Migration of Artificial Neural Networks to Smartphones

Milan Kostak(✉), Ales Berger, and Antonin Slaby

Faculty of Informatics and Management, University of Hradec Králové, Rokitanského 62, 50003 Hradec Králové, Czechia
{milan.kostak, ales.berger, antonin.slaby}@uhk.cz

Abstract. The paper explains the process of migration of an artificial neural network (ANN) to a smartphone device. It focuses on a situation when the ANN is already deployed on a desktop computer. Our goal is to describe the process of the migration of the network to a mobile environment. In the current system we have, images have to be scanned and fed to a computer that is applying the ANN. However, every smartphone has a camera that can be used instead of a scanner. Migration to such a device should save the overall processing time. ANNs in the field of computer vision have a long history. Despite that, mobile phones were not used as a target platform for ANNs because they did not have enough processing power. In the past years, smartphones have developed dramatically, and they have the processing power necessary for deploying ANNs now. Also, major mobile operating systems, Android and iOS, have included the support for the deployment.

Keywords: Artificial neural networks · Smartphones · Android · Machine learning

1 Introduction

In the last decade, with the rise of available computational power, artificial neural networks (ANNs) became a popular and useful tool for solving problems that do not have an easy solution otherwise. Computer vision, time series prediction, text filtering, and speech recognition are just a few of many disciplines that utilize ANNs.

ANNs are one of the computational models used in artificial intelligence. They are vaguely inspired by the biological structures and intelligence as we know it. When a human is trying to learn something, he usually looks at a couple of examples. By learning these examples, he is later able to apply this knowledge in new situations that he did not see before. ANNs use this concept, and it is called training. The general process is called machine learning. Every case of a training dataset is fed to the ANN, which processes it and improves itself based on the error that the input caused on the output. This is repeated many times, and the process is called supervised learning. If the process is successful, then the trained ANN can take the input that it did not see before and produce the correct output with a certain probability. There is also a process of unsupervised learning, but that goes beyond the scope of this short introduction.

© Springer Nature Switzerland AG 2020
O. Gervasi et al. (Eds.): ICCSA 2020, LNCS 12249, pp. 845–858, 2020.
https://doi.org/10.1007/978-3-030-58799-4_61

Typical ANN includes so-called neurons (from tens to billions), which are the primary unit of any neural network. These neurons are grouped into layers, and they are connected between those layers. Every neuron in one layer is taking one output from every neuron of the previous layer and sends its single result to every neuron in the next layer. Deciding on optimal neurons and layers counts is not an easy problem, and it heavily depends on the application. Usually, a compromise is necessary between the high computational demands and the speed of training. The more neurons and layers the ANN has, the more demanding the training is. However, only bigger networks can grasp more complex problems.

ANNs are used in numerous areas of everyday life:

- speech recognition in virtual assistants like Siri, Cortana, Alexa, or Google Home [1–3],
- checking for malicious posts on social media [4],
- postprocessing of photos on smartphones [5],
- path planning for car navigation that counts for expected traffic [6, 7],
- detection of handwritten zip code in postal services [8],
- other smart experimental solutions [9, 10].

A smartphone is a kind of electronic device. Other examples include smartwatches, smart cars, smart homes, and many others. The great thing about them is that they are ubiquitous, small, connected to the Internet, and have much computational power. That makes them an ideal device for many use cases.

Both major operating systems for smartphones, Android and iOS, support deployment of ANNs in the form of trained models. ANNs are commonly deployed on desktop computers or in the cloud. The significant advantage of both major smartphone platforms is that they are ready for the migration of these trained models. These platforms include support for deploying those models, but each has its process and supported format of models. We are addressing these issues in this paper.

2 Problem Definition

We have a system for image processing. This system consists of several parts (Fig. 1). At first, it is necessary to scan the images. The scanned files are transferred to a desktop computer, which then applies a trained ANN to it. At the final step, the results are automatically sent to a server that stores all data.

We were seeking a solution that will save the time necessary to execute all the steps as mentioned above. We think that the part of scanning and data collecting is the only part, which processing can be significantly shortened. The processing of the image by ANN is taking only milliseconds. Internet communication is also fast enough to transfer the data in a couple of seconds, depending on the image size, so the only part remaining is the scanning. The server part does not need any changes, as its API is universal and does not affect the processing time.

We found a paper [11] in which the authors did a text recognition in document images that are obtained by a smartphone. They did the recognition with deep convolutional and recurrent neural networks. We are trying to solve a different problem,

but we want to apply a similar approach of using an ANN on a smartphone to process a document image. We want to save the processing time by combining scanning and detection processes on a single device – a smartphone. That requires migrating our model, which is designed and trained in a TensorFlow environment. We describe this process in this paper.

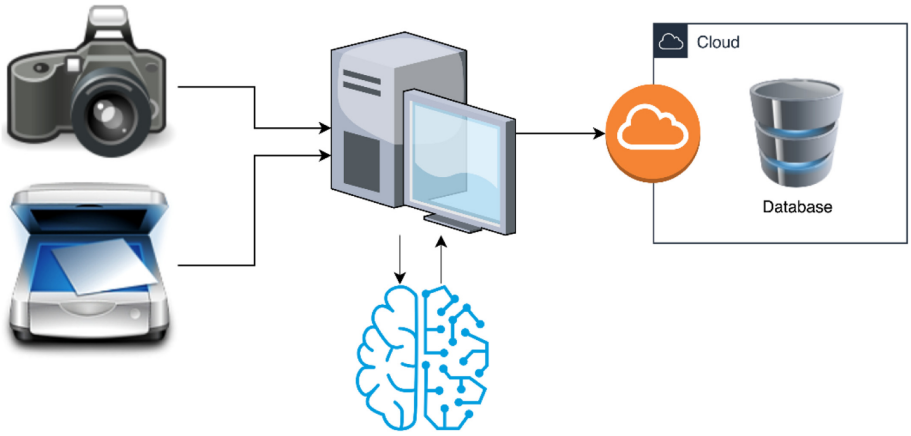

Fig. 1. This figure illustrates the workflow of our previous system. A scanner or a camera is used to obtain the image of a document. This document is processed on a desktop computer by an ANN. The output, together with the document, is sent to the server.

3 Related Work

The most relevant related works on the topics of artificial neural networks, object detection, and deployment of ANNs on smartphones are presented.

3.1 Artificial Neural Networks

ANNs find applications in many areas. One of the most common uses in everyday life are virtual personal assistants. At their core lies the speech detection process. Sriram et al. [2] describe a general framework for robust speech recognition that uses generative adversarial ANN. Their framework focuses on solving problems that current systems suffer from, like ambient noise, different accents, or variations in reverberations. Këpuska and Bohouta [3] propose a solution for multi-modal systems that process two or more user input modes. Apart from the speech itself, the modes include touch, manual gestures, or head and body movements. Their goal is to design a next-generation prototype of virtual personal assistants.

Many people might not realize it, but automatic zip code detection is used in postal services for several decades now. With the advent of ANNs, systems that utilize them were also implemented in these areas. LeCun et al. [8] in 1989 described a method for the use of backpropagation, which is the core of machine learning these days, to the problem of zip code recognition.

Another example of the day-to-day use of ANNs is the postprocessing of photos taken on smartphones. The cameras of these devices usually have to deal with physical limitations like small sensor size, or compact lenses [5]. Ignatov et al. [5] proposed a solution for this problem by postprocessing of taken photos by ANNs, specifically generative adversarial ANNs. These proposed networks help to reconstruct missing details in the pictures.

3.2 Object Detection

Object detection is a core problem in computer vision. Humans can recognize a large variety of objects instantly in almost any image. However, this is a difficult problem for computers because objects of the same category (people, animals, etc.) can vary significantly in appearance. Many different frameworks and systems have been developed in the past. More simplistic approaches include Haar features [12], SIFT (Object Recognition from Local Scale-Invariant Features) [13], HOG (Histograms of Oriented Gradients) [14], or DPM (Deformable Parts Models) [15].

In the last two decades, the use of ANNs became popular in computer vision benefitting from powerful GPU systems. The principle of ANNs was firstly described in the 1960s. It was not until 1986 when Rumelhart et al. [16] introduced a faster approach to backpropagation, which, to this day, is the base technique of learning ANNs. Modern approaches to computer vision include R-CNN (Regions with Convolutional Neural Network features) [17–19], YOLO (You Only Look Once) [20–22], or SSD (Single Shot Detector) [23].

Deformable Parts Models
DPM [15] uses a sliding window approach to object detection. DPM assumes an object is constructed by parts. For instance, a person's face consists of two eyes, a nose, and a mouth. When it finds a match of the object (face), it then fine-tunes the result using the model parts (eyes, nose, mouth). It is based on the idea of penalization when a part is not at the position that it is expected to be – something that is supposed to be a face and does not contain eyes and mouth is probably not a face. DPM uses a multi-stage pipeline for extracting static features, classifying regions, and predicting bounding boxes. This makes it slower in comparison with machine learning approaches like YOLO [20] or R-CNN that use only a single pipeline.

R-CNN
R-CNN is an object detection system first published in 2014 by Girshick et al. [17]. The system consists of three modules. The first generates region proposals, the second is a CNN that extracts features from each region, and the third is a set of class-specific SVMs. The main advantage in comparison to previous systems like DPM [15] is the precision of the detection. It achieves 53.3% mAP (mean average precision) on PASCAL VOC (Visual Object Classes) 2012 dataset [24].

Fast R-CNN [18] is an improvement to R-CNN. It is focused on increasing speed and detection accuracy. Training is 9 times faster, and testing is 213 times faster than for R-CNN. The system consists of only one pipeline compared to three in R-CNN. The architecture contains several convolutional and max-pooling layers, which, together with fully connected layers, replace all three modules of the previous solution. On

the output, there are two sibling layers. The first outputs probability distribution for categories, and the second returns bounding box offsets for each category. The system achieves 68.4% mAP on PASCAL VOC 2012 dataset [24].

Faster R-CNN [19] further improved Fast R-CNN. This detection system achieves 5 FPS at test time while having 70.4% mAP on PASCAL VOC 2012 dataset [24].

In the later years, the method was further improved by some authors, e.g., Mask R-CNN [25], Cascade R-CNN [26], Deep Residual Learning [27], or Rethinking the Faster R-CNN [28].

YOLO

YOLO [20] is an object classifier and detector developed in 2016. The architecture focuses on real-time detection of objects while maintaining good precision, which is something that previous approaches did not achieve. YOLO uses only a single neural network that predicts bounding boxes and class probabilities in one evaluation. The network has 24 convolutional layers, followed by 2 fully connected layers. The base YOLO model processes an image at 45 FPS and achieves more than twice the mAP of other real-time systems. On PASCAL VOC 2012 dataset [24], it reaches 57.9% mAP. Although, as the authors point out, the architecture has problems with precise localization. However, at the same time, it is less likely to predict false positive on the background because it looks at the whole image, and the neural network can decide on the global context. DPM [15] and R-CNN [17] use different approaches. YOLO also struggles with detections of small objects [20].

YOLOv2 [21] is an improved version of YOLO from the same authors. Their main goal was to improve localization when maintaining the speed of detection and classification accuracy. The reworked network has only 19 convolutional layers and 5 max-pooling layers. They added batch normalization, which led to better convergence and a 2% improvement in mAP. The original version of YOLO was trained with 224×224 resolution, and YOLOv2 increases it to 448. Further improvements include dimension clusters, which make it easier for the network to learn. Systems like R-CNN [17] use handpicked anchor boxes. YOLOv2 detection was improved with anchor boxes, but the improvement was small. The authors decided to run k-means clustering on the training set bounding boxes to find them automatically. That provided better improvements that manually picked anchor boxes. They did not use standard k-means with Euclidean distance but rather Intersection over Union (IoU) scores, which are independent of the size of the box. In the end, $k = 5$ was chosen. YOLOv2 achieves 73.4% mAP on the PASCAL VOC 2012 dataset [24]. The performance is comparable with other detectors like Faster R-CNN and SSD while being faster [21].

YOLOv3 [22] is the latest version of the YOLO detector and classifier and was published in 2018. The new model contains 53 convolutional layers. K-means clustering now uses 9 clusters. YOLOv3 predicts boxes at three different scales using a similar concept to feature pyramid network. The original YOLO model struggled with small objects. YOLOv3 uses multi-scale predictions and has worse performance on larger objects. YOLOv3 has a comparable mAP to RetinaNet [29] and Faster R-CNN [19] while being 3-4 times faster [22].

3.3 ANNs on Smartphones

ANNs deployed on smartphones usually must deal with a lower memory capacity and lower computational power. These devices also usually run on a battery, and this fact must be taken into consideration too. Some authors investigated the possibilities of deploying neural networks on smartphones and other devices. The most relevant and interesting works are summarized.

El Bahi and Zatni [11] propose a system of processing document images directly on a smartphone with the use of ANNs. Their goal is to recognize text in the documents. The processing includes pre-processing that detects the document and an ANN architecture that detects the text by lines. The architecture combines convolutional and recurrent neural networks.

Bhandari et al. [30] present a solution for driving lane detection on a smartphone. They argue that providing lane information with only GPS is not accurate enough. Lane detection is vital to deciding whether a car is in the correct lane for making a turn or if the car's speed is compliant with a lane-specific speed limit. Sensors like an accelerometer can detect a lane change, but they are not able to keep the information fresh over a long period when the car is not changing lanes. The authors propose a system for lane detection with a smartphone camera and processing the images with ANNs. They achieve over 90% accuracy of the vehicle's lane position. The system is implemented as an Android application.

Ignatov et al. [31] did an artificial intelligence benchmark of running ANNs on Android smartphones. They present several tests that measure the performance of running ANNs, for instance, image recognition, face recognition, image deblurring, image enhancement, and more. They also present an overview of hardware acceleration for the execution of ANNs. The authors did follow-up research in 2019 with similar tests but with to date hardware [32]. However, a significant drawback of both papers is that they are focused solely on the Android platform.

Niu et al. [33] present a work that focuses on the optimization of ANNs for running on smartphones. Model compression usually degrades the accuracy, so they propose a framework for more efficient and advanced compression to minimize the accuracy loss and maximize the compression rate. They were able to achieve up to 8.8 times faster test time over TensorFlow Lite compression.

Begg and Hasan [34] describe the use of ANNs in home automation. Sensors data are too varied, and a neural network can find relationships and patterns in them. In the paper, they sum up the basics of neural networks and present relevant works to the field of deployment ANNs in smart homes. This means that ANNs are not limited only to smartphones but can be deployed to a broad range of smart devices.

4 Technologies

Many technologies are assisting with training, deployment, and migration of ANNs. Among the most popular libraries and frameworks for training and deployment are Keras, TensorFlow, and PyTorch. Both major smartphone operating systems, Google's Android and Apple's iOS, have their tools for the migration of ANN models. Migration

on Android is possible via TensorFlow Android runtime, and migration on iOS is done via CoreML.

4.1 Keras

Keras [35] is a high-level artificial neural networks API (application programming interface). It is written in Python, and it allows designing and training of ANNs. It offers a set of abstractions that make it easy to develop neural network models. Keras models can be deployed to a range of platforms and environments, including Android, iOS, web browser, Google Cloud, or Raspberry Pi [36].

Keras provides support for many kinds of layers, activation function, optimizers, loss functions, and more. Apart from standard layers, Keras offers convolutional layers, pooling layers, recurrent layers, and others less commonly used. [35].

Keras itself currently has three backend implementations: TensorFlow by Google, CNTK by Microsoft, and Theano by the University of Montreal [37].

4.2 TensorFlow

TensorFlow [38, 39] is a library for machine learning developed by Google. It is one of the implemented backends for Keras. TensorFlow can also be used in the web browser or Node.js environments with TensorFlow.js [40] version of the library, which supports not only deployment but also building and training of the models. TensorFlow Lite (TF Lite) [41] is a variant for mobile and embedded devices. It provides compression of trained models, so they run smoothly on devices that have less computational power.

Many companies are using TensorFlow in their commercial applications [42]. Airbnb uses TensorFlow to categorize listing photos of people's homes. PayPal is using the library for payment fraud detection. Twitter developed a system for ranking tweets to show users the most important tweets first.

4.3 PyTorch

PyTorch is a machine learning library. It supports the designing and training of ANNs on GPUs. It has interfaces written in Python and C++ [43].

PyTorch Mobile is a version of the library that allows the deployment of PyTorch models on Android and iOS devices. The mobile version is focused on improved performance on mobile hardware [44].

4.4 Migration Frameworks

The utilization of the potential of ANNs on mobile devices requires the use of technologies that can migrate these networks. The following subsections describe several approaches that are available today. During the research, we have focused on the two most popular platforms – Android and iOS.

TensorFlow Lite and Android

TensorFlow Lite (TF Lite) is a system that provides support for migrating ANN models to Android and iOS devices. It is designed to execute models efficiently by taking into

consideration the limitations that smartphone environments pose, like less memory, running on a battery, or limited computational power. TensorFlow models must be converted into a format that can be used by TF Lite. Converting the model is necessary to reduce their file size and to optimize it for running on a mobile device while not affecting accuracy. It is also possible to convert the model more drastically, which can affect the accuracy of the trained model. TF Lite supports only a limited number of functions, which means that not all models can be converted. TF Lite converter is a special tool available in the Python environment which converts the models [45].

TF Lite is prepared to be used on both major mobile platforms – Android and iOS. Android uses Java and TF Lite Android Support Library. For the iOS environment, native iOS libraries written in Swift and Objective-C are available [45].

Google also offers the ML Kit for Firebase. It is possible to use it on Android and iOS, and it helps with the online distribution of the trained model through the Firebase services. This kit makes it easy to apply technologies like Google Cloud Vision API, or the Android Neural Networks API in a single SDK (software development kit) [46].

Neural Networks API (NNAPI) is available since Android 8.1 (API level 27). NNAPI offers support for efficient distribution of the computation workload across available processors, including dedicated neural network hardware (sometimes called TPU – Tensor Processing Unit), graphics processing units (GPUs), and digital signaling processors (DSPs). If the devices lack specialized hardware, the NNAPI executes the workload on CPU [47].

Core ML

Apple, as the developer of the iOS platform, also provides tools that help with the migration of existing ANNs to their devices. The tool is called Core ML [48], and it defines the format of models that can be deployed. The technology effectively uses the computational power of the devices, mainly CPU and GPU. Modern Apple devices also contain a so-called Neural Engine, which helps minimize memory and battery consumption [48].

The models are used directly on the device, which therefore does not require an Internet connection, and the data remains on the device as a safety precaution. Core ML itself contains a Vision module for analyzing images, Natural Language module to help with text processing, Speech module for audio to text conversion, and Sound Analysis module to support identifying sounds in audio. Before deploying the ANN model on the device, it is necessary to convert it into a format that is required by Core ML. Several tools support such conversion, for example, MXNet converter [49], or TensorFlow converter [50]. Apple recommends both converters [48].

5 Implementation

After careful consideration and analysis of all possibilities, we designed the following solution. Currently, the majority of users of our system are using Android devices, so we decided to make the pilot implementation for this platform. We are using TF Lite, so there is an easy possibility to migrate the model to iOS devices in the future. We decided not to use the ML Kit, which offers advantages like the possibility to update

the underlying model without updating the application. Our application is not publicly available, and it is not necessary to update the model often. Therefore, we distribute the model inside the application installation file. The development was done in Android Studio 3.6.1.

The computational power of a typical mobile device is sufficient for running the application. The most crucial factor is the input image from the camera, which replaces an external scanner or manual inserting of images. This means that the quality of images from the camera is essential.

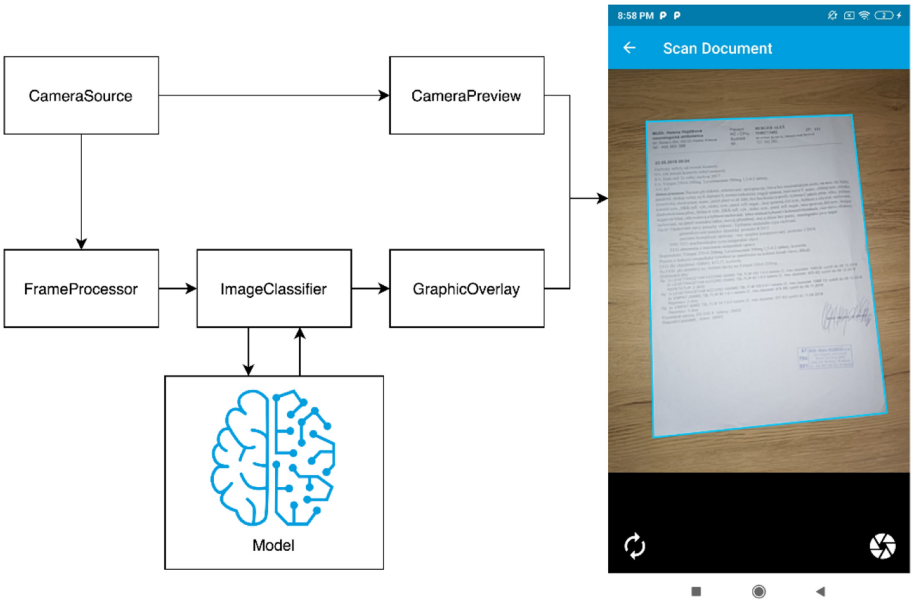

Fig. 2. The figure illustrates the workflow of our new solution in the form of a pseudo-class model.

The workflow of the application is as follows (Fig. 2). The first step is obtaining an image of the document from the camera. Class `CameraSource` is working with the camera stream, and its main task is to distribute the individual images taken by the camera in the highest frequency possible. Class `FrameProcessor` takes only a single image on its input. At first, it does a detection of the document in the input image. That is done with a custom function implemented with the help of OpenCV functions. Then the class transforms the document image into a format that is suitable as input for the detection. The detection is under the control of the class `ImageClassifier`. Its main job is processing the image by the neural network and receiving the results from it. Components `CameraPreview` and `GraphicOverlay` oversee rendering. `CameraPreview` renders the camera stream, and `GraphicOverlay` draws the results from the neural network.

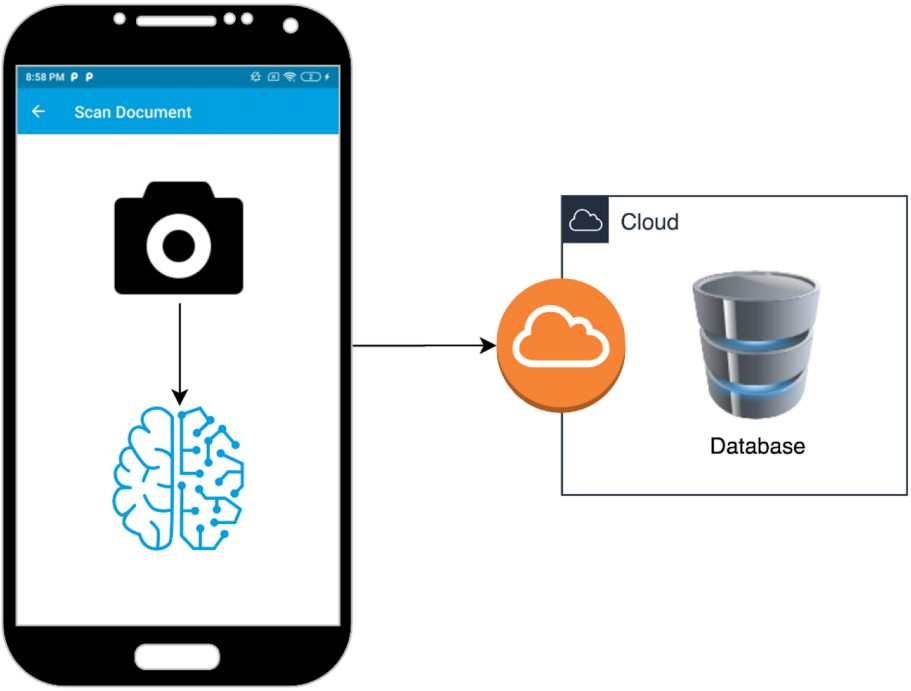

Fig. 3. The figure illustrates the new workflow of the system as compared to the workflow of the old system in Fig. 1. The user takes a picture of the document with the smartphone, which processes the document and sends the result on the server.

6 Testing and Results

Testing proved that the new solution provides faster processing than the old solution. The average time of processing one document using the old system was 75 s under ideal conditions. The time includes putting the paper on the scanner, the scanning process, and the detection process on the computer. Scanning was done with a standard end-user scanner, and since we require high-resolution images, the scanning process was taking a long time. Images are scanned in A4 format with 600 dpi and with deblurring and correction of document skew.

The average time of processing with the new solution is just 26 s. That includes putting the paper on a table, pointing at it with a smartphone, and pressing a button to take a picture. The application does the processing of the document, and the user is only required to confirm sending the data to the server. The document processing is taking only milliseconds even on low-end smartphones, and that means that it almost does not affect the total time. The new system is, on average, almost 3 times faster than the old solution. The testing was done on two devices, Xiaomi Mi MIX 2 and Huawei Honor 9.

7 Discussion

The main advantage of migrating the system to a smartphone is a faster overall procedure. By eliminating the slow scanning process and by automated image handling, we were able to achieve around 3 times faster processing.

Another main advantage of the solution is the adoption of cheaper hardware. Now we need only a smartphone instead of a scanner or standalone camera and a desktop computer. The user experience is better as the user is required only to take a picture with the smartphone, and all the next steps are automatic. The user is only asked to confirm to send the results to the server. That means that the overall solution is much simpler and less error-prone. The application requires only access to the camera and access to the Internet.

One possible disadvantage is battery consumption. The application requires much computational time, which leads to fast drainage of the battery. In our use case, this does not pose a disadvantage because the smartphone can always be plugged in.

In the end, we were able to process a single document almost 3 times faster while using fewer devices, which leads to cost saving in hardware and software.

8 Conclusions

Our main goal of migrating an existing artificial neural network (ANN) to a smartphone device was successfully achieved. Our testing proved that both major smartphone platform vendors, Google for Android and Apple for iOS, provide tools for easy conversion and deployment of trained ANNs. A detailed description of these tools was presented.

Our existing ANN was migrated from a desktop computer to a smartphone. Testing has shown that the network's outputs are the same as expected. With this approach, we were able to save time that is required to process a document. The processing is also more straightforward and, therefore, less error-prone.

In the future, our research will focus on the possibility of lowering the battery consumption and improvements in the deployment of the ANN.

Acknowledgement. This work and the contribution were supported by the project of Students Grant Agency – FIM, University of Hradec Kralove, Czech Republic.

References

1. Michel, E.: Using deep neural networks for automated speech recognition, p. 19 (2015)
2. Sriram, A., Jun, H., Gaur, Y., Satheesh, S.: Robust speech recognition using generative adversarial networks. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, pp. 5639–5643. IEEE (2018). https://doi.org/10.1109/ICASSP.2018.8462456

3. Këpuska, V., Bohouta, G.: Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home). In: 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), pp. 99–103 (2018). https://doi.org/10.1109/CCWC.2018.8301638
4. Shu, K., Sliva, A., Wang, S., Tang, J., Liu, H.: Fake news detection on social media: a data mining. Perspective (2017). https://doi.org/10.1145/3137597.3137600
5. Ignatov, A., Kobyshev, N., Timofte, R., Vanhoey, K., Van Gool, L.: DSLR-quality photos on mobile devices with deep convolutional networks. In: Presented at the Proceedings of the IEEE International Conference on Computer Vision (2017)
6. Amin, S.M., Rodin, E.Y., Liu, A.-P., Rink, K., García-Ortiz, A.: Traffic prediction and management via RBF neural nets and semantic control. Comput. Aided Civil Infrastruct. Eng. **13**, 315–327 (1998). https://doi.org/10.1111/0885-9507.00110
7. Duan, Y., Lv, Y., Wang, F.-Y.: Travel time prediction with LSTM neural network. In: 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, pp. 1053–1058. IEEE (2016). https://doi.org/10.1109/ITSC.2016.7795686
8. LeCun, Y., et al.: Backpropagation applied to handwritten zip code recognition. Neural Comput. **1**, 541–551 (1989). https://doi.org/10.1162/neco.1989.1.4.541
9. Berger, A., Kostak, M., Maly, F.: Mobile AR solution for deaf people. In: Awan, I., Younas, M., Ünal, P., Aleksy, M. (eds.) MobiWIS 2019. LNCS, vol. 11673, pp. 243–254. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-27192-3_19
10. Berger, A., Vokalova, A., Maly, F., Poulova, P.: Google glass used as assistive technology its utilization for blind and visually impaired people. In: Younas, M., Awan, I., Holubova, I. (eds.) MobiWIS 2017. LNCS, vol. 10486, pp. 70–82. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65515-4_6
11. El Bahi, H., Zatni, A.: Text recognition in document images obtained by a smartphone based on deep convolutional and recurrent neural network. Multimedia Tools Appl. **78**(18), 26453–26481 (2019). https://doi.org/10.1007/s11042-019-07855-z
12. Papageorgiou, C.P., Oren, M., Poggio, T.: A general framework for object detection. In: Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271), pp. 555–562 (1998). https://doi.org/10.1109/ICCV.1998.710772
13. Lowe, D.G.: Object recognition from local scale-invariant features. In: Proceedings of the Seventh IEEE International Conference on Computer Vision, vol. 2, pp. 1150–1157 (1999). https://doi.org/10.1109/ICCV.1999.790410
14. Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), vol. 1, pp. 886–893 (2005). https://doi.org/10.1109/CVPR.2005.177
15. Felzenszwalb, P.F., Girshick, R.B., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part-based models. IEEE Trans. Pattern Anal. Mach. Intell. **32**, 1627–1645 (2010). https://doi.org/10.1109/TPAMI.2009.167
16. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986). https://doi.org/10.1038/323533a0
17. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2014)
18. Girshick, R.: Fast R-CNN. In: Presented at the Proceedings of the IEEE International Conference on Computer Vision (2015)
19. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 91–99. Curran Associates, Inc. (2015)

20. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: unified, real-time object detection. arXiv:1506.02640 [cs] (2016)
21. Redmon, J., Farhadi, A.: YOLO9000: Better, Faster, Stronger. arXiv:1612.08242 [cs] (2016)
22. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement. arXiv:1804.02767 [cs] (2018)
23. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
24. Everingham, M., Eslami, S.M.A., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The Pascal visual object classes challenge: a retrospective. Int. J. Comput. Vis. **111**(1), 98–136 (2014). https://doi.org/10.1007/s11263-014-0733-5
25. He, K., Gkioxari, G., Dollar, P., Girshick, R.: Mask R-CNN. Presented at the proceedings of the IEEE international conference on computer vision (2017)
26. Cai, Z., Vasconcelos, N.: Cascade R-CNN: delving into high quality object detection. Presented at the proceedings of the IEEE conference on computer vision and pattern recognition (2018)
27. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. Presented at the proceedings of the IEEE conference on computer vision and pattern recognition (2016)
28. Chao, Y.-W., Vijayanarasimhan, S., Seybold, B., Ross, D.A., Deng, J., Sukthankar, R.: Rethinking the faster R-CNN architecture for temporal action localization. Presented at the proceedings of the IEEE conference on computer vision and pattern recognition (2018)
29. Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollar, P.: Focal loss for dense object detection. Presented at the proceedings of the IEEE international conference on computer vision (2017)
30. Bhandari, R., Nambi, A.U., Padmanabhan, V.N., Raman, B.: Driving lane detection on smartphones using deep neural networks (2020). https://doi.org/10.1145/3358797
31. Ignatov, A., et al.: AI benchmark: running deep neural networks on android smartphones. In: Leal-Taixé, L., Roth, S. (eds.) ECCV 2018. LNCS, vol. 11133, pp. 288–314. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-11021-5_19
32. Ignatov, A., et al.: AI benchmark: all about deep learning on smartphones in 2019. arXiv:1910.06663 [cs] (2019)
33. Niu, W., Ma, X., Wang, Y., Ren, B.: 26 ms inference time for ResNet-50: towards real-time execution of all DNNs on smartphone. arXiv:1905.00571 [cs, stat] (2019)
34. Begg, R., Hassan, R.: Artificial neural networks in smart homes. In: Augusto, J.C., Nugent, C.D. (eds.) Designing Smart Homes. LNCS (LNAI), vol. 4008, pp. 146–164. Springer, Heidelberg (2006). https://doi.org/10.1007/11788485_9
35. Chollet, F., et al.: Keras (2015)
36. Why use Keras - Keras Documentation. https://keras.io/why-use-keras/. Accessed 25 Mar 2020
37. Backend - Keras Documentation. https://keras.io/backend/. Accessed 25 Mar 2020
38. TensorFlow. https://www.tensorflow.org/. Accessed 25 Mar 2020
39. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. Presented at the 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16) (2016)
40. TensorFlow.js|Machine Learning for Javascript Developers. https://www.tensorflow.org/js. Accessed 25 Mar 2020
41. TensorFlow Lite|ML for Mobile and Edge Devices. https://www.tensorflow.org/lite. Accessed 25 Mar 2020
42. Case Studies and Mentions. https://www.tensorflow.org/about/case-studies. Accessed 25 Mar 2020
43. PyTorch, https://www.pytorch.org. Accessed 30 Mar 2020

44. PyTorch Mobile. https://pytorch.org/mobile/home/. Accessed 30 Mar 2020
45. Get started with TensorFlow Lite. https://www.tensorflow.org/lite/guide/get_started?hl=cs. Accessed 01 Apr 2020
46. ML Kit for Firebase. https://firebase.google.com/docs/ml-kit?hl=cs. Accessed 01 Apr 2020
47. Neural Networks API|Android NDK. https://developer.android.com/ndk/guides/neuralnetworks?hl=cs. Accessed 01 Apr 2020
48. Core ML|Apple Developer Documentation. https://developer.apple.com/documentation/coreml. Accessed 01 Apr 2020
49. apache/incubator-mxnet. https://github.com/apache/incubator-mxnet. Accessed 01 Apr 2020
50. tf-coreml/tf-coreml. https://github.com/tf-coreml/tf-coreml. Accessed 01 Apr 2020