APPLICATION OF NEURAL NETWORKS IN ANDROID APPLICATIONS FOR OBJECT RECOGNITION IN REAL TIME

Zeljko Jovanovic[1], Filip Petrovic[1], Mihailo Knezevic[1]

[1] *University of Kragujevac, Faculty of technical sciences Čačak, Serbia*

Abstract

This paper describes how neural networks can be used in Android applications. Specifically, an educational application for language learning based on a neural network model was developed. Several neural network models were trained for object detection as part of the practical part. With the help of these models, an application that detects objects in real-time and translates them into the desired language was created. Several topics were explored in this paper, such as neural networks, artificial intelligence, android platform, TensorFlow and TensorFlow Lite libraries and how they work, and the concept of detection, i.e., object recognition. In addition to the theoretical part, which is necessary to understand how neural networks work and the Android platform, each step of the practical demonstration is described in detail. This includes preparing the working environment, training the data set, training the neural network, and developing the Android application.

Keywords: Android, TensorFlow, neural network, object recognition, image processing.

INTRODUCTION

According to the 2022 report, there are 6.65 billion smartphone users in the world. That's 86% of the global population. As of 2016, the number of users has grown by 50%, from 3.67 billion (45% of the total population at the time). The growth trend is expected to continue, reaching 7.52 billion users by 2026. [1], 85% of the time users spend in applications [1]. The average American spends at least 5 hours on the phone a day [1]. Mobile applications, however, have a huge potential for useful applications. The hardware capabilities of phones are increasing, so the possibilities for realizing more complex activities on phones have become possible. The use of artificial intelligence techniques and the application of neural network algorithms is becoming possible on smartphones. This paper aims to present the use of neural networks in the development of Android applications for real-time object detection and recognition. Edge AI frameworks like TensorFlow Lite and Core ML play a crucial role in deploying lightweight models on Android devices, ensuring efficiency and faster inference.

NEURAL NETWORKS

Neural networks, a fundamental component of artificial intelligence [2][3], have seen significant integration within the realm of Android app development. In the Android applications development neural network are mostly used in:

- **Image Recognition and Classification**: object detection, facial recognition, and scene understanding, automatically categorize and tag images, enhancing user experience.
- **Natural Language Processing (NLP) and Text Analysis**: sentiment analysis, chatbots, language translation, and speech recognition.
- **Recommendation Systems**: offer personalized recommendations in various domains like movies, music, e-commerce, and more. Apps like Netflix and Spotify leverage such recommendation

systems to enhance user engagement.

- **Gesture Recognition and Human Activity Recognition**: gestures, poses, movements types in fitness apps, gaming, and accessibility features.
- **Augmented Reality (AR) and Virtual Reality (VR)**: object recognition, tracking, and scene understanding, enabling compelling AR and VR experiences.
- **Healthcare and Medical Applications**: medical image analysis, disease diagnosis, and personalized health recommendations, identifying anomalies in medical images, aiding medical professionals in diagnostics.

For successful usage in Android application, it is necessary to create and deploy lightweight models, ensuring efficiency and faster user interface. Support for neural networks on Android devices has been available since Android 8.1, released in 2017. Google created the Android Neural Networks API (NNAPI) [4]. This API represents a low-level interface that allows high level libraries to work with that interface. Most popular ones:

TensorFlow Lite: Google's TensorFlow Lite is most widely used libraries for deploying machine learning models on mobile and embedded devices, including Android. It allows the execution of pre-trained models with low latency.

PyTorch Mobile: PyTorch, developed by Facebook, provides a mobile version that allows the deployment of PyTorch models on Android devices.

ML Kit for Firebase: This is a mobile SDK provided by Google that integrates various machine learning features, including image labeling, text recognition, barcode scanning, and more, making it easy to use pre-built models without having to train models from scratch.

NCNN (The NCNN Neural Network Computation Framework): NCNN is a high-performance neural network inference framework optimized for mobile platforms. It's efficient and can be used to deploy models on Android.

Caffe2: Caffe2, now integrated into PyTorch, is a lightweight, modular framework for machine learning. It's suitable for deploying models on Android devices.

MACE (Mobile AI Compute Engine): MACE is another inference framework designed specifically for mobile devices. It supports TensorFlow, Caffe, and ONNX models.

When choosing a neural network library for an Android app, factors such as model compatibility, ease of use, performance on mobile devices, community support, and documentation need to be considered.

TensorFlow Lite was chosen for testing and implementation. The rest will be tested in future.

TENSORFLOF

TensorFlow [5] is an open-source machine learning framework developed by the Google Brain team. It's designed to facilitate the development and deployment of machine learning models, particularly neural networks. TensorFlow allows developers to build, train, and deploy various machine learning models for diverse applications ranging from image and speech recognition to natural language processing and recommendation systems.

At its core, TensorFlow operates on the concept of tensors, which are multidimensional arrays representing the data flow through a computational graph. The framework supports deep learning models through an extensive collection of pre-built operations and functions, making it highly versatile and adaptable for different use cases. The output of the model can be as simple as a number that determines the sequence number of the image class (0 = dog, 1 = cat, 2 = bird), or it can be a complex result, such as the frames of detected objects in the image, with a hit probability expressed by 0 to 1.

Key features of TensorFlow include its flexibility, scalability, and portability. It provides both high-level APIs like Keras [6] for quick model prototyping and low-

level APIs for fine-grained control over the neural network architecture. TensorFlow also supports distributed computing, allowing models to be trained and deployed across multiple devices or distributed computing environments.

Moreover, TensorFlow has a vast and active community, contributing to its continuous improvement, extensive documentation, and a rich ecosystem of tools and libraries. This ecosystem [7][8] includes TensorFlow Serving for model serving, TensorFlow Lite for mobile and edge devices, TensorFlow.js for web-based applications, and TensorFlow Hub for model sharing and reuse.

TensorFlow in object detection:

Object detection is a machine learning task to identify the presence and location of multiple classes of objects within an image. An object detection model is trained on a dataset containing a set of known objects. [9]

The trained model accepts image frames as input and attempts to categorize objects within the image based on a data set of known classes that it was trained to recognize. For each image frame, the object detection model provides a list of objects it detects, the location of the frame for each detected object, and a value indicating the probability that the model successfully classified the object [9][10].

TensorFlow usage environment

For practical usage, it is necessary to prepare a working environment in which all necessary tools are installed. It is necessary to consider the hardware and software elements of the computer in order to prepare the environment in the most efficient way possible.

All scripts for creating data sets and training networks are written in Python version 3.8.0, because it is compatible with version 2.8.0 of TensorFlow which is used for training the neural network.

Installation of the Windows Subsystem for Linux (WSL)[11] enabled efficient working environment. The recommended approach for installing TensorFlow with graphics card support is by using the Miniconda environment [12][13].

DATASET FOR NN TRAINING

FiftyOne [14] is an open-source project that allows the creation of high-quality datasets and computer vision models for 600 different classes. It provides a graphical user interface that enables datasets visualization of and interpret models faster and more effectively. Some of the parameters that can be defined:

- Name of the data set.(Open Images V6, V7, Kinetic 700…)
- Split, i.e. whether a data set is being created for training, testing or validation
- Label types
- Sample classes, ie. object names
- Maximum number of samples

Example Python code for downloading a dataset for training object detection, from the Open Images V7 dataset is presented below. The resulting dataset consist of five hundred samples, featuring dogs and/or cats and/or beer classes:

```
dataset = foz.load_zoo_dataset(
    "open-images-v7",
    split="train",
    label_types=["detections"],
    classes=["Cat", "Dog", "Beer"],
    max_samples=500
)
```

Presentation of the selected dataset is shown in Fig. 1.

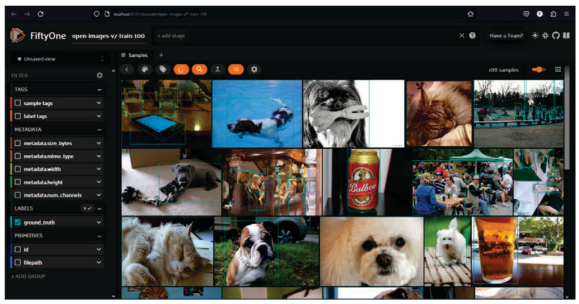

Fig. 1. *Dataset presentation in FiftyOne application*

The dataset is downloaded in next percent distribution: 80% for the training, 10% for the test, and 10% for the validation.

For the neural network model training several datasets presented in Table 1. are used.

Table 1. Used datasets

No.	Number of classes	Number of samples
1	3 “Cat", "Dog", "Beer"	500
2	3 “Car", "Person", "Truck"	500
3	170	5000
4	170	20000

Dataset 3 and 4 contained 170 various classes.

NN MODEL TRAINING

Four model architectures in eight combinations are trained. Parameters set for each model during training, namely:

- Model architecture
- Dataset
- Number of epochs
- Batch size

Training of models is very time consuming and mostly depends of model architecture and number of samples.

Technical specification of a computer used for neural network model training are: processor: Intel Core i5 11400H, graphic card: NVIDIA GeForce RTX 3050Ti with 4GB RAM and 2560 Cuda cores, RAM memory: 32 GB, operating system: Windows 11.

All parameters for model training are shown in Table 2. Batch size was set to the maximum that the computer could support for other given parameters.

Table 2. Parameters setup for models training

No.	Model architecture	Dataset	Number of epochs	Batch size
1	EfficientDet Lite0	1	10	8
2	EfficientDet Lite0	1	20	8
3	EfficientDet Lite0	2	50	8
4	EfficientDet Lite1	2	50	4
5	EfficientDet Lite2	1	50	2
6	EfficientDet Lite3	2	50	1
7	EfficientDet Lite0	3	50	4
8	EfficientDet Lite0	4	50	4

All these architectures are derived from the EfficientDet Lite architecture. EfficientDet Lite0 is the smallest and simplest, but the least accurate. Each subsequent one has more layers and is more precise, but takes up more space. Also EfficientDet Lite0 is the fastest and requires the least hardware resources, while each subsequent one is slower because it requires more memory and computing power. EfficientDet Lite0 is used where real-time, lag-free processing is critical, and EfficientDet Lite4 where accuracy is more important than speed.

NN MODEL EVALUATION

All 8 combinations of settings presented in Table 2. are tested and evaluated with Keras - The high-level API for TensorFlow[5][6]. Its tf.keras.Model.evaluate method provide results for every evaluated class in a range from 0 to 1. Results for the first six are shown in Table 3. Results for detection of selected classes vary from 0.38 to 0.7. EfficientDet Lite3 in theory should achieve best results but that was not the case in all scenarios. Results for the parameter setup 7 and 8 from Table 2. are not presented in Table 3 since there are 170 classes tested. Overall results for No. 7 test case were very bad (average 0.1) since the 5000 number of samples for training of 170 classes was too low. Overall results for No. 8 test case were significantly better (average 0.25, but some classes > 0.5) but not enough for successful usage for all classes. Training phase for No. 8 test case last for 10 hours. Training with a larger dataset would last much longer and is not implemented and tested in this application.

Table 3. Models’ evaluation results

Parameters setup No.	Class 1	Class 2	Class 3
1	0.38	0.52	0.53
2	0.49	0.66	0.65
3	0.49	0.66	0.7
4	0.52	0.68	0.65
5	0.58	0.67	0.59
6	0.38	0.54	0.7

ANDROID APPLICATION FOR OBJCET RECOGNITION

For a practical demonstration, a native Android application developed with Java programming language was created. It is built on MVVM (Model-View-ViewModel) architecture. The model part presents models and data sources. View is the presentation layer. ViewModel serves to connect these two layers.

The application uses neural network model in an innovative way to detect and recognize objects at which the phone's camera is pointed. A trained neural network model is developed with the help of the TensorFlow Lite library. Frames from the camera are sent to the input of the model, and the output of the model gives data about detected objects. Location and name of the object is presented on the screen in real time. Projects detection package contain classes used to integrate the neural network model. These classes pass frames from the camera, and returns the results of the model. In the assets folder there is a trained neural network model.

The application provides the user with information about the detected objects in the local language and in the selected desired language for translation. Google's Cloud Translation API is used for translations. The application must first ask the user for access to the camera, since any application that accesses the camera must first ask for the user's permission to have access.

Translate functionality of the application is developed as a demo for successful NN model integration in Android applications. Only classes that are trained could be detected and translated. Application versions with NN model trained with the first dataset should detect "Cat", "Dog" or "Beer" objects successfully. Application versions with NN model trained with second dataset should detect "Car", "Person" or a "Truck "objects successfully.

All 8 model combinations presented in Table 2. are tested in practice.

Application usage presented in Fig. 2. uses No. 3 parameter setup from Table 2. Application usage presented in Fig. 3. uses No. 4 parameter setup from Table 2. They had best performance and detection results balance.

As presented in Fig. 2. "Cat" and "Dog" object were successfully detected while 1 of 3 "Beers" was not detected and recognized.

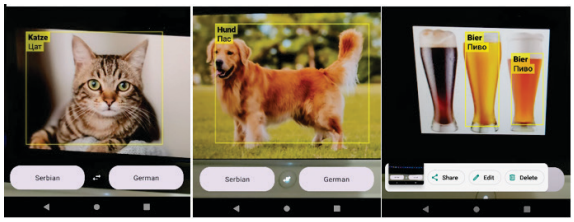

Fig. 2. *Android application usage in object detection for parameter setup No 3 from Table 2.*

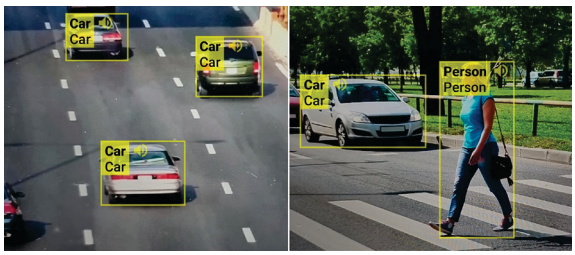

Fig. 3. *Android application usage in object detection for parameter setup No 4 from Table 2.*

On the Fig. 3 all objects were successfully detected. There was no problem with multiple "Car" object occurrence. Even model evaluation results could be better application usage in practice for parameter setup No 4 from Table 2. was successful in almost every situation with trained classes.

Others application versions showed lower results in practice. That was expected since model evaluation results presenter in Table 3. were lower for them. Third and fourth datasets contained 170 classes but for successful detection of all 170 classes larger number of samples should be used for training. Training phase for No. 8 test case from Table 2. lasted for 10 hours. Training with a larger dataset would last much longer and is not implemented and tested in this application. Application usage on object which classes are not trained

showed unexpected results or didn't show any detected object.

CONCLUSION

This paper presented application of neural network models is Android application in real time usage. TensorFlow was successfully implemented and object were detected and recognized.

The quality of the trained model depends on the number of classes, the number of samples for each class in the dataset, the architecture of the model, and the number of epochs. Lower class dataset sample number or uneven data set for training produced lower results in object recognition. Thus, accuracy issues could be addressed by selection of desired classes, improving the dataset creation and preparation scripts. Open Images V7 consists of 9 million images occupying 500 gigabytes of space. Selection of desired classes for detection and increasing number of samples for each class will increase the results of object detection.

Application usage in object detection could be very successful if classes for detection are selected and a NN model is trained with larger number of samples with selected classes.

ACKNOWLEDGEMENT

The research in this paper was supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, as part of the Project grant no. 451-03-47/2023-01/200132 with University of Kragujevac - Faculty of Technical Sciences Čačak.

REFERENCE

[1] 40 Fascinating Mobile App Industry Statistics [2023]: The Success Of Mobile Apps In The U.S, Retrieved from: https://www.zippia.com/advice/mobile-app-industry-statistics/, Date: 01.08. in 2023

[2] Charu C. Aggarwal: Neural Networks and Deep Learning, 2018.

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville: Deep Learning (Adaptive Computation and Machine Learning series), 2016.

[4] Neural Networks API, downloaded from: https://developer.android.com/ndk/guides/neuralnetworks, dated: 08/02/2023.

[5] Amita Kapoor, Antonio Gulli, Sujit Pal, Francois Chollet: Deep Learning with TensorFlow and Keras: Build and deploy supervised, unsupervised, deep, and reinforcement learning models, 3rd Edition, 2022.

[6] Rowel Atienza: Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition, 2020.

[7] Revathi Gopalakrishnan, Avinash Venkateswarlu: Machine Learning for Mobile: Practical guide to building intelligent mobile applications powered by machine learning, 2018.

[8] Vasco Correia Veloso: Hands-On Artificial Intelligence for Android: Understand Machine Learning and Unleash the Power of TensorFlow in Android Applications with Google ML Kit (English Edition), 2022.

[9] Benjamin Planche, Eliot Andres: Hands-On Computer Vision with TensorFlow 2: Leverage deep learning to create powerful image processing apps with TensorFlow 2.0 and Keras, 2019.

[10] Object detection with Android, downloaded from: https://www.tensorflow.org/lite/android/tutorials/object_detection, date: 09.08.2023.

[11] Windows Subsystem for Linux, retrieved from: https://en.wikipedia.org/wiki/Windows_Subsystem_for_Linux, dated: 22.08.2023.

[12] Install Ubuntu on WSL2 and get started with graphical applications, downloaded from: https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-11-with-gui-support#1-overview , dated: 22.08.2023.

[13] Enabling GPU acceleration on Ubuntu on WSL2 with the NVIDIA CUDA Platform, downloaded from: https://ubuntu.com/tutorials/enabling-gpu-acceleration-on-ubuntu-on-wsl2-with-the-nvidia-cuda -platform#2-install-the-appropriate-windows-vgpu-driver-for-wsl, dated: 22.08.2023.