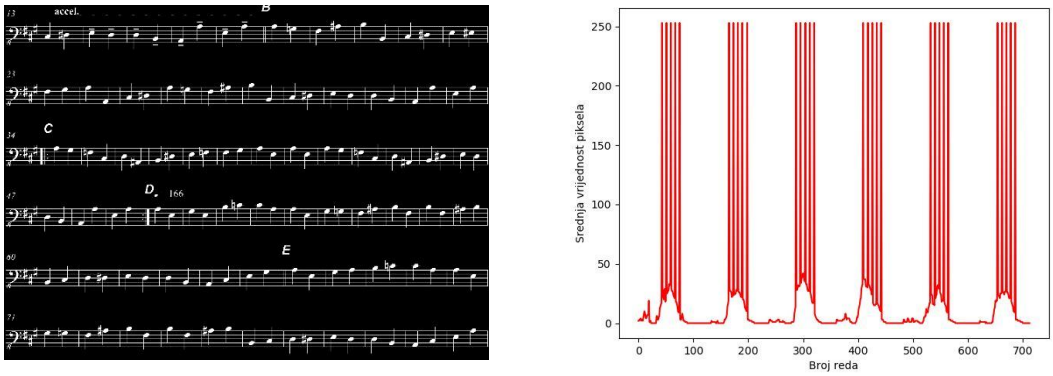
Uzimajući u obzir prethodno poduzete korake, sada se na raspolaganju nalazi slika notnog zapisa koju je moguće dalje obrađivati. Nakon pripreme za obradu slike slijedi pronalaženje korisnih dijelova slike, odnosno simbola notnog zapisa, u svrhu pojednostavljivanja prepoznavanja pojedinih simbola unutar slike notnog zapisa, koje će biti detaljnije objašnjeni niže.

Kao što je prethodno objašnjeno u konceptualnom modelu, izvlačenje simbola izvodi se prvenstveno vertikalnom segmentacijom na redove notnog zapisa, a potom horizontalnim segmentiranjem simbola unutar svakog reda, prepoznavajući i uzimajući u obzir da nisu svi „pronađeni“ dijelovi slike koji sadrže neke pozitivne vrijednosti iskoristivi ili korisni.

Operacija vertikalne segmentacije započinje stvaranjem vektorskog histograma, što je izvedeno sljedećom naredbom:

```
hist = cv2.reduce(rotated, 1, cv2.REDUCE_AVG).reshape(-1)
```

Funkcija `reduce(rotated, 1, cv2.REDUCE_AVG)` uzima priloženu matricu kao ulazni parametar. Dotična matrica dobivena je izvlačenjem vrijednosti piksela iz prethodno pripremljene ulazne slike, u ovom slučaju spremljene kao `rotated`. Sljedeći ulazni parametar funkcije je prvi parametar slike `reduce` koji nam je dostupan (vrijednosti x-osi), koji želimo na izlazu funkcije. Kao treći ulazni parametar funkcije je tražena izlazna funkcija `REDUCE_AVG`, koja daje vektor srednjih vrijednosti piksela po redovima, dobivenih sumom svih vrijednosti piksela po redu te dijeljenjem s brojem piksela unutar tog istog reda. Konačno, funkcija `reshape(-1)` uzima izlaze funkcije `reduce` te ih transformira u kompatibilan oblik za daljnje operacije, odnosno horizontalni vektor srednjih vrijednosti piksela redova slike notnog zapisa.
Na slici 3.9. nalazi se grafički prikaz objašnjenja operacije pronalaženja korisnih piksela u redovima slike notnog zapisa.



Slika 3.9. Vertikalni histogram

Kao što je prethodno spomenuto, ovdje do izražaja dolazi odabir vrste praga kao invertirani binarni. Iz grafičkog prikaza jasno je vidljivo u kojim redovima se nalaze korisni pikseli, te je dalje lako segmentirati one iskoristive (svi veći od definirane granice praga) od onih beskorisnih (svi manji od definirane granice praga).

Nakon što su pronađene vrijednosti korisnih piksela po redovima slike potrebno je definirati vrijednost praga koja će služiti kao granica segmentacije korisnih od beskorisnih piksela. Eksperimentalnim empirijskim pristupom utvrđena je vrijednost `th = 2` kao optimalna za segmentaciju redova notnog zapisa koji sadrže korisne piksele od onih koji ne.

S poznatom vrijednosti praga, sljedećim petljama kreiraju se liste koordinata po y-osi slike notnog zapisa (koordinate redova) početaka i kraja grupa onih redova slike notnog zapisa koji sadrže korisne piksele, usporedbom s prethodno definiranom graničnom vrijednošću praga.

```
# gornje granice
upperBound = [y for y in range(H - 1) if hist[y] <= th < hist[y + 1]]
# donje granice
lowerBound = [y for y in range(H - 1) if hist[y] > th >= hist[y + 1]]
```

Lista `upperBound` inkrementalno pronalazi i sprema koordinate onih redova za koje je utvrđeno da je njihova srednja vrijednost trenutno manja od definirane vrijednosti praga, a u sljedećem koraku postaje veća, čineći time prijelaz iz reda beskorisnih piksela u onaj s korisnim te tako definirajući gornju granicu korisne skupine redova.
Lista `lowerBound` radi upravo suprotno, gledajući prijelaze s redova korisnih piksela u red beskorisnih, definirajući tako koordinate krajeva one skupine onih redova koja sadrži korisne piksele.

Slijedi naredba koja proširuje prethodno definirane izreske (eng. *slice*) redova slike s korisnim pikselima za empirijski određen udio piksela relativno s obzirom na ukupnu visinu slike, radi bolje preglednosti izvučenih izrezaka te ih sprema u listu `slices`.

```
slices = []
# izrezivanje redova
for i in range(len(up_array)):
    # prosiri slice vertikalno
    if (low_array[i] + 1) + int(H / 350) < H and up_array[i] - int(H / 70)>0:
            h_slice = rotated[up_array[i] - int(H / 70):(low_array[i] + 1)
             + int(H / 350), 0:W]
    # ne prosiruj na rubovima
    else:
            h_slice = rotated[up_array[i]:(low_array[i] + 1), 0:W]
    # dodaj slice u listu
    slices.append(h_slice)
```

Drugi dio naredbe, zadovoljavanjem uvjeta `else`, prepoznaje da se dotični izresci nalazi uz rub slike te na njega ne primjenjuje operaciju proširivanja, zbog toga što bi pokušaj izvedbe te naredbe rezultirao greškom jer pikseli potrebni za proširivanje dotičnog izreska nisu raspoloživi jer se njihove koordinate nalaze izvan granica slike.
U tom slučaju, izrezak je spremljen u originalnom stanju, tj. u vertikalnom intervalu od gornje granice (`up_array`) do donje granice (`low_array`) te u horizontalnom intervalu od početka do kraja slike, odnosno od lijevog do desnog ruba slike (`0:W`).

Nakon izvršavanja posljednje operacije kreirana je lista skupina koordinata redova slike s korisnim pikselima, uključujući proširenja.

Naravno, potrebno je uzeti u obzir kako to uključuje kompletno sve redove piksela koji zadovoljavaju uvjet prethodno definiran pragom, što nerijetko uključuje glazbene oznake, komentare, naslove, ritam, podcrtavanja te ostale simbole koji su nepotrebni.
Sljedeća operacija bazirana je na činjenici da ulazna slika mora biti omjera stranica formata A4. Utvrđeno je kako je standardizirana visina retka notnog zapisa otprilike 4% ukupne visine stranice notnog zapisa. Prema tome, možemo definirati filter koji pretražuje sve prethodno spremljene izreske i eliminira sve one koji su manji od 4% parametra visine ulazne slike.

```
# brisanje nepotrebnih sliceova
slices[:] = [s for s in slices if not len(s) < 0.04 * H]
```

Obavljajući prethodnu operaciju, svi preostali izresci u listi `slices` trebali bi biti isključivo oni koji sadrže korisne informacije, odnosno crtovlja notnog zapisa uz par iznimaka. Uzimajući tu činjenicu u obzir, sljedećom operacijom uklanjamo spomenute iznimke.

Kreira se nova lista izrezaka, `valid_slices_pixel_mean` koja sadrži prethodno objašnjene korisne izreske.

```
valid_slices_pixel_mean = []
# mean svakog slicea
for s in slices:
    valid_slices_pixel_mean.append(np.mean(s))
```

Spomenute iznimke mogle bi biti npr. naslovi skladbi velikog fonta, mrlje na papiru, komentari ili bilo što u vertikalnoj dimenziji zauzima više od 4% ukupne visine slike notnog zapisa.
Specifičnost takvih iznimaka je ta što gotovo uvijek imaju malu horizontalnu dimenziju, te je na tome bazirana sljedeća operacija. Koncipirana je na način da se ponovno prolazi kroz sve korisne izreske u listi `valid_slices_pixel_mean` te se računa globalna srednja vrijednost piksela *mean*, koja će se koristiti kao referenca za eliminaciju svih prethodno spomenutih iznimaka.

```
# traženje srednje vrijednosti svih izrezaka
mean = np.mean(valid_slices_pixel_mean)
```

S definiranom globalnom srednjom vrijednosti piksela po redovima unutar izreska ulazimo u sljedeću operaciju, koja zadržava sve izreske čija je srednja vrijednost piksela po redovima unutar 15% tolerancije od referentne vrijednosti definirane prethodnom operacijom, a odbacuje sve ostale.

```
# spremi valjane izreske
j = 0
for i in range(len(slices)):
    #  valjani izresci imaju otprilike istu sr vrijednost piksela,
#ignoriraj beskorisne izreske (+- 15% sr vrijednosti)
    if 1.15 * mean > valid_slices_pixel_mean[i] > 0.85 * mean:
            # imenovanje sliceova
        sliceName = "slice" + str(j) + ".jpg"
            # directory za izreske
        path = "resources/" + image_filename[:-4] + "/"
            # kreiraj dir ako ne postoji
        try:
            os.makedirs(path)
        except FileExistsError:
            pass
            # pohrani izreske u taj dir
        cv2.imwrite(path + sliceName, slices[i])
            # iterativno imenovanje izrezaka
        j = j + 1
```

Drugi dio gornje operacije inkrementalno imenuje i numerira svaki izrezak koji zadovoljava gore navedeni uvjet, te kreira novu mapu na adresi učitane slike u koju spremna novonastale slike izrezaka.

U konačnici, na kraju vertikalne segmentacije dobivaju se izresci slike svakog pojedinog reda notnog zapisa koji sadrži korisne informacije o samoj skladbi, tj. simbole koje je potrebno prepoznati. Primjer jednog takvog izreska nalazi se na slici 3.10.

Slika 3.10. Primjer izreska

3.2.3 HORIZONTALNO SEGMENTIRANJE

Nastavno na vertikalno segmentiranje, gdje se u konačnici završilo sa izrescima slike koji su od interesa proces obrade slike nastavlja se duž druge osi ulazne slike, segmentiranjem u horizontalnom smjeru. Iako vrlo sličan procesu dobivanja vertikalnih segmenata, ovaj proces ima neke specifičnosti koje će niže biti razmotrene.

Sam proces horizontalnog segmentiranja započinje sekvencijalnim učitavanjem prethodno kreiranih, numeriranih i pohranjenih slika izrezaka. To je postignuto petljom:

```
images = []
for r, d, f in os.walk(path):       # r=root, d=directories, f = files
    for file in f:
        if '.jpg' in file:
            if 'el' not in file:
                images.append(file)
```

U zadanom direktoriju pretražuju se datoteke te im se očitavaju 3 podatka: root, directory i naziv. U datom direktoriju, kreiranom u prethodnoj operaciji vertikalne segmentacije nalaze se isključivo izresci koji su prethodno kreirani izrezivanjem iz slike notnog zapisa.
Posljednji uvjet provjerava nalazi li se dio teksta „el“ u nazivu bilo kojeg dokumenta. Razlog tomu je što će svaka nova kreirana slika na kraju ove operacije započinjati nazivom „el“, te se na ovaj način eliminira bilo kakva potencijalna greške očitavanja pogrešnog dokumenta.

Nadalje, unutar crtovlja notnog zapisa nalazi se mnogo više podataka nego izvan njega, na što je potrebno dodatno obratiti pažnju. Prije samo početka segmentiranja ponovno je potrebno dodatno pripremiti sliku radi lakšeg čitanja željenih podataka te zanemarivanja nebitnih.
To uvelike postižemo kreiranjem funkcije `erode_dilate`, koja ima dvojako svojstvo.

```
def erode_dilate(img):
    kernel1 = np.ones((2, 2), np.uint8)  # kernel s dim 2x2, 1 if all = 1
    eroded_img = cv2.erode(img, kernel1, iterations=1)  # prvo erode
    dilated_img = cv2.dilate(eroded_img, None, iterations=1)  # zatim dilate
    return dilated_img
```

Naredba `erode` funkcija je za obradu slike, gdje je svakoj pronađenoj skupini piksela bijele boje odstranjen obrub. Funkcija `erode` to postiže na način da redom pretražuje sliku te ukoliko naiđe na piksel vrijednosti '1' provjerava okolne piksele. Ukoliko su svi pikseli okolo njega jednaki '1', taj piksel ostaje '1', u suprotnom on se briše (pretvara u '0').

Slika 3.11. Erozija slike [2]

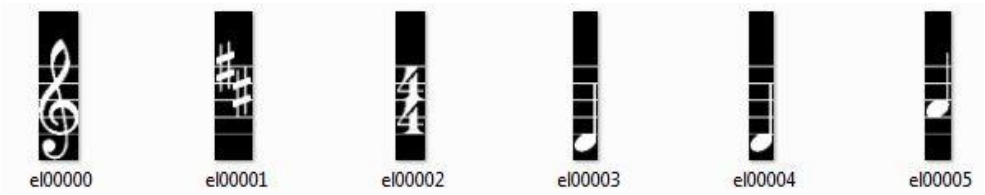
Na taj način vrlo efektivno možemo ukloniti bilo kakav zaostali bijeli šum ili nečistoće na slici.

Suprotno tome, koristi se funkcija `dilate`. Analogno `erode`, ona pretražuje sliku te takoreći stvara obrub oko konture skupine pronađenih piksela definiranih s vrijednosti '1'.

Slika 3.12. Dilatacija slike [2]

Ova nam funkcija koristi za vraćanje prethodno erodiranih korisnih simbola u iskoristivo početno stanje. Konačno, ovom funkcijom eliminira se bilo kakav remanentni bijeli šum na slikama.

Primijenivši gore navedene operacije na izrezak prethodno dobiven vertikalnom segmentacijom raspolaže se svakim pojedinim simbolom, koji se sada može dalje procesirati učitavanjem u neuronsku mrežu. Nekolicina primjera takvih konačnih izrezaka nalazi se na slici 3.13.



Slika 3.13. Primjeri konačnih izrezaka

Potpuni kod obrade slike nalazi se u Prilogu 1.

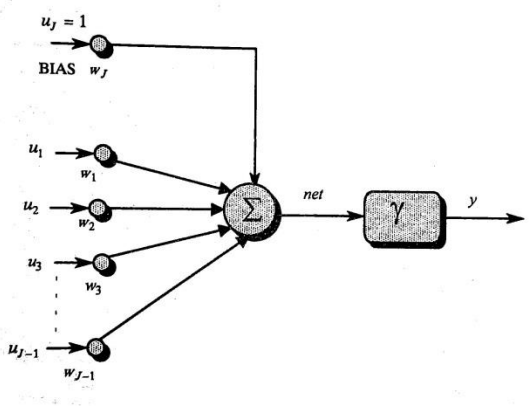
3.3 PREPOZNAVANJE SIMBOLA

Sam postupak „prepoznavanja“ pojedinih simbola unutar slike notnog zapisa može biti izveden na više načina. U ovom poglavlju razmatrat će se moguće metode iskorištavanja umjetnih neuronskih mreža za postizanje upravo toga.

Prije detaljnog opisa umjetnih neuronskih mreža koje se koriste pri prepoznavanju nota, potrebno je objasniti općeniti model umjetnih neuronskih mreža te kako one u osnovi funkcioniraju.

3.3.1 OSNOVE UMJETNIH NEURONSKIH MREŽA

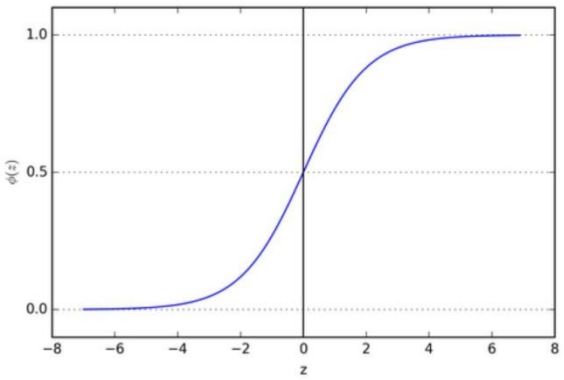
Iz samog imena, „umjetne neuronske mreže“, vidljivo je kako su one ime dobile po stvarnim neuronskim mrežama, odnosno ljudskom mozgu, čiji rad oponašaju. Stvorene su iz potrebe kreiranja modela koji je sposoban generirati informacije, prihvaćati ih, obrađivati te prenositi dalje, odnosno procesirati informacije analogno ljudskom mozgu. Sukladno tome, kao što je ljudski mozak građen od bioloških neurona, tako je i svaka umjetna neuronska mreža građena od umjetnih neurona.



Slika 3.14. Model umjetnog neurona [3]

Iz slike 3.14 vidljivo je da se model umjetnog neurona ugrubo sastoji od četiri dijela – ulaza, sumatora, aktivacijske funkcije i izlaza.

Za aktivacijsku funkciju za potrebe ovog rada prikladna bi bila nelinearna bipolarna sigmoidalna funkcija. Ova aktivacijska funkcija globalnog je karaktera i pokazuje dobre rezultate primjenom bilo na statičkim ili dinamičkim sustavima, što je ujedno i razlog zašto je upravo ovo u primjeni najčešće korištena aktivacijska funkcija. Grafički je prikazana na slici 3.15.

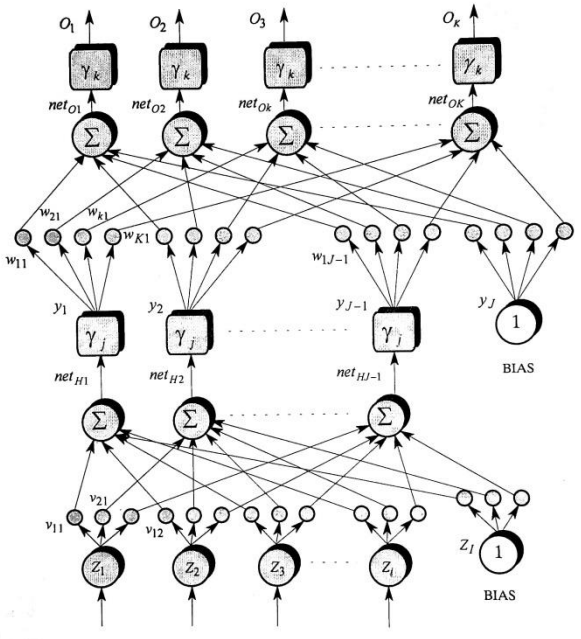


Slika 3.15. Sigmoidalna aktivacijska funkcija

Aktivnost umjetnog neurona modelira se kao zbroj ulaza pomnoženih nekim faktorom (težinama), a ovisi o broju ulaza, odnosno veza s okolinom neurona, intenzitetu tih veza te pragu osjetljivosti.

STRUKTURA MREŽE

Da bi se dobila struktura (topologija ili konfiguracija) neuronske mreže, potrebno je neurone organizirati u slojeve, a slojeve međusobno povezati vezama opterećenim težinskim koeficijentima. Kod toga razlikujemo tri vrste slojeva mreže: ulazni sloj, skriveni sloj i izlazni sloj neurona. Ulazni i izlazni sloj u direktnoj su interakciji s okolinom, dok skriveni sloj nije, otkuda i naziv „skriveni“.
Najpoznatiji i najčešće upotrebljavani tip neuronske mreže je statička unaprijedna višeslojna neuronska mreža (SNN, Static Neural Network), čiji je model prikazan na slici 3.16. [3]



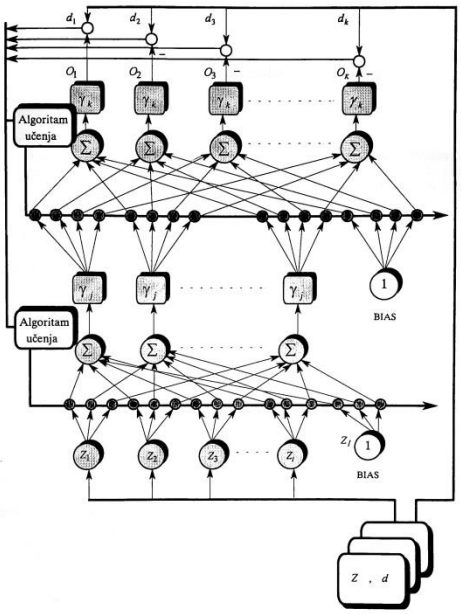
Slika 3.16. Model statičke unaprijedne neuronske mreže [3]

Ulazi neurona ulaznog sloja ujedno su i ulazi u mrežu. Uobičajeno je da neuroni u ulaznom sliju nisu neuroni u pravom smislu, već su to u biti čvorovi za distribuciju ulaza neuronske mreže prema prvom skrivenom sloju neurona.
Treba naglasiti da su svu slojevi mreže potpuno umreženi, što znači da je svaki neuron promatranog sloja vezan sa svakim neuronom iz prethodnog sloja. Izuzetci tomu jedino su i uvijek neuroni označeni s *bias*, dakle neuroni konstantne izlazne vrijednosti jednake '1'. [3]

Broj skrivenih slojeva je proizvoljan, a najčešće se koristi jedan do dva skrivena sloja. On direktno utječe na kompleksnost same neuronske mreže, na način da se povećanjem broja slojeva odnosno neurona skrivenih slojeva ostvaruje veći potencijal za dobivanje veće preciznosti izlaznih podataka mreže, ali proporcionalno time unosi i dodatno kašnjenje, odnosno povećava vrijeme potrebno za procesiranje ulaznih podataka u mrežu, a time i zahtjev za više računalnih resursa.
Teoretski radovi Cybenko i Funanashi [3] pokazuju da je jedan jedini skriveni sloj neurona dostatan za dobru aproksimaciju bilo koje kontinuirane funkcije, uz uvjet da takav sloj ima dovoljan broj neurona.
Potrebno je naglasiti i da je broj ulaznih i izlaznih neurona uvijek definiran karakterom preslikavanja koje mreža mora obaviti. Ako se primjerice radi o učenju ponašanja nekog dinamičkog sustava, broj ulaznih neurona u pravilu odgovara broju ulaza sustava, a broj izlaznih neurona odgovara broju izlaza sustava.

UČENJE MREŽE

Razlikujemo tri osnovna karaktera učenja: učenje s učiteljem (eng. *Supervised*), učenje bez učitelja (eng. *Unsupervised*) te učenje koje je kombinacija prethodnih (eng. *Reinforcement learning*). Ovdje je detaljnije razrađen oblik učenja s učiteljem, kako je on najčešći te ujedno i onaj korišteni za potrebe ovog rada. Postupak učenja neuronske mreže nije ništa drugo nego podešavanje težinskih koeficijenata veza između slojeva mreže s ciljem da se izlazi mreže što više približe poznatim, željenim iznosima izlaza mreže. To istovremeno znači da postupak učenja ne može osigurati sto postotnu točnost. Izlaz mreže uvijek je neka aproksimacija željenog izlaza. Kvaliteta te aproksimacije u funkciji je više čimbenika, kao što su sam zadatak učenja, odabrana topologija mreže te odabrani algoritam učenja.
Osnovna karakteristika učenja s učiteljem je da se u iterativnom postupku učenja, tj. iterativnom postupku adaptacije težinskih koeficijenata veza, mreži uzastopce prikazuju ulazne veličine i za njih odgovarajuće željene izlazne veličine, što je shematski prikazano slikom 3.17. [3]



Slika 3.17. Shematski prikaz postupka učenja statičke mreže [3]

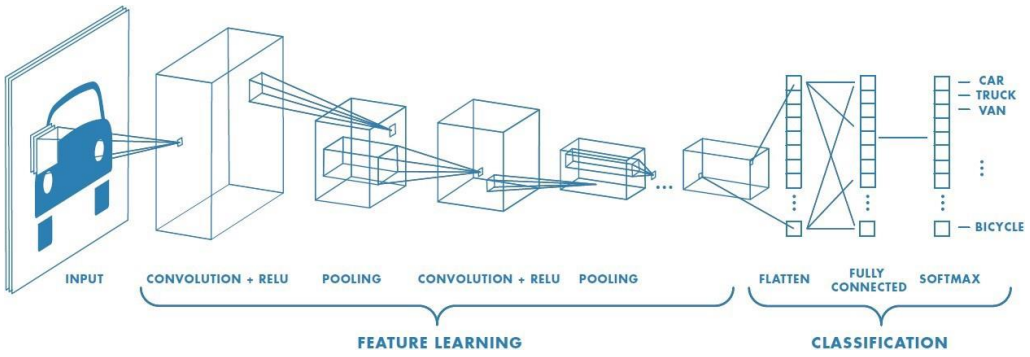
3.3.2 KONVOLUCIJSKA NEURONSKA MREŽA

Za potrebe ovog rada razmatrana je nadograđena i kompleksnija izvedba klasične statičke povratne neuronske mreže.

Konvolucijske neuronske mreže (ConvNets ili CNN) su neuronske mreže koje se koriste prvenstveno za klasifikaciju slika (tj. imenovanje onoga što vide), grupiranje slika po sličnosti (pretraživanje fotografija) i obavljanje prepoznavanja objekata unutar slika. Na primjer, konvolucijske neuronske mreže koriste se za identificiranje lica, pojedinaca, uličnih znakova, tumora, i mnogih drugih aspekata vizualnih podataka. [4]

Učinkovitost konvolucijskih mreža u prepoznavanju slike jedan je od glavnih razloga zašto se svijet probudio do učinkovitosti dubokog učenja. U određenom smislu, CNN-ovi su razlog zašto je duboko učenje poznato. CNN-ovi postižu snažni napredak u računalnom vidu (CV), koji ima očite primjene za autonomna vozila, robotiku, dronove, sigurnost, medicinske dijagnoze i liječenje slabovidnih osoba. Konvolucijske mreže mogu obavljati i banalnije (i profitabilnije), poslovno orijentirane zadatke, poput optičkog prepoznavanja simbola (OCR), digitalizaciju teksta i omogućavanje obrade na analognim i rukom pisanim dokumentima, gdje su slike simboli koje je potrebno prepisati.

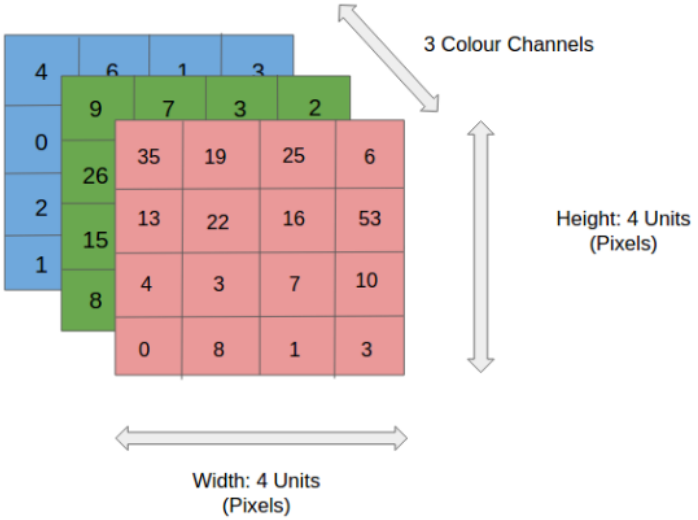
Međutim, CNN-ovi nisu ograničeni na prepoznavanje slike. Primjenjive su i za obradu i prepoznavanje zvuka, kada je on vizualno prikazan spektrogramom.



Slika 3.18. Shema principa rada konvolucijske neuronske mreže [4]

Konvolucijske mreže učitavaju te obrađuju slike zapisane u obliku tenzora.

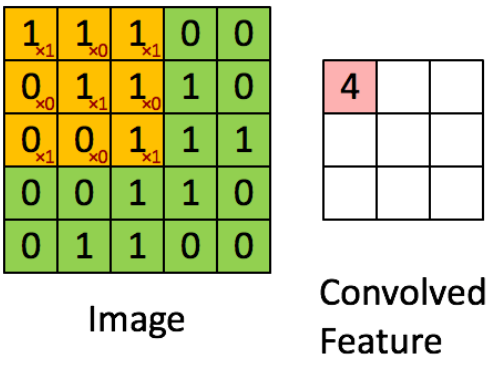
Širina i visina slike se lako razumiju. Treći parametar slike koji konvolucijska neuronska mreža očitava iz slike jest dubina, koja je potrebna zbog načina kodiranja boja unutar slike. Na primjer, kodiranje crveno-zeleno-plave boje (RGB) stvara sliku „duboku“ tri sloja. Svaki se sloj naziva „kanal“, a kroz konvoluciju stvaraju se mape značajki. Značajke su samo detalji slika, poput crte ili krivulje, koje konvolucijske neuronske mreže mapiraju radi kasnije obrade, grupiranja te prepoznavanja.



Slika 3.19. Simbolički matrični zapis slike u boji

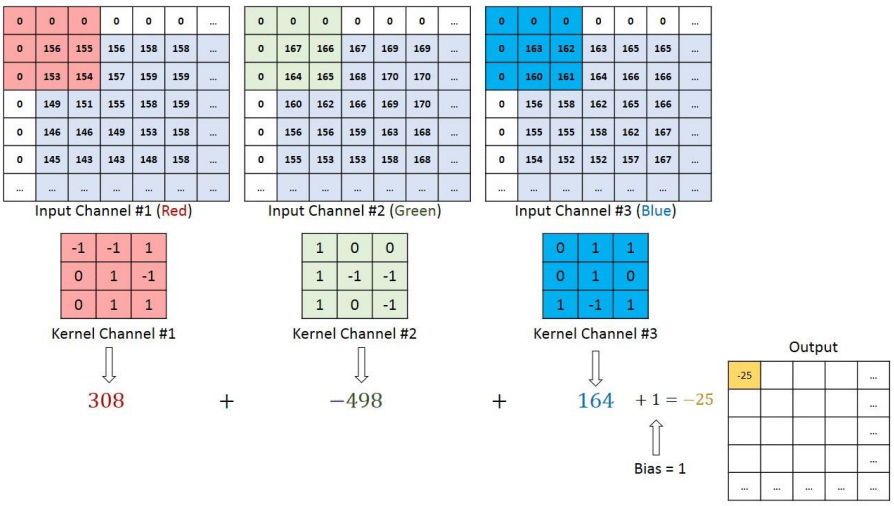
Na slici 3.19 zapis je RGB slike koja je podijeljena u tri ravnine boja - crvenu, zelenu i plavu. Postoji nekoliko takvih prostora u bojama u kojima postoje slike - siva, RGB, HSV, CMYK itd.

Uloga CNN-a je smanjiti slike u oblik koji je lakši za obradu, bez gubitka značajki koje su ključne za dobru pretpostavku. Ovo je važno kada se dizajnira arhitekturu koja nije dobra samo za učenje značajki, već i skalabilna za velike skupove podataka.



Slika 3.20. Konvolucijski filter

Na slici 3.20. prikazana je binarna slika dimenzija 5x5x1. Element koji sudjeluje u izvođenju operacije konvolucije u prvom dijelu konvolucijskog sloja naziva se kernel / filter, prikazan žutom bojom. U ovom slučaju odabrani kernel matrica je dimenzija 3x3 koja ima vrijednosti [1 0 1; 0 1 0; 1 0 1]. Kernel se pozicionira nad sliku te množi vrijednosti svojih ćelija s vrijednostima odgovarajućih piksela nad kojima se one nalaze, te ih u konačnici sumira i tu vrijednost upisuje (mapira) na odgovarajuću poziciju, kreirajući tako novi, pojednostavljeniji zapis originalne slike. Kernel operaciju filtriranja započinje u lijevom gornjem kutu slike te se pomiče za jedan piksel udesno, dok ne dođe do kraja slike. U tom slučaju, vraća se natrag na lijevi rub, spušta za jedan piskel te nastavlja proces dok se u konačnici ne nađe u desnom donjem rubu slike.

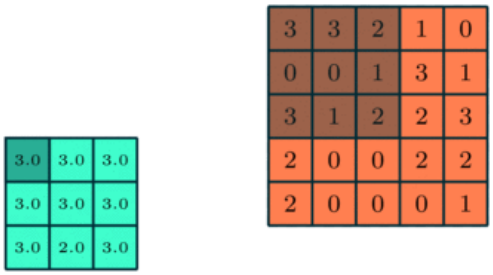


Slika 3.21. Konvolucijski filter za RGB sliku [4]

Radi li se o slici u boji, nakon što je ona segmentirana po kanalima kao što je prethodno pokazano, kernel prolazi kroz nju analogno operacijama s jednim kanalom. U konačnici rezultantni izresci slike svakog kernela sumiraju se zajedno s *biasom* te se mapiraju u novi zapis slike (slika 3.21.).

Konačni cilj operacije konvolucije je iz ulazne slike izvući i prepoznati značajke. Konvolucijske neuronske mreže ne moraju biti ograničene na samo jedan konvolucijski sloj. Uobičajeno, prvi sloj konvolucije odgovoran je za prepoznavanje značajki niske razine kao što su rubovi, boja, orijentacija gradijenta itd. Uz dodate slojeve, arhitektura se prilagođava i značajkama visoke razine, pružajući nam mrežu koja ima dobro razumijevanje slika u skupu podataka.

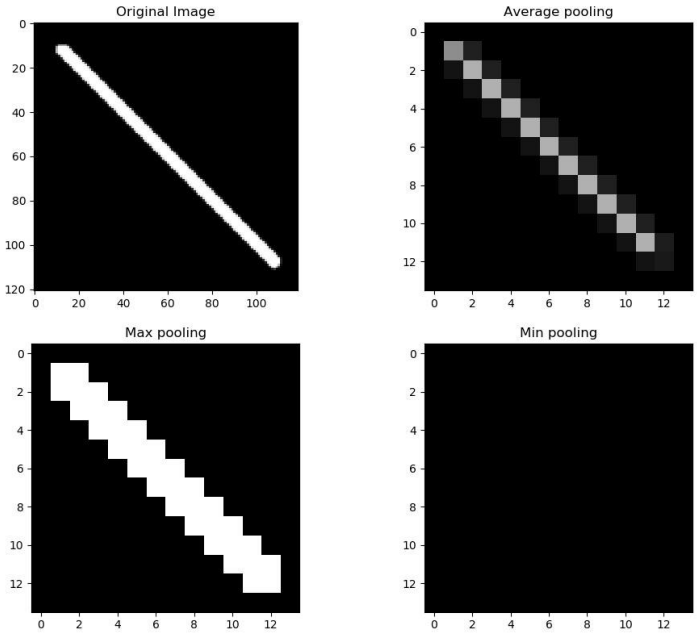
Slično konvolucijskom sloju, sloj združivanja (eng. *Pooling layer*) odgovoran je za smanjenje prostorne veličine (dimenzije) konvolvirane značajke (eng. *Convolved Feature*) . Primjenom te operacije smanjivanja dimenzija, smanjuje se računska snaga potrebna za obradu podataka. Nadalje, korisna je za izdvajanje dominantnih značajki koje nisu osjetljive na promjene kuta rotacije ili promjene pozicije, čime se održava proces učinkovitog treniranja modela.



Slika 3.22. Filter sažimanja

Postoje tri vrste sažimanja: maksimalno sažimanje (eng. *Max pooling*), minimalno sažimanje (eng. *Min pooling*) i prosječno sažimanje (eng. *Average pooling*). Maksimalno sažimanje daje maksimalnu vrijednost iz dijela slike koji pokriva kernel, minimalno sažimanje daje minimalnu vrijednost, dok prosječno sažimanje daje prosječnu. Iako svaki ima svoje prednosti te se primjenjuju ovisno o potrebama koje se postavljaju na sliku, maksimalno sažimanje također ima najbolje svojstvo prigušivanja šuma unutar slike. Princip rada filtera maksimalnog sažimanja prikazan je na slici 3.21.

Za potrebe ovog rada, kako se radi o prikazima bijelih simbola nad crnom pozadinom (slika 3.13.), idealan odabir metode sažimanja bio bi maksimalno sažimanje jer jedino se tom metodom ne gube podaci o slici, kao što je prikazano na slici 3.23.

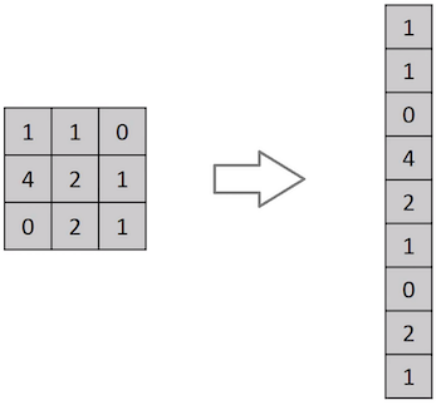


***Slika 3.23.* Prikaz *sažimanja* nad crnom pozadinom**

Kovolucijski sloj te sloj sažimanja zajedno tvore osnovni proces predobrade slika u oblik pogodniji za unos u umjetnu neuronsku mrežu. Nerijetko se te dvije operacije sekvencijalno izmjenjuju više puta, ovisno o potrebi za smanjenjem rezolucije slike te time postizanja kraćeg potrebnog vremena za obradu te iste slike kroz umjetnu neuronsku mrežu.

Ovisno o složenosti slike te zahtjevima nad njom, moguće je napraviti upravo suprotno – provesti konvolucijski filter te filter sažimanje i kernelom mapirati dobivene vrijednosti u zapis dimenzija većih od onih ulazne slike. Povećavajući broj slojeva konvolucijskog filtriranja te sažimanja, može se postići daleko detaljnije snimanje te prepoznavanje značajki na puno manjim razinama, ali uz cijenu više potrebnih računskih resursa.

Po završetku prethodno navedenih operacija, kovolucijska neuronska mreža konačni matrični zapis obrađene reducirane slike transformira u oblik pogodan za unos u ulazne neurone umjetne neuronske mreže, tj. vektor stupac, na način prikazan na slici 3.24.



Slika 3.24. Vektorski oblik zapisa obrađene slike

Ta operacija postiže se pozivom ugrađene funkcije unutar programskog paketa Python zvanom `softmax`.

Uzevši u obzir da su slike notnog zapisa koje su razmatrane u okviru ovog rada prethodno pripremljene za obradu prije unosa u konvolucijsku neuronsku mrežu, na način naveden u poglavlju 2., tj. da su savršeno binarne, u kombinacijama crne i bijele boje, matrica, odnosno vektor na slici 3.22., sadržavat će isključivo varijacije znamenaka '0' i '1'.

Za potrebe ovog rada razmatrana je prethodno definirana i razrađena konvolucijska neuronska mreža tipa VGGNet, usavršena 2014. godine.
VGG konvolucijska neuronska mreža sastoji se od 16 konvolucijskih slojeva te je vrlo privlačna zbog svoje uniformne strukture, prikazane na slici 3.25.

Slika 3.25. Struktura VGG neuronske mreže

Odabrana je iz razloga jer je trenutno globalno najkorišteniji izbor za izvlačenje značajki iz slika.

Konfiguracije težina za ovu mrežu javno su dostupne te je često bila korištena u mnogim aplikacijama i izazovima kao referenca za izvlačenje značajki. Ipak, treba uzeti u obzir da VGG neuronska mreža sadrži 138 milijuna parametara, čime je često teško baratati.
Prosječno vrijeme učenja (treniranja) ovakve mreže u 2014. godini, paralelno odrađivano na 4 grafičke procesorske jedinice bilo je između dva i tri tjedna.
Optimizirana za programski paket Python, unutar biblioteke Keras, VGG neuronska mreža poziva se vrlo jednostavno:

```
from keras.applications.vgg16 import VGG16
model = VGG16()
```

Svakoj, pa tako i ovoj, umjetnoj neuronskoj mreži potrebno je priložiti bazu podataka iz koje se moguće referencirati, te na temelju koje se bazira prepoznavanje svake nove sljedeće značajke.
Iako VGG neuronska mreža ne zahtjeva traženje težina za određene slučajeve za koje već postoje prethodno definirani skupovi podataka za učenje (eng. *dataset*), za potrebe ovog rada to bi bilo potrebno učiniti iz razloga objašnjenih u podpoglavlju 3.3.3.

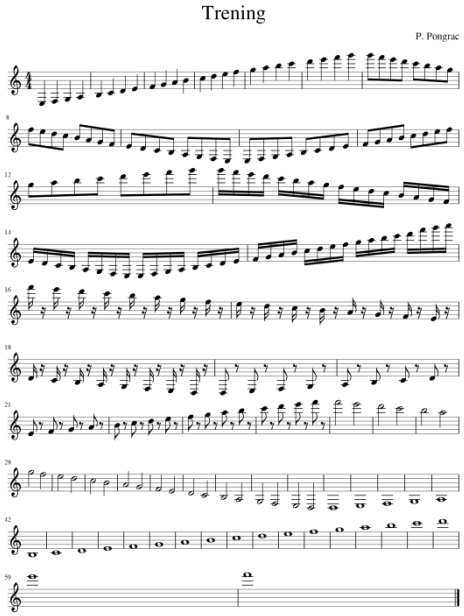
Po učitavanju novog skupa podataka za učenje neuronske mreže, ona je spremna za učenje, tj. definiranje težina mreže. Niže je prikazan isječak koda u kojem je prikazan postupak učenja VGG konvolucijske neuronske mreže iz lista koje sadrže skup očekivanih izlaza za učenje (`train_images`) te podatke o njihovim parametrima (`train_labels`).

```
modelnote = VGG16(weights='imagenet', include_top=False,
input_shape=(IMG_SIZE, IMG_SIZE, 3))
layerInputGender = modelnote.input
layerOutputGender = Dense(6, activation='softmax', name='output_note')
layerFlatten = Flatten()
layerOutputGender = layerOutputGender(layerFlatten(modelnote.output))
modelNOTE = Model(layerInputGender, layerOutputGender)

sgd = SGD(lr=LEARNING_RATE_NOTE, decay=1e-6, momentum=0.9, nesterov=True)
modelNOTE.compile(loss='categorical_crossentropy', optimizer=sgd)

modelNOTE.fit(x_train, z_train, batch_size=BATCH_SIZE_NOTE,
epochs=EPOCH_NOTE, validation_data = (x_val, z_val))
modelNOTE.save('{}NOTE.h5'.format(MODEL_NAME))
#scoreNOTE = modelNOTE.evaluate(x_test, z_test, batch_size=32)
#print(scoreNOTE)
```

3.3.3 IZRADA SKUPA PODATAKA ZA UČENJE NEURONSKE MREŽE

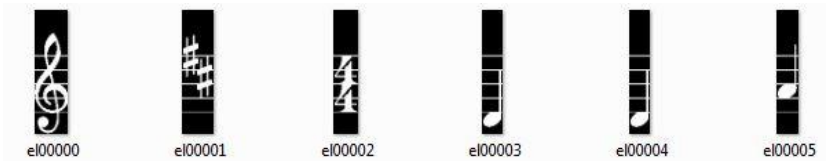
Za većinu klasičnih problema vizualnog prepoznavanja koje rješava neuronska mreža već postoje službeni skupovi podataka za procesiranje, pomoću kojih se dobivaju optimalne vrijednosti težina mreže za dobivanje što boljih rezultata u pogledu preciznijeg prepoznavanja odnosno minimiziranja greške. Kako ovaj rad obrađuje podosta specifičan problem, ne postoji službeni skup podataka za učenje mreže, stoga ga je potrebno izraditi.

Kreiran je novi notni zapis, jedan koji sadrži primjerak svakog simbola u svim mogućim pozicijama, prikazan na slici 3.26.



Slika 3.26. Notni zapis za učenje mreže

Izrada skupa podataka učenja započinje učitavanjem slike notnog zapisa „Trening“ u prethodno objašnjenu Python skriptu za segmentaciju simbola, unutar poglavlja 2. Kao izlazne vrijednosti dobivaju se izresci koji su prikazani slikom 3.27.



Slika 3.27. Izresci notnog zapisa za učenje mreže

Nakon dobivanja slikovnih ulaznih podataka potrebno je imenovati svaki pojedini izrezak simbola kako bi se dobio skup definiranih točnih vrijednosti kakav se očekuje na izlazu mreže, koji će dalje služiti kao referenca svakom novom učitanom notnom zapisu.

Taj postupak provediv je na više načina. Za potrebe ovog rada problemu se pristupilo na način da se kreira novi tekstni dokument u koji se sekvencijalno unose podaci o imenu (vrsti) pronađenog simbola te vremenu njegova trajanja, ukoliko je to vrsta simbola koja posjeduje tu informaciju. Taj se dokument unosi i obrađuje, te se konkretni zahtijevani podaci izvlače iz njega i unose u listu iz koje im neuronska mreža može pristupiti.

Dio tog dokumenta prikazan je na slici 3.28. :

```
el00000     X       gclef
el00001     X       dmaj
el00002     X       4/4
el00003     1/1     d1
el00004     1/1     d1
el00005     1/1     a1
el00006     1/1     a1
el00007     X       barline
el00008     1/1     h1
el00009     1/1     h1
el00010     2/1     a1
el00011     X       barline
el00012     1/1     g1
```

Slika 3.28. Parametri podataka skupa za učenje mreže

Prvi stupac označava naziv dokumenta unutar skupa podataka u pitanju, drugi vrijeme trajanja note, ukoliko je riječ o noti (ukoliko nije, stavljen je znak 'x'), te treći gdje su navedeni željeni nazivi simbola koji je prikazan dotičnom slikom.

Samo egzaktne očekivane vrijednosti nisu dovoljne za učenje neuronske mreže. Postoji mogućnost odstupanja ili grešaka prilikom obrade slike koje treba uzeti u obzir. Da bi se izbjegla situacija da mreža naiđe na ulazni podatak koji ne zna klasificirati jer odstupa za, primjerice, jedan piksel, svaki od podataka unutar skupa podataka za učenje neuronske mreže množi se više puta, svaki put sa različitim varijacijama pomaka unutar slike. Primjer rezultata te operacije prikazan je na slici 3.29 .

Slika 3.29. Varijacije podatka za učenje

Time se postiže više verzija jednog referentnog podatka, te se na taj način postiže veća vjerojatnost da mreža prepozna ulazni podatak ispravno.
Za postizanje ovog procesa, kako se radi o količini podataka (slika) reda 10^3, napravljena je kratka skripta čiji se potpuni kod nalazi u Prilogu 2.

4 ZAKLJUČAK

U opsegu ovog završnog rada dan je pregled osnova sustava za prepoznavanje nota iz notnog zapisa, od obrade i manipulacije slikama do prepoznavanja značajki. Naglasak je bio na prepoznavanju simbola unutar cjeline računalnim vidom, obrade ulazne slike te klasifikaciji pronađenih simbola konvolucijskim neuronskim mrežama.
Za razliku od klasičnih neuronskih mreža, konvolucijske neuronske mreže zahtijevaju velike količine računalnih resursa, a vremena učenja mreže postaju sve duža. Iz toga razloga potrebno je optimirati ulaze u neuronsku mrežu kako bi proces prepoznavanja simbola postao što brži i precizniji. Daljnja poboljšanja izuzev povećanja broja neurona i skrivenih slojeva, promjena veličina i količina filtera i koraka učenja, mogu biti implementirana u obliku predobrade podataka.. Zbog kompleksnosti i teškog razumijevanja problema koje umjetne neuronske mreže danas rješavaju, često prevladava mišljenje da je to područje rezervirano isključivo za pojedince sa programerskom pozadinom. Pojavom raznih novih programskih paketa te biblioteka, jednako kao i građenje zajednice ljudi sa sličnim afinitetima, široj javnosti se sve više približava korištenje algoritama neuronskih mreža, što dodatno doprinosi njihovom razumijevanju i razvoju, jednako kao i njihovoj popularizaciji. Time postaju sve korisniji alat za rješavanje problema, bilo u istraživačke, profesionalne ili privatne svrhe.
Iz ovog završnog rada može se zaključiti koliko je pristupačnost realizacije izrade vizijskog sustava za relativno neistraženu primjenu pristupačna.

5 LITERATURA

1. Marko Blašković*: Ne dirajte mi ravnicu*; *Nek' zvone tambure*; *Čardaš; Podmostec i Biškupec* (Autorske obrade narodnih pjesama)
2. Open Source Computer Vision –*Thresholding, Eroding and Dilating*, https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html
3. 6. B. Novaković, D. Majetić, M. Široki: Umjetne neuronske mreže, Fakultet strojarstva i brodogradnje, Sveučilište u Zagrebu, Zagreb, 2011.
4. S. Sasha: *Convolutional Neural Networks in Image Processing*, 2018.

Prilog 1

```
import cv2
import numpy as np
import os

file_name = "image_wb.jpg"                                  # image Name
img = cv2.imread(file_name)                                 # read

# def segmentacija
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)                # transform into gray img
th, thr = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
non_zero_elements = cv2.findNonZero(thr)                    # find the non zero pixels
non_zero_min_area = cv2.minAreaRect(non_zero_elements       # mark the area with a rectangle

(cx, cy), (w, h), ang = non_zero_min_area                   # find rotated matrix
M = cv2.getRotationMatrix2D((cx, cy), ang, 1.0)
rotated = cv2.warpAffine(thr, M, (img.shape[1], img.shape[0]))     # do the rotation

# expand to fit A4 format
bordered = cv2.copyMakeBorder(rotated, 0, int((w*1.414)-h), 0, 0, cv2.BORDER_CONSTANT,
value=[0, 0, 0])

hist = cv2.reduce(rotated, 1, cv2.REDUCE_AVG).reshape       # reduce matrix to a vector

import matplotlib.pyplot as plt

print(hist)
print(len(hist))
print(range(len(hist)))
plt.plot(range(len(hist)), hist, "r")
plt.xlabel('br retka')
plt.ylabel('hist val')
plt.show()

th = 2                                                      # change the threshold (empirical)
H, W = img.shape[:2]                                        # picture dimensions

upperBound = [y for y in range(H - 1) if hist[y] <= th < hist[y + 1]]     # upper bounds
lowerBound = [y for y in range(H - 1) if hist[y] > th >= hist[y + 1]]     # lower bounds

up_array = np.asarray(upperBound)                           # list to array conversion
up = (H - up_array)
low_array = np.asarray(lowerBound)
low = (H - low_array)

slices = []
for i in range(len(up_array)):                             # row slicing

        # expand the slice vertically
 if(low_array[i] + 1) + int(H/350) < H and up_array[i] - int(H / 70) > 0:
h_slice = bordered[up_array[i] - int(H / 70):(low_array[i] + 1) + int(H / 350), 0:W]
    # don't expand on the edges
 else:
 h_slice = bordered[up_array[i]:(low_array[i] + 1), 0:W]
    slices.append(h_slice)                                 # save all the slices in a list

# on standard A4 paper(21 cm height), 1 note line is 1 cm -> 1/21 ~= 0.04,
# ignore smaller slices
slices[:] = [s for s in slices if not len(s) < 0.04 * H]       # remove unwanted slices
```

```
valid_slices_pixel_mean = []                    # find the mean value of each slice
for s in slices:
    valid_slices_pixel_mean.append(np.mean(s))
mean = np.mean(valid_slices_pixel_mean)         # find the mean value of all slices

j = 0
for i in range(len(slices)):                    # save the valid slices
    # wanted slices have approximately the same mean of pixels,
    # ignore the unwanted lines(+- 15% of mean)
    if 1.30 * mean > valid_slices_pixel_mean[i] > 0.70 * mean:
        sliceName = "slice" + str(j) + ".jpg"               # slice naming
        path = "resources/" + file_name[:-4] + "/"          # directory for the slices
        try:                                    # create the dir if it doesn't exist
            os.makedirs(path)
        except FileExistsError:
            pass
        cv2.imwrite(path + sliceName, slices[i]  # save the slices in that directory
        j = j + 1                                           # name slices iteratively

def crop_image(img):

    h, w = img.shape[:2]  # image dimensions
    img_mean = np.mean(img)  # find the mean of the image

    for i in range(len(img[0])):  # go horizontally; len(img[0]) = no. of columns
        column_mean = 0  # calculate the mean of every column
                # start at the middle of a picture
        for j in range(int(len(img) / 2), int(3 * len(img) / 4), 1):
            # add the means of all the pixels in a column
                column_mean = column_mean + np.mean(img[j][i])
                # divide by the number of elements(rows) in column
        column_mean = column_mean / (len(img) / 2)
         if column_mean > img_mean:  # cut away the spaces before score lines
            img = img[0:h, i:len(img[0])]  # crop the image
            break  # break when done

    for i in range(len(img[0]) - 1, 0, -1):  # go backwards (end to 0, with step being -1)
        column_mean = 0  # calculate the mean of every column
        for j in range(len(img)):
                # add the means of all the pixels in a column
            column_mean = column_mean + np.mean(img[j][i])
                # divide by the number of elements(rows) in column

        column_mean = column_mean / len(img)
       if column_mean > img_mean:  # cut away the spaces after score lines
            img = img[0:h, 0:i]  # crop the image
            break  # break when done
    return img


def erode_dilate(img):
    kernel1 = np.ones((2, 2), np.uint8)  # kernel with dimensions 2x2, 1 if all = 1
    eroded_img = cv2.erode(img, kernel1, iterations=1)  # first erode
    dilated_img = cv2.dilate(eroded_img, None, iterations=1)  # then dilate
    return dilated_img
```

```
def find_histogram(dilated_img):
    min_mean = 10000  # result of the minimum mean of a vertical line

    for i in range(len(dilated_img[0])):  # go horizontally; len(img[0]) = no. of columns
        column_mean = 0  # calculate the mean of every column
        for j in range(len(dilated_img)):
                # add the means of all the pixels in a column
            column_mean = column_mean + np.mean(dilated_img[j][i])
                # divide by the number of elements(rows) in column

        column_mean = column_mean / len(dilated_img)
         if column_mean < min_mean:  # if it is smaller than the minimum...
            min_mean = column_mean  # ... make it a new minimum

    hist = [0]  # histogram of the picture
    for i in range(len(dilated_img[0])):  # go horizontally; len(img[0]) = no. of columns
        column_mean = 0  # calculate the mean of every column
        for j in range(len(dilated_img)):
                # add means of all pixels in a column
           column_mean = column_mean + np.mean(dilated_img[j][i])
            column_mean = column_mean / len(dilated_img)  # divide by the nr of (rows) in column

        if column_mean > 1.1 * min_mean:  # put 1 in a histogram if a line is not empty
            hist.append(1)
        else:  # put 0 in a histogram if a line is empty
            hist.append(0)
    return hist


def get_element_coordinates(dilated_img, hist):
    x_cut_start = []  # coordinates for the left side of the element
    x_cut_end = []  # coordinates for the right side of the element
    for i in range(len(hist)):  # find the edges (rising and falling edge)
        if i > 0:
            if hist[i - 1] == 0 and hist[i] == 1:  # find the starting x coordinate
x_cut_start.append(i - 1)
            elif hist[i - 1] == 1 and hist[i] == 0:  # find the starting x coordinate
                x_cut_end.append(i - 1)
    x_cut_end.append(len(dilated_img[0] - 1))  # last coordinate is the end of the picture
    return x_cut_start, x_cut_end


def get_elements_from_image(path, x_cut_start, x_cut_end, img, element_number):
    large_elements = []
    large_elements_index = []
    h, w = img.shape[:2]  # image dimensions
    for i in range(len(x_cut_start)):
        if x_cut_start[i] - 3 > 0 and x_cut_end[i] + 3 < w - 1:
                # cut the element from the image
            element = img[0:h, x_cut_start[i] - 3:x_cut_end[i] + 3]
        else:
            element = img[0:h, x_cut_start[i]:x_cut_end[i]]
                # generate the element name
        element_name = "el" + str(element_number).zfill(5) + ".jpg"
      if 5 < len(element[0]) < 40:  # if the element is valid
            try:  # if the element is not null
                # save the elements in the directory
                cv2.imwrite(path + element_name, element)
```

```
except:  # else, skip that element
                pass

        elif len(element[0]) > 40:
            large_elements.append(element)
            large_elements_index.append(element_number)
        element_number = element_number + 1  # increase the indexing number

    index = 0
    for el in large_elements:
        el = crop_image(el)
        changed_el = erode_dilate(el)
        hist = find_histogram(changed_el)
        x_cut_start, x_cut_end = get_element_coordinates(changed_el, hist)

        h, w = el.shape[:2]  # image dimensions
        for i in range(len(x_cut_start)):
            if x_cut_start[i] - 3 > 0 and x_cut_end[i] + 3 < w - 1:
                 # cut the element from the image
                element = el[0:h, x_cut_start[i] - 3:x_cut_end[i] + 3]
             else:
                element = el[0:h, x_cut_start[i]:x_cut_end[i]]
            element_name = "el" + str(large_elements_index[index]).zfill(5) +\
                           "part" + str(i) + ".jpg"         # generate the element name
            if 4 < len(element[0]):
                try:  # if the element is not null
                         # save the elements in the directory

                    cv2.imwrite(path + element_name, element)
                except:  # else, skip that element
                    pass
        index = index + 1

    return element_number



path = ".\\resources\\image_wb\\"                                    # image location

images = []                                          # get all the image names in the directory
for r, d, f in os.walk(path):                               # r=root, d=directories, f = files
    for file in f:
        if '.jpg' in file:
            if 'el' not in file:
                images.append(file)

elementNumber = 0                                    # indexing number for extracted elements
for image_name in images:                                                # process every slice
    img = cv2.imread(path + image_name)                                  # read the image
    img = cv2.bitwise_not(img)                                           # convert colors
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)                         # transform into gray img
    th, thr = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)
    img = crop_image(thr)                             # crop the left and right side of the image
    changed_img = erode_dilate(img)                   # erode and dilate to filter out noise
    hist = find_histogram(changed_img)                # find the histogram of symbol occurrences
    # find the start and end coordinates of the symbols
    x_cut_start, x_cut_end = get_element_coordinates(changed_img, hist)
    # get the updated element number and cut out all the symbols into separate images
```

```
    elementNumber = get_elements_from_image(path, x_cut_start, x_cut_end, img, elementNumber)

for fileName in os.listdir(path):              # delete redundant images from the previous step
    if fileName.startswith("slice"):
        os.remove(os.path.join(path, fileName))
```

Prilog 2

```
import numpy as np
import cv2
import os

path = ".\\resources\\img02\\"    # dir

images = []
for r, d, f in os.walk(path):     # walk through dir
    for file in f:
        if '.jpg' in file:
            if 'el' in file:
                images.append(file)
                print(images)

for i in images:
    for k in range(0, 6):
        img = cv2.imread(path + i, 0)
        rows, cols = img.shape
        M = np.float32([[1, 0, k - 2], [0, 1, 0]])
        dst = cv2.warpAffine(img, M, (cols, rows))  # ofset
        element_name = "warp" + str(i[:-4]) + str(k) + ".jpg"
        cv2.imwrite("resources/img02/warp/" + element_name, dst)
```