KONVOLŪCIJU NEIRONU TĪKLU ARHITEKTŪRU ĀTRDARBĪBAS EKSPERIMENTĀLAIS SALĪDZINĀJUMS

EXPERIMENTAL COMPARISON OF CONVOLUTION NEURON NETWORK ARCHITECTURES

Authors: **Ilmārs APEINĀNS**, e-mail: ia16024@edu.rta.lv
Vitālijs ŽUKOVS, e-mail: vz16020@edu.rta.lv
Scientific supervisors: **doc., Dr.sc.ing. Sergejs KODORS,** e-mail: sergejs.kodors@rta.lv
doc., Dr.sc.ing. Imants ZAREMBO, e-mail: imants.zarembo@rta.lv
Rēzeknes Tehnoloģiju Akadēmija
Atbrīvošanas aleja 115, Rēzekne, Latvija

Abstract. *In this work, authors experimentally compare latencies of convolution neuron network architectures. Authors measured only recognition time. Four architectures were applied in the experiment: AlexNet, AlexNet Separated, MobileNetV1 and MobileNetV2. Models were trained using Fruits360 dataset. The Android mobile application was developed to measure latency on mobile devices. The smallest latency authors obtained using AlexNet Separable model, but the smallest size was provided by MobileNetV2.*

Keywords: *CNN, Convolution Neuron Network, Model, Tensorflow*

Ievads

Konvolūciju neironu tīkli (*convolutional neural networks*, tālāk *CNN*) [1] ir speciālas neironu tīklu struktūras, kuras pārsvarā pielieto vizuālu attēlu apstrādei. Tā kā *CNN* ir izstrādāti, lai darbotos, imitējot cilvēka smadzeņu neironu savienojumu modeli, kad atsevišķi neironi reaģē uz stimuliem tikai ierobežotā redzes laukā, kas pazīstams kā "uztverošais lauks". Šie lauki pārklājas, lai aptvertu visu redzes laukumu.

Neironu tīkli, kuri ir apmācīti un darbojas izmantojot personālos datorus, veiksmīgi izmanto datora pieejamos resursus ievaddatu apstrādei, toties reālos apstākļos bieži nav pieejams personālais dators, kurš varētu apstrādāt ievaddatus pietiekoši ātri. Lai izvairītos no tāda veida problēmām, ir iespējams izveidot neironu tīklu modeļus, kuriem nav nepieciešami lieli skaitļošanas resursi rezultāta iegūšanai. Tāpēc tādi neironu tīklu modeļi ir spējīgi pilnvērtīgi strādāt, izmantojot mobilo ierīču resursus.

Šī pētījuma **mērķis** ir eksperimentāli salīdzināt četru konvolūciju neironu tīklu arhitektūru ātrdarbību, mērot attēla apstrādes laiku, pielietojot *Android* mobilo telefonu.

Materiāli un metodes

Darba ietvaros tika apskatītas četras konvolūciju neironu tīklu arhitektūras: *AlexNet* [2], [3], *AlexNet Separable*, *MobileNetV1* [4] un *MobileNetV2* [5].

Lai eksperimentāli salīdzinātu arhitektūras, sākumā četri modeļi tika apmācīti atpazīt ābolu un bumbieru attēlus, pielietojot *Fruits360* datukopu [6], *TensorFlow 2.0* un *Jupyter Notebook*. Eksperimentālos apstākļos tika iegūta precizitāte ~99% (skat. 1. att.). Katrs *CNN* modelis tika eksperimentāli/manuāli samazināts, lai tas aizņemtu pēc iespējas mazāk atmiņas, nezaudējot precizitāti (skat. 1. tabulu). *AlexNet Separable* tika iegūts, aizvietojot *AlexNet* modeļa *Conv2D* slāņus ar *SeparableConv2D* slāņiem.

```
Epoch 35/50
552/552 [==============================] - 113s 205ms/step - loss: 0.0175 - accuracy: 0.9950 - val_loss: 0.0980 - val_accurac
y: 0.9780
Epoch 00035: early stopping
MAX acc: 0.99504644
MAX val_acc: 0.9930186
```

1. attēls. Piemērs ar modeļa apmācības rezultātu

https://doi.org/10.17770/het2020.24.6742

1. tabula

Iegūtu modeļu izmēri un precizitāte

AlexNet: 952 835 *params*, 97,1%	*AlexNet Separable:* 866 750 *params*, 99,8%
MobileNetV1: 801 795 *params*, 99,2%	*MobileNetV2:* 98 379 *params*, 99,8%

Pēc modeļu apmācības nākamais solis, kas tika veikts, bija modeļa sagatavošana izmantošanai mobilajās ierīcēs. Tam bija nepieciešams pārveidot iepriekš apmācīto modeli ar datnes paplašinājumu "*.h5" Tensorflow Lite* [7] modelī ar paplašinājumu "*.tflite"* (skat 2. att.)

```
In [ ]: import tensorflow as tf
        from tensorflow import keras
        path = 'C:/JupyterProjects/TFlite model/mobilenetV1_sk.h5'
        model = keras.models.load_model(path, custom_objects=None, compile=True)
```

```
In [ ]: #prepare for model saving

        export_dir = 'saved_tflite_model'
        tf.saved_model.save(model, export_dir)
```

```
In [ ]: #select mode of conversion

        converter = tf.lite.TFLiteConverter.from_saved_model(export_dir)
        # FOR SPEED
        converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_LATENCY]
        # # #FOR STORAGE
        # converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
        # #DEFAULT
        # converter.optimizations = [tf.lite.Optimize.DEFAULT]
        tflite_model = converter.convert()
```

```
In [ ]: # create tflite file and export it

        tflite_model_file = 'model.tflite'
        with open(tflite_model_file, "wb") as f:
            f.write(tflite_model)
```

2. attēls. *Python* kods tflite modeļa izveidei

Mērījumi tika veikti uzstādot izstrādātu *Android* mobilo aplikāciju uz telefona *Samsung SM-G950F* (*CPU: Exynos 8895, 8 Cores - 2.3GHz Quad + 1.7GHz Quad, GPU: Mali-G71 MP20, RAM: 4GB*) [8].

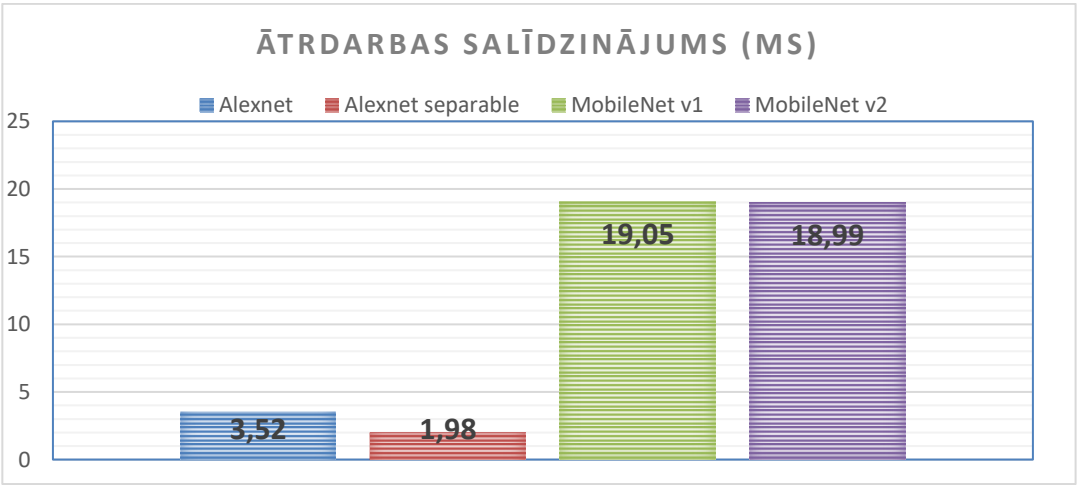
Mērījumi tika veikti, lai noskaidrotu laiku, kas nepieciešams *tflite* modelim attēla atpazīšanai un tā klasificēšanai. Mērījumu mērvienība ir nanosekundes [9], kas dod lielāko precizitāti. Tika ņemts laika mērījums pirms funkcijas uzsākšanas un pēc funkcijas darbības beigām, kas dod divas laika vienības. Laiks tiek ņemts no ierīces precīzākā pieejamā sistēmas taimera pašreizējās vērtības nanosekundēs. Pēc divu vērtību iegūšanas un saglabāšanas mainīgajos, tiek veikta atņemšana, kur no otrā mainīgā tiek atņemts pirmais mainīgais, lai noskaidrotu cik laika tika patērēts noteiktas darbības veikšanai nanosekundēs. Dotā vērtībā tālāk tiek noapaļota līdz 0,01 milisekundei. Kas spēj precīzi attēlot patērēto laiku gan uz vecākām ierīcēm, gan uz jaunākām (skat 3. att.). Pēc mērījumu veikšanas, ierīces operatīvā atmiņa tika attīrīta, lai neietekmētu mērījumu rezultātus.

```
//get start time in nanoseconds
long lStartTime = System.nanoTime();

//run interpreter and save results
tflite_alexnet_seperable.run(imageBytrString, result);

//get end time in nanoseconds
long lEndTime = System.nanoTime();

//get time of interpreter run in nanoseconds
long output = lEndTime - lStartTime;

//set precision to o.1 miliseconds
double floatOutput = output/10000;
byteBuffer.clear();
```

3. attēls. Funkcijas ātrdarbības noteikšanas metode

Rezultāti

Apmācības rezultātā tika iegūts laiks, cik bija nepieciešams *tflite* modelim, lai atpazītu un klasificētu attēlu. Mērījumi tika veikti 100 reizes, pēc katras reizes ierīces operatīvā atmiņa tika attīrīta, lai neietekmētu rezultātus. Darbības laikā tika aprēķināts vidējais laiks, kas nepieciešams *tflite* modelim, lai atpazītu attēlu (skat. 4. att.).



4. attēls. *Tflite* modeļu ātrdarbības rezultāti

Visātrāk klasifikāciju veica modelis ar arhitektūru *AlexNet Separable*, tomēr modelis ar arhitektūru *MobileNetV2* saturēja vismazāko parametru (saišu) skaitu.

Secinājumi

Darba uzsākšanas laikā iepriekš apmācīti modeļi tika optimizēti un pārveidoti *tflite* modeļos. Katras *CNN* arhitektūras darbības laiks tika mērīts 100 reizes un tika izrēķināts vidējais laiks darbības veikšanai. Pētījuma laikā tika novērota liela atšķirība cik daudz bija nepieciešams modelim, lai veiktu attēla atpazīšanu un tā klasificēšanu. Tā kā modeļa apmācībā izmantotie dati bija identiski visiem četriem modeļiem, modeļu ātrdarbību ietekmē neironu tīkla arhitektūra. Situācijas uzlabošanai nepieciešams veikt papildus testēšanu ar modeļu ielādi un

attēlu sagatavošanu. Vēl viena iespēja situācijas uzlabošanai ir izpētīt konvolūciju neironu tīklu arhitektūru struktūru un novērtēt to iespējamu uzlabošanu atbilstoši uzdotajam mērķim.

Summary

At the start of the work, previously trained models were optimized and transformed into tflite models. The running time of each CNN architecture was measured 100 times and the average running time was calculated. During the study period, a large difference was observed in how much time it took for the model to recognize the image and classify it. Since the data used in the model training were identical for all 4 models, the speed of the models is influenced by the neural network architecture. To improve the situation, it is necessary to perform additional testing with model loading and image preparation. Another way to improve the situation is to study the structure of convolutional neural network architectures and evaluate their possible improvement according to the set goal.

Acknowledgement

Funding institution: Latvian Council of Science
Funding number: lzp-2019/1-0094
Acronym: FLPP-2019-1
Funding text: This research is funded by the Latvian Council of Science, project "Application of deep learning and datamining for the study of plant-pathogen interaction: the case of apple and pear scab", project No. lzp-2019/1-0094

Literatūra

[1] Konvolucionāli neironu tīkli vizuālai atpazīšanai [tiešsaiste], [atsauce uz 07.04.2020.]. Pieejams: https://cs231n.github.io/convolutional-networks/
[2] *Alexnet* arhitektūra [tiešsaiste], [atsauce uz 07.04.2020.]. Pieejams: https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637
[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deepconvolutional neural networks.* In Advances in neural information processing systems, pp. 1097–1105, 2012
[4] Andrew G., et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017. [tiešsaiste], [atsauce un 07.04.2020.]. Pieejams: https://arxiv.org/abs/1704.04861
[5] M. Sandler, et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks,* 2019. [tiešsaiste], [atsauce un 07.04.2020.]. Pieejams: https://arxiv.org/pdf/1801.04381.pdf
[6] H. Mureșan & O. Mihai. Fruit recognition from images using deep learning. Acta Universitatis Sapientiae, Informatica, vol. 10, pp. 26-42, 2018.
[7] *Tensorflow Lite* [teiššaiste], [atsauce uz 07.04.2020.]. Pieejams: https://www.tensorflow.org/lite
[8] *Samsung S8* mobilā telefona specifikācija [tiešsaiste], [atsauce uz 07.04.2020.]. Pieejams: https://www.sammobile.com/samsung/galaxy-s8/specs/
[9] *Android Studio* laika mērīšanas funkcija [tiešsaiste], [atsauce un 07.04.2020.]. Pieejams: https://developer.android.com/reference/java/lang/System