Prepoznavanje karaktera sa registarskih tablica

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Filip Jovašević, Stefan Lazović, Luka Milošević
filip.jovasevic@gmail.com, stefanlazovic1010@gmail.com,
lukamilosevic11@gmail.com

20. maj 2019

Sažetak

U radu će biti prikazana primena neuronskih mreža i fazi logike u prepoznavanju karaktera sa tablica. Objašnjeni su algoritmi koje smo podelili na četiri dela, a zajedno čine celinu rada. Kodovi su pisani u programskom jeziku Python. Neke od biblioteka koje su korišćene su Numpy, Keras, CV2. U radu će biti reči i o vremenu izvršavanja programa, kao i poređenje različitih algoritama i njihove preciznosti.

Sadržaj

1 Uvod i opis algoritma

Kao što smo već rekli rad se bavi prepoznavanjem karaktera sa tablice koja je data na slici. Skup podataka koji koristimo sastoji se od tablica koje su izgenerisane automatski, to jest ne predstavljaju prave tablice. Na slikama se nalaze isključivo tablice i ništa drugo.

Proces se sastoji iz četiri dela. Prvo pretražimo ceo skup podataka i na osnovu te pretrage izdvojimo sve karaktere koji se mogu javiti na nekoj tablici. Nakon toga se vrši priprema podataka. Slike iz skupa podataka učitavamo pomoću funkcije „imread“ koja slike prebacuje u grayscale model. Zatim se vrši fazifikacija koja popravlja kvalitet slike podešavanjem kontrasta. Treniranje modela smo vršili korišćenjem konvolutivnih i rekurentnih neuronskih mreža. Na kraju vrši se testiranje istreniranog modela. O svim ovim delovima više reči će biti u nastavku.

2 Prepoznavanje karaktera sa tablice

Prva stvar koju radimo jeste izvlačenje svih slova i brojeva koji se pojavljuju u našoj bazi slika[5], kao i upoređivanje da li se u sva tri skupa (test, train, val) nalaze svi karakteri koje smo prethodno našli. U našoj bazi se nalaze dva foldera, jedan za trening podatke koji sadrži i validacione podatke i jedan za test podatke. U okviru svakog od tih foldera postoji folder sa slikama i folder u kojem se nalaze fajlovi u json formatu koji opisuju slike. Ovaj algoritam se zasniva na tome da u zavisnosti od toga koji tag prosledimo (može biti „val“, „train“, „test“) vršimo izvlačenje svih slova i brojeva koji se javljaju na tablicama. Na kraju, izvučene karaktere stavljamo u skup i dajemo njihov prikaz. U nastavku se može videti izlaz opisane funkcije, kao i informacije koje šaljemo mreži da bi joj pomogle u treniranju modela.

```
Characters in train, val and test images do match!

All characters that appear in plates:
    0 1 2 3 4 5 6 7 8 9 A B C E H K M O P T X Y

Informations that we feed into NN:

1. Input image of vechicle registration.
2. Characters from above (encoded respectively):
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
3. Input length is
 (image width / 4 - downsample_factor) => 128 / 4 - 2 = 30.
4. Plate length which is 8.
```

3 Generisanje i parsiranje ulaznih podataka

Svaku sliku obrađujemo na način tako što pravimo uređene parove (putanja do slike, karakteri od kojih se slika sastoji). Ovaj proces vršimo samo za slike koje zadovoljavaju uslov, a to je da se sastoji samo od mogućih karaktera.

3.1 Tehnika za poboljšavanje kvaliteta crno-belih slika tablica pomoću fazi logike i INT algoritma

Svaka slika sadrži nekoliko bitnih svojstava koja su važna za sam doživljaj slike. Jedno od tih svojstava jeste kontrast. Kontrast predstavlja odnos između svetlih i tamnih delova u slici[7]. Pomoću fazi logike izoštrićemo brojeve i slova na tablici.
INT algoritam, ili algoritam intenziteta, je algoritam koji spada u grupu fazi algoritama i koristi se za popravljanje slike isticanjem kontrasta. Glavna odlika ovog algoritma je da koristi dve karakteristične funkcije i da se te dve karakteristične funkcije primenjuju direktno na svaki pojedinačni piksel u slici[7].
Vrednosti ulazne slike X, dimenzija M x N, fazifikuju se primenom funkcije μ:

$$\mu_{ij} = \frac{x_{ij} - x_{min}}{x_{max} - x_{min}},$$

gde je $0 \leq i < M, 0 \leq j < N$, x_{max} maksimalna, a x_{min} minimalna vrednost piksela na slici X i x_{ij} vrednost piksela na mestu ij slike X. Za 8-bitnu sliku, vrednost piksela može biti u intervalu $[0, 2^8 - 1]$.
μ_{ij} predstavlja stepen osvetljenosti koju ima piksel ij na ulaznoj slici X. U fazi terminologiji, μ je funkcija pripadnosti i omogućava prevođenje nefazi ulaza (slika X) u njen odgovarajući fazi oblik. Primetimo da je $0 \leq \mu_{ij} \leq 1$.

Nakon što smo napravili fazi skup prelazimo na fazu modifikacije. Tu se primenjuje druga karakteristična funkcija koja će poboljšati kontrast. Ideja je da oni pikseli čija karakteristična funkcija μ ima vrednost manju od 0.5 „sabijemo“ bliže nuli, a da one veće od 0,5 „sabijemo“ bliže jedinici, dakle da ih potamnimo i posvetlimo respektivno. Za rešenje tog problema koristimo sledeću karakterističnu funkciju [7]:

$$\bar{\mu}_{ij}(\mu_{ij}) = \begin{cases} 2 \cdot \mu_{ij}^2 & 0 \leq \mu_{ij} \leq \mu_c \\ 1 - (2 \cdot (1 - \mu_{ij})^2) & \mu_c < \mu_{ij} \leq 1 \end{cases} \quad (1)$$

Tačka μ_c je granična vrednost koja se može menjati. Najčešće uzimana vrednost za nju je 0.5 (takođe korišćenja i u našoj implementaciji).

Praktično, nakon primene INT operatora, vrednost piksela sa funkcijom pripadnosti manjom od 0.5 će se smanjiti. Oni pikseli sa većom vrednošću funkcije pripadnosti od 0.5 imaće veću vrednost.
Sada je neophodno vratiti se iz fazi domena u domen slike. Nakon modifikacije funkcije pripadnosti, dobijene vrednosti treba defazifikovati i na taj način dobiti nove nijanse sive, izlaznu sliku.
Izlazna slika Y se dobija primenom inverzne funkcije μ^{-1}:

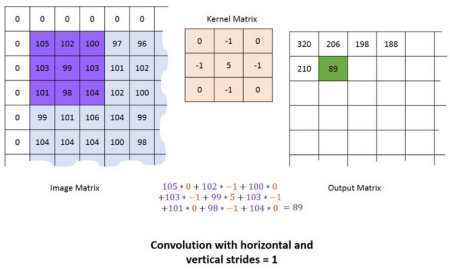
$$y_{ij} = \mu^{-1}(\bar{\mu}_{ij}) = x_{min} + \bar{\mu} \cdot (x_{max} - x_{min}).$$

Slika 1: Pre

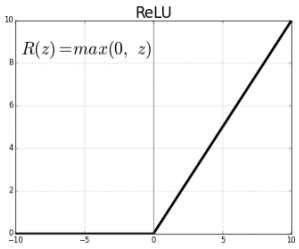
Slika 2: Posle

4 Treniranje modela pomoću neuronskih mreža

Treniranje modela vršimo korišćenjem neuronskih mreža. Prvo koristimo konvolutivnu neuronsku mrežu [3]. Ovom neuronskom mrežom vrši se obrada slika, ali se i usložnjava izlaz, koji je neophodno ponovo uprostiti. Broj neurona po sloju ove mreže je 16, kernel size je (3,3), dok je za funkciju aktivacije korišćena ReLU funkcija („ispravljena linearna“ funkcija aktivacije).

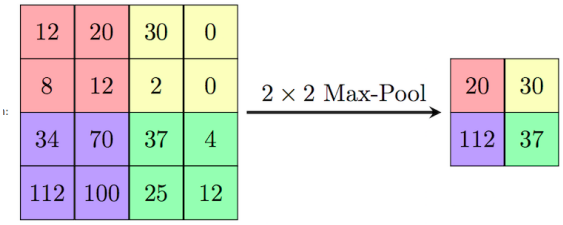


Slika 3: Rad konvolutivne mreže.



Slika 4: Funkcija aktivacije.

Za uprošćavanje koristimo MaxPooling u kojem smo pool size postavili na (2, 2)[4].

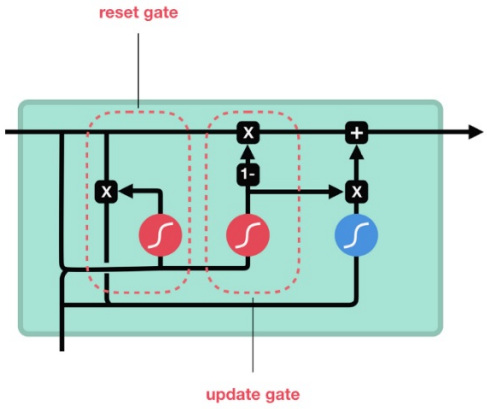


Slika 5: MaxPooling

Primer možete videti ispod.

```
inner = Conv2D(conv_filters, kernel_size, padding='same',
        activation=act, kernel_initializer='he_normal',
        name='conv1')(input_data)
inner = MaxPooling2D(pool_size=(pool_size, pool_size),
        name='max1')(inner)
inner = Conv2D(conv_filters, kernel_size, padding='same',
        activation=act, kernel_initializer='he_normal',
        name='conv2')(inner)
inner = MaxPooling2D(pool_size=(pool_size, pool_size),
        name='max2')(inner)
```

GRU predstavlja skraćenicu od „Gated Recurrent Units“ i ona je jedan od načina za rešavanje problema kratkotrajnog pamćenja kod rekurentnih neuronskih mreža. Za to je zaslužan njen unutrašnji mehanizam „gejtova“ koji pomaže da mreža razluči bitne informacije od nebitnih i tako pomogne u budućim predikcijama.



Slika 6: shema rekurentne mreže GRU

Na shemi možete videti dva gejta koja se nalaze u GRU mehanizmu. Update gejt pomaže u biranju informacija koje će odbaciti iz mreže a koje će zadržati. Ovaj gejt zna šta da uradi sa informacijama na osnovu sigmoidnih rezultata koji su izvršeni nad istom. Ako su rezultati za neku informaciju bliže 0 - znači da treba odbaciti tu informaciju, u suprotnom da je treba zadržati. Nekada se u kombinaciji sa sigmoidnom funkcijom koristi i tangens hiperbolički.
Reset gejt je fokusiran samo na informacije koje treba da zaboravi, radi po sličnom principu kao Update gejt.[1]

Da bismo obradili karaktere koristili smo rekurentne neuronske mreže. GRU je neuronska mreža koju smo koristili jer smo videli na sajtu Kerasove dokumentacije da daje iste, ako ne i bolje rezultate od LSTM-a. Da bismo prilagodili ulaz za rekurentnu neuronsku mrežu, neophodno je bilo da izvršimo Reshape nad izlazom iz konvolutivne neuronske mreže, kaoi da definišemo potpuno povezan sloj. Pošto smo to uradili, primenili smo GRU gde je veličina mreže 512, a inicijalizacija vrednosti se vrši hi-normalnom raspodelom:

```
gru_1 = GRU(rnn_size, return_sequences=True,
    kernel_initializer='he_normal', name='gru1')(inner)
gru_1b = GRU(rnn_size, return_sequences=True,
    go_backwards=True, kernel_initializer='he_normal',
    name='gru1_b')(inner)
gru1_merged = add([gru_1, gru_1b])
gru_2 = GRU(rnn_size, return_sequences=True,
 kernel_initializer='he_normal', name='gru2')(gru1_merged)
gru_2b = GRU(rnn_size, return_sequences=True,
    go_backwards=True, kernel_initializer='he_normal',
    name='gru2_b')(gru1_merged)
```

U nastavku se nalazi izgled našeg modela:

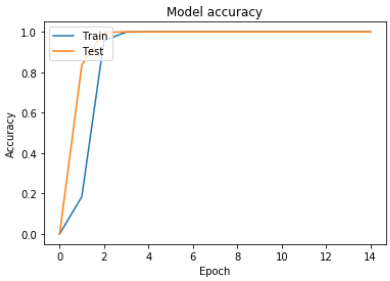
```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================
the_input (InputLayer)       (None, 128, 64, 1)        0
----------------------------------------------------------------------------------
conv1 (Conv2D)               (None, 128, 64, 16)       160         the_input[0][0]
----------------------------------------------------------------------------------
max1 (MaxPooling2D)          (None, 64, 32, 16)        0           conv1[0][0]
----------------------------------------------------------------------------------
conv2 (Conv2D)               (None, 64, 32, 16)        2320        max1[0][0]
----------------------------------------------------------------------------------
max2 (MaxPooling2D)          (None, 32, 16, 16)        0           conv2[0][0]
----------------------------------------------------------------------------------
reshape (Reshape)            (None, 32, 256)           0           max2[0][0]
----------------------------------------------------------------------------------
dense1 (Dense)               (None, 32, 32)            8224        reshape[0][0]
----------------------------------------------------------------------------------
gru1 (GRU)                   (None, 32, 512)           837120      dense1[0][0]
----------------------------------------------------------------------------------
gru1_b (GRU)                 (None, 32, 512)           837120      dense1[0][0]
----------------------------------------------------------------------------------
add_1 (Add)                  (None, 32, 512)           0           gru1[0][0]
                                                                   gru1_b[0][0]
----------------------------------------------------------------------------------
gru2 (GRU)                   (None, 32, 512)           1574400     add_1[0][0]
----------------------------------------------------------------------------------
gru2_b (GRU)                 (None, 32, 512)           1574400     add_1[0][0]
----------------------------------------------------------------------------------
concatenate_1 (Concatenate)  (None, 32, 1024)          0           gru2[0][0]
                                                                   gru2_b[0][0]
----------------------------------------------------------------------------------
dense2 (Dense)               (None, 32, 23)            23575     concatenate_1[0][0]
----------------------------------------------------------------------------------
softmax (Activation)         (None, 32, 23)            0           dense2[0][0]
==================================================================================
```

Parametri učenja koje smo koristili su različiti. Neki od onih koje smo koristili bili su SGD(stohastički gradijentni spust), RMSprop, Nadam i dobijene rezultate možete videti na sledećim slikama.

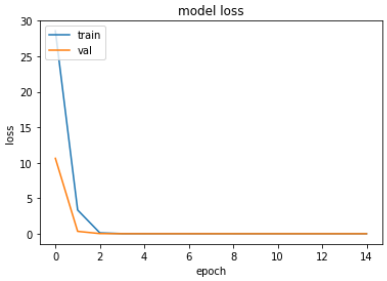
Prvo smo pokušali da treniramo model sa više epoha a koristeći manje slika po epohi uz pomoć SGD optimizer-a da bismo videli da li model dobro uči iz ovog skupa podataka, kao i koje mu je vreme potrebno da bi dao neke rezultate.
Zatim smo to prikazali i grafički.

Primeri za SGD:

```
Epoch 1/15
    109/108 [==============================] - 912s 8s/step
    - loss: 28.5811 - acc: 0.0000e+00
    - val_loss: 10.6289 - val_acc: 0.0000e+00
Epoch 2/15
    109/108 [==============================] - 902s 8s/step
    - loss: 3.3413 - acc: 0.1843
    - val_loss: 0.3364 - val_acc: 0.8359
Epoch 3/15
    109/108 [==============================] - 911s 8s/step
    - loss: 0.1255 - acc: 0.9573
    - val_loss: 0.0224 - val_acc: 0.9980
Epoch 4/15
    109/108 [==============================] - 904s 8s/step
    - loss: 0.0145 - acc: 0.9991
    - val_loss: 0.0078 - val_acc: 1.0000
Epoch 5/15
    109/108 [==============================] - 901s 8s/step
    - loss: 0.0065 - acc: 1.0000
    - val_loss: 0.0053 - val_acc: 1.0000
Epoch 6/15
    109/108 [==============================] - 901s 8s/step
    - loss: 0.0047 - acc: 1.0000
    - val_loss: 0.0041 - val_acc: 1.0000
Epoch 7/15
    109/108 [==============================] - 899s 8s/step
    - loss: 0.0037 - acc: 1.0000
    - val_loss: 0.0033 - val_acc: 1.0000
Epoch 8/15
    109/108 [==============================] - 901s 8s/step
    - loss: 0.0030 - acc: 1.0000
    - val_loss: 0.0028 - val_acc: 1.0000
Epoch 9/15
    109/108 [==============================] - 900s 8s/step
    - loss: 0.0026 - acc: 1.0000
    - val_loss: 0.0024 - val_acc: 1.0000
Epoch 10/15
    109/108 [==============================] - 896s 8s/step
    - loss: 0.0022 - acc: 1.0000
    - val_loss: 0.0021 - val_acc: 1.0000
Epoch 11/15
    109/108 [==============================] - 899s 8s/step
    - loss: 0.0020 - acc: 1.0000
    - val_loss: 0.0019 - val_acc: 1.0000
Epoch 12/15
    109/108 [==============================] - 908s 8s/step
    - loss: 0.0018 - acc: 1.0000
    - val_loss: 0.0017 - val_acc: 1.0000
Epoch 13/15
    109/108 [==============================] - 902s 8s/step
    - loss: 0.0016 - acc: 1.0000
    - val_loss: 0.0015 - val_acc: 1.0000
Epoch 14/15
    109/108 [==============================] - 899s 8s/step
    - loss: 0.0015 - acc: 1.0000
    - val_loss: 0.0014 - val_acc: 1.0000
Epoch 15/15
    109/108 [==============================] - 898s 8s/step
    - loss: 0.0014 - acc: 1.0000
    - val_loss: 0.0013 - val_acc: 1.0000
```



Slika 7: Preciznost



Slika 8: Gubici

Iz prethodnog primera možete videti da već posle četvrte epohe imamo maksimalnu preciznost, pa smo zaključili da možemo smanjiti broj epoha, a samim ti i vreme izvršavanja.

```
Epoch 1/2
 10799/10799 [========================] - 1912s 177ms/step
 - loss: 0.9540 - acc: 0.8994
 - val_loss: 3.3165e-04 - val_acc: 1.0000
Epoch 2/2
 10799/10799 [========================] - 1912s 177ms/step
 - loss: 2.5733e-04 - acc: 1.0000
 - val_loss: 2.1359e-04 - val_acc: 1.0000
```

Videvši da vec nakon 2 epohe, ukoliko u oviru svake od njih koristimo ceo skup podataka, dobijamo maksimalnu preciznost na trening i val skupu i to čak za manje vremena, nastavili smo sa ovom strategijom.

Primer za RMSprop:

```
Epoch 1/2
 10799/10799 [========================] - 1886s 175ms/step
 - loss: 0.9570 - acc: 0.9472
 - val_loss: 6.5268e-05 - val_acc: 1.0000
Epoch 2/2
 10799/10799 [========================] - 1885s 175ms/step
 - loss: 6.5221e-05 - acc: 1.0000
 - val_loss: 6.5190e-05 - val_acc: 1.0000
```

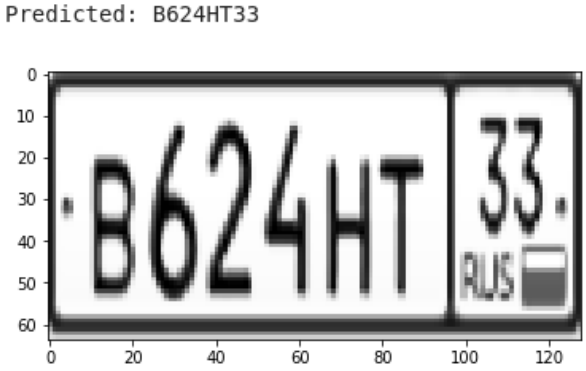
Primer za NADAM:

```
Epoch 1/2
 10799/10799 [========================] - 1907s 177ms/step
 - loss: 0.4874 - acc: 0.9796
 - val_loss: 7.1639e-05 - val_acc: 1.0000
Epoch 2/2
 10799/10799 [========================] - 1858s 172ms/step
 - loss: 6.6347e-05 - acc: 1.0000
 - val_loss: 6.5141e-05 - val_acc: 1.0000
```

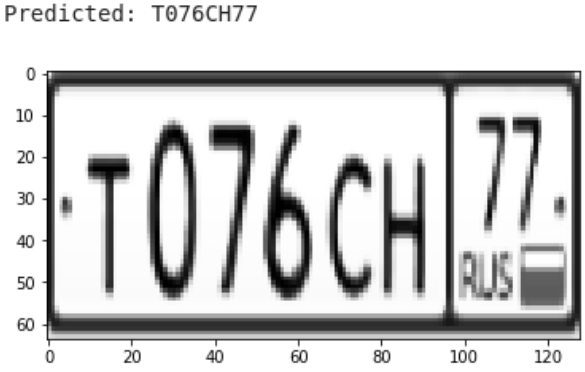
Iz gorenavedenih primera možemo zaključiti da je najbolje rešenje dobijeno korišćenjem Nadam metode učenja. Preciznost dobijena u prvoj epohi je 97 procenata, dok već u drugoj epohi dobijamo preciznosti od 100 procenata. I druge dve metode učenja u drugoj metodi imaju preciznost 100 procenata, ali su preciznosti u prvoj nešto manje.

5 Test podaci

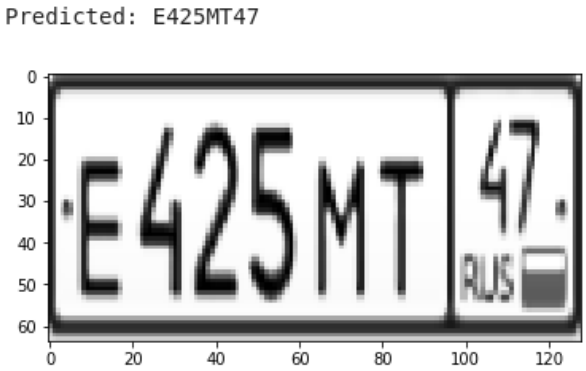
Na kraju, želimo da proverimo kroz nekoliko primera da li su tačne predviđene vrednosti u odnosu na sliku sa ulaza. Rezultate tih primera možete videti u nastavku.



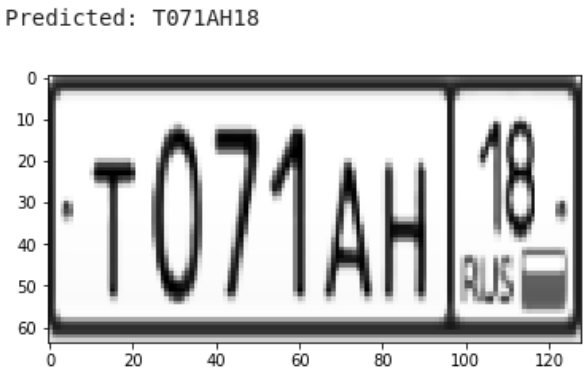
Slika 9: Primer 1



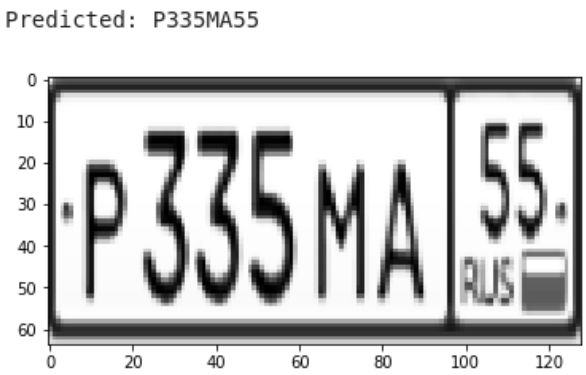
Slika 10: Primer 2



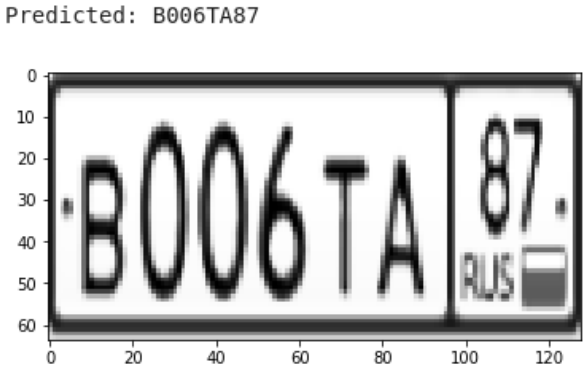
Slika 11: Primer 3



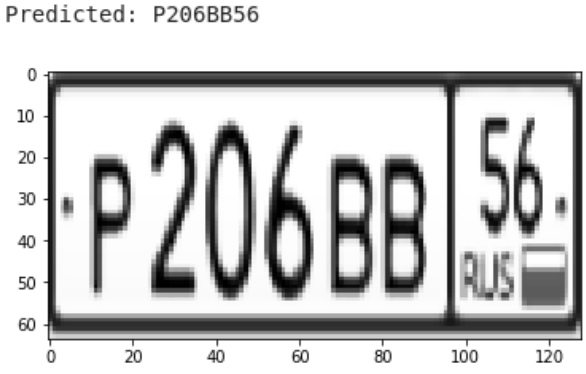
Slika 12: Primer 4



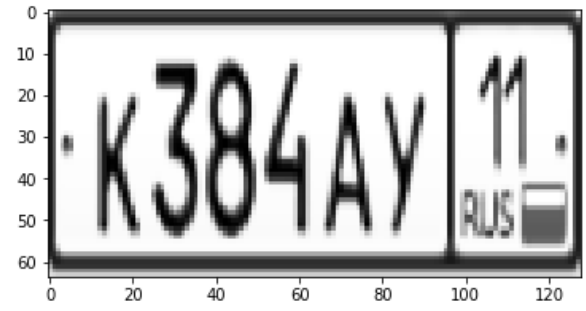
Slika 13: Primer 5



Slika 14: Primer 6



Slika 15: Primer 7



Slika 16: Primer 8

6 Zaključak

U ovom radu opisan je postupak prepoznavanja karaktera sa registarskih tablica. Postoji više načina za rešavanje problema detekcije karaktera sa registarskih tablica, program koji je napisan i čiji su rezultati predstavljeni u ovom radu, daje jedno od mogućih rešenja za ovaj problem. U radu je takođe prikazana optimizacija slike tablica korišćenjem fazi logike, koja je imala ulogu izoštravanja karaktera na tablici radi lakšeg prepoznavanja istih. Prikazane su različite tehnike optimizacije pri traženju rešenja. Jedan od najbitnijih delova rada je sam skup podataka koji sadrži veliki broj tablica i zahvaljujući istom preciznost na test skupu je jako velika i prepoznavanje karaktera jako dobro radi. Parametri kao što su broj epoha, grupe slike pomoću kojih se uči tokom svake epohe ili tehnika optimizacije pomažu da se pronađe najkvalitetniji i najprecizniji model za što kraće vreme.

Literatura

[1] Illustrated Guide to LSTM's and GRU's: A step by step explanation. on-line at: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21/.

[2] Image ocr. on-line at: https://keras.io/examples/image_ocr/.

[3] What is kernel size, 2017. on-line at: https://stats.stackexchange.com/questions/296679/what-does-kernel-size-mean.

[4] Max-pooling / Pooling, 2018. on-line at: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling.

[5] ANPR OCR dataset, 2019. on-line at: https://supervise.ly/explore/projects/anpr-ocr-21232/datasets.

[6] Model visualization, 2019. on-line at: https://keras.io/visualization/.

[7] Nebojša Perić. Neke primene teorije fazi skupova i fazi logike u procesiranju slika, 2014. on-line at: http://www.racunarstvo.matf.bg.ac.rs/MasterRadovi/2014_01_29_Nebojsa_Peric/rad.pdf.