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: Emilis Ruzveltas (parašas)

Darbo vadovas: dr. Vytautas Valaitis (parašas)

Recenzentas: lekt. Tomas Smagurauskas (parašas)

Vilnius
2019

Santrauka

Summary

TURINYS

Priedas Nr.2
Priedas Nr.3
Priedas Nr.4

Įvadas

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, tokių kaip automobilių patikra, automatinis kelių mokesčių surinkimas, išmanus eismo reguliavimas [BSS13]. Didžioji dauguma automobilio numerių atpažinimo sistemų remiasi optine ženklų atpažinimo sistema. Jų apdorojimo greitis yra pakankamai greitas, kad būtų efektyviai išnaudojama įvairiose srityse. Tačiau dažniausiai yra kuriamos specializuotos atpažinimo programos skirtingiems regionams. Panaudojus neuroninius tinklus galima būtų apmokyti atpažinti numerius, kurių formatai yra skirtingi. 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 [LS16]. Šiame darbe naudosime kursinio darbo metu sukurtą konvoliucinį 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 [Smi07].

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 konvoliucinį 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 konvoliuciniu 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. Analizuojant bei tobulinant dirbtinio intelekto sistemas, galima pasiekti greitesnių bei efektyvesnių rezultatų nei naudojant tradicinius atpažinimo metodus.

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

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

Darbo atlikimo procesas

Pirmiausia bus gilinamasi į Tesseract LSTM neuroninio tinklo veikimo principus [BSS13]. Išanalizavus, bus bandoma apmokyti neuroninį tinklą su kursinio darbo metu sugeneruotais paveikslėliais. Atlikus apmokymą, reikės analizuoti ir gerintį 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 konvoliucinį neuroninį tinklą 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. Rėmelio 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 pajėgus suprasti paveikslėlio turinį, o dydis pakankamai mažas, kad būtų galima turėti efektyviai veikiantį neuroninį tinklą.

Pirmoji paveikslėlio rezultato dalis nurodo, koks yra teisingas numeris. Antroji - numerio rėmelio egzistavimas paveikslėlyje. Jei reikšmė 1 - numeris yra tinkamas nuskaitymui, 0 - neatitinka kriterijų (2 pav.).

Kriterijai, kurie nusako ar numerio rėmelis yra tinkamas apdorojimui:

- Visas rėmelio plotas yra paveikslėlyje.
- Rėmelio plotis yra mažesnis nei 80% paveikslėlio pločio.
- Rėmėlio aukštis yra mažesnis nei 87,5% paveikslėlio aukščio.
- Rėmelio plotis yra didesnis nei 60% paveikslėlio pločio.
- Rėmėlio 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 rėmelį $\sqrt{2}$ 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**.

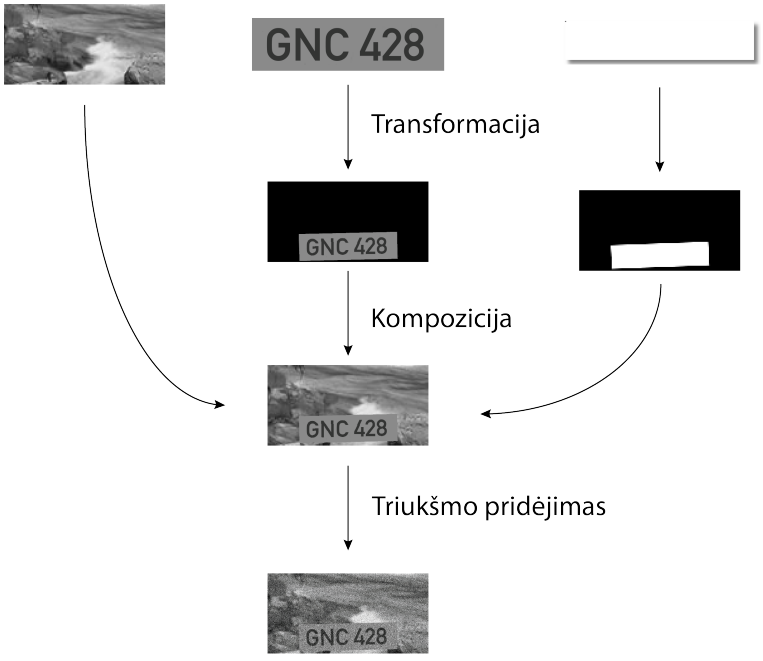
(c) Tikimasis rezultatas **COH150 0** (per mažas numeris).

(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 rėmelio spalva generuojama atsitiktinai, tačiau tekstas turi būti tamsesnis negu rėmelis. 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 raides) 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 mokytųsi ir atpažintų paveikslėlius realaus pasaulio sąlygomis, generuojant duomenų rinkinį buvo pritaikyta remėlio transformacija. Tai atlikti buvo pasitelktas metodas generuoti atsitiktines reikšmes X, Y, Z ašims ir pritaikyti Oilerio kampų metodą[Sla99]. Reikšmių rėžiai pasirinkti tokie, kuriuos labiausiai tikėtina sutikti 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 ašį:
    - Apskaičiuojamos cos(Y) ir sin(y) reikšmės,
    - Sudaroma 3x3 matrica su reikšmėmis,

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

2. Sukama aplink X ašį

- Apskaičiuojamos cos(X) ir sin(X) reikšmės,
- Sudaroma 3x3 matrica su reikšmėmis,

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

- Sudauginama su praeitame žingsnyje gauta matrica

3. Sukama aplink Z ašį
   - Apskaičiuojamos cos(Z) ir sin(Z) reikšmės,
   - Sudaroma 3x3 matrica su reikšmėmis,

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

   - Sudauginama su praeitame 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 rėmelio kampus „nesukčiaudamas“. Naudojant juodą foną, tinklas gali daryti prielaidą, kad remėlis yra ten, kur nėra juodos spalvos, o tai būtų netikslu realiame pasaulyje. Transformuotas automobilio numerio rėmelis sukomponuojamas su atsitiktiniu paveikslėliu atsitiktinėje vietoje. Atsitiktinių paveikslėlių šaltiniu naudojamas daugiau nei 100.000 paveikslėlių duomenų rinkinys [XHE+10]. Labai svarbu didelis kiekis paveikslėlių, taip sumažinant riziką, kad neuroninis tinklas atsimins kiekvieną paveikslėlį. Kaip atrodo kompozicija galima matyti 3 paveikslėlyje.

$$A = 0.6,$$

$$B = 0.875,$$

$$C = 1.5.$$

kur:

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

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

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

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

$$min = 0.7375 - 0.20625,$$

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

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

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

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

$$max = 0.7375 + 0.20625,$$

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

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

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

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

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 nepersimokytų 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 skaičius 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://autoplius.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.

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 *xxxyyy.tif* formatu, kur x - raidė, y - skaičius.
- Šalia išsaugomas tekstinė rinkmena *xxxyyy.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 rėmelio atpažinimui naudojamas konvoliucinis neuroninis tinklas bei numeryje esančių simbolių atpažinimui naudojamas rekurentinis neuroninis tinklas.

2.1. Numerio rėmelio atpažinimui skirtas neuroninis tinklas

Numerio rėmelio atpažinimui panaudotas konvoliucinis neuroninis tinklas.

2.1.1. Modelis

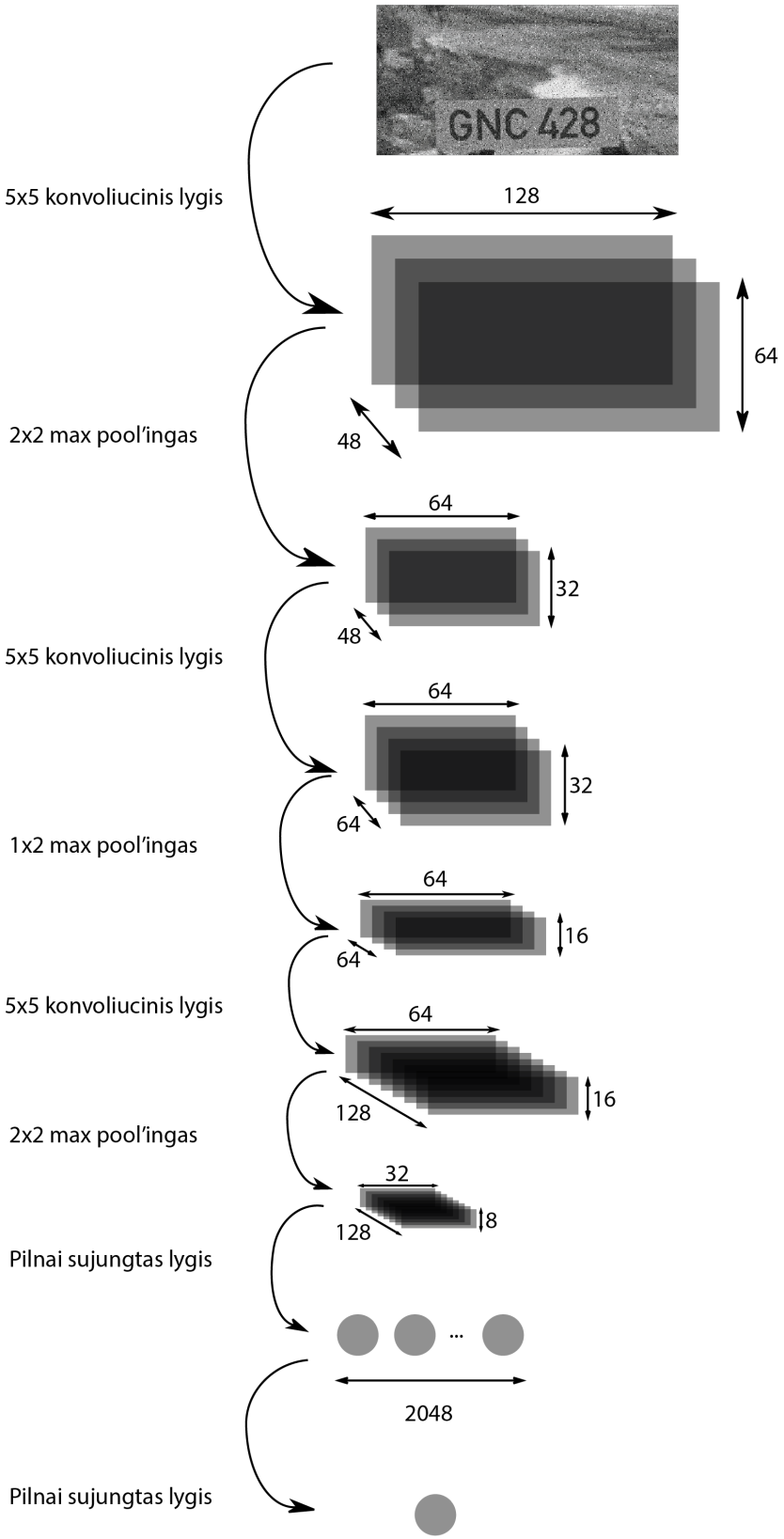
Neuroninis tinklas kurtas su Tensorflow bibliotekomis. Kuriant dirbtinį konvoliucinį neuroninį tinklą buvo pasirinkta architektūra pavaizuota 6 pav. Iš viso yra 3 konvoliuciniai lygiai, kurių dydžiai yra 48, 64 ir 128[GBI+13]. Visų jų langelio dydis yra vienodas - 5x5. Taip pat yra 3 max pool'ingo lygmenys, 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 stadijoje. 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[SHT+15]. 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 – langelio dydis 2x2, įeinančio paveikslėlio dimensijos 128x64x48, išeinančio paveikslėlio dimensijos 64x32x48.
    2. Max pool'ingo lygis – langelio dydis 1x2, įeinančio paveikslėlio dimensijos 64x32x64, išeinančio paveikslėlio dimensijos 64x16x64.
    3. Max pool'ingo lygis – langelio 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)[1].

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 [HS97].

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 tėkmę. Kiekvienas atminties blokas originalioje architektūroje yra sudarytas iš įvesties ir išvesties vartų. Įvesties vartai kontroliuoja įvesties aktyvacijų informacijos tėkmę į atminties ląstelę. Išvesties vartai kontroliuoja ląstelių aktyvacijos išvesties informacijos tėkmę atgal į tinklą. Vėliau į atminties bloką pridedami atminties praradimo vartai. Tai sprendžia vieną iš LSTM modelio silpnybių, kai įvestis nėra suskirstyta į atskiras sekas ir apdorojimas niekad nesibaigia. Atminties praradimo vartai keičia vidinę ląstelės būseną prieš patalpinant ją į ląstelę per rekurentinį ryšį su pačia savim, tuo pačiu įgalinant pamiršti ar ištrinti ląstelės atmintį. LSTM tinklas skaičiuoja sąryšius nuo įvesties sekos

$$x = (x1, ..., xT)$$

[1] https://github.com/tesseract-ocr/tesseract/wiki/VGSLSpecs

iki išvesties sekos

$$y = (y1, ..., yT)$$

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

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

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

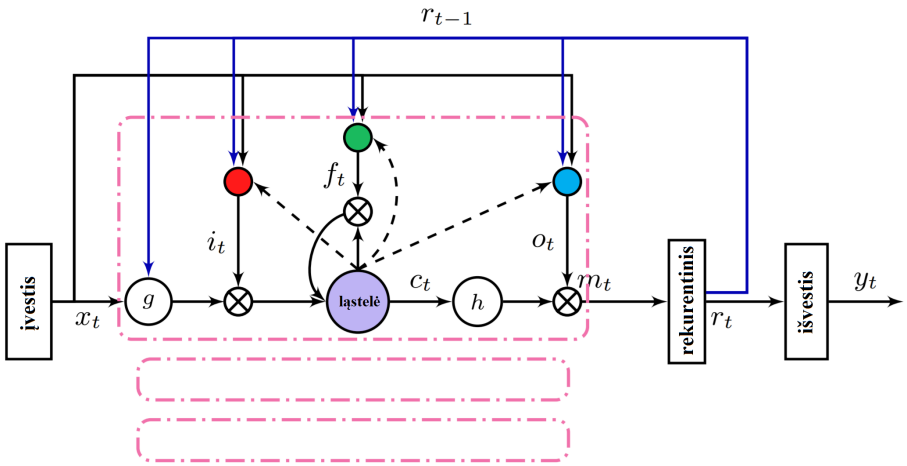
$$c_t = f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c)(3)$$

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

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

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

kur W apibrėžia svorių matricas (pvz.: W_{ix} yra svorių matrica nuo įvesties vartų iki pačios įvesties), W_{ic}, W_{fc}, W_{oc} yra įstrižos svorių matricos skirtos skylučių jungtims [2], b apibūdina išvesties vektorius, kurie neturi jokios įvesties, σ 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 ϕ yra tinklo išvesties aktyvacijos funkcija - *Softmax* [SSB14].



7 pav. LSTM atminties blokai

[2] 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. Atlikti tyrimai 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 atpažįstant tekstą dideliame dokumente arba gali būti naudojama kartu su išoriniu teksto detektoriumi, kad atpažintų tekstą iš vienos teksto eilutės atvaizdo.

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

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 tekstą paveikslėlyje, 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 atlikti 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 sukurtį 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 fiksuoto dydžio) paveikslėlius naudoja kaip įvestį (vienoje ar dvejose dimensijose).
- Gauna rezultatą kaip reikšmių matricą, tekstą ar kategoriją.
- Konvoliucijos ir LSTM tinklai yra pagrindinis skaičiavimo komponentas.

Modelį aprašančios teksto eilutės įvestis ir išvestis Neuroninio tinklo modelį aprašo teksto eilutė, kurioje yra aprašomos įvesties, išvesties ir tinklo lygių specifikacijos. Pavyzdys:

```
[1,0,0,3 Ct5,5,16 Mp3,3 Lfys64 Lfx128 Lrx128 Lfx256 O1c105]
```

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 klasių - *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, įgalinant 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.
- Klasių 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 skirtingos funkcinės operacijos:

- C(s|t|r|l|m)<y>,<x>,<d> - vykdoma konvoliucija 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 kiekviena 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 ląstelė su n išvesčių:
  - *f* - leidžia tik į priekį judantį LSTM lygį.
  - *r* - leidžia tik priešinga kryptimi judantį LSTM lygį.
  - *b* - leidžia abiejomis kryptimis judantį LSTM lygį.
  - 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 ląstelė su integruota *Softmax* aktyvacijos funkcija.
- LE<n> - tik į priekį x kryptimi judanti LSTM ląstelė 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 konvoliucija 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:

- [...] – Vykdyti ... neuroninius tinklus nuosekliai lygiais.
- (...) – Vykdyti ... 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 tekstą

```
[1,1,0,48 Lbx256 O1c105]
```

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

- O1c105: 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 atpažįstant tekstą, tol kol įvesties paveikslėlis normalizuotas vertikalioje padėtyje.

Pavyzdys: Keletos lygių LSTM tinklas, galintis tiksliai atpažinti tekstą

```
[1,0,0,1 Ct5,5,16 Mp3,3 Lfys64 Lfx128 Lrx128 Lfx256 O1c105]
```

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

- O1c105: Išvesties lygis, pagaminantis vienos dimensijos seką, treniruotą su CTC, išvedantis 105 klases.
- 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 konvoliucija 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 lygį. 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 fiksuotas 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 uždaviniui, 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 O1c`head -n1 data/unicharset`]
```

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

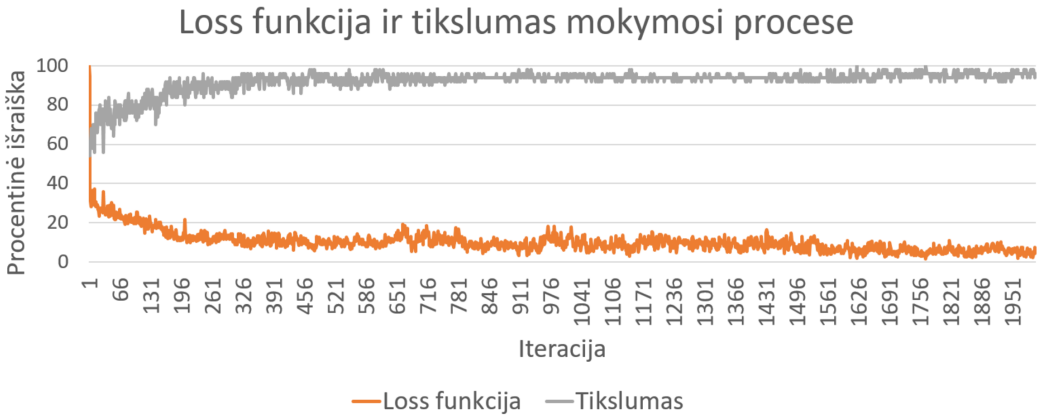
- O1c`head -n1 data/unicharset`: Išvesties lygis, pagaminantis vienos dimensijos seką, treniruotą su CTC, išvedantis n klasių nurodytų faile esančiame *data/unicharset*.
- Lfx256: Tik pirmyn judantis LSTM lygis x kryptimi su 256 išvestimis.
- Lrx96: Tik priešingai judantis LSTM lygis x kryptimi su 96 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 konvoliucija 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.

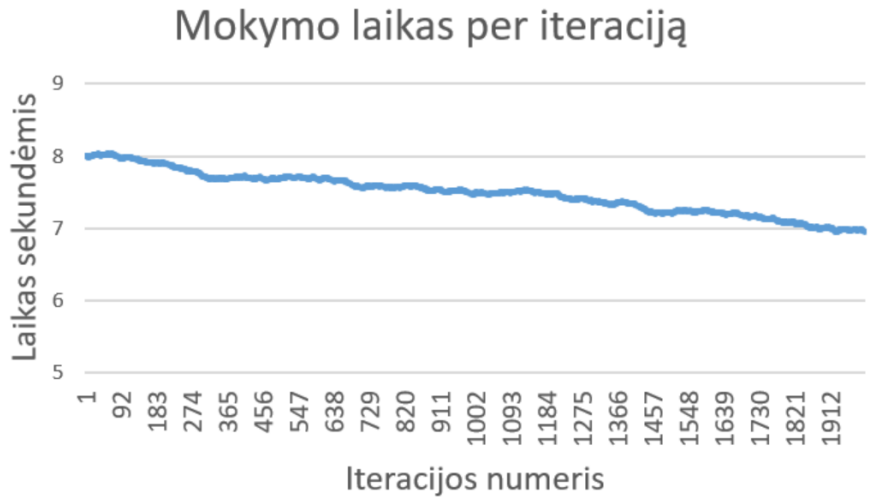
3.1. Konvoliucinio neuroninio tinklo mokymas

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. Apmokyti 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 skirtingų šriftų. 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.

Išskiriami 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 sąlyginai mažu naujų duomenų kiekiu.

- Nuimti viršutinį (ar keletą daugiau) modelio sluoksnių ir pertreniruoti naujus sluoksnius su naujais duomenimis. Jei esamo modelio patobulinimas nesprendžia esamos problemos, 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 pasiektas 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 neatliks 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 nestandartinis 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 paveiksliuką, tai galima atlikti pakeitus konfigūracinio parametro *tessedtessedit_write_image*s 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 tekstą ant juodo fono be jokių problemų, nuo 4.00 versijos būtina sąlyga, kad tekstas būtų juodas, o fonas šviesus. Tam tikslui atlikti užtenka vienos komandos:

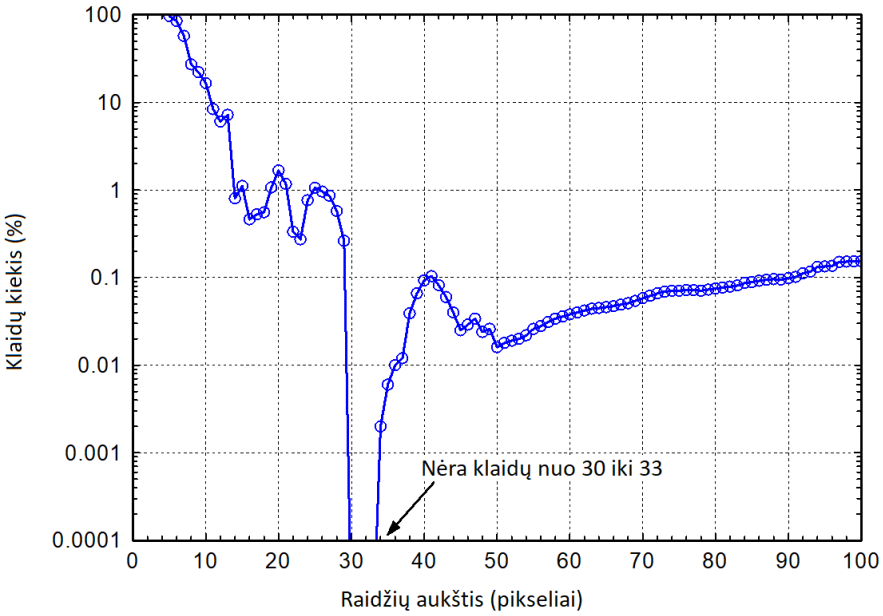
```
numpy.invert(image)
```

Dydžio keitimas

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[3] (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 geriausių rezultatų.

[3] *Willus Dotkom* vartotojo atliktas eksperimentas. Šaltinis https://groups.google.com/forum/#!msg/tesseract-ocr/Wdh_JJwnw94/24JHDYQbBQAJ



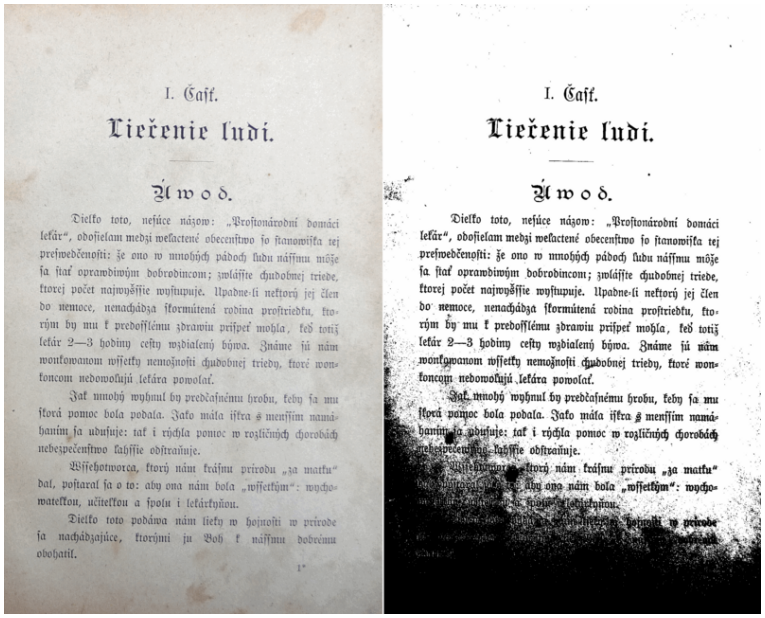
10 pav. Klaidų kiekio priklausomybė nuo raidžių aukščio

Binarizacija

Binarizacija – tai paveiksliuko 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:

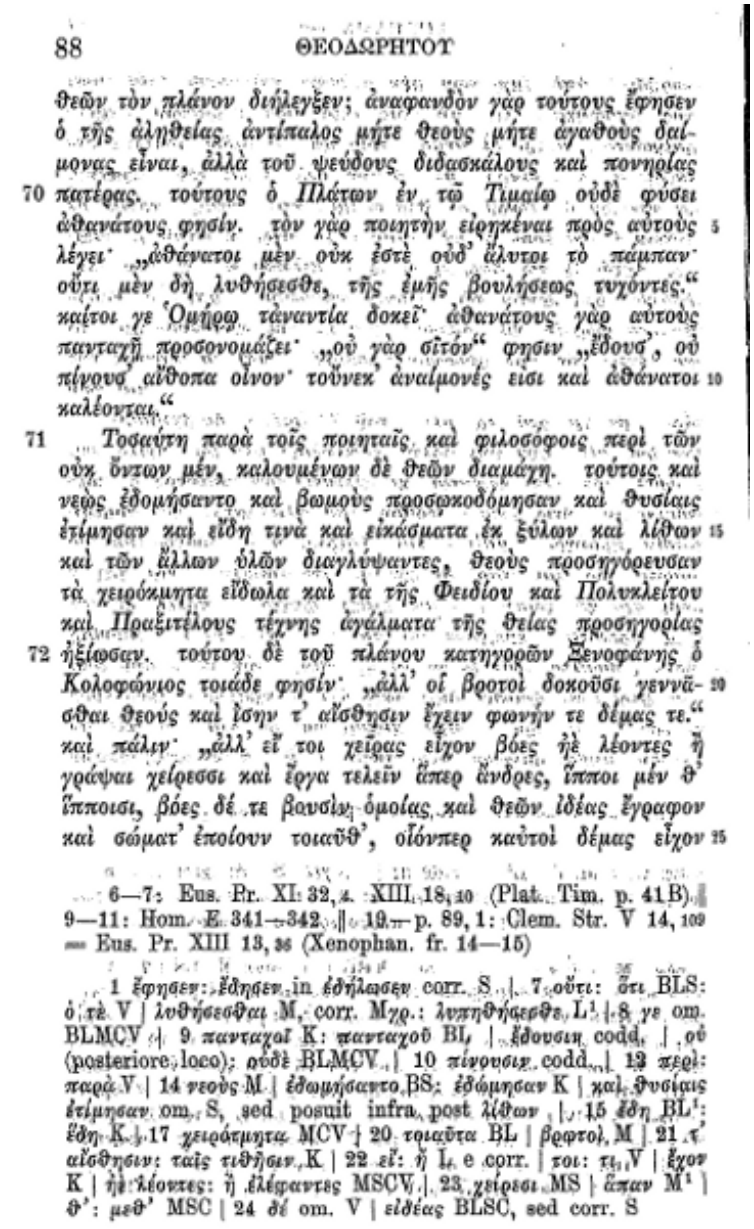
- *ImageJ* automatinis ribinių verčių nustatymo algoritmas (JAVA programavimo kalba).
- *OpenCV* ribinių verčių nustatymo algoritmas (Python programavimo kalba).
- *scikit-image* ribinių verčių nustatymo algoritmas (Python programavimo kalba).



11 pav. Binarizacijos algoritmo taikymo rezultatas

Triukšmo pašalinimas

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



88 ΘΕΟΔΩΡΗΤΟΥ

θεῶν τὸν πλάνον διήλεγξεν; ἀναφανδὸν γὰρ τούτους ἔφησεν
ὁ τῆς ἀληθείας ἀντίπαλος μήτε θεοὺς μήτε ἀγαθοὺς δαί-
μονας εἶναι, ἀλλὰ τοῦ ψεύδους διδασκάλους καὶ πονηρίας
70 πατέρας. τούτους ὁ Πλάτων ἐν τῷ Τιμαίῳ οὐδὲ φύσει
ἀθανάτους φησίν. τὸν γὰρ ποιητὴν εἰρηκέναι πρὸς αὐτοὺς 5
λέγει· „ἀθάνατοι μὲν οὐκ ἐστὲ οὐδ' ἄλυτοι τὸ πάμπαν·
οὔτι μὲν δὴ λυθήσεσθε, τῆς ἐμῆς βουλήσεως τυγόντες.“
καίτοι γε Ὁμήρῳ τἀναντία δοκεῖ· ἀθανάτους γὰρ αὐτοὺς
πανταχῇ προσονομάζει· „οὐ γὰρ σῖτον“ φησιν „ἔδουσ', οὐ
πίνουσ' αἴθοπα οἶνον· τοὔνεκ' ἀναίμονές εἰσι καὶ ἀθάνατοι 10
καλέονται.“

71 Τοσαύτη παρὰ τοῖς ποιηταῖς καὶ φιλοσόφοις περὶ τῶν
οὐκ ὄντων μέν, καλουμένων δὲ θεῶν διαμάχη. τούτοις καὶ
νεὼς ἐδομήσαντο καὶ βωμοὺς προσῳκοδόμησαν καὶ θυσίαις
ἐτίμησαν καὶ εἴδη τινὰ καὶ εἰκάσματα ἐκ ξύλων καὶ λίθων 15
καὶ τῶν ἄλλων ὑλῶν διαγλύψαντες, θεοὺς προσηγόρευσαν
τὰ χειρόκμητα εἴδωλα καὶ τὰ τῆς Φειδίου καὶ Πολυκλείτου
καὶ Πραξιτέλους τέχνης ἀγάλματα τῆς θείας προσηγορίας
72 ἠξίωσαν. τούτου δὲ τοῦ πλάνου κατηγορῶν Ξενοφάνης ὁ
Κολοφώνιος τοιάδε φησίν· „ἀλλ' οἱ βροτοὶ δοκοῦσι γεννᾶ- 20
σθαι θεούς καὶ ἶσην τ' αἴσθησιν ἔχειν φωνήν τε δέμας τε.“
καὶ πάλιν· „ἀλλ' εἴ τοι χεῖρας εἶχον βόες ἠὲ λέοντες ἢ
γράψαι χείρεσσι καὶ ἔργα τελεῖν ἅπερ ἄνδρες, ἵπποι μέν θ'
ἵπποισι, βόες δέ τε βουσὶν ὁμοίας καὶ θεῶν ἰδέας ἔγραφον
καὶ σώματ' ἐποίουν τοιαῦθ', οἷόνπερ καὐτοὶ δέμας εἶχον 25

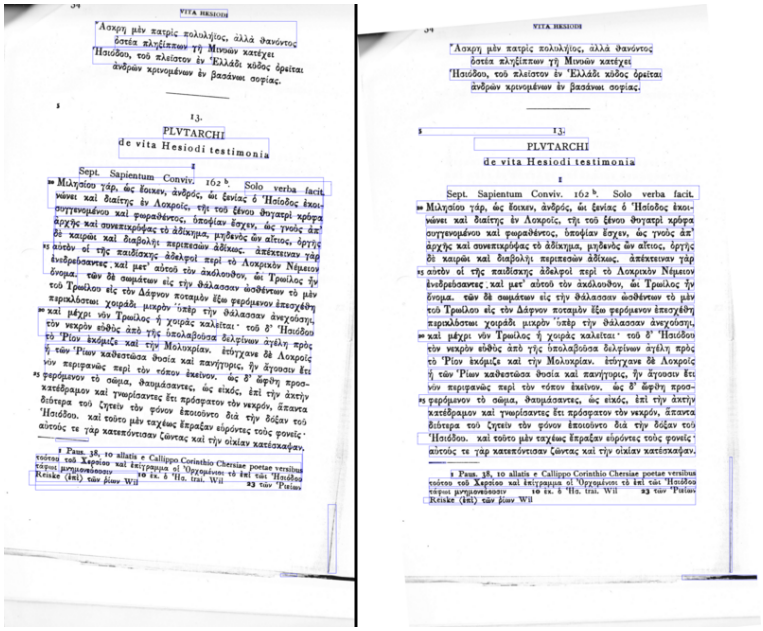
6—7: Eus. Pr. XI 32, 4. XIII 18, 10 (Plat. Tim. p. 41 B). ‖
9—11: Hom. E 341—342. ‖ 12—p. 89, 1: Clem. Str. V 14, 109
= Eus. Pr. XIII 13, 36 (Xenophan. fr. 14—15)

1 ἔφησεν: ἔδησεν in ἐδήλωσεν corr. S | 7 οὔτι: ὅτι BLS: ὁ τὲ V | λυθήσεσθαι M, corr. Mχρ.: λυπηθήσεσθε L¹ | 8 γε om. BLMCV | 9 πανταχοῖ K: πανταχοῦ Bl | ἔδουσιν codd. | οὐ (posteriore loco): οὐδὲ BLMCV | 10 πίνουσιν codd. | 12 περὶ: παρὰ V | 14 νεοὺς M | ἐδομήσαντο βS: ἐδόμησαν K | καὶ θυσίαις ἐτίμησαν om. S, sed posuit infra post λίθων | 15 ἔδη BL¹: ἔδη K | 17 χειράτμητα MCV | 20 τοιαῦτα BL | βροτοὶ M | 21 τ' αἴσθησιν: ταῖς τιθήσιν K | 22 εἴ: ἢ L, e corr. | τοι: τι V | ἔχον K | ἠὲ λέοντες: ἢ ἐλέφαντες MSCV | 23 χείρεσι MS | ἅπαν M¹ | 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.

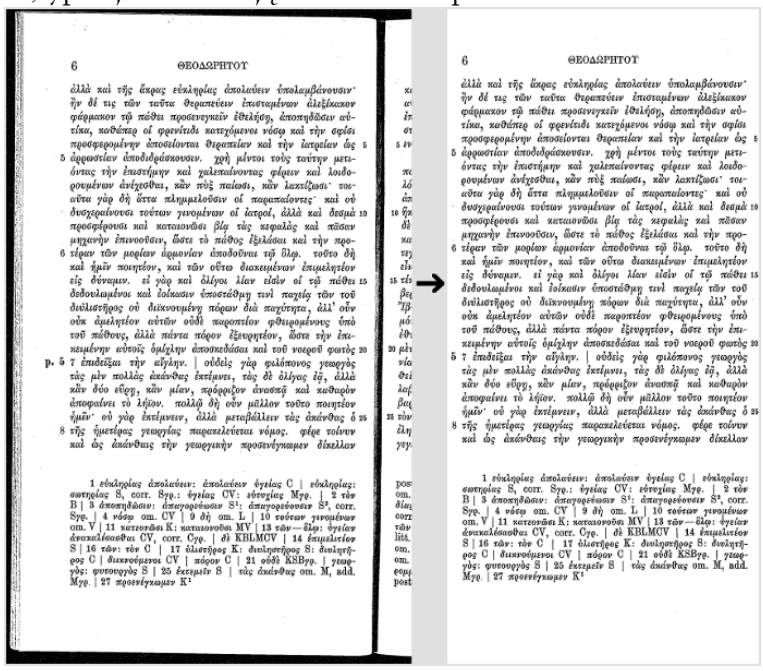


13 pav. Iškreipto puslapio išlyginimas

Kraštinės

Skanuotų puslapių kraštinių naikinimas

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



14 pav. Puslapio kraštinių naikinimas

Tekstas be kraštinių Jei norimas atpažinti tekstas visiškai neturių kraštinių ir yra nuo krašto iki krašto, Tesseract programa gali turėti sunkumų bandant atpažinti tekstą. 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 paveiksliuką 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 jį su baltu 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. Puslapių skirstymo metodas

Tesseract programos standartinis veikimo principas pagrįstas tuo, kad programa tikisi paveikslėlio puslapio pavidalu su jame 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) (Numatytasis režimas).
4. Vienas teksto stulpelis.
5. Vienas vertikaliai išlygiuoto teksto blokas.

6. Vienas teksto blokas.

7. Paveikslėlį laikyti kaip vieną teksto liniją.

8. Paveikslėlį laikyti kaip vieną žodį.

9. Paveikslėlį laikyti kaip vieną žodį apskritime.

10. Paveikslėlį laikyti kaip vieną simbolį.

11. Išmėtytas tekstas. Rasti kuo daugiau teksto nesilaikant jokios tvarkos.

12. Atpažinti išmėtytą tekstą su rašto aptikimu (OSD).

13. Neapdorota eilutė. Paveikslėlį 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 nestandartinės strukturos tekstus (pvz.: sąskaitas, 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 prisidėti norimus žodžius į Tesseract programą, kurie padės atpažinimo varikliui geriau suprasti žodžius. Be žodžių, yra galimybė prisidėti simbolių sekų šablonus, kurie dar labiau padės pagerinti tikslumą.

3.2.2. Rinkmenų pasiruošimas

Bendrai mokymo žingsniai yra tokie:

1. Pasiruošti norimą apmokyti tekstą.

2. Sugeneruoti paveiksliuką 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 paveikslėlį ir *box* failą bei sukurtų apmokymo duomenų rinkinį.

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
```

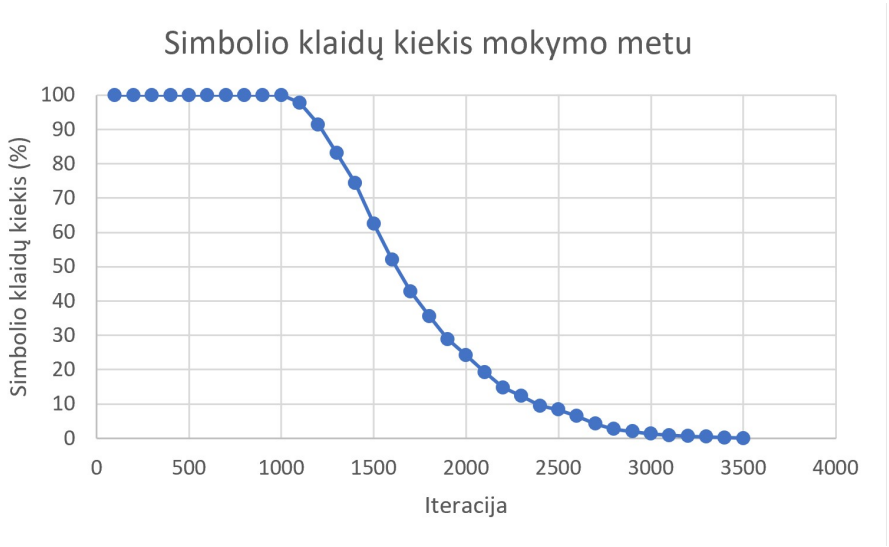
Ši komanda yra trumpinys pilnos komandos:

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

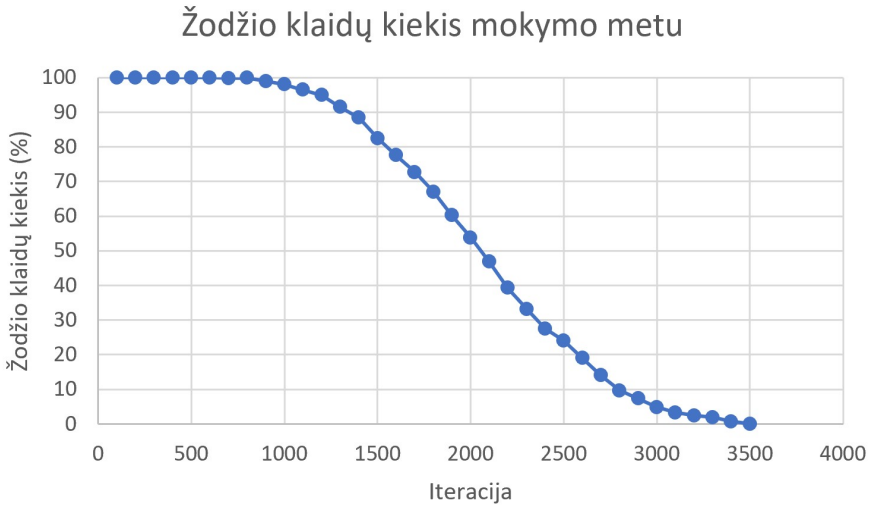
Vykdomos operacijos:

- *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 jiems paskirtas užduotis.

4.1. Numerio rėmelio atpažinimas

Norint aptikti ir atpažinti realiuose paveikslėliuose numerio rėmelį, į neuroninį tinklą paduodamos 128x64 pikselių dydžio paveikslėlio dalys, kaip jau buvo aprašyta 1.1 skyriuje. Atpažįstant realius paveikslėlius, naudojama kitokia neuroninio tinklo architektūra. Paskutiniai du lygmenys vietoj to, kad būtų pilnai sujungti, yra konvoliuciniai. Taip pat pradinio paveikslėlio dydis neturi būti 128x64 pikselių, o gali būti bet koks. Idėja tokia, kad pilno dydžio paveikslėlis gali būti paduodamas į neurorinį tinklą suskaidant jį į dalis slenkančio langelio principu, bei kiekvienai iš jų grąžinant rezultatą, ar rėmėlis egzistuoja. Naudojant vienodą neuroninį tinklą visoms paveikslėlio dalims yra pranašesnis nei atskiri neuroniniai tinklai, kadangi slenkantys langai dalinsis dauguma konvoliucinių 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 rėmelis egzistuoja yra didiesnė arba lygi 99%. Tai padaryta tokiu tikslu, kadangi mokymo duomenų aibėje apie 50% paveikslėlių yra su egzistuojančiu numerio rėmeliu, kai realiame pasaulyje paveikslėlių su numerio rėmeliais yra daug mažiau. Jeigu būtų naudojama 50% tikimybė atrinkti teisingiems paveikslė-

liams, tai būtų neapsisaugota nuo pasitaikančių panašių paveikslėlių atitikmenų.

Norint panaikinti perteklinius dublikatus, pritaikomas Non-Maximum Suppresion [4] algoritmas, kuris tarp visų besikertančių stačiakampių palieka tik didžiausią tikimybę turinčią reikšmę[GDD+14].

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

--TODO Sukurta programa, kuri naudodama apmokytą LSTM pagrindu veikiantį neuroninį tinklą atpažįsta automobilio numerius (žiūrėti priede Nr. 4) paveiksliukai 1h

[4]Non-Maximum Suppresion angl. - ne maksimalios reikšmės slopinimo algoritmas

Rezultatai

--TODO 20min

1. Pasinaudojus paveikslėlių duomenų rinkiniu susigeneruoti 100.000 atsitiktinių paveikslėlių su automobilio numeriais.
2. Apmokyti kursinio darbo metu sukurtą konvoliucinį 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 konvoliuciniu 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.

Išvados

--TODO 20min Išvadų skyriuje daromi nagrinėtų problemų sprendimo metodų palyginimai, siūlomos rekomendacijos, akcentuojamos naujovės. Išvados pateikiamos sunumeruoto (gali būti hierarchinis) sąrašo pavidalu. Darbo išvados turi atitikti darbo tikslą.

LITERATŪROS SĄRAŠAS

[BSS13] Bharat Bhushan, Simranjot Singh ir Ruchi Singla. License plate recognition system using neural networks and multithresholding technique. *International journal of computer applications*, 84(5), 2013.

[GBI+13] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud ir Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *Arxiv preprint arxiv:1312.6082*, 2013.

[GDD+14] Ross Girshick, Jeff Donahue, Trevor Darrell ir Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the ieee conference on computer vision and pattern recognition*, 2014, p.p. 580–587.

[HS97] Sepp Hochreiter ir Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[LS16] Hui Li ir Chunhua Shen. Reading car license plates using deep convolutional neural networks and lstms. *Arxiv preprint arxiv:1601.05610*, 2016.

[SHT+15] Fabian Stark, Caner Hazırbas, Rudolph Triebel ir Daniel Cremers. Captcha recognition with active deep learning. *Workshop new challenges in neural computation 2015*. Citeseer, 2015, p. 94.

[Sla99] Gregory G Slabaugh. Computing euler angles from a rotation matrix. *Retrieved on august*, 6(2000):39–63, 1999.

[Smi07] Ray Smith. An overview of the tesseract ocr engine. *Ninth international conference on document analysis and recognition (icdar 2007)*. Tom. 2. IEEE, 2007, p.p. 629–633.

[SSB14] Haşim Sak, Andrew Senior ir Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Fifteenth annual conference of the international speech communication association*, 2014.

[XHE+10] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva ir Antonio Torralba. Sun database: large-scale scene recognition from abbey to zoo. *Computer vision and pattern recognition (cvpr), 2010 ieee conference on*. IEEE, 2010, p.p. 3485–3492.

Sąvokų apibrėžimai

- Tesseract – optinė ženklų atpažinimo programa, kuri geba naudoti neuroninius tinklus atpažinimui.
- Leptonica
- Tensorflow
- OpenMP
- Tenzorius – geometrinis objektas, susidedantis iš sumos komponenčių, kurios yra transformuojamos 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

```
#!/usr/bin/env python

import io
import argparse
import unicodedata
from PIL import Image

#
# command line arguments
#
arg_parser = argparse.ArgumentParser('''Creates tesseract box files for given (line) image text pairs''')

# Text ground truth
arg_parser.add_argument('-t', '--txt', nargs='?', metavar='TXT', help='Line text (GT)', required=True)

# Image file
arg_parser.add_argument('-i', '--image', nargs='?', metavar='IMAGE', help='Image file', required=True)

args = arg_parser.parse_args()

#
# main
#

# load image
with open(args.image, "rb") as f:
    width, height = Image.open(f).size

# load gt
with io.open(args.txt, "r", encoding='utf-8') as f:
    lines = f.read().strip().split('\n')

```

```
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 generuojanti automobilių numerius

```
import numpy as np
from PIL import Image as img
from os import listdir
from os.path import isfile, join
from random import randint
import os.path
import numpy

letters_path = "d:\\bakalaurinis\\generating\\images\\letters"
numbers_path = "d:\\bakalaurinis\\generating\\images\\numbers"
test_data_path = "D:\\BAKALAURINIS\\ocrd-train\\data\\"

letters = [join(letters_path, f) for f in listdir(letters_path) if isfile(join
    (letters_path, f))]
numbers = [join(numbers_path, f) for f in listdir(numbers_path) if isfile(join
    (numbers_path, f))]

imgs    = [ img.open(i) for i in letters + numbers ]

def generate(symbols, name):
    min_shape = sorted( [(np.sum(i.size), i.size ) for i in symbols])[0][1]
    imgs_comb = np.hstack( (np.asarray( i.resize(min_shape) ) for i in symbols
    ) )
    imgs_comb = img.fromarray( imgs_comb)
    imgs_comb.save(test_data_path + name + ".tif")
    with open(test_data_path + name + ".gt.txt", "w") as text_file:
        print(name.upper(), file=text_file)

for x in range(0, 100000):
    letter_1 = imgs[randint(0, len(letters)-1)]
    letter_2 = imgs[randint(0, len(letters)-1)]
    letter_3 = imgs[randint(0, len(letters)-1)]
    number_1 = imgs[randint(21, len(numbers)+len(letters)-1)]
    number_2 = imgs[randint(21, len(numbers)+len(letters)-1)]
    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 neuroninį tinklą

```
export

SHELL := /bin/bash
LOCAL := $(PWD)/usr
PATH := $(LOCAL)/bin:$(PATH)
TESSDATA = $(LOCAL)/share/tessdata

# Name of the model to be built. Default: $(MODEL_NAME)
MODEL_NAME = foo

# Name of the model to continue from. Default: '$(START_MODEL)'
START_MODEL =

LAST_CHECKPOINT = data/checkpoints/$(MODEL_NAME)_checkpoint

# Name of the proto model. Default: '$(PROTO_MODEL)'
PROTO_MODEL = data/$(MODEL_NAME)/$(MODEL_NAME).traineddata

# No of cores to use for compiling leptonica/tesseract. Default: $(CORES)
CORES = 4

# Leptonica version. Default: $(LEPTONICA_VERSION)
LEPTONICA_VERSION := 1.75.3

# Tesseract commit. Default: $(TESSERACT_VERSION)
TESSERACT_VERSION := fd492062d08a2f55001a639f2015b8524c7e9ad4

# Tesseract model repo to use. Default: $(TESSDATA_REPO)
TESSDATA_REPO = _best

# Ground truth directory. Default: $(GROUND_TRUTH_DIR)
GROUND_TRUTH_DIR := data/ground-truth

```

```
# Normalization Mode - see src/training/language_specific.sh for details.
    Default: $(NORM_MODE)
NORM_MODE = 2

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

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

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

help:
	@echo ""
	@echo "  Targets"
	@echo ""
	@echo "    unicharset       Create unicharset"
	@echo "    lists            Create lists of lstmf filenames for training and
      eval"
	@echo "    training         Start training"
	@echo "    proto-model      Build the proto model"
	@echo "    leptonica        Build leptonica"
	@echo "    tesseract        Build tesseract"
	@echo "    tesseract-langs  Download tesseract-langs"
	@echo "    clean            Clean all generated files"
	@echo ""
	@echo "  Variables"
	@echo ""
	@echo "    MODEL_NAME          Name of the model to be built. Default: $(
     MODEL_NAME)"
	@echo "    START_MODEL         Name of the model to continue from. Default: '
     $(START_MODEL)'"
	@echo "    PROTO_MODEL         Name of the proto model. Default: '$(
     PROTO_MODEL)'"
	@echo "    CORES               No of cores to use for compiling leptonica/
     tesseract. Default: $(CORES)"
	@echo "    LEPTONICA_VERSION   Leptonica version. Default: $(
     LEPTONICA_VERSION)"
```

```
65   @echo "    TESSERACT_VERSION  Tesseract commit. Default: $(TESSERACT_VERSION
       )"
66   @echo "    TESSDATA_REPO      Tesseract model repo to use. Default: $(
       TESSDATA_REPO)"
67   @echo "    GROUND_TRUTH_DIR   Ground truth directory. Default: $(
       GROUND_TRUTH_DIR)"
68   @echo "    NORM_MODE          Normalization Mode - see src/training/
       language_specific.sh for details. Default: $(NORM_MODE)"
69   @echo "    PSM                Page segmentation mode. Default: $(PSM)"
70   @echo "    RATIO_TRAIN        Ratio of train / eval training data. Default:
       $(RATIO_TRAIN)"
71
72 # END-EVAL
73
74 ALL_BOXES = data/all-boxes
75 ALL_LSTMF = data/all-lstmf
76
77 # Create unicharset
78 unicharset: data/unicharset
79
80 # Create lists of lstmf filenames for training and eval
81 lists: $(ALL_LSTMF) data/list.train data/list.eval
82
83 data/list.train: $(ALL_LSTMF)
84   total=`cat $(ALL_LSTMF) | wc -l` \
85       no=`echo "$$total * $(RATIO_TRAIN) / 1" | bc`; \
86       head -n "$$no" $(ALL_LSTMF) > "$@"
87
88 data/list.eval: $(ALL_LSTMF)
89   total=`cat $(ALL_LSTMF) | wc -l` \
90       no=`echo "($$total - $$total * $(RATIO_TRAIN)) / 1" | bc`; \
91       tail -n "$$no" $(ALL_LSTMF) > "$@"
92
93 # Start training
94 training: data/$(MODEL_NAME).traineddata
95
96 ifdef START_MODEL
97 data/unicharset: $(ALL_BOXES)
98   mkdir -p data/$(START_MODEL)
```

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

```
$(LAST_CHECKPOINT): unicharset lists $(PROTO_MODEL)
  mkdir -p data/checkpoints
  lstmtraining \
    --traineddata $(PROTO_MODEL) \
          --old_traineddata $(TESSDATA)/$(START_MODEL).traineddata \
    --continue_from data/$(START_MODEL)/$(START_MODEL).lstm \
    --net_spec "[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c`head -
    n1 data/unicharset`]" \
    --model_output data/checkpoints/$(MODEL_NAME) \
    --learning_rate 20e-4 \
    --train_listfile data/list.train \
    --eval_listfile data/list.eval \
    --max_iterations 100000
else
$(LAST_CHECKPOINT): unicharset lists $(PROTO_MODEL)
  mkdir -p data/checkpoints
  lstmtraining \
    --traineddata $(PROTO_MODEL) \
    --net_spec "[1,36,0,1 Ct3,3,16 Mp3,3 Lfys48 Lfx96 Lrx96 Lfx256 O1c`head -
    n1 data/unicharset`]" \
    --model_output data/checkpoints/$(MODEL_NAME) \
    --learning_rate 20e-4 \
    --train_listfile data/list.train \
    --eval_listfile data/list.eval \
    --max_iterations 100000
endif

data/$(MODEL_NAME).traineddata: $(LAST_CHECKPOINT)
  lstmtraining \
  --stop_training \
  --continue_from $(LAST_CHECKPOINT) \
  --traineddata $(PROTO_MODEL) \
  --model_output $@

data/radical-stroke.txt:
  wget -O$@ 'https://github.com/tesseract-ocr/langdata_lstm/raw/master/radical
    -stroke.txt'

# 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 $(TESSDATA) && wget https://github.com/tesseract-ocr/tessdata$(
       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 atpažįstanti automobilio numerius

```
import cv2
import numpy as np
import glob
import pytesseract
import time
import sys

def detect(image):
    gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    threshGauss = cv2.adaptiveThreshold(gray, 255, cv2.
    ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 51, 27)
    ratio = 200.0 / image.shape[1]
    dim = (200, int(image.shape[0] * ratio))
    resizedCubic = cv2.resize(threshGauss, dim, interpolation=cv2.INTER_CUBIC)
    bordersize = 10
    border = cv2.copyMakeBorder(resizedCubic, top=bordersize, bottom=
    bordersize, left=bordersize, right=bordersize, borderType=cv2.
    BORDER_CONSTANT, value=[255, 255, 255])
    edges = cv2.Canny(border, 50, 150, apertureSize=3)
    cv2.imshow('d', border)
    cv2.waitKey(1000)
    lines = cv2.HoughLinesP(image=edges, rho=1, theta=np.pi / 180, threshold
    =100, lines=np.array([]), minLineLength=100, maxLineGap=80)
    a, b, c = lines.shape
    for i in range(a):
        x = lines[i][0][0] - lines[i][0][2]
        y = lines[i][0][1] - lines[i][0][3]
        if x != 0:
            if abs(y / x) < 1:
                cv2.line(border, (lines[i][0][0], lines[i][0][1]), (lines[i
    ][0][2], lines[i][0][3]), (255, 255, 255), 1, cv2.LINE_AA)

        se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))
        gray = cv2.morphologyEx(border, cv2.MORPH_CLOSE, se)

```

```
    # OCR
    config = ''
    text = pytesseract.image_to_string(gray, lang='training', config=config)
    validChars = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
    'M', 'N', 'O', 'P', 'R', 'S', 'T', 'U', 'V', 'Y', 'Z', '0', '1', '2', '3',
     '4', '5', '6', '7', '8', '9']
    print(text)
    cleanText = []

    for char in text:
        if char in validChars:
            cleanText.append(char)

    plate = ''.join(cleanText)
    return plate


start = time.time()
plate = detect(cv2.imread(sys.argv[1]))
print(plate)
print(round(time.time() - start, 2), 'ms')
```