Rašytinės LT kalbos identifikavimas (OCR)

Paulius Milmantas
Vilniaus universitetas
Matematikos ir informatikos fakultetas
Vilnius, Lietuva
paulius.milmantas@mif.stud.vu.lt

***Abstract*—Darbe buvo spręsta rašytinės lietuvių kalbos raidžių atpažinimo problema. Parašyta programa gali aptikti 6 išmokytas raides: A, a, B, C, P, u vidutiniškai su 85% atspėjimo tikimybe. Padarytą sistemą galima plėsti nemodifikuojant esamo kodo: užtenka sukurti naują duomenų rinkinį pagal tam tikrą formatą ir ištreniruoti naują 3 elementų aptikimo tinklą. Kadangi programoje naudojama daug skirtingų tinklų ir kiekvienas jų atskiria tik 3 elementus, plečiant programą, nebereikia apmokyti visos sistemos iš naujo, užtenka tik pridėti naują tinklą, programa jį automatiškai aptinka ir pradeda naudoti. Naujam 3 raidžių poaibiui apmokyti reikia vidutiniškai 50 nuotraukų kiekvienai raidei ir norint pasiekti 85% tikslumą vidutiniškai užtenka 300 epochų. Naudojant Google Colab platformą tai vidutiniškai užtrunka apie 10 minučių. Naudojant daug neuroninių tinklų taip pat išspręsta didelio RAM naudojimo problema: pagal esamus resursus galima apskaičiuoti kiek vienu momentu galima pakrauti tinklų ir pakrauti tik tam tikrą jų kiekį.**

I. Įvadas

Užduoties tikslas: parašyti sprendimą, kuris iš duotos nuotraukos išgautų joje pateiktą Lietuvišką rašytinį tekstą. Atpažinimui buvo naudojamas stochastinio gradientų nuolydis (SGD). Buvo bandyta jį lyginti su naujai išėjusiu SGD modifikuotu variantu (pav. 2). Šio algoritmo nebuvo spėta pilnai realizuoti.

II. Metodai

A. Taikyta nuostolių funkcija

Darbe buvo naudota MSE (Mean Square Error) funkcija. (1)

$$MSE = \frac{1}{n} * \sum_{i=1}^{n} (y_i - y_i^p)^2 \tag{1}$$

B. Optimizavimo funkcija

Optimizavimui buvo naudota stochastinio gradientų nuolydžio (SGD) optimizavimo funkcija (2). Taip pat buvo bandyta realizuoti modifikuotą SGD algoritmą (pav. 2).

$$\theta^{(\tau)} = \theta^{(\tau-1)} - \eta * \bigtriangledown_\theta Loss(\theta^{(\tau-1)}; (x_i, y_i)) \tag{2}$$

C. Naudojamas tinklas

Visos abėcėlės raidės yra skirstomos į poaibius po 3 raides. Tai parodyta pav. 2 Kiekvienam poaibiui yra sukuriama po atskirą neuroninį tinklą. Taip yra lengviau atlikti tinklo treniravimą ir tinklo užkrovimui galima sutaupyti RAM atminties.

Algorithm 1 Alternating Minimization with Stochastic Batch Size

for all epoch **do**
  $\{\beta^t\}$: universal batches given by random shuffling
  $\beta_j^0 = \varnothing$: initialize stochastic batch for all j
  for all t : index for universal batch **do**
    for all j : index for parameter **do**
      $\beta_j^t = \beta_j^{t-1} \cup \beta^t$
      if $\chi_j^t = 1$ **then**
        $g_j^t = \frac{1}{|\beta_j^t|} \sum_{i \in \beta_j^t} \nabla_j f_i(w^t)$
        $w_j^{t+1} = w_j^t - \chi_j^t(\eta_j^t g_j^t)$
        $\beta_j^t = \varnothing$
      end if
    end for
  end for
end for

Figure 1 – SGD modifikuotas algoritmas.

Norint pridėti daugiau duomenų, užtenka tinklą apmokyti tik vienam naujam poaibiui.

Tinklą sudaro 2 paslėpti sluoksniai, 1 įvesties ir 1 išvesties sluoksnis. Duomenys yra 64x64 dydžio pilki vaizdai, todėl įvesties sluoksnis yra 4096 dydžio. Išvesties sluoksnis yra 3 dydžio, nes visi poaibiai yra sudaryti iš 3 narių. Visi sluoksniai naudoja RELU aktyvacijos funkcijas.

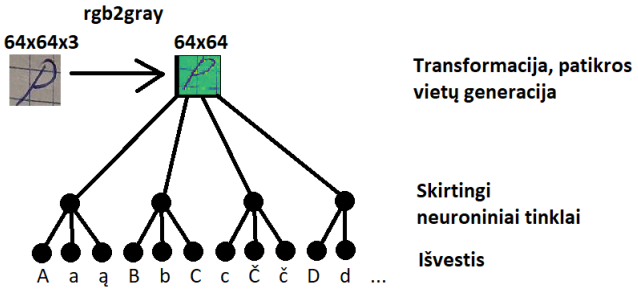

Figure 2 – Tinklo sudėtis.

D. Apdorojimo sričių paieška

Sričių radimui, kurias norima leisti per neuroninius tinklus, buvo naudojama objektų kraštinių paieška. Kadangi kiekvieną pikselį apibrėžia tik vienas skaičius: pilkos spalvos stiprumas, galima eiti per paveiksliuko kiekvieną pikselį ir tikrinti ar jo reikšmė labai skiriasi nuo praeitos. Šiuo metodu gaunamos visos kraštinės, tačiau atsiranda ir daug triukšmo. Triukšmas paprastai aiškiuose vaizduose būna nedidelis, todėl jį galima pašalinti tikrinant kraštinių vientisumas: jeigu pikselis yra aptiktas kaip kraštinė, tai jį turi supti dar nors vienas pikselis, kuris yra laikomas kraštine. Jeigu tokio pikselio šalia nėra,

reiškia tikrinamas pikselis nėra kraštinė ir jo nereikia įtraukti. Rezultatas pateiktas pav. 4.

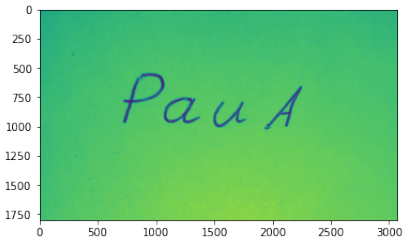

Figure 3 – Vaizdas prieš apdorojimą.

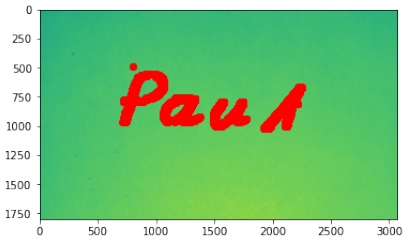

Figure 4 – Rastos kraštinės.

Radus kraštines ir taikant sąlygą, kad vaizde nėra daug pašalinių objektų galima aproksimuoti Y ašies padėtį, ties kuria yra parašytas tekstas, apskaičiuojant rastų kraštinių taškų Y koordinates. Rezultatas pateiktas pav. 5.

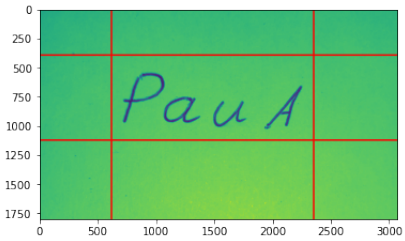

Figure 5 – Y ašies radimas.

Radus teksto kraštines yra naudojamas slenkančio lango algoritmas: einama pro paveikslėlio pikselius ir didinant imamo vaizdo plotis ir vaizdo variantai yra siunčiami vaizdai pro tinklus. Jeigu didinant vaizdo plotį tikimybė sumažėja arba spėjama jog yra kitas objektas, tada yra ieškoma kitos raidės.

III. Duomenys

Sužymėtų duomenų, kurių reikia norint išmokyti modelį, internete nėra, todėl jie buvo renkami ranka. Ant lapo buvo surašomos raidės ir visas lapas buvo nufotografuojamas. Gautos fotografijos buvo apdorojamos duomenų žymėjimo programa, kuri kaip rezultatą eksportavo JSON formato failą su kiekvienos raidės pozicija nuotraukoje. Pagal gautą JSON failą kiekviena raidė buvo eksportuota į atskirą JPG failą ir atitinkamai apdorota: naudojant nearest neighbour metodą sumažinta iki 64x64 dydžio ir panaikintas RGB kanalas.

IV. Rezultatai

Parašyta programa gali aptikti 6 išmokytas raides: A, a, B, C, P, u vidutiniškai su 85% atspėjimo tikimybe. Raides A, B, C aptinka su 82% procentų tikimybę, raides a, P, u su 92% tikimybe. Padarytą sistemą galima plėsti nemodifikuojant esamo kodo: užtenka sukurti naują duomenų rinkinį pagal tam tikrą formatą ir ištreniruoti naują 3 elementų aptikimo tinklą. Kadangi programoje naudojama daug skirtingų tinklų ir kiekvienas jų atskiria tik 3 elementus, plečiant programą, nebereikia apmokyti visos sistemos iš naujo, užtenka tik pridėti naują tinklą, programa jį automatiškai aptinka ir pradeda naudoti. Naujam 3 raidžių poaibiui apmokyti reikia vidutiniškai 50 nuotraukų kiekvienai raidei ir norint pasiekti 85% tikslumą vidutiniškai užtenka 300 epochų. Naudojant Google Colab platformą tai vidutiniškai užtrunka apie 10 minučių. Naudojant daug neuroninių tinklų taip pat išspręsta didelio RAM naudojimo problema: pagal esamus resursus galima apskaičiuoti kiek vienu momentu galima pakrauti tinklų ir pakrauti tik tam tikrą jų kiekį.

Dabartinei programos realizacijai trūksta geresnio radžių sričių radimo algoritmo. Šiuo metu programa aptinka daugiau raidžių nuotraukoje, negul jų yra. Pateikus programai 3 pav., programa vietoj rastų raidžių 'PauA', aptinka 'aPAPAPa'. Programos realizacijoje taip pat trūksta ir modifikuoto SGD algoritmo implementacijos pav. 1. Tikėtina, jog ši modifikacija galėtų pagerinti tinklo tikslumą.

V. References

[1] Stochastic batch size for adaptive regularization in deep network optimization. Kensuke Nakamura and Stefano Soatto and Byung-Woo Hong.