Dirbtiniai neuroniniai tinklai

„Šnekos atpažinimo uždavinys“

3-4 Etapas

Laimonas Beniušis
Informatika Magistras 1k

Turinys

1 Pradinė informacija

- Programavimo kalba Java.
- Mašininio mokymo paketas Deeplearning4j (Java interfeisas).
- Mokymui skirti duomenys – kepstriniai.

2 Paveikslėlių filtras

Pastebėjau, kad duomenų kokybė yra prastoka. Buvo parašytas rankinis pikselių filtras. Jis susideda iš dviejų dalių:

- Binarinis filtras
- Triukšmo pašalinimo filtras

2.1 Binarinis filtras

Filtras kuris skanuoja kiekvieną pikselį ir jeigu jis yra visiškai baltas, t. y. jeigu jo RGB (*angl.* Red Green Blue) reikšmės yra (255, 255, 255) tada jis toks ir lieka, priešingu atveju pikselis yra paverčiamas juodu (0, 0, 0).

2.2 Triukšmo filtras

Filtras kuris skenuoja kiekvieną pikselį ir jo kaimynus.

Turi 3 parametrus: *dydis, diagonal*. *sutapimai*

Dydis nusako kaip tolis yra skenuojami jo kaimynai. Pikselio kaimynas yra jam gretimas pikselis. Jeigu parametras *diagonal* yra tiesa, tai skenuojami visi 8 kaimynai (iš šonų ir įstrižai), priešingu atveju yra skenuojami tik iš šonų esantys kaimynai. Dydis yra suprantamas kaip rekursyvių skanavimų iteracijų kiekis. Tarkime *diagonal* yra tiesa ir pikselis nėra krašte, dydis yra 1, tai bus nuskenuoti 8 kaimynai .Jeigu dydis yra 2, tai bus nuskenuoti 24 kaimynai, nes bus nemažai pakartotinų kaimynų.

Sutapimai parametras yra naudojamas nustatyti ar dabartinis pikselis yra „išsišokėlis“. Kitaip tariant, patikrinama ar šis pikselis yra triukšmas savo kaimynų kontekste. Pikselio spalva yra sulyginama su jo kaimynais ir jeigu sutampančių spalvų yra mažiau negu nurodyta parametre *sutapimai*, pikselis yra klasifikuojamas kaip triukšmas ir paverčiamas fono spalva (balta).

Šį filtrą prasminga naudoti po binarinio filtro, nes tada sumažėja skirtingų pikselių rūšių kiekis.

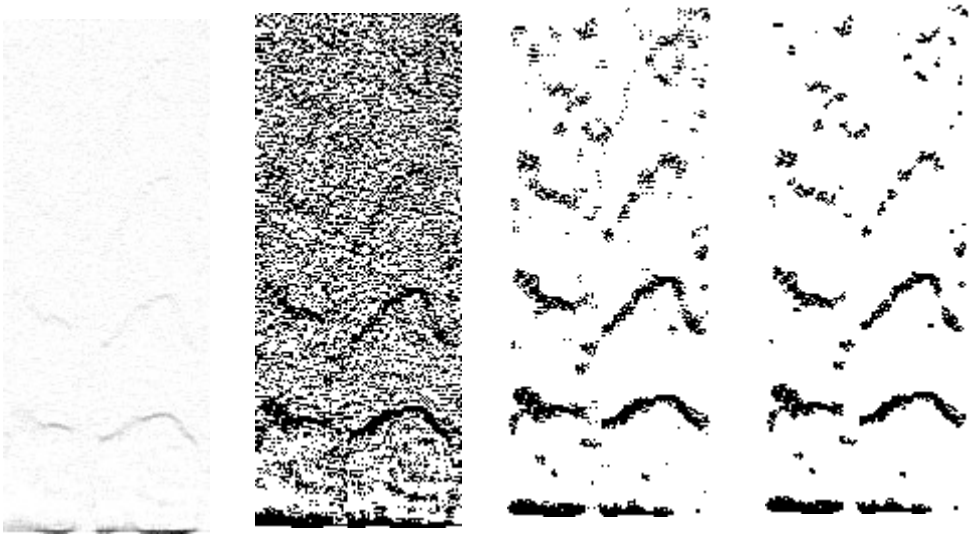
2.3 Filtro panaudojimas

Visiems duomenims buvo panaudota tokia filtrų kombinacija:

1. Binarinis filtras
2. Triukšmo filtras(3, true, 25)
3. Triukšmo filtras(1, true, 3)

Tai reiškia, kad pirmiausiai buvo panaudojamas binarinis filtras, o po to 2 triukšmo filtrai su skirtingais parametrais. Binarinis filtras padaro maksimalų kontrastą, po to pirmas triukšmo filtras išvalo pagrindinį triukšmą ir galiausiai antras triukšmo filtras išvalo mažas pikselių grupes, nuo 1 iki 3 pikselių dydžio juodas „salas“.

Toliau yra pavaizduota neapdoroto paveikslėlio progresija pereinant filtrus iš eilės (nuo kairės):



Šitoks duomenų apdorojimas padėjo išryškinti garso bangų paliktus „pėdsakus“, ir kardinaliai padidino paveikslėlių kontrastą, kas leidžia geriau įsisavinti paveikslėlio formą, tačiau matomai gali būti prarasta informacija, jeigu bangos intensyvumas turi prasmę klasifikavime. Kadangi įvairaus garsumo bangos yra sugrupuojamos, jų **intensyvumas nėra svarbus**.

Filtrų kodas pridedamas ir gali būti naudojamas nepriklausomai.

3 Mokymo parametrai

Kaip jau buvo minėta, buvo panaudotas Deeplearning4j Java mašininio mokymosi karkasas. Tinklas buvo pratestuotas su MNIST aibės duomenimis. Ši duomenų aibė buvo naudojama patikrinti ar programa iš viso korektiškai parašyta, ar teisingai parengti failai bei jų skaitymas, paveikslėlių formatavimas ir išmokimo testavimas.

3.1 Paveikslėlių formatavimas

Visi duomenys yra paverčiami į **30x60** dydžio matricas. Daugumą paveikslėlių yra portreto formato, kur aukščio:pločio santykis yra 1:2-2.5, todėl didelės prasmės galutinis santykis neturi, svarbu tik išlaikytas portreto formatas. Buvo pratestuotas ir su kvadrato dydžio paveikslėliais, tačiau rezultatai buvo prasti. Duomenys normalizuojami iš intervalo [0, 255] į intervalą [0, 1].

Tikriausiai butų įmanoma įterpti minėtus filtrus (binarinis ir triukšmo) į šią sistemą, tačiau tam reiktų gilintis į karkaso duomenų saugojimo ypatumus, apdorojimo eigą, kas užimtų dar daugiau laiko. Galiausiai tos idėjos buvo atsisakyta, nes alternatyva (tiesiog konvertuoti visus paveikslėlius), mano atveju, buvo paprastesnė ir greitesnė. Visgi reikėtų paminėti, kad visą duomenų aibę (~600 000 failų) konvertuoti pilnu pajėgumu užtruko 12 valandų. Konvertuoti duomenys, png kompresijos formato dėka, užėmė daugiau nei 10 kartų mažiau vietos, nes paveikslėliuose liko tik dviejų rūšių spalvos (balta ir juoda).

3.2 Dirbtinis neuronų tinklas

DNT buvo paprastesnis negu panaudotas užduoties demo versijoje, todėl mokymas, t. y., tinklo svorių apskaičiavimas bet keitimas vyko greičiau, lyginant su demo tinklu. Toliau yra pateikta tinklo sluoksnių lentelė.

Layer	Out	Type	Size	Stride	Activation
1a	20	Convolution	5x5	1x1	Identity
1b	-	Subsampling Max Pooling	2x2	2x2	-
2a	50	Convolution	5x5	1x1	Identity
2b	-	Subsampling Max Pooling	2x2	2x2	-
2c	500	Dense	-	-	RELU
3	Class count	Output	-	-	SOFTMAX

Toliau yra pateiktas kodo fragmentas, kuris aprašo tinklą detaliai.

```
MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
        .seed(seed)
        .l2(0.0005)
        .updater(new Nesterovs(0.01, 0.9))
        .weightInit(WeightInit.XAVIER)
        .list()
        .layer(0, new ConvolutionLayer.Builder(5, 5)
                .nIn(channels)
                .stride(1, 1)
                .nOut(20)
                .activation(Activation.IDENTITY)
                .build())
        .layer(1, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
                .kernelSize(2, 2)
                .stride(2, 2)
                .build())
        .layer(2, new ConvolutionLayer.Builder(5, 5)
                .stride(1, 1) // nIn need not specified in later layers
                .nOut(50)
                .activation(Activation.IDENTITY)
                .build())
        .layer(3, new SubsamplingLayer.Builder(SubsamplingLayer.PoolingType.MAX)
                .kernelSize(2, 2)
                .stride(2, 2)
                .build())
        .layer(4, new DenseLayer.Builder().activation(Activation.RELU)
                .nOut(500).build())
        .layer(5, new OutputLayer.Builder(LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
                .nOut(outputNum)
                .activation(Activation.SOFTMAX)
                .build())
        .setInputType(InputType.convolutional(height, width, 1)) // InputType.convolutional for normal image
        .build();
```

Kaip matome, buvo panaudotas Nesterov tipo optimizatorius, su 0.01 mokymosi greičiu ir 0.9 inercija. Taip pat naudojamos Xavier tipo svorių inicializacija, bei Negative log likelihood paklaidos funkcija.

4 Rezultatai

Rezultatų kartėlis buvo demo tinklo rezultatai, kuriuos pageidautina buvo pagerinti. **Tą padaryti pavyko.** Rezultatai prilygsta demo tinklo gautiems ir netgi juos pranoksta. Toliau parodyti geriausi klasifikavimo rezultatai (po 30-os epochos) naudojant 10, 70 ir 111 klasių rinkinius.

# of classes: 10	# of classes: 70	# of classes: 111
Accuracy: 0.8941	Accuracy: 0.7115	Accuracy: 0.6634
Precision: 0.8959	Precision: 0.7382	Precision: 0.6797
Recall: 0.8941	Recall: 0.7115	Recall: 0.6634
F1 Score: 0.8945	F1 Score: 0.7142	F1 Score: 0.6642

Visų užduočių tipų atveju programa vyksta 30-35 epochas, kas yra žymiai mažiau negu demo tinkle. Taip pat, kadangi tinklas yra mažesnis, skaičiavimai vyksta trumpiau, todėl **tinklas yra pranašesnis ir greičiu, ir tikslumu.**

Kaip matome, nebuvo ženklaus pokyčio rezultatuose 10-ies klasių uždavinyje, tačiau pastebimas mažas pokytis (0-2%) rezultatuose 70-ies klasių uždavinyje ir galiausiai akivaizdus rezultatų pagerinimas (2-3% visuose tikslumo matuose) 111-os klasių uždavinyje.

Tikslas sukurti mažesnį tinklą, kuris klasifikuotų duomenis bent jau taip pat gerai ar geriau buvo pasiektas.

Visų rezultatų progresijos yra pridedamos atskiruose failuose.