Integration of Convolutional Neural Networks in Mobile Applications

Roger Creus Castanyer
Universitat Politècnica de Catalunya
Barcelona, Spain
roger.creus@estudiantat.upc.edu

Silverio Martínez-Fernández
Universitat Politècnica de Catalunya
Barcelona, Spain
silverio.martinez@upc.edu

Xavier Franch
Universitat Politècnica de Catalunya
Barcelona, Spain
franch@essi.upc.edu

***Abstract*—When building Deep Learning (DL) models, data scientists and software engineers manage the trade-off between their accuracy, or any other suitable success criteria, and their complexity. In an environment with high computational power, a common practice is making the models go deeper by designing more sophisticated architectures. However, in the context of mobile devices, which possess less computational power, keeping complexity under control is a must. In this document we study the performance of a system that integrates a DL model as a trade-off between the accuracy and the complexity. We present a practical study that aims to explore the challenges met when the performance efficiency of DL models becomes a requirement. Concretely, we aim to identify: *(i)* the most concerning challenges when deploying DL-based software in mobile applications; and *(ii)* the role of the performance efficiency and the path for its achievement. We obtain results that verify many of the identified challenges in the related work such as the availability of frameworks and the software-data dependency. We provide a documentation of our experience when facing the identified challenges together with the discussion of possible solutions to them. Additionally, we implement a solution to the sustainability of the DL models when deployed in order to reduce the severity of other identified challenges. Moreover, we relate the performance efficiency to a new defined challenge featuring the impact of the complexity in the obtained accuracy. Finally, we discuss and motivate future work that aims to provide solutions to the more open challenges found.**

***Index Terms*—DL-based software, Convolutional Neural Networks, Software Engineering, AI Model Development Process, DevOps**

I. Introduction

Convolutional Neural Networks (CNN) have become a widely used architecture since they have proven to be an accurate universal function estimator in many domains, such as Image Classification and Natural Language Processing [1]. As a particular type of Neural Network (NN), a CNN consists of a collection of parameters, which represent weights and biases, that are tuned depending on the problem that is faced. These parameters put together after long training sessions have proven to be able to solve many complex problems even outperforming humans [2]. The tuning is performed automatically and the values that the parameters take are outside of the human comprehension, turning the models into black boxes.

The use of CNNs in custom applications demands high computational power, since there is no physical limitation in the complexity of the designed models. Simple augmentation of the size of the collection of trainable parameters has proven to work as long as new data keeps being fed to the model. Obtaining huge collections of data is not a big problem nowadays due to the Big Data hype. Furthermore, following the exponential evolution of the availability of computational power, the tendency of solving complex tasks with complex models is increasing. Language models are good examples of this, e.g. Mann et al. with the description of a NN with 175 billion parameters [3]. According to this tendency, the most common deployment platforms used historically for DL-based software are servers that enable cloud computing. For the purpose of making use of a trained NN, a data pipeline is developed manually so that it links the pre-processing steps, computation of results, post-processing and eventually the display of the predictions. When all these steps are implemented in a server/cloud, the requirement of an internet connection for accessing it is a must. However, with the growth of the mobile applications offer and demand, approaches on the integration of NNs in lightweight devices are becoming popular for their ability to provide great services to the user. Within this settings, DL-based software consists of software-reliant systems that **include data and components** that implement algorithms mimicking learning and problem solving [4].

In this work we aim to perform an exploratory and descriptive analysis on: *(i)* what are the current challenges regarding the deployment of DL-based software as mobile applications; and *(ii)* how is accuracy achieved while keeping the complexity under control. The subject is analysed by means of a practical study in which we perform all the required cycles from data acquisition, DL modelling, classification and application, and operation in real context. In this way, we study the relation between

the DL-based system development and the accuracy when operating with it in the production settings [5]. Ensuring the trustworthiness of the DL-based system in the operation environment is key and can be specially concerning in safety-critical applications [6].

Our work is performed over the German Traffic Sign Recognition Benchmark dataset [7], created in order to standardize the traffic sign recognition literature. In the DL literature, this faced task is an Image Classification problem. Each image in the data is from one and only one class so the models will be trained using classification accuracy as the success criteria. CNNs are the standard approach to Image Classification problems for their ability to fit in image processing. The CNN architecture has already shown that can achieve very good results in the literature but also for this specific dataset.

This work has four main contributions:

- Show experiences of the end-to-end DL lifecycle from a practical study on traffic sign recognition.
- Discuss the challenges encountered on the deployment of CNNs.
- Discuss the trade-offs between accuracy and complexity in environments with limited computation power (e.g., mobile applications).
- An open science package of the developed DL-based software freely available on GitHub following the "Cookiecutter Data Science" project structure [1] to foster correctness and reproducibility.

The document is structured as follows. In Section 2 we describe Related Work on both the problematic of integrating complex models in mobile applications and the focuses on achieving performance efficiency in the deployment of DL-based software. In Section 3 we describe the Study Goal and Research Questions. In Section 4 we describe the study design and the end-to-end lifecycle, which goes from the description of the data, to the modelling part and to the system deployment. In Section 5 we show the results obtained in the practical study that allow to give an answer to the Research Questions. In Section 6 we discuss what is strictly analysed in this document and what specific parts of the whole topic are addressed. To end with, in Section 7 we draw conclusions from the research performed over the development of the project.

II. Related Work

In this section, we respectively describe the challenges of deploying CNNs in DL-based software, and related work on the achievement of performance efficiency in the performance of embedded systems or systems with limited computation power.

[1] https://drivendata.github.io/cookiecutter-data-science/

A. Challenges in deployment

Regarding the recent interests in the deployment of DL-based software, Chen et al. state that mobile applications come with the strongest popularity trend over other platforms like servers/clouds or browsers for integrating DL models [8]. They also show that this increasing trend in the deployment of DL models in mobile applications comes with an inverse proportional relation with the knowledge that the community has about the subject. This way, they demonstrate that the deployment of DL-based software becomes the most challenging part in the life cycle of DL, and that this effect is even more critical when the target are mobile devices.

When analysing the challenges of deployment of DL-based software, several studies focus on the selection of the necessary target quality attributes of this type of software. The importance of each quality attribute varies depending on the application but it is always clear that DL-based software implies taking care of measures that might not be considered when building traditional software systems. In this way, a common key point in these studies is that in DL-based software there is a major influence that data has over the system performance, as this has a strong dependency on the data structure.

Indeed, Pons and Ozkaya clearly state that compared to software systems that do not integrate DL-based components, the deployment of systems that do have a DL-based component increases the risk in many quality attributes [9]. They state that Data centricity is the cause that affects the robustness, security, privacy and sustainability of these systems, so they further analyse the methodology for architecting in Artificial Intelligence (AI) engineering. Following this principle, Ozkaya makes the difference between the development of software systems and DL-based software in its processes of building and sustaining [10]. Also, in [11] a list of challenges and solutions regarding the deployment of DL-based software within industry settings is provided.

The identified challenges in the above related work that are objects of study in this work are:

- (C1) **Frameworks in early stage** of its development to support DL-based software implementations.
- (C2) **Software–Data dependency**.
- (C3) **Explainability** of the models.
- (C4) **Sustainability** of the models when deployed.

B. In the quest for efficiency

In the pursuit of efficiency when deploying DL-based software, there exist several key points that affect the overall system implementation. Optimizations need to be present during the cycle from development to deployment. Important bottlenecks that might involve limitations to efficiency are found in both the modelling stage and in the usage of different frameworks to deploy the models. On the one hand, recent contributions [12] show an optimized CNN architecture for enabling its usage in devices with less energy capacity. This allows to reduce the model's complexity (e.g., storage weight, computing power, memory size) while obtaining the desired accuracy. On the other hand, the availability and capabilities of different frameworks that support the implementation of DL-based software in all its stages has a strong influence on the performance of the systems, as shown in [13]. Several researchers have investigated the performance of the whole DL development versus other techniques [14].

Compared to the aforementioned studies, in our work we focus on analysing the challenges when deploying a specific type of NN, namely CNN, into mobile devices. Furthermore, we do so by means of a practical study. We relate the challenges found with the analysis of the performance, a quality attribute that is studied as the trade-off between two other quality attributes: accuracy and complexity. We provide a comparative analysis between different configurations for each of the models in order to gather evidence of the implications of the accuracy and complexity. Moreover, the sustainability of the models once deployed is also studied to enhance the capabilities of the applications during the DL-based software lifecycle. Our work is developed under the PyTorch framework and Android operating system.

III. Study Goal and Research Questions

In this Section, we define the Goal and Research Questions (RQs), following the GQM guidelines [15].

- RQ1 - What are the challenges of the creation and integration of complex CNNs in DL-based mobile applications?
- RQ2 - How can we optimize the trade-off between accuracy and complexity along the DL-based software lifecycle?

The motivation of RQ1 within this project is to study the **challenges of the creation, training and integration of a CNN in a mobile application**, while identifying, exploring, documenting and facing as many challenges as possible. The obtained conclusions aim to be generic for anyone whose desire is deploying complex models in quotidian software applications.

Furthermore, RQ2 motivates a **comparative analysis between different solutions to the problem of achieving efficiency in the performance**, since the trade-off between the accuracy of the model and its complexity can be studied from different viewpoints (e.g. design of optimized operations or limitation of architecture's complexity).

TABLE I: Summary of the application functionalities

ID	Description
1	Use of the device integrated camera
2	Real-time response for a call to the model
3	High accuracy in traffic sign recognition
4	Data augmentation by the community
5	Integration of updated models

IV. Study design

In this section we describe the proposed pipeline for achieving a functional implementation of the traffic sign recognition mobile app.

There are four main cycles in the DL-based software lifecycle [5]. First, the *Data Management* which consists of the collection and processing of raw data. Second, the *DL Modelling* phase which consists of adjusting different model fits to obtain the best-performing possible solution. Third, the *Development* of the environment (mobile application in our work) and the integration of the DL model in it. Fourth, the *DL-based System Operation* with the application that integrates the DL model. This last phase enables sustainability of the model and ensures proper performance within the production settings.

A global overview of the study design and DL-based software lifecycle is shown in Figure 1. The functionalities of the system are shown in Table I. We implement these through iterations on the aforementioned DL-based software lifecycle phases. Furthermore, we provide a solution for the *Send Data* operation in Figure 1. This allows the possibility that the user collects the generated data in run-time during the DL-based system operation, which is then manually annotated and used for re-training the models on the performant computer. Then, this updated models can be integrated again for inference in the mobile application.

A. Platforms and technologies used

For joining the DL modelling and the development of DL-based software, there exist recent DL frameworks,

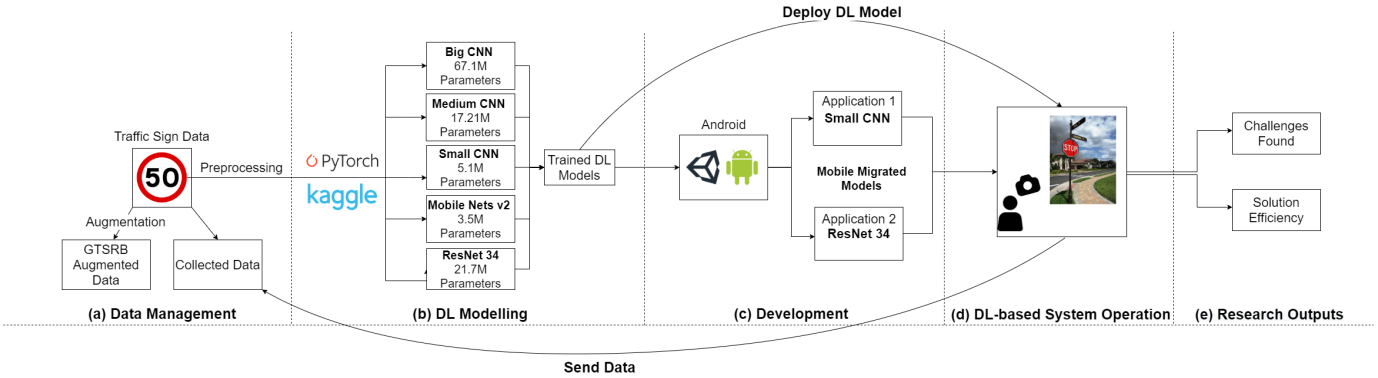

Fig. 1: Software development lifecycle in our DL-based software

all of them under continuous development and evolution. These frameworks offer up to two different approaches to model transferring operations. On the one hand, *model conversion*. CoreML for iOS software and TensorFlow-Light for Android arguably are the most famous examples that offer this capability. On the other hand, *model export* using the recent Open Neural Network Exchange (.ONNX) file format, which was presented in 2017 by Microsoft and Facebook. This file format allows to store serialized versions of trained networks in compressed files.

Since the .ONNX file format is widely used in the current state of the art, it is defined as a key part part of the proposed project pipeline for the practical study.

With this taken into account, the focus is put on finding two different platforms that allow development-sided and operation-sided development. The first one has to allow training and writing of a CNN into a .ONNX file. The latter has to be capable of reading and using the model and also of supporting the use of the camera in a mobile device.

The development-sided framework used in this project is set to be PyTorch, which is based on Python and primarily developed by Facebook and key for global DL development around the world. The operation-sided platform is set to be Unity3D, which supports development with C#, is compatible with Android and has powerful standard libraries that come with it. A useful package in this platform for this purpose is Barracuda, which is now in a very first stage and still being developed by a team in Unity.

Also, as observed in Figure 1 the models built on PyTorch are trained on the Kaggle GPU, which is available freely for 40 hours a week.

Moreover, it is also seen in Figure 1 that only two of the five models that are trained are later deployed to the application. For an appropriate analysis of the performance of these two models when deployed, two applications that only vary in the integrated models are built.

B. The Dataset

The GTSRB traffic sign data consists of 144,769 labelled images of 2,416 traffic sign instances of 70 classes belonging to a portion of the real traffic signs that can be found in the roads of Germany. However, after following several designed criteria for guaranteeing the quality of data, explained in [7], the final dataset consists of a collection of 51,840 images of 43 classes fulfilling: *(i)* images are of sizes from 15×15 to 222×193 pixels, stored in PPM format; *(ii)* it contains the definitions of the region of interest with a separating margin of the size equal to the 10% of the pixels; *(iii)* it records the temporal order of the creation of the images; *(iv)* it provides post-processed features like HOG descriptors, Haar-like features and Color Histograms. No usage of the post-processed features is made for the purpose of the deployment of a full-stack convolutional-based architecture for solving the image classification problem.

1) The Pre-processing: When processing images for DL purposes, in this case for model training, there are critical steps that are key for successful learning. For this purpose, several pre-processing steps are listed as follows.

The images are resized to a fixed and squared size, these vary in different experiments from 256x256 to 512x512. Then, a center crop is applied to the image. Next, images are encoded as tensors of the form *[B, C, H, W]* where B is the batch size, C is the number of channels, H is height and W is width. Then, the values of these tensors are normalized into the 0-1 range. Finally, the tensors are standardized in the HxW channels with the pre-computed means and standard deviations of the 3 RGB channels of the images in the dataset. The values are shown in Table II:

TABLE II: Mean and standard deviation of the RGB channels in the images from the dataset.

Stat	R	G	B
Mean	0.3418	0.3126	0.3224
Std. dev.	0.1627	0.1632	0.1731

2) The Augmentation: For the purpose of supporting a comparative analysis of the performance of different models, we implemented two variants for applying the technique of data augmentation to the pre-processed dataset: *(i)* rotation in 90º degrees plus the vertical and horizontal flips to the images; *(ii)* addition of manually tagged images by the users of the application during the system operation phase.

A different approach on data augmentation for the GTSRB dataset is presented in [16]. They augment the original dataset with the creation of new images with quality deficits.

C. The DL/CNN models

The models that are applied for solving the task are the standard CNN, the MobileNet v2 (MBv2) [12] and the ResNet34 (RN34) [17]. For the first one, we test and evaluate different configurations in order to choose an optimal one that balances accuracy and complexity. For the last two, we test and compare two different ways of training the model: pre-training with feature extraction and fine-tuning, and full-training from scratch. The CNN works with standard convolutional blocks that consist of convolutions followed by batch normalization, an activation function, a max-pooling layer and a dropout layer. It can learn representations of images from scratch which are then passed to a classifier that is built on top of the convolutional blocks. The MBv2 is similar but works with separable depthwise convolutions. It is a form of factorized convolutions which factorize a standard convolution into a depthwise one and a 1x1 one called a pointwise convolution. The depthwise convolution applies a single filter to each input channel and the pointwise convolution then applies a 1x1 convolution to combine the outputs of the depthwise convolution [12]. Finally, the RN34 is a more complex CNN that has been successfully applied in the image classification domain and has become a benchmark model for related tasks in the DL literature [17]. Both the MBv2 and the RN34 used in the practical study are downloaded from PyTorch and are pre-trained on ImageNet data [18]. See the comparative analysis of the developed models at https://github.com/yuyecreus/CNN-in-mobile-device/blob/master/reports/Appendix.pdf.

D. The DL-based software system

The architecture of the DL-based software system operation is shown in Figure 2, following design patterns from [19]. The application that loads the SmallCNN has a total weight of 68.84Mb and the one that loads the RN34 of 131Mb. The baseline weight of the application bundle generated by Unity3D is of 46.73Mb without the models.

The two built applications provide the functionalities 1,2 and 3 in Table I. However, for ensuring the sustainability of the models when deployed, and for including the functionalities 4 and 5, an external server/computer is needed for performing the re-training of the model with the new tagged data added by the community. This implementation still lets the user obtain recognition predictions without external calls. The user has this option because the application client will only make use of the server/computer for downloading new updates on the model and for manually submit new tagged data in a personally controlled way, whenever it is desired. We consider allowing the user to annotate the data that it generates on run-time an efficient solution to the sustainability of the model when deployed. This could be done either by verifying correct model recognition predictions or by correcting wrong ones. However, the architecture that we implement in this practical study makes use of the mobile device storage to save the taken pictures, which are then manually annotated and sent to the local machine, where they are fed to the model again in the Kaggle GPU. This is not a scalable implementation but releasable for research purposes.

The implementation of the two DL-based systems and of the data modelling of the problem is available on our Open Science package, freely available on https://github.com/yuyecreus/CNN-in-mobile-device.

V. Results

In this section we present and discuss the results of our practical study for each RQ. A demo video of the developed and evolved DL-based software is available at https://www.youtube.com/watch?v=yFsp6kxO5kI.

A. RQ1

For answering the RQ1, we analyze both the identified challenges in the state-of-the-art (see Section II-A) and our results. This analysis can be seen in Table III, where: **v** indicates that the challenge has been verified to be real and concerning; ± indicates that the challenge has been identified although it has not been faced in the practical study; **new** defines newly spotted challenges up to our knowledge.

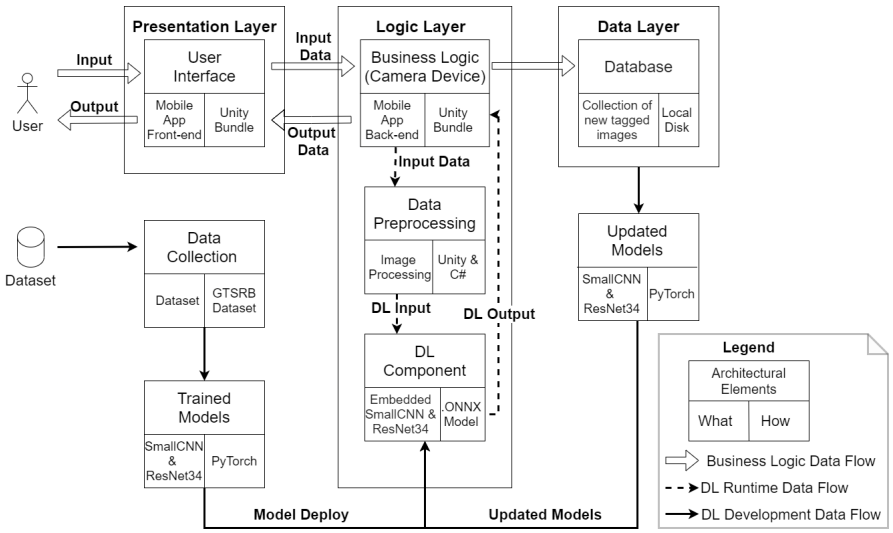

Fig. 2: DL-based System Operation Architecture

TABLE III: Diagnoses of identified challenges

Challenges	Diagnostic
Frameworks in early stage (C1)	v
Software-Data dependency (C2)	v
Explainability (C3)	±
Sustainability (C4)	v
Software dependencies (C5)	new
Model efficiency (C6)	new

Regarding C1 and C5, a constraint we found in DL-based software is that **no matter what framework one wishes to use, it will always need to wrap at least two different technologies**: one for the data management and model training outside the device, and the other for the model operation inside the user's device. This implies challenges regarding the mutual support between the two technologies. In the following, the faced challenges regarding these aspects are listed.

First, **the pre-processing steps that are applied to both the initial dataset in the development phase and the ones in the operation phase (which are applied in real time to the images taken by the user) have to be the same despite being implemented in different frameworks**. DL-based software developers shall ensure that the inputs on both platforms are the same" to the network, so it gives the same predictions for the same images. Secondly, **the operation-sided framework must support model importing and must provide functional libraries that enable the appropriate usage of the model in its exported format**. In Unity Barracuda, a challenging limitation found is the inability to support a lot of the basic operations in the treatment of DL models. For example, let's consider the *view()* function from PyTorch. For the hand-made CNNs, this operation has been replaced by *Flatten()* which does exactly the same, but for the pre-trained MBv2 this method is used and hence the architecture cannot be deployed directly to Unity. In conclusion, due to the limitations in the used operation-sided framework, a model architecture that is built for its optimized performance on mobile devices cannot be deployed to the mobile application.

Regarding C2, **we have experienced high volatility in the models' performance when testing them with real-world data outside of the training set**. This fact makes sustainability (C4) **become a critical quality attribute for reducing data dependency and ensuring the correct model performance when deployed**. In the practical study, this has been achieved by updating the models with the user's taken images. By doing this, we smooth the differences between the development and operation phases of the DL-based software lifecycle by adapting the models to the real world environment. In general, our solution to the sustainability of the models over time consists of augmenting the training dataset with data of the operational environment.

Regarding C3, in the built applications we provide standard confidence indicators of the models' predictions. These consist of the output probabilities of class membership which have been scaled to become percentages of confidence for a given input. **These simple indicators provide a level of explainability of the models** that is acceptable for the applications in the practical study, meaning that **they allow some level of interpretation to the user**. We do not consider ensuring this quality attribute a critical challenge because it might not generalize to lots of applications that integrate a DL-based component. However, **we recognize the importance of the presence of any**

measure that provides the user with information about why the model has given a result.

Our practical study revealed C6. Compared to the usage of a server, the model exporting operation and its integration into the mobile application bundle **has the drawback of the inclusion of the model's storage weight in the application bundle, hence, in the mobile storage**. For this reason the **optimization of efficiency is a must when deploying DL-based software to mobile devices**. We have shown in the practical study that not limiting the complexity of the model's architecture, as in the case of the RN34, yields a cost in performance in the mobile application in terms of real time response, memory cost, and storage weight. For the deployment of DL-based software in mobile devices, efficiency is not an option, it is a must, and facing it becomes a critical challenge.

B. RQ2

Most of the optimizations applied to achieve efficiency are approaches based on the reduction of the models complexity. In an environment where the developer has few alternatives in the choice of both development-sided and operation-sided frameworks that allow the deployment of DL-based software, the deployment of these systems becomes a bottleneck of the DL pipeline and the focus is only put in the modelling stages for achieving efficiency. This way, **the evolution of the performance of DL-based software is strictly linked to the evolution of the current model conversion and exporting technologies**, a field that is in its very early stages, where few file formats can be used and with several limitations. For this reason, the deployment of DL-based software to the server cloud and the remote usage through internet connections is still a more flexible choice in terms of DL-based services that can be provided to a mobile user. However, this is not fault-tolerant, due to the dependency on hosting and internet connections.

A fact that makes the deployment of DL-based software in mobile devices not always suitable is that it collides with the benefits of transfer learning. The major capabilities of transfer learning include great results with little task-specific data, the ease in fine-tuning pre-built architectures and omitting the model design stage. The common available models for transferring learning are of massive weight, and the revolution of transfer learning has created a motivation to use these largest-scaled models for solving any task before designing task-specific architectures [20]. This has never been a problem until now, when one wants the massive model to be deployed in a low-performance environment.

We have shown in the study that the most famous architecture amongst all the experimented ones, which is the RN34 [17], is the one carrying the biggest drawbacks in terms of real-time response and storage weight. In this way, the simple CNN architecture with the appropriate configuration outperforms the massive model in terms of efficiency because of the ability to provide almost the same results regarding the accuracy when solving the classification problem and because of its reduction in the complexity of the architecture. Of course, this might not happen for all the tasks that can be solved with AI, where the RN34 might provide better accuracy due to the wider generalization in its extracted features.

Finally, the MBv2 looks very promising in this field for its low weight and good performance in terms of accuracy. Although it has not been deployed for the proposed practical study, it is concluded that the modifications in its architecture provide the best capabilities for the purpose of deploying DL-based software into low-performance environments, in this case the mobile devices.

VI. Discussion

Although the focus is put on the deployment of the DL-based software in a mobile application our study goes through the entire DL-based software lifecycle. For this reason, we now discuss other stages of the DL-based software lifecycle that also influence the overall performance of this type of software when deployed.

To begin with, we have verified that the identified challenges that regard the software-data dependency and sustainability of the models deployed in the software are of major concern. In addition to this, we have implemented a data augmentation technique that ensures the adaption of the models to the real world environment, hence providing sustainability. We have experienced very significant increases in accuracy (without loss of efficiency) when sustainability is ensured in the two systems. This step is represented by the "Send Data" operation in Figure 1. The data that is "sent" in this case consists of the images taken by the users during the DL-system operation phase, which is manually tagged and used for re-training the models.

Regarding the differences between the efficiency of the two DL-based systems, an option for obtaining a functional and efficient implementation could consist of the integration of the two deployed models in the same DL-based mobile application. The default model could be set to the most efficient one, and in case the results given by this were not satisfactory the user would have the option to switch to the more robust and less efficient model.

Finally, the systems built in the practical study solve the task of traffic sign recognition over the assumption that all the input images given to the model consist of pictures of

traffic sign instances. Hence, the model is not capable of learning what is a traffic sign, it only learns to distinguish between them.

VII. Conclusions

In this work we have studied the DL-based software lifecycle in a practical manner. We have provided an analysis of the challenges found when developing this type of software. Specifically, we have put the focus on the deployment of DL-based software in mobile applications. Concretely, we have tested the performance of different CNN architectures under the PyTorch and Android frameworks. With all this, we have experienced, solved and documented many of the previously identified challenges in the related work and also have highlighted new ones. Furthermore, we have analysed the impact of efficiency in the overall performance, and we have related this quality attribute to a newly identified critical challenge of the deployment of DL-based software.

This study motivates several key points of future work. First, the search for optimized model architectures that allow more efficient solutions to the deployment in mobile applications scenario. These can both implement more efficient operations in its architecture or have limitations in its complexity when designed. The study also encourages the community to develop alternative solutions to the conversion and export of models together with software technologies that support mobile application development with DL-based components. The latter would reduce the severity of the challenges that are met while implementing this type of software. Moreover, the study motivates the implementation of more sophisticated confidence indicators that describe in a more precise way the uncertainty in the models predictions. The uncertainty of a model prediction given the input conditions is a measure that is hard to determine but useful to influence the decisions taken automatically by autonomous DL-based systems. These can be key in many applications related to safety-critical environments. A part from this, sophisticated confidence indicators increase the quality of the system by enhancing the explainability to the user.

Acknowledgment

We thank Lisa Jöckel for having reviewed our work and for being supportive and useful to enhance it.

References

[1] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[4] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 50–59.

[5] L. E. Lwakatare, I. Crnkovic, and J. Bosch, "DevOps for AI – Challenges in Development of AI-enabled Applications," in *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020, pp. 1–6.

[6] S. Martínez-Fernández, X. Franch, A. Jedlitschka, M. Oriol, and A. Trendowicz, "Research Directions for Developing and Operating Artificial Intelligence Models in Trustworthy Autonomous Systems," 2020.

[7] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[8] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750–762.

[9] L. Pons and I. Ozkaya, "Priority Quality Attributes for Engineering AI-enabled Systems," *arXiv preprint arXiv:1911.02912*, 2019.

[10] I. Ozkaya, "What Is Really Different in Engineering AI-Enabled Systems?" *IEEE Software*, vol. 37, no. 4, pp. 3–6, 2020.

[11] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," *Information and Software Technology*, vol. 127, p. 106368, 2020.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. A. Mobilenets, "Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[13] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 810–822.

[14] W. Fu and T. Menzies, "Easy over Hard: A Case Study on Deep Learning," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 49–60. [Online]. Available: https://doi.org/10.1145/3106237.3106256

[15] V. R. Basili, G. Caldiera, and D. H. Rombach, "The Goal Question Metric Approach. Encyclopedia of Software Engineering 1 (1994)," 1994.

[16] L. Jöckel, M. Kläs, and S. Martínez-Fernández, "Safe Traffic Sign Recognition through Data Augmentation for Autonomous Vehicles Software," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2019, pp. 540–541.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] J. Deng, W. Dong, R. Socher *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[19] H. Washizaki, H. Uchida, F. Khomh, and Y.-G. Guéhéneuc, "Machine learning architecture and design patterns," 2020.

[20] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.