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

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

--TODO

1. Pasinaudojus paveikslėlių duomenų rinkiniu susigeneruoti 1.000.000 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.

Literatūros šaltinių apžvalga

Bendrai apie LSTM --TODO 2h

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

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

--TODO aprašymas 10min

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**.

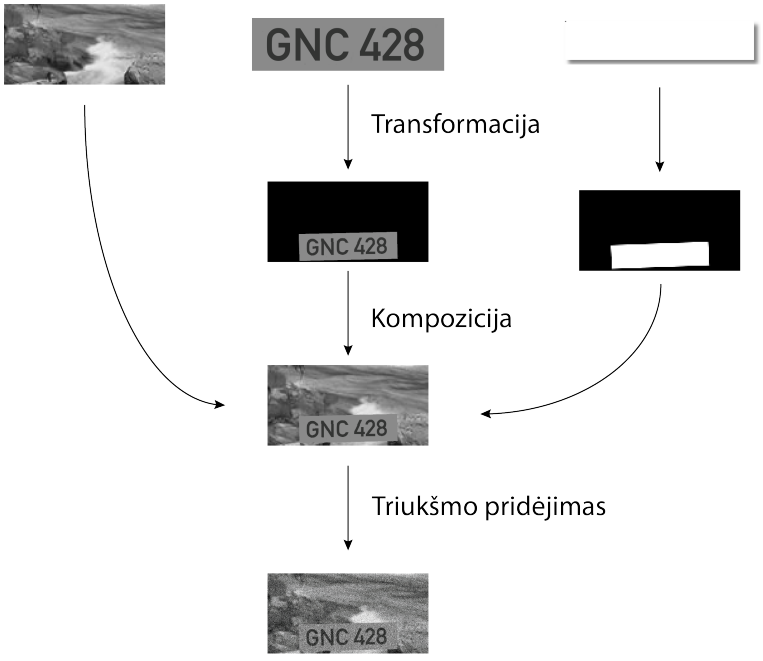
(c) Tikimasis rezultatas **COH150 0** (per mažas numeris).

(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.

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

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

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

$$min = 0.7375 - 0.20625,$$

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

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

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

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

$$max = 0.7375 + 0.20625,$$

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

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

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

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

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 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.

1.1.2. Duomenų generavimo patobulinimai

--TODO 1h kas patobulinta remeliams kaip pasikeite statistika

1.2. Numerio atpažinimui skirtų duomenų generavimas

--TODO aprašymas 10min

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. 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

--TODO aprašymas 10min

2.1. Numerio rėmėlio atpažinimui skirtas neuroninis tinklas

--TODO aprašymas 10min

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:

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

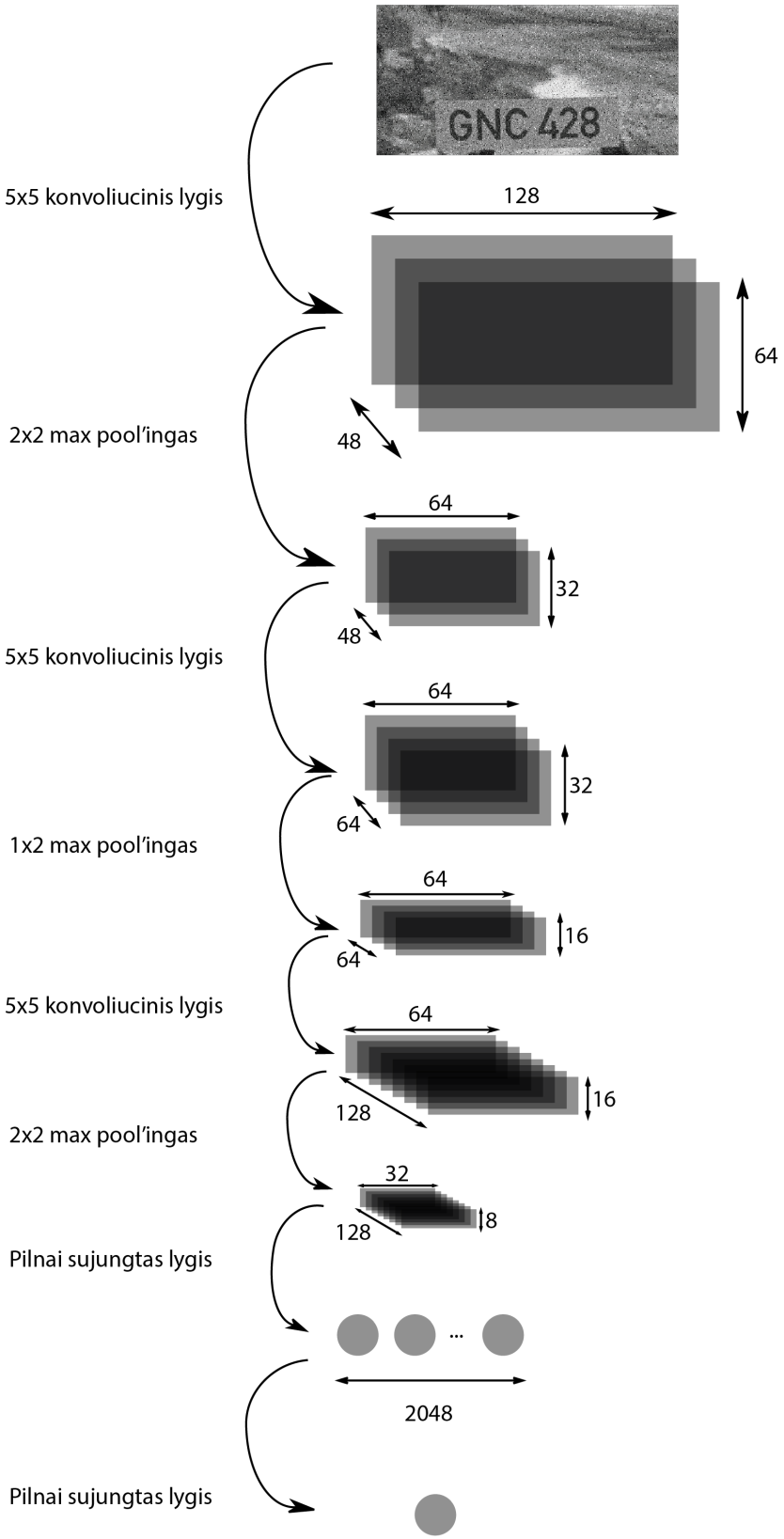
Neuroninį tinklą sudaro:

- 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).

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. 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.

2.2.2. 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.

2.2.3. 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.

2.2.4. 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.

2.2.5. Modelis

--TODO aprašymas 10min

2.2.5.1. VGSL specifikacijos

VGSL Specs - rapid prototyping of mixed conv/LSTM networks for images Variable-size Graph Specification Language (VGSL) enables the specification of a neural network, composed of convolutions and LSTMs, that can process variable-sized images, from a very short definition string.

Applications: What is VGSL Specs good for?

VGSL Specs are designed specifically to create networks for:

Variable size images as the input. (In one or BOTH dimensions!) Output an image (heat map), sequence (like text), or a category. Convolutions and LSTMs are the main computing component. Fixed-size images are OK too!

Model string input and output

A neural network model is described by a string that describes the input spec, the output spec and the layers spec in between. Example:

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

The first 4 numbers specify the size and type of the input, and follow the TensorFlow convention for an image tensor: [batch, height, width, depth]. Batch is currently ignored, but eventually may be used to indicate a training mini-batch size. Height and/or width may be zero, allowing them to be variable. A non-zero value for height and/or width means that all input images are expected to be of that size, and will be bent to fit if needed. Depth needs to be 1 for greyscale and 3 for color. As a special case, a different value of depth, and a height of 1 causes the image to be treated from input as a sequence of vertical pixel strips. NOTE THAT THROUGHOUT, x and y are REVERSED from conventional mathematics, to use the same convention as TensorFlow. The reason TF adopts this convention is to eliminate the need to transpose images on input, since adjacent memory locations in images increase x and then y, while adjacent memory locations in tensors in TF, and NetworkIO in tesseract increase the rightmost index first, then the next-left and so-on, like C arrays.

The last "word" is the output specification and takes the form:

O(2|1|0)(l|s|c)n output layer with n classes. 2 (heatmap) Output is a 2-d vector map of the input (possibly at different scale). (Not yet supported.) 1 (sequence) Output is a 1-d sequence

of vector values. 0 (category) Output is a 0-d single vector value. l uses a logistic non-linearity on the output, allowing multiple hot elements in any output vector value. (Not yet supported.) s uses a softmax non-linearity, with one-hot output in each value. c uses a softmax with CTC. Can only be used with s (sequence). NOTE Only O1s and O1c are currently supported. The number of classes is ignored (only there for compatibility with TensorFlow) as the actual number is taken from the unicharset.

2.2.5.2. Syntax of the Layers in between

NOTE that all ops input and output the standard TF convention of a 4-d tensor: [batch, height, width, depth] regardless of any collapsing of dimensions. This greatly simplifies things, and allows the VGSLSpecs class to track changes to the values of widths and heights, so they can be correctly passed in to LSTM operations, and used by any downstream CTC operation.

NOTE: in the descriptions below, <d> is a numeric value, and literals are described using regular expression syntax.

NOTE: Whitespace is allowed between ops.

Functional ops

C(s|t|r|l|m)<y>,<x>,<d> Convolves using a y,x window, with no shrinkage, random infill, d outputs, with s|t|r|l|m non-linear layer. F(s|t|r|l|m)<d> Fully-connected with s|t|r|l|m non-linearity and d outputs. Reduces height, width to 1. Connects to every y,x,depth position of the input, reducing height, width to 1, producing a single <d> vector as the output. Input height and width *must* be constant. For a sliding-window linear or non-linear map that connects just to the input depth, and leaves the input image size as-is, use a 1x1 convolution eg. Cr1,1,64 instead of Fr64. L(f|r|b)(x|y)[s]<n> LSTM cell with n outputs. The LSTM must have one of: f runs the LSTM forward only. r runs the LSTM reversed only. b runs the LSTM bidirectionally. It will operate on either the x- or y-dimension, treating the other dimension independently (as if part of the batch). s (optional) summarizes the output in the requested dimension, outputting only the final step, collapsing the dimension to a single element. LS<n> Forward-only LSTM cell in the x-direction, with built-in Softmax. LE<n> Forward-only LSTM cell in the x-direction, with built-in softmax, with binary Encoding. In the above, (s|t|r|l|m) specifies the type of the non-linearity:

s = sigmoid t = tanh r = relu l = linear (i.e., No non-linearity) m = softmax Examples:

Cr5,5,32 Runs a 5x5 Relu convolution with 32 depth/number of filters.

Lfx128 runs a forward-only LSTM, in the x-dimension with 128 outputs, treating the y dimension independently.

Lfys64 runs a forward-only LSTM in the y-dimension with 64 outputs, treating the x-dimension independently and collapses the y-dimension to 1 element.

Plumbing ops

The plumbing ops allow the construction of arbitrarily complex graphs. Something currently missing is the ability to define macros for generating say an inception unit in multiple places.

[...] Execute ... networks in series (layers). (...) Execute ... networks in parallel, with their output concatenated in depth. S<y>,<x> Rescale 2-D input by shrink factor y,x, rearranging the data by increasing the depth of the input by factor xy. **NOTE** that the TF implementation of VGSLSpecs has a different S that is not yet implemented in Tesseract. Mp<y>,<x> Maxpool the input, reducing each (y,x) rectangle to a single value.

Full Example: A 1-D LSTM capable of high quality OCR

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

As layer descriptions: (Input layer is at the bottom, output at the top.)

O1c105: Output layer produces 1-d (sequence) output, trained with CTC, outputting 105 classes. Lbx256: Bi-directional LSTM in x with 256 outputs 1,1,0,48: Input is a batch of 1 image of height 48 pixels in greyscale, treated as a 1-dimensional sequence of vertical pixel strips. []: The network is always expressed as a series of layers. This network works well for OCR, as long as the input image is carefully normalized in the vertical direction, with the baseline and meanline in constant places.

Full Example: A multi-layer LSTM capable of high quality OCR

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

As layer descriptions: (Input layer is at the bottom, output at the top.)

O1c105: Output layer produces 1-d (sequence) output, trained with CTC, outputting 105 classes. Lfx256: Forward-only LSTM in x with 256 outputs Lrx128: Reverse-only LSTM in x with 128 outputs Lfx128: Forward-only LSTM in x with 128 outputs Lfys64: Dimension-summarizing LSTM, summarizing the y-dimension with 64 outputs Mp3,3: 3 x 3 Maxpool Ct5,5,16: 5 x 5 Convolution with 16 outputs and tanh non-linearity 1,0,0,1: Input is a batch of 1 image of variable size in greyscale []: The network is always expressed as a series of layers.

The summarizing LSTM makes this network more resilient to vertical variation in position of the text.

2.2.5.3. Variable size inputs and summarizing LSTM

NOTE that currently the only way of collapsing a dimension of unknown size to known size (1) is through the use of a summarizing LSTM. A single summarizing LSTM will collapse one dimension (x or y), leaving a 1-d sequence. The 1-d sequence can then be collapsed in the other dimension to make a 0-d categorical (softmax) or embedding (logistic) output.

For OCR purposes then, the height of the input images must either be fixed, and scaled (using Mp or S) vertically to 1 by the top layer, or to allow variable-height images, a summarizing LSTM must be used to collapse the vertical dimension to a single value. The summarizing LSTM can also be used with a fixed height input.

2.2.5.4. Programos įgyvendinimui panaudotas modelis

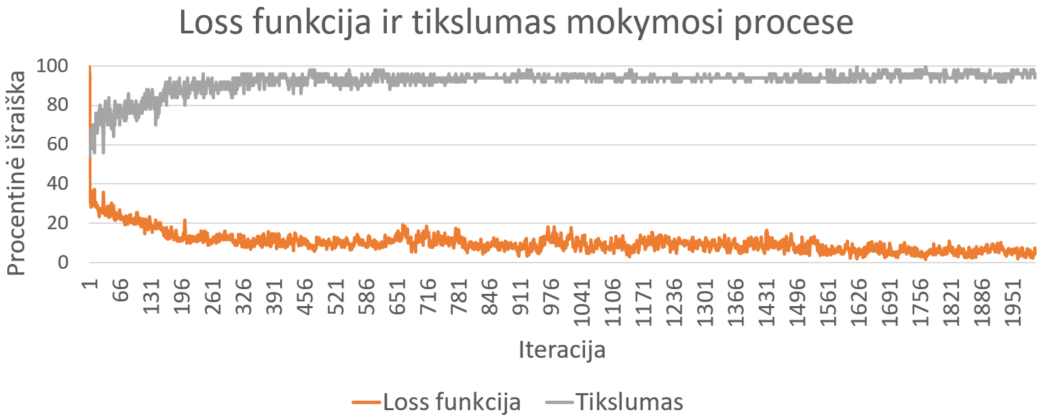
--net$_spec$"[1,36,0,1Ct3,3,16Mp3,3$Lfys$48Lfx96Lrx96Lfx256O1c`$head$ $-$ $n$1$data/unicharset$`]" $--learning_rate$20$e-4$ $--max_iterations$100000

3. Neuroninių tinklų apmokymas

--TODO aprašymas 10min

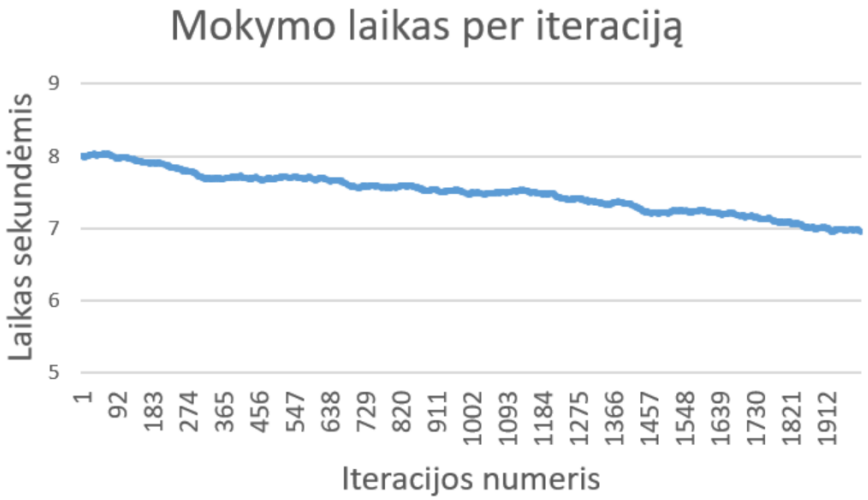
3.1. Konvoliucinio neuroninio tinklo mokymas

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 7 paveikslėlyje, mokymosi proceso metu tikslumas priartėjo prie 100% bei pasiekė vidutinį 98% tikslumą mokymo pabaigoje.



7 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 (8 pav.). Paveikslėlių generavimas buvo vykdomas tuo pačiu metu naudojantis CPU.



8 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[1] 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)
```

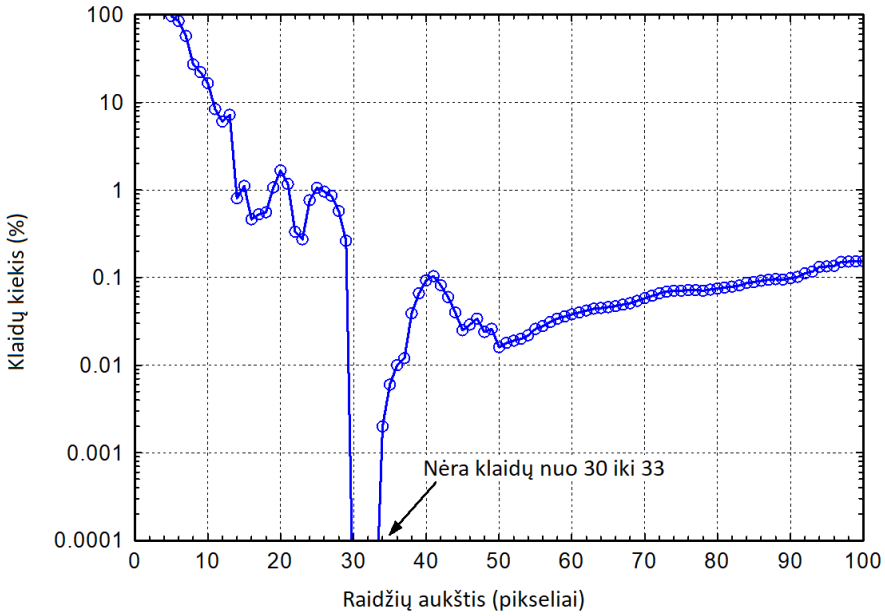
Dydžio keitimas

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[2] (žiūrėti 9 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.

[1] Leptonica - --TODO

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



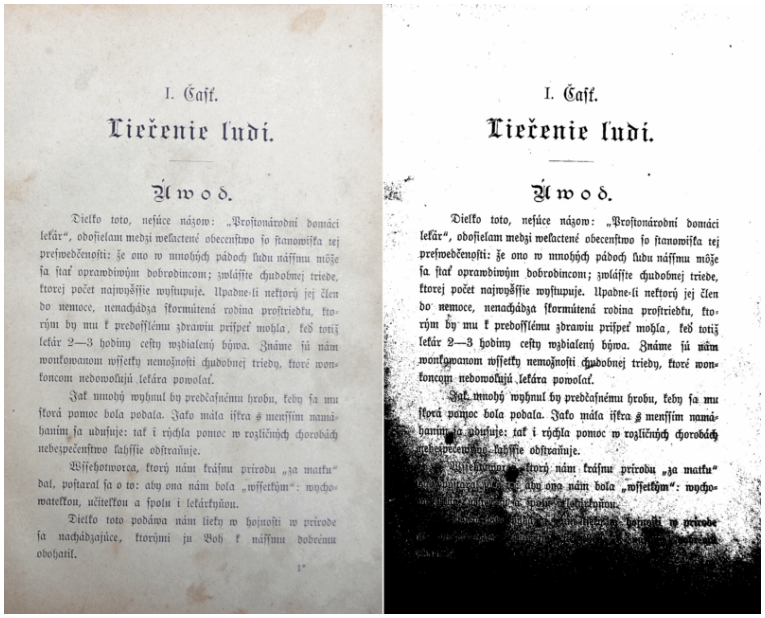
9 pav. Klaidų kiekio priklausomybė nuo raidžių aukščio

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).



I. Časť.

Liečenie ľudí.

Úvod.

Dielko toto, nesúce názov: „Prostonárodní domáci lekár“, obsielam medzi welactené obecenstwo so stanowiska tej presvedčenosti: že ono w mnohých pádoch ľudu nášmu môže sa stať oprawdivým dobrodincom; zwlášte chudobnej triede, ktorej počet najwyššie wystupuje. Upadne-li niektorý jej člen do nemoce, nenachádza skormútená rodina prostriedku, ktorým by mu k predošlému zdraviu prispieť mohla, keď totiž lekár 2—3 hodiny cesty wzdialený býwa. Známe sú nám wonkajšnom wšetky nemožnosti chudobnej triedy, ktoré wonkoncom nedowoľujú lekára powolať.

Ak mnohý wyhnul by predčasnému hrobu, keby sa mu skorá pomoc bola podala. Jako máta iskra s menším namáhaním sa udušuje: tak i rýchla pomoc w rozličných chorobách nebezpečenstwo ľahšie odstraňuje.

Wšemohúca, ktorý nám krásnu prírodu „za matku“ dal, postaral sa o to: aby ona nám bola „wšetkým“: wychowateľkou, učiteľkou a spolu i lekárkyňou.

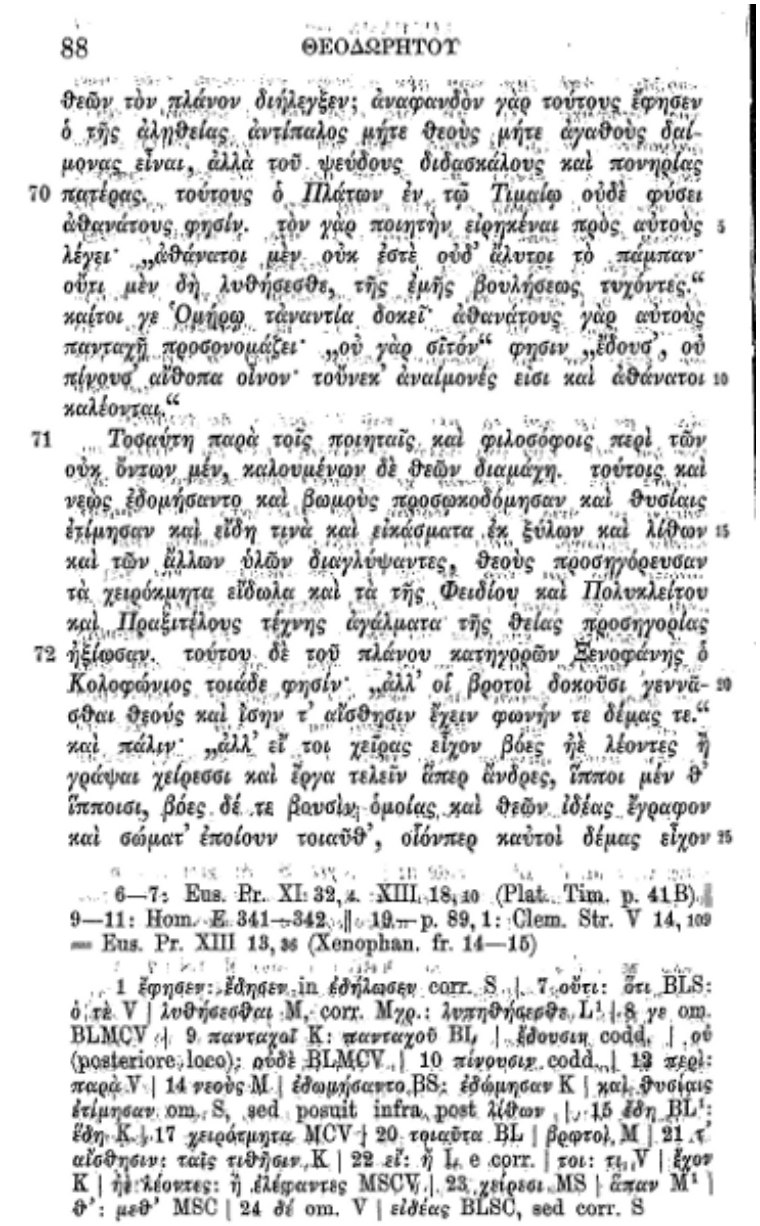
Dielko toto podáwa nám lieky w hojnosti w prírode sa nachádzajúce, ktorými ju Boh k nášmu dobrému obohatil.

1*

10 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

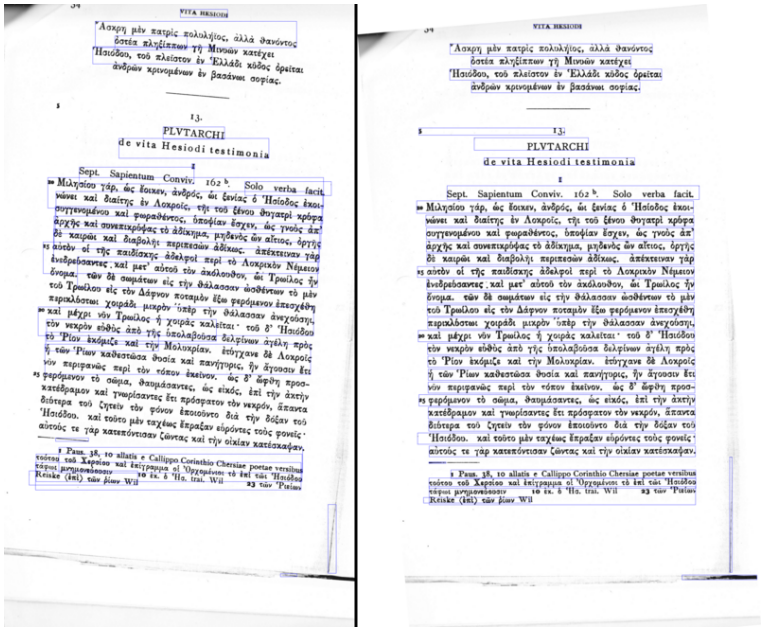
6—7: Eus. Pr. XI 32, 4. XIII 18, 10 (Plat. Tim. p. 41 B). ‖
9—11: Hom. E 341—342. ‖ 12—p. 89, 1: Clem. Str. V 14, 109
= Eus. Pr. XIII 13, 36 (Xenophan. fr. 14—15)

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 τιθῆσιν 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

11 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.

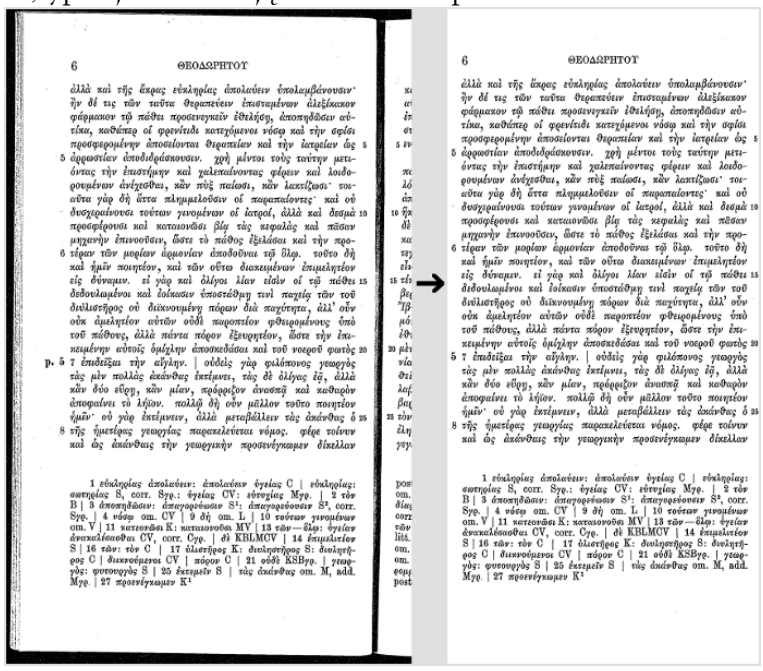


12 pav. Iškreipto puslapio išlyginimas

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.



13 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

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:

--TODO.box formato - .lstmf formato - .gt.txt formato - tekstinė rinkmena, kurioje yra tekstas, kuris yra pavaizduotas paveikslėlyje. .txt formato - .tif formato - paveiksliukas, išsaugotas TIFF formatu.

--TODO https://github.com/tesseract-ocr/tesseract/wiki/TrainingTesseract-4.00 prideti kaip vyko tinklo mokymas 2h Makefile 1h statistika 1h

4. Vaizdo atpažinimas

--TODO aprašymas 10min

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:

14 pav. Slenkančio lango principu gauti rezultatai

Žali stačiakampiai (14 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 paveikslėliams, tai būtų neapsisaugota nuo pasitaikančių panašių paveikslėlių atitikmenų.

Norint panaikinti perteklinius dublikatus, pritaikomas Non-Maximum Suppresion [3] algoritmas, kuris tarp visų besikertančių stačiakampių palieka tik didžiausią tikimybę turinčią reikšmę[GDD+14].

Gavus likusį vieną stačiakampį (15 pav.), pagal to objekto koordinates iškerpamas paveikslėlis ir gaunamas toks rezultatas:

15 pav. Iškirptas gautas rezultatas

4.2. Numerio simbolių atpažinimas

--TODO programa kuri naudoja tsseract su istreniruotu tinklu 1h paveiksliukai 1h

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

Rezultatai

--TODO 20min Rezultatų skyriuje išdėstomi pagrindiniai darbo rezultatai: kažkas išanalizuota, kažkas sukurta, kažkas įdiegta. Rezultatai pateikiami sunumeruotų (gali būti hierarchiniai) sąrašų pavidalu. Darbo rezultatai turi atitikti darbo tikslą.

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.

[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.

[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

Sutartinių ženklų, simbolių, vienetų ir terminų sutrumpinimų sąrašas (jeigu ženklų, simbolių, vienetų ir terminų bendras skaičius didesnis nei 10 ir kiekvienas iš jų tekste kartojasi daugiau nei 3 kartus).

Santrumpos

--TODO

- LSTM - trumpinys angl. Long short-term memory - rekurentinio neuroninio tinklo architektūra.
- Tesseract - optinė ženklų atpažinimo programa, kuri geba naudoti neuroninius tinklus atpažinimui.
- TIFF
- Leptonica
- DPI
- PNG
- OSD
- OCR
- VGSL
- Tensorflow
- OpenMP
- SSE
- AVX
- SIMD
- Tenzorius - geometrinis objektas, susidedantis iš sumos komponenčių, kurios yra transformuojamos pagal tiesinius sąryšius.

-
-
-
-
-