

VILNIAUS UNIVERSITETAS  
INFORMATIKOS INSTITUTAS  
PROGRAMŲ SISTEMŲ KATEDRA

**Automobilių numerių atpažinimas naudojant Tesseract  
LSTM rekurentinį neuroninį tinklą**

Car Number Plate Recognition Using Tesseract LSTM Recurrent Neural  
Network

Bakalauro baigiamasis darbas

Atliko: Emilio Ruzveltas (parašas)

Darbo vadovas: dr. Vytautas Valaitis (parašas)

Recenzentas: lekt. Tomas Smagurauskas (parašas)

Vilnius  
2019

## **Santrauka**

Darbe analizuojami dviejų tipų neuroniniai tinklai, kurių pagalba galima iš nuotraukos atpažinti automobilio numerius.

Šio darbo tikslas – sukurti programą, kuri gebėtų atpažinti lietuviškus automobilio numerius paveikslėlyje. Šiam tikslui atlikti buvo atskirtos dvi dalys. Pirmoje dalyje sugeneruojama, kuo našesnių į realaus pasaulio sąlygas, paveikslėlių, kurie panaudoti apmokyti konvoliucinė neuroninė tinklą. Vėliau tinklas apmokomas su sugeneruotais duomenimis. Galiausiai parašoma programa, kuri randa automobilio numero rėmelio koordinates ir išsaugo iškirptą paveikslėlį. Antroje dalyje sugeneravus atsitiktinių automobilių numerių simbolių, buvo apmokyta Tesseract LSTM rekurentinis neuroninis tinklas. Vėliau sukurta programa, kuri pasinaudojus gautu neuroniniu tinklu, atpažista simbolius esančius automobilio numeryje. Darbe apžvelgiami būdai kaip reikia generuoti mokymo duomenis. Kaip modifikuoti paveikslėlius, norint, kad neuroninio tinklo mokymo metu būtų pasiekti kuo geresni rezultatai.

Raktiniai žodžiai: konvoliucinis neuroninis tinklas, rekurentinis neuroninis tinklas, Tesseract, LSTM, automobilių numerių atpažinimas.

## **Summary**

The paper analyzes two types of neural network, which help to recognize car number plates from image.

Aim of this work is to create a program, which can detect and recognize car number plate from image. There are two separated parts in this paper. In the first part, there were generated images with number plates in it which are close to real world conditions and fed to convolutional neural network to train. Then the program was developed to detect number plate borders using trained neural network and save cropped image. In the second part, there were generated random sequences of car number plates and then fed to the Tesseract LSTM recurrent neural network. After that, the program was developed, which recognize car number plate symbols. This paper analyzes ways of generating different training data, how to modify images, to better train neural networks.

Key words: convolutional neural network, recurrent neural network, Tesseract, Long short-term memory, car number plate recognition.

## TURINYS

IVADAS .....	5
1. Duomenų generavimas .....	8
1.1. Rémelio atpažinimui skirtų duomenų generavimas.....	8
1.1.1. Pirminis duomenų generavimo variantas.....	8
1.1.1.1. Generavimo parametrai .....	8
1.1.1.2. Numerio generavimas .....	10
1.1.1.3. Transformacija .....	10
1.1.1.4. Kompozicija .....	12
1.1.1.5. Triukšmo pridėjimas .....	13
1.1.2. Numerio atpažinimui skirtų duomenų generavimas .....	13
1.1.2.1. Idėja .....	13
1.1.2.2. Simbolių rinkimas.....	14
1.1.2.3. Generavimo algoritmas .....	14
2. Neuroninių tinklų architektūra .....	15
2.1. Numerio rémelio atpažinimui skirtas neuroninis tinklas .....	15
2.1.1. Modelis .....	15
2.2. Numerio simbolių atpažinimui skirtas neuroninis tinklas .....	18
2.2.1. Bendrai apie LSTM .....	18
2.2.1.1. Rekurentinis neuroninis tinklas .....	20
2.2.1.2. Integracija su Tesseract .....	20
2.2.1.3. Sisteminiai reikalavimai .....	20
2.2.1.4. Igyvendinimo pagrindai.....	21
2.2.1.5. Naujo tinklo lygio pridėjimas .....	21
2.2.1.6. VGSL specifikacijos .....	21
2.2.1.7. Vidinių tinklo lygių sintaksė .....	23
2.2.1.8. Kintamo dydžio įvestis ir apibendrinantis LSTM lygis .....	25
2.2.1.9. Modelis .....	26
3. Neuroninių tinklų apmokymas .....	27
3.1. Konvoliucinio neuroninio tinklo mokymas .....	27
3.2. LSTM rekurentinio neuroninio tinklo mokymas .....	28
3.2.1. Atpažinimo kokybės gerinimas .....	29
3.2.1.1. Paveikslėlio apdorojimas .....	29
3.2.1.2. Puslapių skirstymo metodas .....	34
3.2.1.3. Žodynai, žodžių sąrašai, šablonai .....	35
3.2.2. Rinkmenų pasiruošimas .....	35
3.2.3. Modelio apmokymas .....	36
3.2.4. Mokymo statistika .....	37
4. Vaizdo atpažinimas .....	38
4.1. Numerio rémelio atpažinimas .....	38
4.2. Numerio simbolių atpažinimas .....	39
REZULTATAI .....	42
IŠVADOS .....	42
SĀVOKŲ APIBRĖŽIMAI .....	43
SANTRUMPOS .....	43
Priedas Nr.1	
Priedas Nr.2	

Priedas Nr.3

Priedas Nr.4

## **Ivadas**

Pagrindinis automobilių atpažinimo sistemų tikslas yra automatizuoti vaizdo stebėjimą ir apdorojimą bei automatiškai surinkti įvairią informaciją apie transporto priemonę. Automobilių numerių atpažinimo sistemos remiasi tuo, kad kiekviena transporto priemonė turi unikalų identifikacinį kodą, kuris leidžia vienareikšmiškai nustatyti transporto priemonės savininką. Techniškai automobilių numerių atpažinimas yra paveikslėlių apdorojimo programa, naudojantis specialiu algoritmu išgauti rezultatus iš paveikslėlio. Automatinis paveikslėlių atpažinimas turi platų spektrą pritaikimo sričių, tokį kaip automobilių patikra, automatinis kelių mokesčių surinkimas, išmanus eismo reguliavimas [**bhushan2013license**]. Didžioji dauguma automobilio numerių atpažinimo sistemų remiasi optine ženklu atpažinimo sistema. Jų apdorojimo greitis yra pakankamai greitas, kad būtų efektyviai išnaudojama įvairose srityse. Tačiau dažniausiai yra kuriamos specializuotos atpažinimo programos skirtiniams regionams. Panaudojus neuroninius tinklus galima būtų apmokyti atpažinti numerius, kurių formatai yra skirtiniai. Taip pat galima būtų pagreitinti procesą iškerpant numerio rėmelį iš paveikslėlio pasinaudojus neuroniniais tinklais. Norint pagreitinti patį teksto atpažinimą galima naudoti rekurentinį neuroninį tinklą su LSTM savybėmis [**li2016reading**]. Šiame darbe naudosime kursinio darbo metu sukurtą konvoluciinį neuroninį tinklą, kuris yra skirtas atpažinti numerio rėmelio koordinates. Taip pat pritaikysime bei modifikuosime Tesseract LSTM rekurentinį neuroninį tinklą, kuris sugebės atpažinti automobilio numerio simbolius [**smith2007overview**].

## **Darbo tikslas**

Sukurti programą, kuri gebėtų atpažinti lietuviškus automobilio numerius paveikslėlyje panaudojant Tesseract LSTM rekurentinį neuroninį tinklą.

## **Uždaviniai**

1. Pasinaudojus paveikslėlių duomenų rinkiniu susigeneruoti 100.000 atsitiktinių paveikslėlių su automobilio numeriais.
2. Apmokyti kursinio darbo metu sukurtą konvoluciinį neuroninį tinklą (rėmelio atpažinimui) pateikiant sugeneruotus paveikslėlius.

3. Apmokyti Tesseract LSTM rekurentinį neuroninį tinklą (teksto atpažinimui) pateikiant sugeneruotus paveikslėlius.
4. Pasinaudojus kursinio darbo metu sukurtu ir apmokytu konvoluciiniu neuroniniu tinklu atpažinti numerio rėmelį paveikslėlyje ir gauti jo koordinates.
5. Pagal gautas koordinates, iškirpti rėmelį ir pasinaudojus Tesseract LSTM neuroniniu tinklu atpažinti numerį bei atvaizduoti gautus rezultatus pradiniame paveikslėlyje.
6. Ištestuoti tinklą su tikrais paveikslėliais, kuriuose yra lietuviški automobilių numeriai.

## Darbo prielaidos ir metodika

Šiais laikais, kai dominuoja naujosios technologijos, paremtos dirbtiniu intelektu, svarbu analizuoti ir gilintis į procesus, kurie nusako kaip veikia neuroniniai tinklai. Analizujant bei tobulinant dirbtinio intelekto sistemas, galima pasiekti greitesnių bei efektyvesnių rezultatų nei naudojant tradicinius atpažinimo metodus.

Tyrimo objektas yra automobilių numerių atpažinimas. Bus tiriamas kaip vyksta teksto atpažinimas pasitelkiant dirbtinius neuroninius tinklus.

Pagrindinis šio tyrimo metodas - rekurentinio LSTM dirbtinio neuroninio tinklo veikimo analizė. Analizujama pasitelkiant įvairius mokslinius šaltinius, straipsnius, publikacijas, knygas. Kitoje darbo dalyje bus atliekamas eksperimentas pritaikant teoriją.

## Darbo atlikimo procesas

Pirmiausia bus gilinamas į Tesseract LSTM neuroninio tinklo veikimo principus [**bhushan2013license**]. Išanalizavus, bus bandoma apmokyti neuroninį tinklą su kursinio darbo metu sugeneruotais paveikslėliais. Atlikus apmokymą, reikės analizuoti ir gerinti tikslumą keičiant neuroninio tinklo specifikaciją. Galiausiai norint pasiekti dar didesnę spartą ir tikslumą, tinklas bus pritaikytas atpažinti lietuviškus automobilio numerius. Atlikus šį eksperimentą bus sukurta programa, kuri naudos kursinio darbo metu sukurtą konvoluciinių neuroninių tinklų skirtą atpažinti rėmelį bei šiame darbe sukurtą bei modifikuotą Tesseract LSTM neuroninio tinklo konfigūraciją.

## **Eksperimente naudojami instrumentai**

Atlikti šiam eksperimentui buvo naudojami šie pagrindiniai įrankiai:

- Tesseract - skirta atpažinti numeryje esančius simbolius.
- Python - programavimo kalba naudota kurti programoms.
- TensorFlow - skirta neuroninio tinklo pagalba atpažinti numerio rėmelį nuotraukoje.
- OpenCV - skirta apdoroti paveikslėlius.

# **1. Duomenų generavimas**

## **1.1. Rèmelio atpažinimui skirtų duomenų generavimas**

Norint sukurti realiai veikiančią programą, kuri naudotų neuroninį tinklą išgauti tikėtinam rezultatui, tinklą reikia apmokyti su dideliu kiekiu duomenų. Apmokant bet kokį neuroninį tinklą turi būti būti pateiktas duomenų rinkinys su norimu gauti rezultatu.

### **1.1.1. Pirminis duomenų generavimo variantas**

Pirminis duomenų generavimo variantas, kuris buvo įgyvendintas bei sėkmingai sugeneruoti 100.000 paveikslėlių skirtų neuroninio tinklo apmokymui.

#### **1.1.1.1. Generavimo parametrai**

Šiam tyrimui buvo pasirinkta generuoti duomenų rinkinį, kurių kiekvienas paveikslėlis būtų 128 pikselių pločio ir 64 pikselių ilgio. Toks pasirinktas būdas užtikrina, kad neuroninis tinklas bus pajęsus suprasti paveikslėlio turinį, o dydis pakankamai mažas, kad būtų galima turėti efektyviai veikiantį neuroninį tinklą.

Pirmaoji paveikslėlio rezultato dalis nurodo, koks yra teisingas numeris. Antroji – numero rèmelio egzistavimas paveikslėlyje. Jei reikšmė 1 – numeris yra tinkamas nuskaitymui, 0 – neatitinka kriterijų (2 pav.).

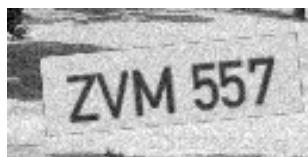
Kriterijai, kurie nusako ar numero rèmelis yra tinkamas apdorojimui:

- Visas rèmelio plotas yra paveikslėlyje.
- Rèmelio plotis yra mažesnis nei 80% paveikslėlio pločio.
- Rèmèlio aukštis yra mažesnis nei 87,5% paveikslėlio aukščio.
- Rèmelio plotis yra didesnis nei 60% paveikslėlio pločio.
- Rèmèlio aukštis yra didesnis nei 60% paveikslėlio aukščio.

Su tokiais parametrais galima naudoti judantį 128x64 pikselių langelį, kuris judėtų po 8 pikselius ir kas kartą padidintų rèmelį  $\sqrt{2}$  kartų. Tokiu būdu užtikrinama, kad nebus praleista nei viena paveikslėlio vieta, o taip pat pakankamai efektyviai ir greitai pereinamas visas paveikslėlis.



(a) Tikimasis rezultatas **MNI144 1.**



(b) Tikimasis rezultatas **ZVM557 1.**



(c) Tikimasis rezultatas **COH150 0** (per mažas numeris).



(d) Tikimasis rezultatas **GTK311 0** (ne pilnas numeris).

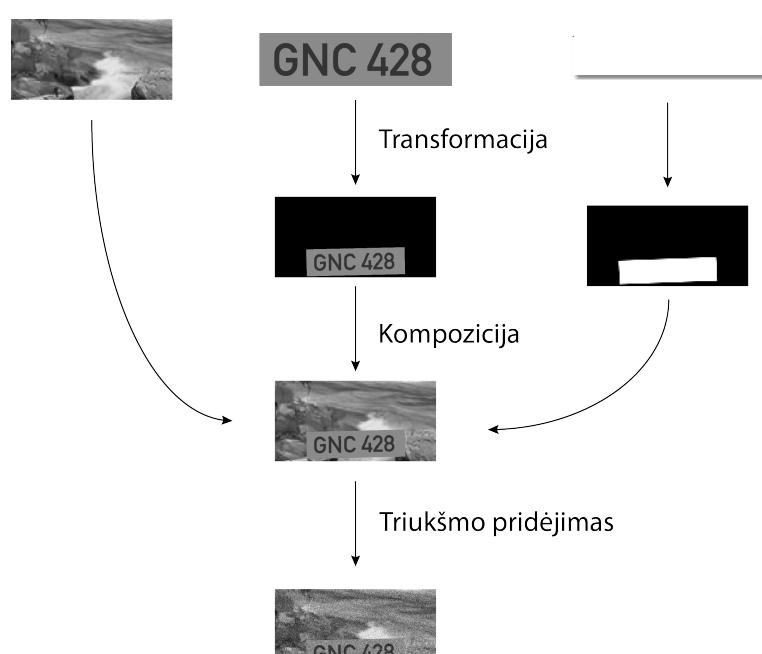


(e) Tikimasis rezultatas **ABN046 0** (ne pilnas numeris).



(f) Tikimasis rezultatas **KLF155 0** (nėra numerio).

2 pav. Sugeneruotų paveikslėlių pavyzdžiai



3 pav. Paveikslėlių generavimo schema

### 1.1.1.2. Numerio generavimas

Numeris ir rēmelio spalva generuojama atsitiktinai, tačiau tekstas turi būti tamšesnis negu rēmelis. Tokiu būdu bandoma atkurti realaus pasaulio apšvietimo variacijas. Numeris generuojamas pagal Lietuvos Respublikos Valstybinių numerių formatą, kuris yra - 3 lotyniško alfabeto raidės (išskyrus lietuvių kalboje nenaudojamas raidės) ir 3 arabiški skaičiai. Generuojant numerį, atsitiktine tvarka parenkamos trys raidės iš 23 raidžių žodyno *ABCDEFGHIJKLMNOPRSTUVYZ* bei 3 skaičiai iš skaičių žodyno *0123456789*. Maksimalus galimas unikalių numerių skaičius siekia:

$$23^3 * 10^3 = 12.167.000.$$

### 1.1.1.3. Transformacija

Norint, kad tinklas efektyviai mokytusi ir atpažintų paveikslėlius realaus pasaulio sąlygomis, generuojant duomenų rinkinį buvo pritaikyta remelio transformacija. Tai atlikti buvo pasitelktas metodas generuoti atsitiktines reikšmes X, Y, Z ašims ir pritaikyti Oilerio kampų metodą [slabaugh1999computing]. Reikšmių rėžiai pasirinkti tokie, kuriuos labiausiai tikėtinai sutiki realiame pasaulyje. Kaip atrodo transformacija galima matyti 3 paveikslėlyje. Ašių atsitiktinių reikšmių rėžiai:

$$-0.3 \leq X \leq 0.3,$$

$$-0.2 \leq Y \leq 0.2,$$

$$-1.2 \leq Z \leq 1.2.$$

Transformacijos vykdomos 3 etapais (programinis kodas matomas 4 pav.):

1. Sukama aplink Y ašę:

- Apskaičiuojamos  $\cos(Y)$  ir  $\sin(Y)$  reikšmės,
- Sudaroma  $3 \times 3$  matrica su reikšmėmis,

$$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}.$$

2. Sukama aplink X ašę

- Apskaičiuojamos  $\cos(X)$  ir  $\sin(X)$  reikšmės,
- Sudaroma  $3 \times 3$  matrica su reikšmėmis,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix}.$$

- Sudauginama su praėitame žingsnyje gauta matrica

### 3. Sukama aplink Z ašį

- Apskaičiuojamos  $\cos(Z)$  ir  $\sin(Z)$  reikšmės,
- Sudaroma  $3 \times 3$  matrica su reikšmėmis,

$$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix},$$

- Sudauginama su praėitame žingsnyje gauta matrica.

```
def euler_to_mat(yaw, pitch, roll):
    # Rotate clockwise about the Y-axis
    c, s = math.cos(yaw), math.sin(yaw)
    M = numpy.matrix([[ c, 0., s],
                      [ 0., 1., 0.],
                      [ -s, 0., c]])

    # Rotate clockwise about the X-axis
    c, s = math.cos(pitch), math.sin(pitch)
    M = numpy.matrix([[ 1., 0., 0.],
                      [ 0., c, -s],
                      [ 0., s, c]]) * M

    # Rotate clockwise about the Z-axis
    c, s = math.cos(roll), math.sin(roll)
    M = numpy.matrix([[ c, -s, 0.],
                      [ s, c, 0.],
                      [ 0., 0., 1.]]) * M

return M
```

4 pav. Oilerio kampų metodo kodas

#### 1.1.1.4. Kompozicija

Turėti realų foną svarbu, kadangi tinklas turi išmokti surasti rēmelio kampus „nesukčiaudamas“. Naudojant juodą foną, tinklas gali daryti prielaidą, kad remėlis yra ten, kur nėra juodos spalvos, o tai būtų netikslu realiamame pasaulyje. Transformuotas automobilio numerio rēmelis su komponuojamas su atsitiktiniu paveikslėliu atsitiktinėje vietoje. Atsitiktinių paveikslėlių šaltiniu naudojamas daugiau nei 100.000 paveikslėlių duomenų rinkinys [xiao2010sun]. Labai svarbu didelis kiekis paveikslėlių, taip sumažinant riziką, kad neuroninis tinklas atsimins kiekvieną paveikslėli. Kaip atrodo kompozicija galima matyti 3 paveikslėlyje.

$$A = 0.6,$$

$$B = 0.875,$$

$$C = 1.5.$$

kur:

- A – minimalus numerio rēmelio plotis,
- B – maksimalus numerio rēmelio aukštis,
- C – dydžio variacijos koeficientas.

$$\min = (A + B) * 0.5 - (B - A) * 0.5 * C,$$

$$\min = ((0.6 + 0.875) * 0.5) - ((0.875 - 0.6) * 0.5 * 1.5),$$

$$\min = (1.475 * 0.5) - (0.275 * 0.5 * 1.5),$$

$$\min = 0.7375 - 0.20625,$$

$$\mathbf{\min = 0.53125},$$

$$\max = (A + B) * 0.5 + (B - A) * 0.5 * C,$$

$$\max = ((0.6 + 0.875) * 0.5) + ((0.875 - 0.6) * 0.5 * 1.5),$$

$$\max = (1.475 * 0.5) + (0.275 * 0.5 * 1.5),$$

$$\max = 0.7375 + 0.20625,$$

$$\mathbf{\max = 0.94375},$$

$$x = R[min, max],$$

$$p = 1, \text{ kai } x \in [A, B],$$

$$p = 0, \text{ kai } x \in [A, B].$$

kur:

- min - minimalus generuojamas dydžio koeficientas,
- max - maksimalus generuojamas dydžio koeficientas,
- R - atsitiktinio skaičiaus generavimo funkcija tarp dviejų reikšmių,
- x - numerio rėmelio dydžio koeficientas lyginant su pradiniu dydžiu,
- p - jei 1 - rėmelis tinkamai egzistuoja paveikslėlyje, 0 - rėmelis neegzistuoja arba yra netinkamas.

Atsitiktinių reikšmių rėžis, kuris nusako kurioje vietoje turėtų atsidurti rėmelis.

#### 1.1.1.5. Triukšmo pridėjimas

Triukšmas paveikslėlyje reikalingas, kadangi realiame pasaulyje pasitaiko, kad kameros sensorius generuoja triukšmus, o taip pat, kad neuroninis tinklas nepersimokyti ir neskirstytų paveikslėlių pagal vieną konkrečią spalvą ar būtų priklausomas nuo „aštrių“ kampų. Triukšmas paveikslėliui pridedamas pritaikant Gauso normalųjį skirstinį su reikšme 0.05. Kaip atrodo triukšmo pridėjimas galima matyti 3 paveikslėlyje.

### 1.2. Numerio atpažinimui skirtų duomenų generavimas

Norint sėkmingai apmokyti neuroninį tinklą, reikia daug pradinių mokymo duomenų. Šiam tikslui pasiekti nuspręsta duomenis susigeneruoti, kadangi tiek daug lietuviškų numerių tikrų nuotraukų nėra įmanoma gauti.

#### 1.2.1. Idėja

Norint kuo efektyviau apmokyti LSTM rekurentinį neuroninį tinklą, reikia sugeneruoti tokio pačio šrifto atsitiktinius numerius, kurie kuo panašiau atkurtų realią situaciją. Idėja buvo susirinkti visas galimas raides ir skaicius iš realių automobilių nuotraukų. Atrinktus simbolius išsikirpti bei sugeneruoti atsitiktinius raidžių ir skaičių kratinius.

### 1.2.2. Simbolių rinkimas

Realių automobilių numerių nuotraukų paieška buvo vykdoma <http://autoplus.lt> puslapyje. Norint iškirpti kokybiškas raides bei skaičius reikia aukštos kokybės nuotraukų. Atrinkus tinkamas nuotraukas, buvo iškirptos visos galimos raidės ir skaičiai (5 pav.). Kai kuriems simboliams reikėjo pritaikyti transformacijas, kad jų orientacija būtų horizontaliai tiesi.

**A B C D E F G H J K L M N O P R S T U V Z 0 1 2 3 4 5 6 7 8 9**

5 pav. Atrinktos raidės ir skaičiai

### 1.2.3. Generavimo algoritmas

Generuoti atsitiktiniams automobilio numeriams parašyta Python programėlė (žiūrėti priede Nr. 2). Veikimo eiga:

- Masyve *letters* saugomi paveikslėliai atitinkantys raides.
- Masyve *numbers* saugomi paveikslėliai atitinkantys skaičius.
- Sukamas ciklas  $N$  kartų.
- Kiekvieno iteracijos metu atsitiktiniu būdu atrenkamos trys raidės ir trys skaičiai iš atitinkamo masyvo.
- Sukuriamas naujas masyvas, kuriame iš eilės sudedami atrinkti paveikslėliai.
- Surandamas mažiausias paveikslėlis iš atrinktų, ir pagal jo dydį sumažinamas likusių paveikslėlių aukštis proporcingai.
- Sujungiamas vienas paveikslėlis iš atrinktų simbolių.
- Paveikslėlis išsaugomas *xxxxyy.tif* formatu, kur x - raidė, y - skaičius.
- Šalia išsaugomas tekstinė rinkmena *xxxxyy.gt.txt*, kur x - raidė, y - skaičius, kurio viduje yra XXXYYY formatu išsaugotas sugeneruotas numeris.

## 2. Neuroninių tinklų architektūra

Šiame skyriuje aprašyti dviejų skirtingų neuroninių tinklų architektūriniai sprendimai. Numerio rēmelio atpažinimui naudojamas konvoliucinis neuroninis tinklas bei numeryje esančių simbolių atpažinimui naudojamas rekurentinis neuroninis tinklas.

### 2.1. Numerio rēmelio atpažinimui skirtas neuroninis tinklas

Numerio rēmelio atpažinimui panaudotas konvoliucinis neuroninis tinklas.

#### 2.1.1. Modelis

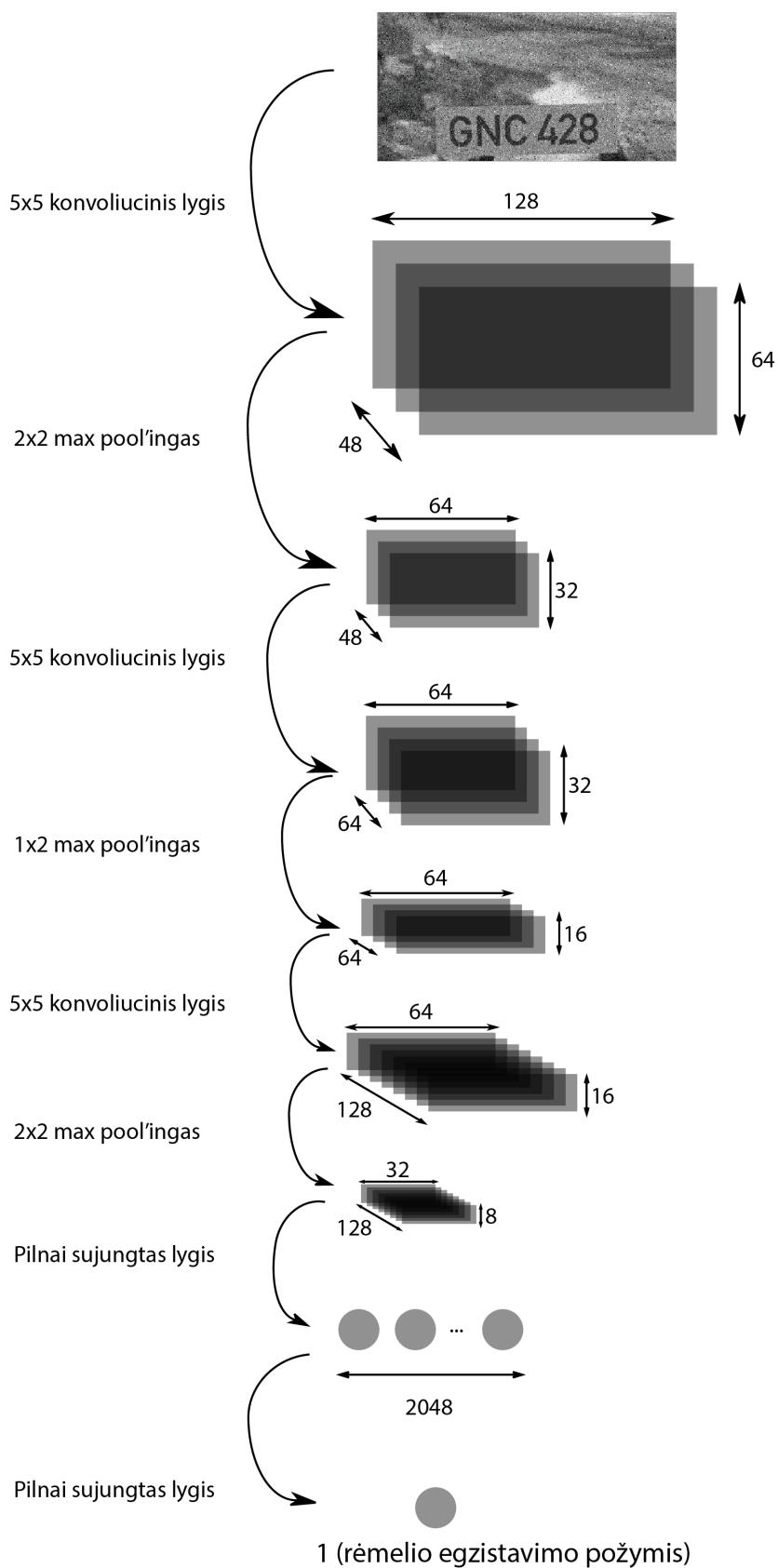
Neuroninis tinklas kurtas su Tensorflow bibliotekomis. Kuriant dirbtinį konvoliucinį neuroninį tinklą buvo pasirinkta architektūra pavaizuota 6 pav. Iš viso yra 3 konvoliuciniai lygiai, kurių dydžiai yra 48, 64 ir 128[**goodfellow2013multi**]. Visų jų langelio dydis yra vienodas - 5x5. Taip pat yra 3 max pool'ingo lygmenys, kurių pirmo ir trečio langelio dydis yra 2x2, o antro - 1x2. Tada neuroninis tinklas turi du pilnai sujungtus lygius, kurių pirmojo dydis - 2048, o antrojo (klasifikatoriaus) - 1. Po pirmojo konvoliucinio lygio pritaikyta neuronų atmetimo operacija, norint nepermokyti tinklo pirminėje stadioje. Po trečiojo konvoliucinio lygio taip pat pritaikyta neuronų atmetimo operacija, norint padidinti neuroninio tinklo tikslumą, kadangi pastebėta, kad ignoruojant 50% neuronų, tinklas turi didesnį atpažinimo tikslumą[**stark2015captcha**]. Kiekvieno mokymo ciklo metu imties dydis yra 50. Galutinis tinklo išvedamas rezultatas yra:

$$0 \leq x \leq 1, x \in N.$$

Neuroninį tinklą sudaro:

- 3 konvoliuciniai lygiai:
  1. Konvoliucinis lygis - 48 filtrų, langelio dydis 5x5, įeinančio paveikslėlio dimensijos 128x64x3, išeinančio paveikslėlio dimensijos 128x64x48.
  2. Konvoliucinis lygis - 64 filtrų, langelio dydis 5x5, įeinančio paveikslėlio dimensijos 64x32x48, išeinančio paveikslėlio dimensijos 64x32x64.
  3. Konvoliucinis lygis - 128 filtrų, langelio dydis 5x5, įeinančio paveikslėlio dimensijos 64x16x64, išeinančio paveikslėlio dimensijos 64x16x128.

- 3 max pool'ingo lygiai:
  1. Max pool'ingo lygis - lanelio dydis 2x2, įeinančio paveikslėlio dimensijos 128x64x48, išeinančio paveikslėlio dimensijos 64x32x48.
  2. Max pool'ingo lygis - lanelio dydis 1x2, įeinančio paveikslėlio dimensijos 64x32x64, išeinančio paveikslėlio dimensijos 64x16x64.
  3. Max pool'ingo lygis - lanelio dydis 2x2, įeinančio paveikslėlio dimensijos 64x16x128, išeinančio paveikslėlio dimensijos 32x8x128.
- 2 pilnai sujungti lygiai:
  1. Pilnai sujungtas lygis - įeinančio paveikslėlio dimensijos 32x8x128, išeinančių signalų kiekis - 2048.
  2. Pilnai sujungtas lygis - įeinančių signalų kiekis - 2048, išeinančių signalų kiekis - 1.



6 pav. Neuroninio tinklo architektūra

## 2.2. Numerio simbolių atpažinimui skirtas neuroninis tinklas

Tesseract programa nuo 4.00 versijos integravo naują neuroninio tinklo pagrindu veikiančią teksto eilučių atpažinimo posistemę. Pirminis idėjos šaltinis kilo iš *OCRopus* sistemos, kuri panaudodama Python programavimo kalbą įgyvendino LSTM veikimą. Tačiau tai buvo visiškai perdaryta panaudojus C++ kalbos ypatumus. Neuroninio tinklo sistema Tesseract programoje egzistuoja jau nuo *TensorFlow* atsiradimo ir taip pat su ja yra suderinama, kadangi naudojama tos pačios sintaksės neuroninio tinklo modelio aprašymo kalbą (VGSL)<sup>1</sup>.

Pagrindinė VGSL idėja yra, kad nebūtina išmokti daug naujų dalykų, kad būtų įmanoma sukurti ir apkomyti neuroninį tinklą. Nereikia mokytis *Python* programavimo kalbos, *TensorFlow* bibliotekos ar net rašyti C++ programinio kodo. Užtenka įvaldyti VGSL kalbos sintaksines ypatybes, kad būtų įmanoma taisyklingai sudaryti neuroninį tinklą.

### 2.2.1. Bendrai apie LSTM

LSTM yra rekurentinio neuroninio tinklo architektūra naudojama gilaus mokymosi srityse. Skirtingai nuo standartinių „feedforward“ neuroninių tinklų, LSTM turi grįžtamuosius ryšius, kurie tai padaro „universaliu kompiuteriu“ (t.y. galima skaičiuoti viską, ką gali Turingo mašina). LSTM neuroniniai tinklai gali apdoroti ne tik statinius objektus (paveikslėlius), bet ir informacijos sekas (garso įrašas ar video įrašas). LSTM dažniausiai naudojama ranka rašyto teksto ar garso atpažinimui [**hochreiter1997long**].

LSTM tinklo paslėptame rekurentiniame lygyje yra specialūs vienetai – atminties blokai. Atminties blokuose yra atminties ląstelių saugančių laikiną tinklo būseną pridedant dar specialius dauginamuosius vienetus, taip vadinamus vartus, kurie kontroliuoja informacijos srauto tékmę. Kiekvienas atminties blokas originalioje architektūroje yra sudarytas iš įvesties ir išvesties vartų. Įvesties vartai kontroliuoja įvesties aktyvacijų informacijos tékmę į atminties ląstelę. Išvesties vartai kontroliuoja ląstelių aktyvacijos išvesties informacijos tékmę atgal į tinklą. Vėliau į atminties bloką pridedami atminties praradimo vartai. Tai sprendžia vieną iš LSTM modelio silpnybių, kai įvestis néra suskirstyta į atskiras sekas ir apdorojimas niekad nesibaigia. Atminties praradimo vartai keičia vidinę ląstelęs būseną prieš patalpinant ją į ląstelę per rekurentinį ryšį su pačia savimi, tuo pačiu įgalinant pamiršti ar ištinti ląstelęs atmintį. LSTM tinklas skaičiuoja sąryšius nuo įvesties sekos

$$x = (x_1, \dots, x_T)$$

---

<sup>1</sup><https://github.com/tesseract-ocr/tesseract/wiki/VGSLSpecs>

iki išvesties sekos

$$y = (y_1, \dots, y_T)$$

atlikdamas tinklo vienetų aktyvacijų funkcijas pagal žemiau nurodytas formules iš eilės nuo  $t = 1$  iki  $T$ :

$$i_t = \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \quad (1)$$

$$f_t = \sigma(W_{fx}x_t + W_{fm}m_{t-1} + W_{fc}c_{t-1} + b_f) \quad (2)$$

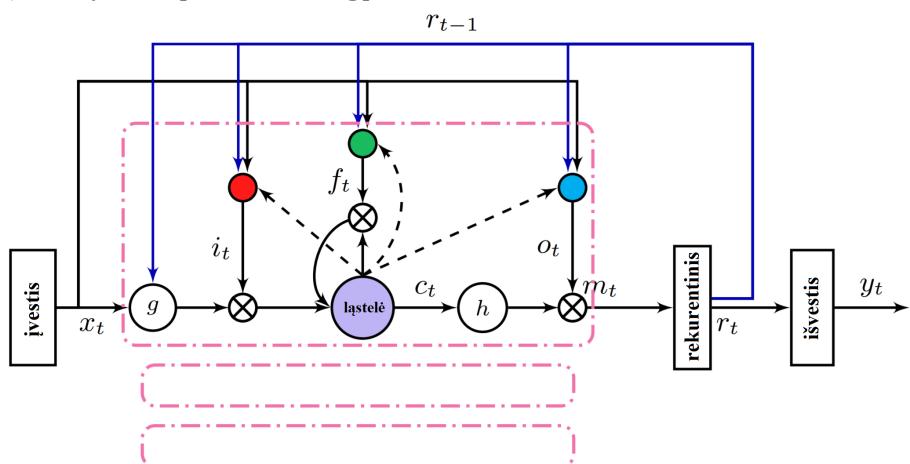
$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \quad (3)$$

$$o_t = \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \quad (4)$$

$$m_t = o_t \odot h(c_t) \quad (5)$$

$$y_t = \phi(W_{ym}m_t + b_y) \quad (6)$$

kur  $W$  apibrėžia svorių matricas (pvz.:  $W_{ix}$  yra svorių matrica nuo įvesties vartų iki pačios įvesties),  $W_{ic}$ ,  $W_{fc}$ ,  $W_{oc}$  yra ištrižos svorių matricos skirtos skyliučių jungtimis<sup>2</sup>,  $b$  apibūdina išvesties vektorius, kurie neturi jokios įvesties,  $\sigma$  yra logistinė sigmoidinė funkcija,  $i$ ,  $f$ ,  $o$ ,  $c$  yra atitinkamai įvesties vartai, atminties praradimo vartai, išvesties vartai ir ląstelės aktyvacijos vektoriai, kurių visų dydis yra toks pat kaip ir ląstelės išvesties aktyvacijos vektoriaus  $m$ ,  $\odot$  yra vektorių elementas,  $g$  ir  $h$  yra ląstelės įvestis ir ląstelės išvesties aktyvacijos funkcijos,  $\tanh$  ir  $\phi$  yra tinklo išvesties aktyvacijos funkcija - **Softmax [sak2014long]**.



7 pav. LSTM atminties blokai

<sup>2</sup>Skylutės jungtis (angl. peephole connection) - LSTM tinklo ypatybė, kai vartai priklausoma ne tik nuo buvusios paslėptos būsenos, bet ir nuo buvusios vidinės būsenos)

### **2.2.1.1. Rekurentinis neuroninis tinklas**

Rekurentinis neuroninis tinklas – skirtingai nuo grįžtamuoju ryšiu grįstų neuroninių tinklų, yra rekursinio dirbtinio neuroninio tinklo variantas, kuriame ryšiai tarp neuronų sudaro apskritą ratą. Tai reiškia, kad išvedimo rezultatas priklauso ne tik nuo dabartinių įvesčių, bet ir nuo ankstesnio etapo neuronų būklės. Šis metodas leidžia vartotojams išspręsti problemas, susijusias su balso ar kalbos atpažinimu. Atlirktyrimai rodo, kad įmanoma sukurti rekurentinį neuroninį tinklą, kuris gali generuoti naujus sakinius ir dokumentų santraukas.

### **2.2.2. Integracija su Tesseract**

Integruota neuroninio tinklo posistemė gali būti panaudojama kaip papildinys esamai analizės sistemai atpažstant tekštą dideliame dokumente arba gali būti naudojama kartu su išoriniu teksto detektoriumi, kad atpažintų tekštą iš vienos teksto eilutės atvaizdo.

Nuo 4.00 versijos neuronio tinklo pagrindu veikiantis atpažinimo būdas Tesseract programoje yra numatytais.

### **2.2.3. Sisteminiai reikalavimai**

Nauja programos versija naudoja iki 10 kartų daugiau kompiuterio procesoriaus resursų nei senesnės Tesseract versijos, tačiau jei naudojamas kompiuteris ir platforma palaiko žemiau aprašytas funkcijas, resursų naudojimas gali sumažėti:

- *OpenMP* leidžia naudoti iki 4 procesoriaus branduolių vienu metu, jei juos procesorius turi.
- *Intel/AMD* procesoriai, kurie palaiko *SSE* ir/ar *AVX* technologiją, turi pranašumą naudojant *SIMD* branduolio matricų daugybos operacijų išlygiagretinimą.
- Kompiuteryje, kuris turi bent 4 branduolius, *AVX*, nesudėtingą anglų kalbos tekštą parveikslėlyje, atpažinimas užtrunka dvigubai ilgiau bei naudoja 7 kartus daugiau procesoriaus resursų nei ankstesnės versijos, nors Hindi kalbos atpažinimas trunka netgi greičiau nei senesnėse versijose bei naudoja tik nežymiai daugiau procesoriaus resursų.

Jei šių paminėtų komponentų nėra sistemoje, egzistuoja lėtesnė C++ kalbos implementacija, kuri vis dėlto sugeba atlirktyrimai paskirtą darbą.

#### **2.2.4. Įgyvendinimo pagrindai**

Visi neuroninio tinklo lygių tipai yra paveldėti iš bazinės *Network* klasės. *Plumbing* subklasė yra bazinė kitų tinklo lygių, kurie įvairiomis operacijomis (grupuojant keletą lygių; keičiant įvestį ir išvestį) manipuliuoja kitais lygiais, klasė.

#### **2.2.5. Naujo tinklo lygio pridėjimas**

Naujas tinklo lygis turi būti paveldimas iš klasės *Network* ar *Plumbing* ir įgyvendinti bent vieną virtualų metodą:

- *spec*, kuris grąžina *String* tipo eilutę, kuri buvo naudojama sukurti šiam tinklo lygiui.
- *Serialize/DeSerialize* – skirtas išsaugoti/atkurti tinklo lygį iš/į failą.
- *Forward* – skirtas treniravimo metu vykdyti tinklo lygį nurodant kryptį į priekį.
- *Backward* – skirtas treniravimo metu vykdyti tinklo lygį nurodant kryptį atgal.

Lygiai, kurie turi svorius taip pat turi įgyvendinti *Update* metodą, kuris atnaujina svorius naudodamas rinkinį nuolydžių. Taip pat yra keletas kitų metodų, kurie turėtų būti įgyvendinti, priklausomai nuo specifinių tinklo lygio reikalavimų:

- *NetworkBuilder* klasė turi būti pakeista, kad būtų galima apdoroti naujo tipo specifikaciją.
- *NetworkType* klasifikatorius turi būti papildytas nauju tipu.
- Naujo tipo atitinkamas įrašas turi būti pridėtas į lauką *Network::kTypeNames*.
- *Network::CreateFromFile* metodas turi būti modifikuotas, kad galėtų būti deserializuotas naujo tinklo lygio tipas.
- Kaip ir su kiekvienu nauju kodu, *lstm/Makefile.am* failas turi būti papildytas naujais failų pavadinimais.

#### **2.2.6. VGSL specifikacijos**

Kintamo dydžio grafų aprašymo kalba (angl. Variable-size Graph Specification Language) įgalina lengvai aprašyti neuroninį tinklą, susidarantį iš konvoliucijų ar LSTM tinklo ypatybių, kuris gali apdoroti kintamo dydžio paveikslėlius panaudojant vienos teksto eilutės ilgio aprašytą tinklo specifikaciją.

## VGSL pritaikymas

VGSL kalba sukurta aprašyti neuroniniems tinklams, kurie:

- Kintamo dydžio (tinka ir fiksuočio dydžio) paveikslėlius naudoja kaip įvestį (vienoje ar dviejose dimensijose).
- Gauna rezultatą kaip reikšmių matricą, tekštą ar kategoriją.
- Konvolucijos ir LSTM tinklai yra pagrindinis skaičiavimo komponentas.

**Modelių aprašančios teksto eilutės įvestis ir išvestis** Neuroninio tinklo modelių aprašo teksto eilutę, kurioje yra aprašomos įvesties, išvesties ir tinklo lygių specifikacijos. Pavyzdys:

[1,0,0,3 Ct5,5,16 Mp3,3 Lfys64 Lfx128 Lrx128 Lfx256 01c105]

Pirmi 4 numeriai aprašo įvesties dydį ir tipą. Tai atitinka TensorFlow sistemos paveikslėlio tenzoriaus konvenciją: [paketas, aukštis, plotis, gylis]. Šiuo atveju paketas yra ignoruojamas, bet gali būti panaudotas aprašant treniravimo paketo dydį. Aukštis ir/ar plotis gali būti lygus 0, tokiu būdu jie tampa kintamo dydžio. Nenulinės aukščio ir/ar pločio reikšmės reiškia, kad visi įvesties paveikslėliai bus vienodo dydžio arba bus suspausti iki galimo dydžio jei reikės. Gylio reikšmė 1 nurodo, kad paveikslėlis yra juodai baltas, reikšmė 3 nurodo, kad naudojamos visos spalvos. Yra specialus atvejis, kai nurodomas gylis su kitokia reikšme nei 1 ar 3 ir aukščiu - 1. Tokiu atveju tai bus traktuojama kaip vertikalių pikselių juostų seka. Paskutinis žodis nurodo apibūdina išvestį:

- Bendrinis išvesties formatas su n klasii -  $O(2|1|0)(l|s|c)n$ :
  - 2 (reikšmių matrica) – išvestis yra dviejų dimensijų įvesties vektorių žemėlapis.
  - 1 (seka) – išvestis yra vienos dimensijos vektoriaus reikšmių seka.
  - 0 (kategorija) – išvestis yra vieno vektoriaus reikšmė.
  - l naudoja logistinę netiesinę funkciją, išgalinant išvesti keletą rezultatų bet kuriai išvesties vektoriaus reikšmei.
  - s naudoja Softmax netiesinę aktyvacijos funkciją, išvedant vieną rezultatą kiekvienai reikšmei.
  - c naudoja Softmax su CTC aktyvacijos funkciją. Gali būti naudojama tik su seka.
- Klasii skaičius yra ignoruojamas (palikta dėl suderinamumo su TensorFlow) ir tikras skaičius paimamas iš *unicharset* failo.

## 2.2.7. Vidinių tinklo lygių sintaksė

Žemiau aprašomos funkcinės, *plumbing* operacijos bei pateikiami jų pavyzdžiai.

**Funkcinės operacijos** Egzistuoja 5 skirtinės funkcinės operacijos:

- $C(s|t|r|l|m)<y>,<x>,<d>$  - vykdoma konvoluciija naudojant  $y$ ,  $x$  langelį, nenaudojant sutraukimo, su atsitiktiniu užpildu,  $d$  išvestimi bei  $s|t|r|l|m$  aktyvavimo funkcija.
- $F(s|t|r|l|m)<d>$  - pilnai jungus lygis su  $s|t|r|l|m$  aktyvavimo funkcija ir  $d$  išvestimi. Sumažina aukštį ir plotį iki 1. Susijungia su kiekvienu įvesties  $y$ ,  $x$  bei gylio pozicija, sumažindamas aukštį, plotį iki 1 ir sugeneruodamas  $<d>$  vektorių kaip išvestį. Įvesties aukštis ir plotis turi būti konstantos.
- $L(f|r|b)(x|y)[s]<n>$  - LSTM laštelė su  $n$  išvesčių:
  - $f$  - leidžia tik į priekį judantį LSTM lygi.
  - $r$  - leidžia tik priešinga kryptimi judantį LSTM lygi.
  - $b$  - leidžia abiejomis kryptimis judantį LSTM lygi.
  - Operacija veiks tik su  $x$  arba  $y$  kryptimi, ignoruojant kitą kryptį.
  - $s$  - neprivalomas argumentas, kuris grąžina kaip rezultatą tik paskutinį žingsnį, sutraukdamas dimensiją iki vieno elemento.
- $LS<n>$  - tik į priekį  $x$  kryptimi judanti LSTM laštelė su integruota *Softmax* aktyvacijos funkcija.
- $LE<n>$  - tik į priekį  $x$  kryptimi judanti LSTM laštelė su integruota *Softmax* aktyvacijos funkcija ir binariniu atkodavimu.

Aukščiau paminėtos raidės  $(s|t|r|l|m)$  reiškia vieną iš aktyvacijos funkcijų:

- $s$  - sigmoido funkcija.
- $t$  - hiperbolinio tangento funkcija.
- $r$  - *Relu* funkcija.
- $l$  - linijinė funkcija.
- $m$  - *Softmax* funkcija.

Pavyzdžiai:

- Cr5,5,32 - 5x5 Relu konvolucija su 32 filtrais.
- Lfx128 - tik į priekį judantis LSTM lygis, x dimensijoje turintis 128 išvestis, laikydamas y dimensiją nepriklausoma.
- Lfys64 - tik į priekį judantis LSTM lygis, y dimensijoje turintis 64 išvestis, laikydamas x dimensiją nepriklausoma ir sutraukdamas y dimensiją iki 1 elemento.

**Plumbing ops** *Plumbing* operacijos leidžia konstruoti pakankamai kompleksiškus grafus:

- [...] - Vykdysi ... neuroninius tinklus nuosekliai lygiais.
- (...) - Vykdysi ... neuroninius tinklus lygiagrečiai, jungiant jų išvestis į gylį.
- S<y>,<x> - Pakeisti dviejų dimensijų įvestį susitraukimo koeficientu y,x, sutvarkant duomenis padidinant įvesties gylį koeficientu xy.
- Mp<y>,<x> - pritaikyti *Maxpool* operaciją kiekvienam stačiakampiui (y, x), gaunant vienintelę reikšmę.

**Pavyzdys: Vienos dimensijos LSTM tinklas, galintis tiksliai atpažinti tekštą**

[1,1,0,48 Lbx256 01c105]

Lygių aprašymas (įvesties lygis apačioje, išvesties lygis viršuje):

- 01c105: Išvesties lygis, pagaminantis vienos dimensijos seką, treniruotą su CTC, išvedantis 105 klases.
- Lbx256: Dvikryptis LSTM lygis judantis x kryptimi su 256 išvestimis.
- 1,1,0,48: Įvestis yra juodai baltas paveikslėlis, kurio aukštis yra 48 pikseliai, laikomas kaip vienos dimensijos vertikalių pikselių seka.
- [ ]: Tinklas visada vykdo lygius nuosekliai.

Šis sukurtas tinklas gerai veikiai atpažstant tekštą, tol kol įvesties paveikslėlis normalizuotas vertikalioje padėtyje.

**Pavyzdys: Keletos lygių LSTM tinklas, galintis tiksliai atpažinti tekštą**

[1,0,0,1 Ct5,5,16 Mp3,3 Lfys64 Lfx128 Lrx128 Lfx256 01c105]

Lygių aprašymas (įvesties lygis apačioje, išvesties lygis viršuje):

- O1c105: Išvesties lygis, pagaminantis vienos dimensijos seką, treniruotą su CTC, išvedantis 105 klasses.
- Lfx256: Tik pirmyn judantis LSTM lygis x kryptimi su 256 išvestimis.
- Lrx128: Tik priešingai judantis LSTM lygis x kryptimi su 128 išvestimis.
- Lfx128: Tik pirmyn judantis LSTM lygis x kryptimi su 128 išvestimis.
- Lfys64: Dimensiją apibendrinantis LSTM lygis, apibendrinantis y dimensiją su 64 išvestimis.
- Mp3,3: 3x3 *Maxpool* operacija.
- Ct5,5,16: 5x5 konvoluciija su 16 išvesčių ir hiperbolinio tangento aktyvacijos funkcija.
- 1,0,0,1: Įvestis yra juodai baltas paveikslėlis.
- [ ]: Tinklas visada vykdo lygius nuosekliai.

Šis sukurtas LSTM tinklas yra atsparesnis vertikaliems teksto nuokrypiams.

### 2.2.8. Kintamo dydžio įvestis ir apibendrinantis LSTM lygis

Kol kas vienintelis būdas sumažinti nežinomo dydžio dimensiją iki žinomo dydžio (1) yra naudojant apibendrinantį LSTM lygi. Vienas apibendrinantis LSTM lygis sumažins vieną dimensiją (x arba y), palikdamas vienos dimensijos seką. Tada vienos dimensijos seka gali būti sumažinta iki *Softmax* ar logistinės aktyvacijos funkcijos išvesties.

Toliau norint atpažinti tekstą, įvesties paveikslėlių aukštis turi būti fiksotas arba pakeistas jų vertikalus dydis (panaudojant *Mp* ar *S* funkcijas) iki 1, arba leidžiant kintamo aukščio paveikslėlius, apibendrinantis LSTM lygis turi sumažinti vertikalią dimensiją iki vienintelės reikšmės. Apibendrinantis LSTM lygis taip pat gali būti naudojamas su fiksuoto aukščio įvestimis.

## 2.2.9. Modelis

Šiam konkrečiam uždavinui, kuris turi atpažinti automobilio numerio simbolius, pasirinktas tokios konfigūracijos neuroninis LSTM tinklas:

```
[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 01c`head -n1 data/unicharset`]
```

Lygių aprašymas (įvesties lygis apačioje, išvesties lygis viršuje):

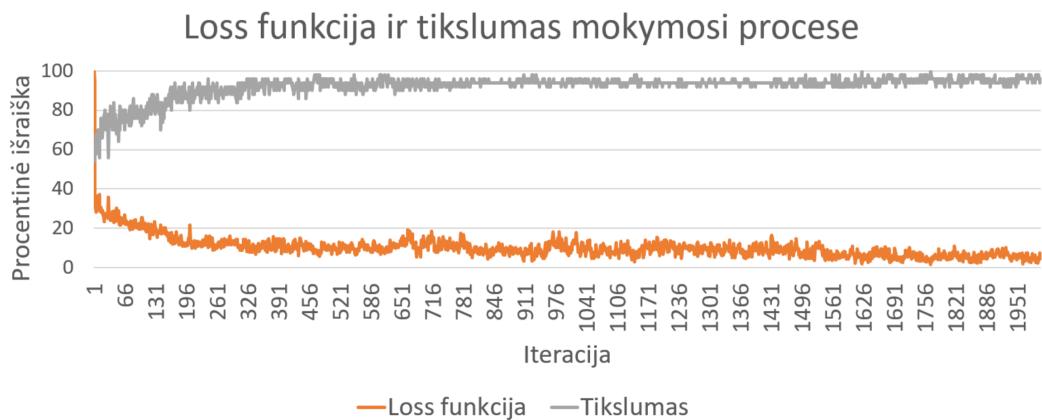
- 01c‘head -n1 data/unicharset’: Išvesties lygis, pagaminantis vienos dimensijos seką, treniuotą su CTC, išvedantis n klasių nurodytų faile esančiame *data/unicharset*.
- Lfx256: Tik pirmyn judantis LSTM lygis x kryptimi su 256 išvestimis.
- Lrx96: Tik priešingai judantis LSTM lygis x kryptimi su 96 išvestimis.
- Lfx96: Tik pirmyn judantis LSTM lygis x kryptimi su 96 išvestimis.
- Lfys48: Lfys64: Dimensiją apibendrinantis LSTM lygis, apibendrinantis y dimensiją su 48 išvestimis.
- Mp3,3: 3x3 *Maxpool* operacija.
- Ct3,3,16: 3x3 konvoluciija su 16 išvesčių ir hiperbolinio tangento aktyvacijos funkcija.
- 1,36,0,1: Įvestis yra juodai baltas paveikslėlis, kurio aukštis 36 pikseliai.

### 3. Neuroninių tinklų apmokymas

Šiame skyriuje aprašomi veiksmai skirti apmokyti neuronius tinklus panaudojant sugeneruotus paveikslėlius.

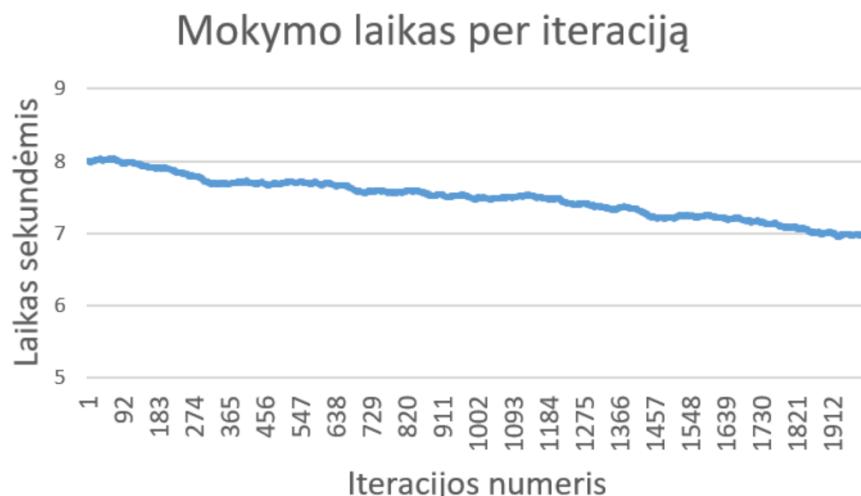
#### 3.1. Konvoliucinio neuroninio tinklo mokymas

Apmokymui buvo naudoti 100.000 paveikslėlių, iš kurių 75.000 sudarė mokymo duomenys, o 25.000 testavimo duomenys. Vienoje iteracijoje buvo apmokoma po 50 paveikslėlių. Kas 20 iteracijų išvedami statistiniai duomenys. Kaip matome 8 paveikslėlyje, mokymosi proceso metu tikslumas priartėjo prie 100% bei pasiekė vidutinį 98% tikslumą mokymo pabaigoje.



8 pav. Loss funkcijos ir tikslumo statistika mokymosi procese

Apmokymas buvo vykdomas su ASUS GeForce GTX 1070 8GB vaizdo plokšte. Apmokyt 100.000 paveikslėlių truko apytiksliai 4h. Vidutiniškai viena iteracija truko apytiksliai 7.2s (9 pav.). Paveikslėlių generavimas buvo vykdomas tuo pačiu metu naudojantis CPU.



9 pav. Neuroninio tinklo mokymo greitis

### **3.2. LSTM rekurentinio neuroninio tinklo mokymas**

Tesseract 4.00 versijoje pridėtas naujas atpažinimo variklis, kuris remiasi LSTM tipo rekurentiniu neuroniniu tinklu. Lyginant su ankstesnėmis versijomis, ženkliai padidėjo dokumentų tipo nuotraukų teksto atpažinimas, tačiau tai reikalauja ženkliai didesnių kompiuterio skaičiavimo resursų. Atpažįstant sudėtingas kalbas, yra didelė tikimybė, kad atpažinimas truks greičiau nei bazinė pirminė Tesseract versija.

Naudojant neuroninius tinklus teksto atpažinimui yra reikalinga žymiai daugiau duomenų modelio treniravimui, taip pat pats treniravimas trunka ilgiau nei pirminėje Tesseract versijoje. Visoms lotynų rašmenimis pagrįstoms kalboms treniravimas vyko naudojant daugiau nei 400.000 teksto eilučių bei apie 4.500 skirtinį šriftą. Su nauja versija ženkliai išaugo mokymosi laikas. Jei su ankstesne versija mokymas trukdavo nuo kelių minučių iki kelių valandų, tai su nauja 4.00 versija tai gali trukti nuo kelių dienų iki kelių savaičių. Tačiau ne visais atvejais yra naudinga treniruoti modelį nuo pradžių, priklausomai nuo situacijos, kartais užtenka pertreniruoti egzistuojantį modelį.

Išskiriame trys pagrindiniai modelio apmokymo principai:

- Esamo modelio patobulinimas. Naudojant egzistuojantį pasirinktos kalbos modelį, papildomai apmokomas su papildomais specifiniais duomenimis. Tai gali išspręsti problemas, kai norimas rezultatas nedaug skiriasi nuo jau apmokyto modelio, pvz.: truputį nestandartinis šriftas. Gali veikti su salyginai mažu naujų duomenų kiekiu.
- Nuimti viršutinį (ar keletą daugiau) modelio sluoksnį ir pertreniruoti naujus sluoksnius su naujais duomenimis. Jei esamo modelio patobulinimas nesprendžia esamos problemas, šis būdas dažniausiai būna kitas pasirinkimas. Viršutinio sluoksnio permokymas vis dar gali veikti treniruojant visiškai naują kalbą, tačiau tos kalbos turi būti labai panašios, kad būtų pasiekta norimas efektas.
- Apmokymas nuo nulio. Tai gali būti labai sunki užduotis, jei nėra pakankamai daug reprezentatyvių duomenų spręsti konkrečiai problemai. Jei duomenų nėra pakankamai daug, galiausiai tinklas bus permokytas, kuris puikiai susidoros tik su mokymo duomenimis, tačiau visiškai neatlikis savo užduoties, kai bus paduodami realūs duomenys. Nors mokymas atrodo skiriasi, matys treniravimo žingsniai yra beveik identiški aukščiau aprašytiems, taigi tai yra visai paprasta išbandyti, atsižvelgiant į turimų duomenų bei kompiuterio resursų kiekį.

### **3.2.1. Atpažinimo kokybės gerinimas**

Egzistuoja įvairiausių priežasčių, kodėl Tesseract atpažinimo programa nesugeba atpažinti jai paduoto teksto. Svarbu pabrėžti, kad Tesseract modelio permokymas retai padės, nebent naudojamas labai nestandardinis šriftas arba nauja dar netreniruota ir neapmokyta kalba.

#### **3.2.1.1. Paveikslėlio apdorojimas**

Pati Tesseract sistema savyje atlieka įvairius paveikslėlių apdorojimo veiksmus, pasinaudojant Leptonica biblioteka, prieš pradedant pati teksto atpažinimą. Dažniausiai Tesseract puikiai susitvarko su šita užduotimi, tačiau neišvengiamai atsiranda situacijų, su kuriuomis automatiškai susidoroti nepavyksta ir dėl to pastebimai nukenčia atpažinimo tikslumas.

Jei norima pamatyti, kaip Tesseract apdorojo paveikslėliuką, tai galima atlikti pakeitus konfigūracinio parametru *tessedit\_write\_images* reikšmę į **true** kai yra leidžiama Tesseract programa. Jei paruoštas tinklo apmokymui paveikslėlis atrodo problematiškai, neišvengiamai reikės pritaikyti viena ar daugiau paveikslėlių apdorojimo technikų prieš siunčiant apdorojimui.

## **Spalvų inversija**

Nors senesnės Tesseract versijos (<= 3.05) palaikė šviesų tekštą ant juodo fono be jokių problemų, nuo 4.00 versijos būtina salyga, kad tekstas būtų juodas, o fonas šviesus. Tam tikslui atlikti užtenka vienos komandos:

```
numpy.invert(image)
```

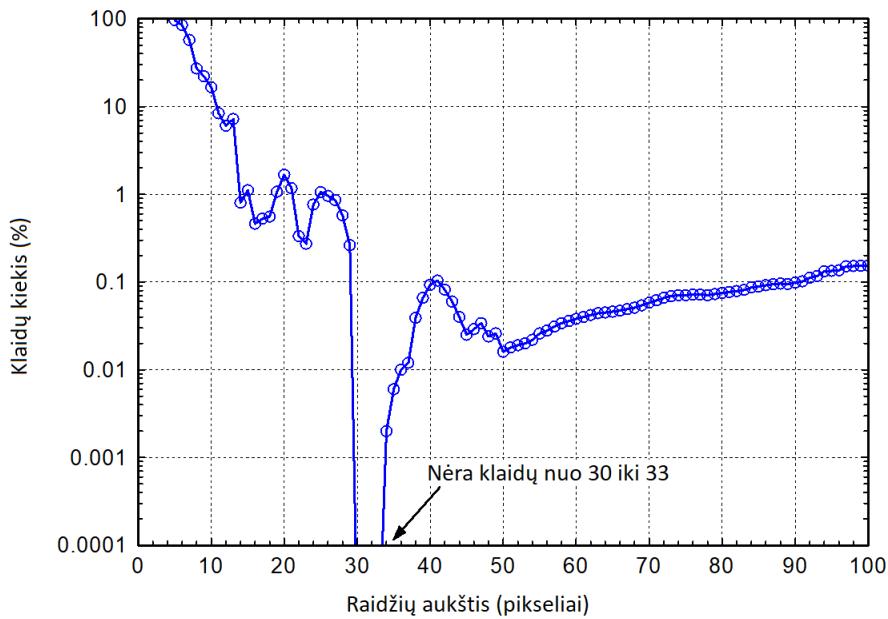
## **Dydžio keitimasis**

Tesseract programa geriausiai veikia, kai paduodamų paveikslėlių taškų viename colyje (angl. DPI) dydis yra bent 300, todėl labai svarbu užtikrinti, kad dydis nebūtų mažesnis.

Atliktas eksperimentas<sup>3</sup> (žiūrėti 10 pav.) parodė, kad egzistuoja optimalus raidžių aukštis, kuriam esant klaidų tikimybė mažėja iki 0. Raidžių aukščiui esant tarp 30 ir 33 pikselių, klaidų tikimybė visiškai sumažėja, todėl galima daryti prielaidą, kad labai svarbu pasirinkti tinkamą šrifto dydį ruošiant mokymo duomenis, norint pasiekti geriausią rezultatą.

---

<sup>3</sup>Willus Dotkom vartotojo atliktas eksperimentas. Šaltinis [https://groups.google.com/forum/#!msg/tesseract-ocr/Wdh\\_JJwnw94/24JHDYQbBQAJ](https://groups.google.com/forum/#!msg/tesseract-ocr/Wdh_JJwnw94/24JHDYQbBQAJ)



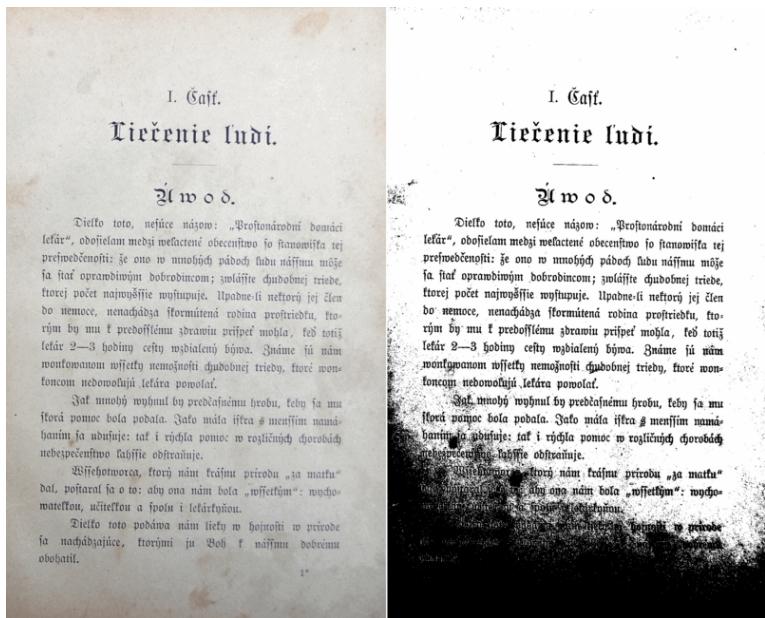
10 pav. Klaidų kiekių priklausomybė nuo raidžių aukščio

## Binarizacija

Binarizacija - tai paveiksluko spalvų keitimas į juodą ir baltą. Tesseract jau turi integruotą funkcionalumą atlikti šiai užduočiai (naudojamas *Otsu* algoritmas), tačiau ne visada rezultatas gaunasi optimalus. Tai dažniausiai lemia netolygus fono tamsumas.

Jei nepavyksta išgauti geresnės kokybės nuotraukos, kuriame fono spalva būtų tolygi, yra alternatyvių ribinių verčių nustatymo algoritmų, kuriuos vertėtų išbandyti:

- *ImageJ* automatinis ribinių verčių nustatymo algoritmas (JAVA programavimo kalba).
- *OpenCV* ribinių verčių nustatymo algoritmas (Python programavimo kalba).
- *scikit-image* ribinių verčių nustatymo algoritmas (Python programavimo kalba).



11 pav. Binarizacijos algoritmo taikymo rezultatas

## Triukšmo pašalinimas

Triukšmas – tai atsitiktinis netolygaus ryškumo išsibarstymas paveikslėlyje, kuris gali padaryti tekštą sunkiai ar visai neįskaitomu. Yra specifiniai triukšmo tipai, kurių Tesseract nesugeba pašalinti vykdyma binarizacijos etapą, todėl ženkliai sumažėja atpažinimo tikslumas.

- θεῶν τὸν πλάνον διέλεγεν; ἀναφανδὸν γὰρ τούτους ἔφησεν  
ὅτις ἀριθμὸς ἀντίπαλος μῆτε θεοῦς μῆτε ἀνθρώπου, δα-  
μονᾶς εἶναι, ἀλλὰ τοῦ φευδόντος διδασκάλοντος καὶ πονηροῖς  
70 πατέρος τούτους ὁ Πλάτων ἐν τῷ Τιμαίῳ οὐδὲ φύσει  
ἀδανάτους φησίν. τὸν γὰρ ποιητὴν εἰρήκεναι πόδες αὐτοὺς  
λέγει· „ἀδανάτοι μὲν οὐκ ἔστε οὖδε ἄλιτοι τὸ παμπαν-  
οῦτι μὲν δὴ λυθῆσθε, τῆς ἡμῆς βουλῆσεως τυχόντες“  
καίτοι γε Ὁμηρος τενάντια δοκεῖ ἀδανάτους γὰρ αὐτοὺς  
πανταχῇ προσονομάζει· „οὐ γὰρ σίτον“ φησίν „ἔδονα“, οὐ  
πίνοντας αἴδοπα οἶνον· τούτην ἀναίμονές εἰσι καὶ ἀδανάτοις  
καλέονται.<sup>10</sup>
- 71 Τοσαύτη παρὰ τοῖς ποιηταῖς καὶ φιλοσόφοις περὶ τῶν  
οὐκ ὄντων μὲν, καλουμένων δὲ θεῶν διαμάχη· τούτοις καὶ  
νεὸς ἔδομήσαντο καὶ βουινὸς προσωποδόμησαν καὶ θυσίας  
ἐξέμησαν καὶ εἰδὴ τινὰ καὶ εἰκάσματα ἐκ ξύλων καὶ λίθων  
καὶ τῶν ἄλλων ὄλων διαγλυφάντες, θεὸν προστυγόσευσαν  
τὰ χειρόσημα εἰδώλα καὶ τὰ τῆς Φειδίου καὶ Πολυκλείτου  
καὶ Προαξιτέλους τέχνης ἀράλματα τῆς θεᾶς προστυγοῖς  
72 ἡγούσαν· τούτον δὲ τοῦ πλάνον κατηγορῶν Σενοφάνης ὁ  
Κολοφώνιος τοιάδε φησίν· „ἄλλ’ οἱ βροτοὶ δοκοῦσι γεννᾶ-  
σθαι θεούς καὶ λογητὸν τὸν αἰσθητὸν ἔχειν φωνὴν τὸ δέμας τε.“  
καὶ πάλιν· „ἄλλ’ εἰ τοι χειρὸς εἶχον βοες ἢ εἰ λέοντες ἢ  
γράφαι χειρέσσει καὶ ἕργα τελεῖν ἀπερ ἀνδρες, ἵπποι μὲν θ-  
ἵπποισι, βοες δέ τε βανσιλεῖ δύοις, καὶ θεῶν ἰδέας ἔργαφον  
καὶ σώματα ἐπολούν τουαῦθ’, ολόνπερ καντοὶ δέμας εἶχον<sup>15</sup>

— 6—7: Eus. Pr. XI: 32, 2. XIII: 18, 10 (Plat. Tim. p. 41B). ||

9—11: Hom. E. 341—342. || 10. π. 89, 1: Clem. Str. V 14, 109

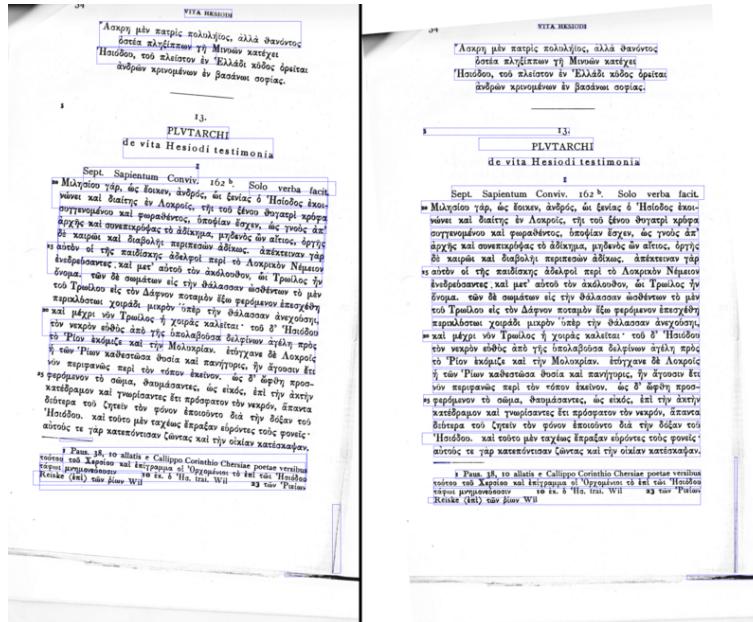
— Eus. Pr. XIII 13, 38 (Xenophan. fr. 14—15)

1 ἔργαφαι· ἔργαφαι, in ἔργαφεν corr. S. |, 7 οὐτι: ὅτι BLS:  
οὐτὲ V | λυθῆσθαι M. corr. ΜΖρ.: λυπηθῆσθαι, L | 8 γε om.  
BLMCV | 9 πανταχοῦ K: πανταχοῦ BL | ἔδουσιν codd. |, οὐ  
(posteriore loco); οὐδὲ BLMCV. | 10 πίγρουσιν codd. |, 12 περι: περι V | 14 νεοῖς M. | ἔδωμησατο BS: ἔδωμησαν K | καὶ θυσίας  
τετιμησαν om. S, sed posuit infra, post λύθων |, 15 ἔδη BL:  
ἔδη K | 17 χειρόσημα MCV | 20 τοιαῦτα BL | βροτοὶ M | 21 οὐ-  
αἰσθητον: ταῖς τιθῆσιν, K | 22 εἰ: ἢ L. e corr. | τοι: τι, V | ἔχον  
K | ἢ λέοντες: ἢ διέργαντες MSCV. | 23 γνήσαι MS | ἔπαν M<sup>1</sup> |  
θ': μεθ' MSC | 24 δέ om. V | εἰδέας BLSC, sed corr. S

12 pav. Pašalintas triukšmas

## Pasukimas / Iškreipimas

Iškreiptas paveikslėlis būna tada, kai yra nuskanuojamas lapas kreivei. Tesseract linijų atpažinimo tikslumas sumažėja jei puslapis nėra visiškai horizontalus, o tai įtakoja patį teksto atpažinimą. Norint išspręsti šią problemą, reikia pakreipti puslapį taip, kad tekslo linijos būtų horizontalios.

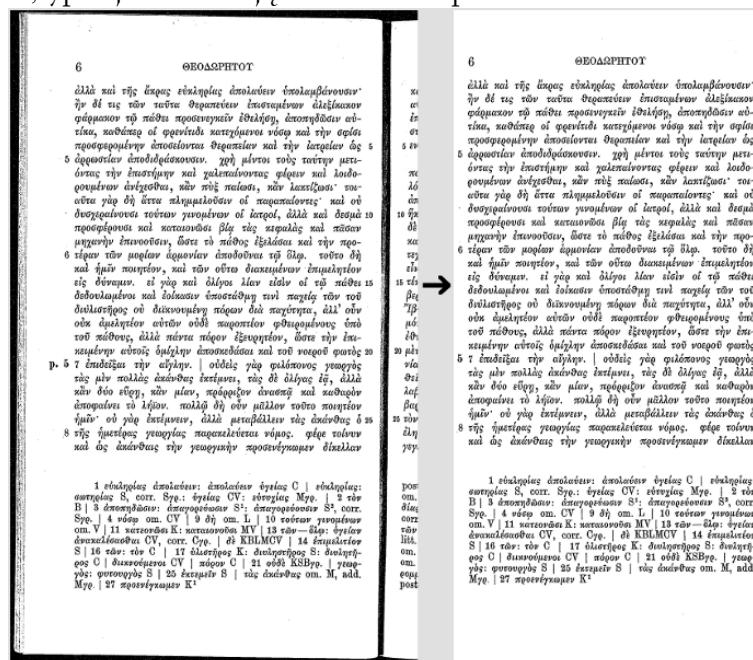


13 pav. Iškreipto puslapio išlyginimas

## Kraštinės

### Skanuotų puslapių kraštinų naikinimas

Skanuoti puslapiai dažnai turi tamsias kraštinės aplinkui tekštą. Tai dažnai gali būti atpažistami kaip papildomi simboliai, ypač jei skiriasi jų formos ir atspalviai.



14 pav. Puslapio kraštinų naikinimas

**Tekstas be kraštinų** Jei norimas atpažinti tekstas visiškai neturių kraštinų ir yra nuo krašto iki krašto, Tesseract programai gali turėti sunkumų bandant atpažinti tekštą. Panaudojant vieną komandą, lengvai galima pridėti kraštines iš visų pusių (naudojama *ImageMagick®* programa):

```
convert input.jpg -bordercolor White -border 10x10 output.jpg
```

## Permatomumas / alfa kanalas

Kai kurie paveikslėlių formatai (pvz.: png) turi alfą kanalą, kuris suteikia galimybę saugoti permatomumo reikšmę nuotraukoje. Alfa kanalu dažniausiai nusakomas paveikslėlio skaidrumas. Paprastai prie 24 nuotraukos bitų, kuriuose kiekvienai iš trijų pagrindinių spalvų skiriama po 8 bitus, pridedami papildomi 8 bitai, kurie saugo skaidrumo informaciją.

Tesseract 3.0x versijos tikisi, kad pats vartotojas pateiks paveiksluką jau su panaikinta alfa kanalu. Tai gali būti padaroma su tokia komanda (naudojama *ImageMagick®* programa):

```
convert input.png -alpha off output.png
```

Tesseract 4.00 versijoje yra funkcionalumas, kuris pats pašalina alfa kanalą naudojant *Leptonica* programos komandą *pixRemoveAlpha()*. Ši komanda panaikina alfa kanalą suliedama ji su balto fonu. Kartais (pvz.: filmų subtitrų atpažinimas) tai gali sukelti problemų, todėl vartotojai turėtų patys panaikinti alfa kanalą arba pritaikyti spalvų inversiją.

### 3.2.1.2. Puslapio skirstymo metodas

Tesseract programos standartinis veikimo principas pagristas tuo, kad programa tikisi paveikslėlio puslapio pavidalu su tame esančiu tekstu. Tačiau, jei norima atpažinti tik dalį teksto, yra įvairiausių teksto skirstymo parametrų, kurį reikia nurodyti naudojant komandą *-psm* ir nurodant komandos numerį.

0. Orientacija ir rašto aptikimas (OSD).
1. Automatinis puslapio skirstymas su rašto aptikimu (OSD).
2. Automatinis puslapio skirstymas, bet be rašto aptikimo (OSD) ir be simbolių atpažinimo (OCR).
3. Pilnai automatinis puslapio skirstymas, bet be rašto aptikimo (OSD) (Numatytais režimas).
4. Vienas teksto stulpelis.
5. Vienas vertikaliai išlygiuoto teksto blokas.

6. Vienas teksto blokas.
7. Paveikslėli laikyti kaip vieną teksto liniją.
8. Paveikslėli laikyti kaip vieną žodį.
9. Paveikslėli laikyti kaip vieną žodį apskritime.
10. Paveikslėli laikyti kaip vieną simbolį.
11. Išmėtytas tekstas. Rasti kuo daugiau teksto nesilaikant jokios tvarkos.
12. Atpažinti išmėtytą tekštą su rašto aptikimu (OSD).
13. Neapdorota eilutė. Paveikslėli laikyti kaip vieną teksto liniją, išvengiant specifinių Tesseract gudrybių.

### **3.2.1.3. Žodynai, žodžių sąrašai, šablonai**

Tesseract programa optimizuota taip, kad geriausiai atpažintų sakinius, susidarančius iš žodžių. Jei yra bandoma atpažinti nestandardinės struktūros tekstus (pvz.: saskaitas, čekius, prekių sąrašus, kodus), yra keletas papildomų būdų, kaip būtų galima pagerinti atpažinimo tikslumą.

Pirmiausiai reikia įsitikinti, kad yra pasirinktas tinkamas puslapio skirstymo būdas. Tai užtikrina, kad bus efektyviausiai ieškoma teksto.

Žodynų atjungimas, kuriuos naudoja Tesseract turėtų pagerinti atpažinimą, jei dauguma teksto nėra žodyne esantys žodžiai. Norint išjungti funkcionalumą, kai naudojami Tesseract žodynai, reikia nurodyti *FALSE* reikšmę šiems konfigūraciniams parametrams: *load\_system\_dawg* ir *load\_freq\_dawg*.

Taip pat yra galimybė pačiam vartotojui prisiidėti norimus žodžius į Tesseract programą, kurie padės atpažinimo varikliui geriau suprasti žodžius. Be žodžių, yra galimybė prisiidėti simbolių sekų šablonus, kurie dar labiau padės pagerinti tikslumą.

### **3.2.2. Rinkmenų pasiruošimas**

Bendrai mokymo žingsniai yra tokie:

1. Pasiruošti norimą apmokyti tekštą.
2. Sugeneruoti paveiksluką su tekstu + *box* failu.

3. Sukurti *unicharset* failą.
4. Iš *unicharset* sukurti pradinę apmokymo duomenų failą ir nebūtiną žodynų informaciją.
5. Paleisti *Tesseract*, kad apdorotų paveikslėli ir *box* failą bei sukurtų apmokymo duomenų rinkini.
6. Paleisti treniravimą su sukurtu duomenų rinkiniu.
7. Sujungti duomenų failus.

Norint atlikti LSTM rekurentinio tinklo mokymą *Tesseract* 4.0 versijos aplinkoje, reikia susigeneruoti atitinkamo formato mokymo rinkmenas. Kiekvienas sugeneruotas automobilio numeris turi turėti 5 skirtingus rinkmenas:

- .box formato – rinkmena, kurią sugeneruoja priede Nr. 1 pridėta programa.
- .lstmf formato – rinkmena, kurią sugeneruoja priedo Nr. 3 komanda: lists.
- .gt.txt formato – tekstinė rinkmena, kurioje yra tekstas, kuris yra pavaizduotas paveikslėlyje.
- .txt formato – rinkmena, kurią sugeneruoja priedo Nr. 3 komanda: lists.
- .tif formato – paveiksliukas, išsaugotas TIFF formatu.

### **3.2.3. Modelio apmokymas**

Norint paleisti modelio apmokymą, reikia paleisti komandą:

```
make training MODEL_NAME=modelis
```

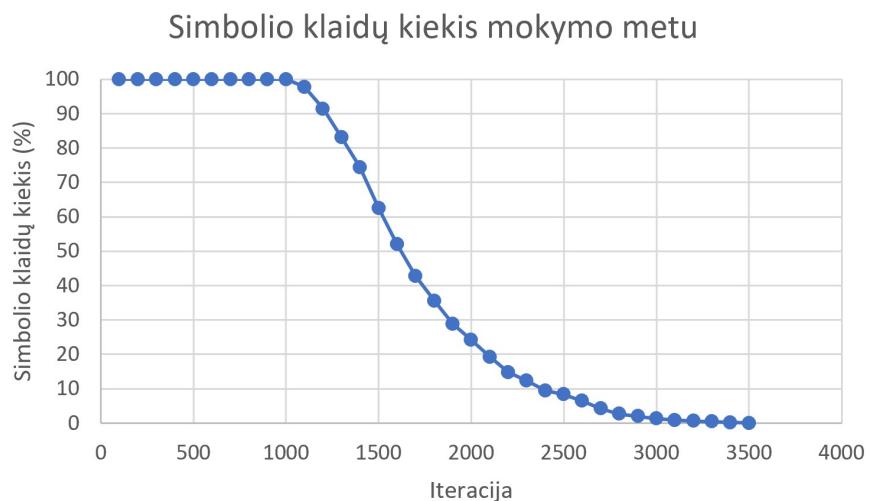
Ši komanda yra trumpinys pilnos komandos:

```
make unicharset lists proto-model training MODEL_NAME=modelis
```

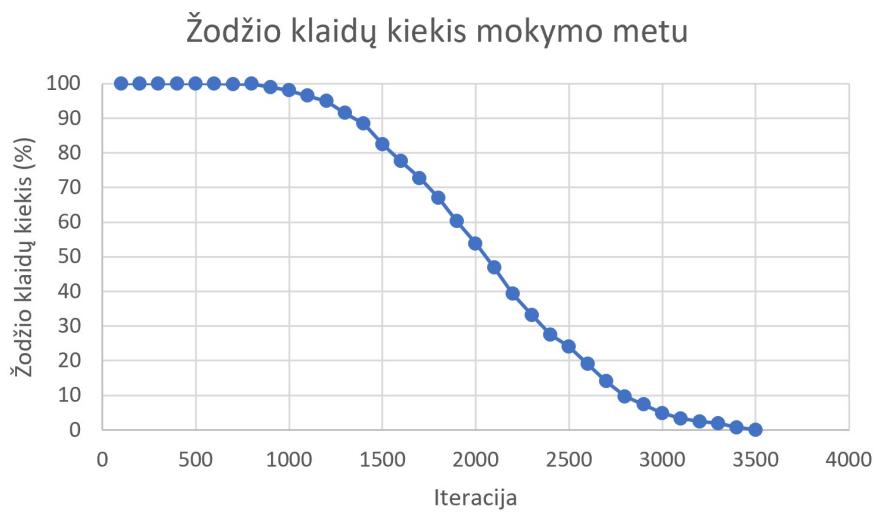
Vykdomos operacijos:

- *unicharset* – Sukurti galimų simbolių sąrašą.
- *lists* – Sukurti sąrašą *.lstmf* failų ir išskirstyti juos į treniravimo ir verifikavimo.
- *training* – Pradėti mokymą.
- *proto-model* – Sugeneruoti modelį.

### 3.2.4. Mokymo statistika



15 pav. Simbolio klaidų kiekis mokymo metu



16 pav. Simbolio klaidų kiekis mokymo metu

## 4. Vaizdo atpažinimas

Šiame skyriuje aprašoma kaip panaudojami apmokyti neuroniniai tinklai skirti atlikti jems paskirtas užduotis.

### 4.1. Numerio rēmelio atpažinimas

Norint aptikti ir atpažinti realiuose paveikslėliuose numerio rēmelį, į neuroninį tinklą paduodamos 128x64 pikselių dydžio paveikslėlio dalys, kaip jau buvo aprašyta 1.1 skyriuje. Atpažstant realius paveikslėlius, naudojama kitokia neuroninio tinklo architektūra. Paskutiniai du lygmenys vietoj to, kad būtų pilnai sujungti, yra konvoluciniai. Taip pat pradinio paveikslėlio dydis neturi būti 128x64 pikselių, o gali būti bet koks. Idėja tokia, kad pilno dydžio paveikslėlis gali būti paduodamas į neuroninį tinklą suskaidant jį į dalis slenkančio lanelio principu, bei kiekvienai iš jų grąžinant rezultatą, ar rēmėlis egzistuoja. Naudojant vienodą neuroninį tinklą visoms paveikslėlio dalims yra pranašesnis nei atskiri neuroniniai tinklai, kadangi slenkantys langai dalinsis dauguma konvoluciinių savybių tarpusavyje, todėl nereikės kiekvieną kart atlikti naujų skaičiavimo operacijų.

Slenkančio lango principu veikiančio neuroninio tinklo rezultatai:



17 pav. Slenkančio lango principu gauti rezultatai

Žali stačiakampiai (17 pav.) vaizduoja regionus kur tikimybė, kad rēmeliis egzistuoja yra didesnė arba lygi 99%. Tai padaryta tokiu tikslu, kadangi mokymo duomenų aibėje apie 50% paveikslėlių yra su egzistuojančiu numerio rēmeliu, kai realiame pasaulyje paveikslėlių su numerio rēmeliais yra daug mažiau. Jeigu būtų naudojama 50% tikimybė atrinkti teisingiemis paveikslė-

liams, tai būtų neapsisaugota nuo pasitaikančių panašių paveikslėlių atitikmenų.

Norint panaikinti perteklinius dublikatus, pritaikomas Non-Maximum Suppresion<sup>4</sup> algoritmas, kuris tarp visų besikertančių stačiakampių palieka tik didžiausią tikimybę turinčią reikšmę [girshick2014rich].

Gavus likusį vieną stačiakampį (18 pav.), pagal to objekto koordinates iškerpamas paveikslėlis ir gaunamas toks rezultatas:



18 pav. Iškirptas gautas rezultatas

## 4.2. Numerio simbolių atpažinimas

Sukurta programa, kuri naudodama apmokytą LSTM pagrindu veikiantį neuroninį tinklą atpažįsta automobilio numerius (žiūrėti priede Nr. 4). Atpažinimo pavyzdžius galima pamatyti 19 pav., 20 pav., 21 pav., 22 pav., 23 pav., 24 pav., 25 pav.

Pavyzdžiai:



19 pav. Tikimasis rezultatas **DGZ473**.



20 pav. Tikimasis rezultatas **EV0741**.

---

<sup>4</sup>Non-Maximum Suppresion angl. - ne maksimalios reikšmės slopinimo algoritmas



21 pav. Tikimasis rezultatas **FON550**.



22 pav. Tikimasis rezultatas **JPC320**.



23 pav. Tikimasis rezultatas **KTP049**.



24 pav. Tikimasis rezultatas **KZF273**.



25 pav. Tikimasis rezultatas **LAN285**.

## **Rezultatai**

1. Sugeneruota 100.000 paveikslėlių, skirtų apmokyti neuroninį tinklą.
2. Apmokytas kursinio darbo metu sukurtas neuroninis tinklas su 100.000 sugeneruotų paveikslėlių.
3. Apmokytas modifikuotas Tesseract LSTM rekurentinis neuroninis tinklas panaudojant sugeneruotus paveikslėlius.
4. Parašyta programa, kuri gauna rėmelio koordinates iš paveikslėlio, kuriame yra numeris, pasinaudojus neuroniniu tinklu.
5. Parašyta programa, kuri iškerpa rėmelį pagal koordinates ir pasinaudojus modifikuotu Tesseract LSTM rekurentiniu neuroniniu tinklu atpažista numerį bei atvaizduoja pradiniai paveikslėlyje.
6. Tinklas ištstuotas su tikrais paveikslėliais, kuriose yra lietuviški automobilių numeriai.

## **Išvados**

Darbe buvo nagrinėta kaip išspręsti vaizdo atpažinimo problemą naudojantis neuroniniais tinklais.

Išnagrinėta kaip veikia konvolucinis neuroninis tinklas, kaip generuoti duomenis jo apmokymui, kaip sudaryti neuroninio tinklo lygius, kaip apmokyti tinklą su sugeneruotais duomenimis bei kaip ji pritaikyti sukurtoje programe, kurį geba iškirpti numerio rėmelį nuotraukoje.

Padaryta išvada, kad įmanoma sukurti konvolucinį neuroninį tinklą, kuris geba nuotraukoję identifikuoti numerio rėmelį bei ji iškirpti. Norint efektyvinti atpažinimą, reiktų išbandyti daugiau neuroninių tinklų rūsių bei rasti efektyviausią.

Taip pat nagrinėtas rekurentinis neuroninis tinklas, o tiksliau, Tesseract atviro kodo programos 4.00 versija. Panaudotas naujas metodas, kuris naudoja LSTM architektūros neuroninį tinklą. Sudarytas neuroninis tinklas bei parašyta programa kuri geba atpažinti raides ir skaičius.

Padaryta išvada, kad Tesseract LSTM yra tinkamas įrankis atpažinti tekstą nuotraukose, turi daug konfigūracijos galimybų ir pakankamai lengvai galima susikurti neuroninio tinklo modelį.

## **Savokų apibrėžimai**

- Tesseract - optinė ženklu atpažinimo programa, kuri geba naudoti neuroninius tinklus atpažinimui.
- Leptonica
- Tensorflow
- OpenMP
- Tenzorius - geometrinis objektas, susidedantis iš sumos komponenčių, kurios yra transformojamos pagal tiesinius sąryšius.
- Softmax
- Relu
- Plumbing
- Maxpool

## **Santrumpos**

-TODO

- LSTM - trumpinys angl. Long short-term memory - rekurentinio neuroninio tinklo architektūra.
- TIFF
- DPI
- PNG
- OSD
- OCR
- VGSL
- SSE

- AVX
- SIMD
- CTC -

## Priedas Nr. 1

### Programa skirta sugeneruoti .box failus

```
1 #!/usr/bin/env python
2
3 import io
4 import argparse
5 import unicodedata
6 from PIL import Image
7
8 #
9 # command line arguments
10 #
11 arg_parser = argparse.ArgumentParser( '''Creates tesseract box files for given
12 (line) image text pairs ''' )
13
14 # Text ground truth
15 arg_parser.add_argument( '-t', '--txt', nargs='?', metavar='TXT', help='Line
16 text (GT)', required=True )
17
18 # Image file
19 arg_parser.add_argument( '-i', '--image', nargs='?', metavar='IMAGE', help='
20 Image file', required=True )
21
22 args = arg_parser.parse_args()
23
24 #
25 # load image
26 with open(args.image, "rb") as f:
27     width, height = Image.open(f).size
28
29 # load gt
30 with io.open(args.txt, "r", encoding='utf-8') as f:
31     lines = f.read().strip().split('\n')
32
```

```
33 for line in lines:
34     if line.strip():
35         for i in range(1, len(line)):
36             char = line[i]
37             prev_char = line[i-1]
38             if unicodedata.combining(char):
39                 print(u"%s %d %d %d %d 0" % ((prev_char + char), 0, 0, width,
height))
40             elif not unicodedata.combining(prev_char):
41                 print(u"%s %d %d %d %d 0" % (prev_char.encode("utf-8"), 0, 0,
width, height))
42             if not unicodedata.combining(line[-1]):
43                 print(u"%s %d %d %d %d 0" % (line[-1], 0, 0, width, height))
44             print(u"%s %d %d %d %d 0" % ("\t", width, height, width+1, height+1))
```

## Priedas Nr. 2

### Programa atsitiktinai generuojant automobilių numerius

```
1 import numpy as np
2 from PIL import Image as img
3 from os import listdir
4 from os.path import isfile, join
5 from random import randint
6 import os.path
7 import numpy
8
9 letters_path = "d:\\bakalaurinis\\generating\\images\\letters"
10 numbers_path = "d:\\bakalaurinis\\generating\\images\\numbers"
11 test_data_path = "D:\\BAKALAURINIS\\ocrd-train\\data\\"
12
13 letters = [join(letters_path, f) for f in listdir(letters_path) if isfile(join(letters_path, f))]
14 numbers = [join(numbers_path, f) for f in listdir(numbers_path) if isfile(join(numbers_path, f))]
15
16 imgs = [img.open(i) for i in letters + numbers]
17
18 def generate(symbols, name):
19     min_shape = sorted([(np.sum(i.size), i.size) for i in symbols])[0][1]
20     imgs_comb = np.hstack((np.asarray(i.resize(min_shape)) for i in symbols))
21     imgs_comb = img.fromarray(imgs_comb)
22     imgs_comb.save(test_data_path + name + ".tif")
23     with open(test_data_path + name + ".gt.txt", "w") as text_file:
24         print(name.upper(), file=text_file)
25
26 for x in range(0, 100000):
27     letter_1 = imgs[randint(0, len(letters)-1)]
28     letter_2 = imgs[randint(0, len(letters)-1)]
29     letter_3 = imgs[randint(0, len(letters)-1)]
30     number_1 = imgs[randint(21, len(numbers)+len(letters)-1)]
31     number_2 = imgs[randint(21, len(numbers)+len(letters)-1)]
32     number_3 = imgs[randint(21, len(numbers)+len(letters)-1)]
```

```
33 symbols = [letter_1, letter_2, letter_3, number_1, number_2, number_3]
34 name = os.path.basename(letter_1.filename).split('.')[0] +
35     os.path.basename(letter_2.filename).split('.')[0] +
36     os.path.basename(letter_3.filename).split('.')[0] +
37     os.path.basename(number_1.filename).split('.')[0] +
38     os.path.basename(number_2.filename).split('.')[0] +
39     os.path.basename(number_3.filename).split('.')[0]
40 generate(symbols, name)
```

## Priedas Nr. 3

### Makefile failas skirtas treniruoti Tesseract 4.00 versijos neuroninių tinklą

```
1 export
2
3 SHELL := /bin/bash
4 LOCAL := $(PWD)/usr
5 PATH := $(LOCAL)/bin:$(PATH)
6 TESSDATA = $(LOCAL)/share/tessdata
7
8 # Name of the model to be built. Default: $(MODEL_NAME)
9 MODEL_NAME = foo
10
11 # Name of the model to continue from. Default: '$(START_MODEL)'
12 START_MODEL =
13
14 LAST_CHECKPOINT = data/checkpoints/$(MODEL_NAME)_checkpoint
15
16 # Name of the proto model. Default: '$(PROTO_MODEL)'
17 PROTO_MODEL = data/$(MODEL_NAME)/$(MODEL_NAME).traineddata
18
19 # No of cores to use for compiling leptonica/tesseract. Default: $(CORES)
20 CORES = 4
21
22 # Leptonica version. Default: $(LEPTONICA_VERSION)
23 LEPTONICA_VERSION := 1.75.3
24
25 # Tesseract commit. Default: $(TESSERACT_VERSION)
26 TESSERACT_VERSION := fd492062d08a2f55001a639f2015b8524c7e9ad4
27
28 # Tesseract model repo to use. Default: $(TESSDATA_REPO)
29 TESSDATA_REPO = _best
30
31 # Ground truth directory. Default: $(GROUND_TRUTH_DIR)
32 GROUND_TRUTH_DIR := data/ground-truth
33
34 # Normalization Mode - see src/training/language_specific.sh for details.
```

```

    Default: $(NORM_MODE)

35 NORM_MODE = 2

36

37 # Page segmentation mode. Default: $(PSM)
38 PSM = 6

39

40 # Ratio of train / eval training data. Default: $(RATIO_TRAIN)
41 RATIO_TRAIN := 0.90

42

43 # BEGIN-EVAL makefile-parser --make-help Makefile

44

45 help:
46     @echo """
47     @echo " Targets"
48     @echo """
49     @echo "      unicharset          Create unicharset"
50     @echo "      lists                 Create lists of lstmf filenames for training and
51           eval"
52     @echo "      training             Start training"
53     @echo "      proto-model         Build the proto model"
54     @echo "      leptonica            Build leptonica"
55     @echo "      tesseract            Build tesseract"
56     @echo "      tesseract-langs     Download tesseract-langs"
57     @echo "      clean                Clean all generated files"
58     @echo """
59     @echo " Variables"
60     @echo """
61     @echo "      MODEL_NAME          Name of the model to be built. Default: $(
62           MODEL_NAME)"
63     @echo "      START_MODEL         Name of the model to continue from. Default: '$(
64           START_MODEL)'"
65     @echo "      PROTO_MODEL         Name of the proto model. Default: '$(
66           PROTO_MODEL)'"
67     @echo "      CORES               No of cores to use for compiling leptonica/
68           tesseract. Default: $(CORES)"
69     @echo "      LEPTONICA_VERSION   Leptonica version. Default: $(
70           LEPTONICA_VERSION)"
71     @echo "      TESSERACT_VERSION   Tesseract commit. Default: $(TESSERACT_VERSION
72           )"

```

```

66 @echo "      TESSDATA_REPO      Tesseract model repo to use. Default: $(  

67   TESSDATA_REPO)"  

68 @echo "      GROUND_TRUTH_DIR    Ground truth directory. Default: $(  

69   GROUND_TRUTH_DIR)"  

70 @echo "      NORM_MODE          Normalization Mode - see src/training/  

71   language_specific.sh for details. Default: $(NORM_MODE)"  

72 @echo "      PSM                Page segmentation mode. Default: $(PSM)"  

73 @echo "      RATIO_TRAIN        Ratio of train / eval training data. Default:  

74   $(RATIO_TRAIN)"  

75  

76 # END-EVAL  

77  

78 ALL_BOXES = data/all-boxes  

79 ALL_LSTMF = data/all-lstmf  

80  

81 # Create unicharset  

82 unicharset: data/unicharset  

83  

84 # Create lists of lstmf filenames for training and eval  

85 lists: $(ALL_LSTMF) data/list.train data/list.eval  

86  

87 data/list.train: $(ALL_LSTMF)  

88   total='cat $(ALL_LSTMF) | wc -l' \  

89   no='echo "$total * $(RATIO_TRAIN) / 1" | bc'; \  

90   head -n "$no" $(ALL_LSTMF) > "$@"  

91  

92 data/list.eval: $(ALL_LSTMF)  

93   total='cat $(ALL_LSTMF) | wc -l' \  

94   no='echo "($total - $total * $(RATIO_TRAIN)) / 1" | bc'; \  

95   tail -n "$no" $(ALL_LSTMF) > "$@"  

96  

97 # Start training  

98 training: data/$(MODEL_NAME).traineddata  

99  

100 ifdef START_MODEL  

101 data/unicharset: $(ALL_BOXES)  

102   mkdir -p data/$(START_MODEL)  

103   combine_tessdata -u $(TESSDATA)/$(START_MODEL).traineddata data/$(  

104     START_MODEL)/$(START_MODEL)

```

```

100 unicharset_extractor --output_unicharset "$(GROUND_TRUTH_DIR)/my.unicharset"
101   --norm_mode $(NORM_MODE) "$(@)"
102 merge_unicharses data/$(START_MODEL)/$(START_MODEL).lstm-unicharset $(
103   GROUND_TRUTH_DIR)/my.unicharset "$@"
104 else
105 data/unicharset: $(ALL_BOXES)
106   unicharset_extractor --output_unicharset "$@" --norm_mode 1 "$(@)"
107 endif
108
109 $(ALL_BOXES): $(sort $(patsubst %.tif,%.box,$(wildcard $(GROUND_TRUTH_DIR)/*.
110   tif)))
111   find $(GROUND_TRUTH_DIR) -name '*.*.box' -exec cat {} \; > "$@"
112
113 $(GROUND_TRUTH_DIR)/*.box: $(GROUND_TRUTH_DIR)/*.tif $(GROUND_TRUTH_DIR)/*.gt.
114   txt
115   python generate_line_box.py -i "$(GROUND_TRUTH_DIR)/*.*.tif" -t "$(GROUND_TRUTH_DIR)/*.*.gt.txt" > "$@"
116
117 $(ALL_LSTMF): $(sort $(patsubst %.tif,%.lstmf,$(wildcard $(GROUND_TRUTH_DIR)/*
118   *.tif)))
119   find $(GROUND_TRUTH_DIR) -name '*.*.lstmf' -exec echo {} \; | sort -R -o "$@"
120
121 $(GROUND_TRUTH_DIR)/*.*.lstmf: $(GROUND_TRUTH_DIR)/*.*.box
122   tesseract $(GROUND_TRUTH_DIR)/*.*.tif $(GROUND_TRUTH_DIR)/*.* --psm $(PSM)
123   lstm.train
124
125 # Build the proto model
126 proto-model: $(PROTO_MODEL)
127
128 $(PROTO_MODEL): data/unicharset data/radical-stroke.txt
129   combine_lang_model \
130     --input_unicharset data/unicharset \
131     --script_dir data/ \
132     --output_dir data/ \
133     --lang $(MODEL_NAME)
134
135 ifdef START_MODEL
136 $(LAST_CHECKPOINT): unicharset lists $(PROTO_MODEL)
137   mkdir -p data/checkpoints

```

```

132 lstmtraining \
133   --traineddata $(PROTO_MODEL) \
134     --old_traineddata $(TESSDATA)/$(START_MODEL).traineddata \
135   --continue_from data/$(START_MODEL)/$(START_MODEL).lstm \
136   --net_spec "[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c'head -n1 data/unicharset']" \
137   --model_output data/checkpoints/$(MODEL_NAME) \
138   --learning_rate 20e-4 \
139   --train_listfile data/list.train \
140   --eval_listfile data/list.eval \
141   --max_iterations 100000
142 else
143 $(LAST_CHECKPOINT): unicharset lists $(PROTO_MODEL)
144   mkdir -p data/checkpoints
145   lstmtraining \
146     --traineddata $(PROTO_MODEL) \
147     --net_spec "[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c'head -n1 data/unicharset']" \
148     --model_output data/checkpoints/$(MODEL_NAME) \
149     --learning_rate 20e-4 \
150     --train_listfile data/list.train \
151     --eval_listfile data/list.eval \
152     --max_iterations 100000
153 endif
154
155 data/$(MODEL_NAME).traineddata: $(LAST_CHECKPOINT)
156   lstmtraining \
157   --stop_training \
158   --continue_from $(LAST_CHECKPOINT) \
159   --traineddata $(PROTO_MODEL) \
160   --model_output $@
161
162 data/radical-stroke.txt:
163   wget -O$@ 'https://github.com/tesseract-ocr/langdata_lstm/raw/master/radical-stroke.txt'
164
165 # Build leptonica
166 leptonica: leptonica.built
167

```

```

168 leptonica.built: leptonica-$(LEPTONICA_VERSION)
169   cd $< ; \
170     ./configure --prefix=$(LOCAL) && \
171       make -j$(CORES) && \
172       make install && \
173       date > "$@"
174
175 leptonica-$(LEPTONICA_VERSION): leptonica-$(LEPTONICA_VERSION).tar.gz
176   tar xf "$<"
177
178 leptonica-$(LEPTONICA_VERSION).tar.gz:
179   wget 'http://www.leptonica.org/source/$@'
180
181 # Build tesseract
182 tesseract: tesseract.built tesseract-langs
183
184 tesseract.built: tesseract-$(TESSERACT_VERSION)
185   cd $< && \
186     sh autogen.sh && \
187       PKG_CONFIG_PATH="$(LOCAL)/lib/pkgconfig" \
188       LEPTONICA_CFLAGS="-I$(LOCAL)/include/leptonica" \
189       ./configure --prefix=$(LOCAL) && \
190       LDFLAGS="-L$(LOCAL)/lib" \
191         make -j$(CORES) && \
192         make install && \
193         make -j$(CORES) training-install && \
194         date > "$@"
195
196 tesseract-$(TESSERACT_VERSION):
197   wget https://github.com/tesseract-ocr/tesseract/archive/$(TESSERACT_VERSION).zip
198   unzip $(TESSERACT_VERSION).zip
199
200 # Download tesseract-langs
201 tesseract-langs: $(TESSDATA)/eng.traineddata
202
203 $(TESSDATA)/eng.traineddata:
204   cd $(TESSDATA) && wget https://github.com/tesseract-ocr/tessdata$(
205     TESSDATA_REPO)/raw/master/$(notdir $@)

```

```
205  
206 # Clean all generated files  
207 clean:  
208     find $(GROUND_TRUTH_DIR) -name '*.box' -delete  
209     find $(GROUND_TRUTH_DIR) -name '*.lstmf' -delete  
210     rm -rf data/all/*  
211     rm -rf data/list.*  
212     rm -rf data/${MODEL_NAME}  
213     rm -rf data/unicharset  
214     rm -rf data/checkpoints
```

## Priedas Nr. 4

### Programa atpažīstanti automobilio numerius

```
1 import cv2
2 import numpy as np
3 import glob
4 import pytesseract
5 import time
6 import sys
7
8 def detect(image):
9     gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
10    threshGauss = cv2.adaptiveThreshold(gray, 255, cv2.
11                                         ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 51, 27)
12    ratio = 200.0 / image.shape[1]
13    dim = (200, int(image.shape[0] * ratio))
14    resizedCubic = cv2.resize(threshGauss, dim, interpolation=cv2.INTER_CUBIC)
15    bordersize = 10
16    border = cv2.copyMakeBorder(resizedCubic, top=bordersize, bottom=
17                                bordersize, left=bordersize, right=bordersize, borderType=cv2.
18                                BORDER_CONSTANT, value=[255, 255, 255])
19    edges = cv2.Canny(border, 50, 150, apertureSize=3)
20    cv2.imshow('d', border)
21    cv2.waitKey(1000)
22    lines = cv2.HoughLinesP(image=edges, rho=1, theta=np.pi / 180, threshold
23                           =100, lines=np.array([]), minLineLength=100, maxLineGap=80)
24    a, b, c = lines.shape
25    for i in range(a):
26        x = lines[i][0][0] - lines[i][0][2]
27        y = lines[i][0][1] - lines[i][0][3]
28        if x != 0:
29            if abs(y / x) < 1:
30                cv2.line(border, (lines[i][0][0], lines[i][0][1]), (lines[i]
31                           [0][2], lines[i][0][3]), (255, 255, 255), 1, cv2.LINE_AA)
32
33    se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
34    gray = cv2.morphologyEx(border, cv2.MORPH_CLOSE, se)
```

```

31 # OCR
32 config = ''
33 text = pytesseract.image_to_string(gray, lang='training', config=config)
34 validChars = [ 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
35   'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'V', 'Y', 'Z', '0', '1', '2', '3',
36   '4', '5', '6', '7', '8', '9' ]
37 print(text)
38 cleanText = []
39
40 for char in text:
41     if char in validChars:
42         cleanText.append(char)
43
44 plate = ''.join(cleanText)
45 return plate
46
47 start = time.time()
48 plate = detect(cv2.imread(sys.argv[1]))
49 print(plate)
50 print(round(time.time() - start, 2), 'ms')

```