A Framework for Developing Neural Networks Based Mobile Appliances

Paolo Motto Ros and Eros Pasero

Abstract— **The aim of our project is to develop a mobile real-time reader device for blind people. It uses Artificial Neural Networks (ANNs) for the core character recognition engine. The hardware constraints led us to develop a cross-platform framework to design and evaluate such subsystem in order to find a good trade off between run-time performances and accuracy of results.**

The goal of this work is to provide a tool for prototyping and evaluating ANNs without requiring any insight knowledge of the top-level design tool. At the same time this relies on the same code base that will be part of the target application, thus shortening and making easier the overall development process.

In this paper we will present both the proposed software solution and its usefulness in our research project; it enabled us to reduce the needed run-time resources while improving the performance results of the ANN subsystem.

I. Introduction

The current trend in technology evolution is to design and develop mobile aids, able to assist the user in everyday life in a smart way. This may mean to prevent his needs or to suggest actions to take, rather than to simply expect commands to be executed. They are progressively shifting from a passive role to an active one in interacting with the user. Think about the increasing popularity gained in last years by devices like smartphones and GPS navigation systems. As the offered services grow, the system complexity increases, mainly due to the additional amount of information to be processed. This in turn can be managed either locally by the same device interacting with the user, or by a remote infrastructure, e.g., using the well established client-server architecture or perhaps the emerging cloud computing model. Both solutions have their pros and cons, mainly related to connectivity and/or computational power issues. The goal is to implement applications with a human-like behavior without compromising run-time performances, e.g., the device responsiveness or the capability of delivering information in a useful time. To achieve such results we can still rely on well-known approaches to information processing, but it is clear that they need to be modified to meet the new resource constraints.

The outlined scenario is exactly the background of our research project, whose aim is to develop a mobile reader device for blind people. It is part of the STIPER2 project, funded by INFN (Istituto Nazionale Fisica Nucleare) and developed in Italy by Dipartimento di Elettronica of Politecnico di Torino and by Istituto Superiore di Sanità of Roma. The idea is to design and develop a portable and autonomous device able to guide the user to read printed text in a interactive way, which means that the acquired images are processed locally in real-time. This follows the same *modus operandi* of the Optacon device developed at Stanford University by Linvill and Bliss about four decades ago [1], [2]. At the time the main limitation was the lack of an embedded platform with enough computational power to develop a full-fledged reading device.

As briefly pointed out before, the technology has been evolving since then, and nowadays there are commonly available software and hardware platforms that should guarantee the requirements for achieving such ambitious goal. This motivated us to carry on the same proposal of Linvill and Bliss and to switch from a direct graphical translation to a recognition approach. Moreover we are now able to integrate a speech synthesis engine, practically replacing the Braille transducer (thus broadening the expected user base). A similar solution, even if with a different approach, has been recently announced and marketed by Kurzweil with its kReader Mobile (see [3]): with such aid the user has first to take a snapshot of the desired information to be read, then the software processes the image and communicate the results to the user (by means of speech synthesis), eventually requiring a new snapshot. The system has not a fully human-friendly interface, since it is based on a trial-and-error interaction.

Our approach to the issue of reading printed text allows the user to actively explore the underlying text by means of a custom camera which acquires a continuous stream of images. These are processed by the CPU, which also extract textual information, both the characters and their arrangement inside the text flow. This task is carried out by an Artificial Neural Network (ANN) subsystem, with the same approach that has already been successfully used in a previous feasibility study [4]. Anyway, a key difference is that now we face with the practical constraints of the run-time platform, different from the development one, while still trying to maintain (and eventually to improve) the performances. It is clear that the character recognition engine is the most time-consuming part of the whole system, and so we will focus our attention on it. This implies to study the new hardware architecture, to setup a cross-development environment and to adopt a set of development policies in order to guarantee the same results on the involved platforms. All these issues are addressed in Sec. II, while in Sec. III we will describe the proposed solution, and in Sec. IV we will use such software to develop a new set of ANNs with improved performances, thus showing the usefulness of the developed software. In the end (Sec. V) we will talk about the future perspectives, since this work is meant to be only

Paolo Motto Ros is research associate at Politecnico di Torino, INFN associate (Gruppo V, sezione di Torino). Email address: paolo.mottoros@polito.it

Eros Pasero is professor at Politecnico di Torino, INFN associate (Gruppo V, sezione di Torino). Email address: eros.pasero@polito.it

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

the first step toward the design of a full-fledged artificial neural network tool.

II. Method and development setup

A. Target platform discussion

The target platform for the reader device is based on the Marvell PXA270 SoC (System-on-a-Chip), equipped with the MS Windows CE OS (Operating System). This hardware solution has been chosen because it is based on ARM technology, the most used in mobile devices. Such CPU has been widely adopted in PDAs (Personal Device Assistants) and smartphones, but also in SBC (Single Board Computer) to be integrated in embedded systems. As a reference we used the Fujitsu-Siemens PocketLook N560 (PDA) and the Toradex Colibri PXA270 (SBC). All the run-time performances reported here have been measured on the Colibri module, whose CPU clock frequency is 520 MHz. The PXA270 is based on the 32 bit XScale architecture (derived from the ARM one, see [5], [6] for further details), with an in-order superscalar pipeline able to complete up to four instructions per cycle. This, together with the Intel Wireless MMX coprocessor [7], leads to obtain performances of about 800 MIPS (according to Intel[1]).

Even if the claimed performances are a considerable amount of computational power for a mobile device, it could be required to optimize anyway the application. It is well known that the first optimizations to look for reside in the algorithm design process rather than in the low-level details. It is also clear that ANNs are not the most resource-friendly solution, nevertheless their effectiveness justifies their use. A possible technique to mitigate their weight in the overall data-processing time is not to use a fully-connected layout. We can roughly estimate their cost as the amount of connections involved, since the multiplication is usually the most time-consuming operation (the activation function can be approximated by a piecewise linear one). So, reducing the connections has a twofold purpose: first, to lower the resources needed and, second, to counteract the overfitting issue (as outlined in [4], where partial connected layers have been already used).

The second useful optimization is the switch from the floating-point arithmetic to the fixed-point one. It is direct consequence of the lack of an FPU inside the PXA270, thus requiring to emulate floating-point operations in software. Although it seems a low-level issue, it has a strong relationship with the model chosen to develop the character recognition engine. Changing the operating mode of such core part means also to ensure the same performances it was originally designed for. In this case, such internal detail could have a performance degradation as a consequence, making necessary to modify the model to overcome it. The relationship between ANNs and fixed-point arithmetic has been widely investigated, both in empirical studies (see for example [8], where the minimum required precision is determined for different problems) and in theoretical work (e.g. see [9], where an analytical solution for the minimal weight representation is formulated given the problem domain). The fixed point issue is not only a matter of defining the requirements for a proper execution of ANNs, but also the more critical issue of how to train the networks. Although there has been a lot of research in this field, many proposed solution are either modifications of already well-known algorithms or novel techniques specifically designed to target low resolution system.

Considering the constraints of our hardware platform, equipped with a 32 bit CPU, from the run-time standpoint, it suffices to adapt all the numeric algorithms (including the ANNs) to a fixed-point arithmetic whose operands are equal to the fundamental CPU data type. Further squeezing the numerical data could be even counter-effective, since it is well known that this kind of CPUs show their best performances when managing data that exactly fits the internal registers length, think about the implemented memory access techniques when fetching/storing data. In this regard, 32 bit are well above the limits found, for example, in [8], but we will still check the behavior to be the same for both the floating-point arithmetic and the fixed-point one.

Given the above discussion, we preferred to use common training algorithms (such as the one proposed in [10]), relying on the floating-point arithmetic, and then to convert the resulting networks to the fixed point one, thanks to an algorithm that inspects the trained weights. We will show that this straightforward solution is sufficient to achieve our goal. Moreover, this choice still leaves the developer free to reconsider the decision about the arithmetic. It is true that, actually, in the mobile word, there is almost no support in hardware for floating-point numbers, but their desktop counterpart, and even emerging technologies such as general purpose GPU computing, are better optimized to execute such kind of computations. In this way we are able to maintain at the same time a double solution, able to take advantage of peculiarities of both type of platforms: the mobile one when executing the system, and the "standard" one for the training (which often takes a long time and should not be underevaluated).

B. Development environment discussion

Beside the study of the underlying hardware, we also need to establish a common software ground upon which to build our framework, in order to end with the double solution pointed out just above. The choice of MS Windows CE (WinCE) for the mobile part and standard MS Windows (Win32) for the desktop one is just a first step, since from the developer standpoint they are similar in the API (Application Program Interface) philosophy, even if with their own peculiarities (see [11] for further details). For this reason we need to design something like a software abstraction layer, which provides advanced functionalities in a independent way from the underlying platform. This includes of course the integration, (or porting whenever the original implementation does not support WinCE OS, of

[1]Intel sold almost all the XScale division to Marvell in 2005, but the PXA270 was originally developed by Intel.

third-party libraries as well as the development of custom specific objects, e.g. those representing neural networks. In both cases we have to ensure the same behavior on all the involved platforms, especially when the code encapsulates low-level details, such as a new fixed-point numeric data type.

In order to achieve the cross-platform feature of the proposed framework is thus needed to adopt the proper development practices: in this regard the Test Driven Development (TDD, see [12], [13]) has been useful. The main idea behind this model is first to write tests and then the code that passes them. First we define the behavior, then we implement such components. This is particularly useful whenever we build a complex system (such as a character recognition engine) using a lot of small components. If we are sure that those basic building blocks work in the right way, then we can concentrate our efforts on designing the higher-level system. We would like to use the framework with enough confidence without bothering with ill-defined behaviors.

The TDD development model also enforces some key concepts as *modularity*, *flexibility*, *extensibility* and *simplicity*. It is important to point out that the proposed framework is by no means intended to be an "all inclusive" solution. It is only a starting point, where the ANN designer can simply use such framework by describing (as we will see later) the desired solution or it can extend its usefulness by integrating new functionalities, e.g., new routines to manipulate the training set or a new training algorithm, in a easy way.

III. Proposed solution

Following the above discussions, we first investigated the solutions to adopt in order to properly implement the framework for developing ANNs based mobile applications. First of all we had to look for a library able to manage the neural networks in our context. We identified in FANN (Fast Artificial Neural Network [14]) the right product, provided with a open-source license and hence with the opportunity to modify it both to run on the mobile platform and to add some facilities. It is entirely written in C, so we developed a set of C++ classes to encapsulate both the involved data structures and the routines, in order to have an effective object-oriented model serving as a collection of basic building blocks for our classifier. The choice of such programming language is also due to the high run-time performances of the generated code; the same code will be used in the target application of the reader device. The identical choice has been made about the image processing functions, which extract the feature vector from the character images (see [4]): all the performance-related routines have been carefully designed and tested using such programming model.

One interesting feature of the FANN library is that it has an algorithm that, given a network, chooses automatically the best format to represent the weights. It considers the greatest number that could be computed, deduced from the weight values, avoiding also to incur into an overflow. Therefore the place of the decimal point depends upon each ANN, and it is (in general) not the same for all the networks belonging to the same configuration. A higher value for the decimal point of course means a greater precision, since it represents the numbers of bits used for the fractional part. In practice, a value greater than 6 should be sufficient to achieve good results (as reported in the documentation [14] provided with the ANN library), but we have to check it anyway. In order to better understand the differences about the execution of the ANNs with the two approaches, it is useful to highlight the average and minimal decimal point used for each configuration, along with the MSE and the BitFail (defined later) obtained from the comparison of the outputs in the two ways. To further exploit whether switching away from the floating-point arithmetic could lead to worse or better results, the ratio between the global MSE in fixed-point mode and the one in floating-point will be reported, and the same for the BitFail performance value.

The C/C++ environment is perfectly suited for self-contained applications, but if we need to modify or improve a part, we need to re-build the overall system from the scratch, even if only one parameter is changed. This could lead to a longer development cycle; it would be better to use something like a scripting language as a glue between the components. This should be an interpreted dynamic language, with good data description facilities, minimal run-time and easy to embed/interface with C++. There are plenty of alternatives, but in the end we preferred the LUA language [15], which is entirely written in C (the run-time code fits in about 150 kB), with a Pascal-like syntax and a lean and mean stack-based API. All the C++ objects have been exported to this environment, thus completing the foundation upon which to build the proposed framework.

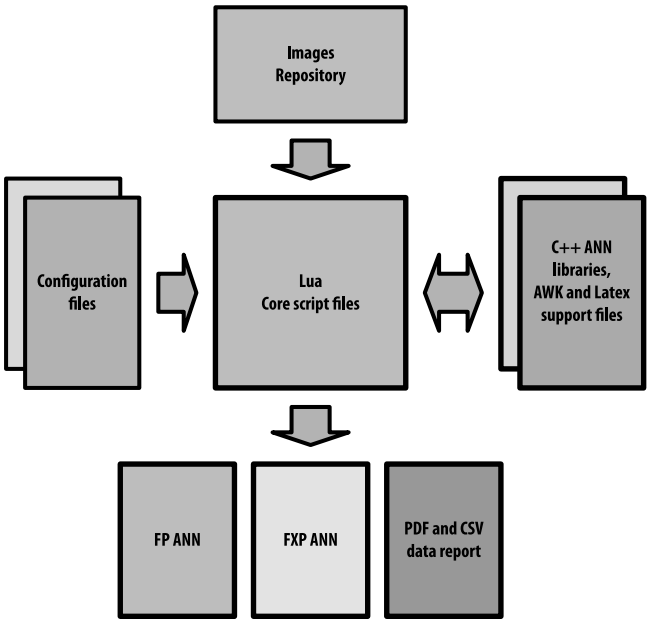

Fig. 1. Overall architecture of the proposed framework for training/evaluating the ANNs.

The architecture of the overall system is depicted in Fig. 1: the core part is a set of LUA scripts that takes

as input the image repository, subdivides in subdirectories (according to the classes of objects to be recognized), reads the configuration files and then processes the data generating the desired output. This is the set of networks both in fixed-point and floating-point format, plus a PDF report and the numerical results arranged in CSV (Comma Separated Values) files. This last feature is accomplished by a set of AWK (a powerful text processing tool [16]) scripts and LaTeX (the well-known text composition system) template files, and its purpose is to enable the ANN designer to compare the performances of different solutions as well as to make available such set of data to further elaborate with external tools, e.g., spreadsheet or statistical applications.

The configuration files are themselves LUA scripts, simple enough to be edited even without knowing the full language: they just contain a list, each item describing either a configuration of ANNs or the feature extraction vector to be used by networks (for practical reasons we have made distinction between these two parts, for further details on the feature vector, see [4]). We talk about "configurations" of ANNs rather than layouts of networks, since a configuration is composed by a network layout, the training parameters, a specification of how the training set is handled, and in the end a set of parameters stating how the single networks are combined together to form the classifier (according to the *winner-takes-all* model described in our previous work).

It is very difficult to predict *a priori* which parameter could lead, taken alone, to better performances without any doubt, because it interacts with all the others. The idea is to further analyze the results and to find some hints on how to train different ANNs, in order to have a set of configurations from which to choose the best one. As already pointed out in [17], there is no scientific algorithm that gives as result the optimal ANN. It is more an iterative process based on a trial and error approach, but we can identify some guidelines on how to design new configurations. All these opportunities are then exposed to the framework user, which is thus able to exploit their pros and cons. The parameters can be divided into three categories:

- Network layout
- Evaluation parameters
- Training parameters

The network layout can be either simply expressed as the number of desired layers and the amount of neurons for each of them, resulting in fully-connected networks, or defined through custom parametrizable functions. These, actually, generate both the not-fully-connected topology presented in [4] or an analogous bidimensional three-layers one (always partially connected). This last opportunity has been designed since in the end we are trying to recognize images, which have an intrinsic 2D structure.

Regarding the evaluation parameters, the MSE (Mean Squared Error) is the most used one, even if it is more suited to regression problems rather than classification ones. MSE can be better interpreted as an estimate of the distance of the proposed model to the desired target; when dealing with pattern recognition systems, it would be better to know how many samples in the validation set make the model fail, i.e., to have an answer to how many errors we had. The *BitFail* parameter serves exactly at this purpose: it is a simple true/false property of the ANN that, given an input, states whether the error of the output is above a certain threshold, called *BitFail limit* or $BitFail_{lim}$; it corresponds to the 0/1 loss function. Hence it is possible to obtain a number representing the *percentage of wrong outputs*, by running the network over a set of samples and counting when such condition is verified. This should give a clearer idea about the classification performances of the tested configuration, although it does not take into consideration how the ANN results are combined together in the final decision algorithm (this is true also for the MSE). Anyway, it can be used as the training goal, in order to create ANNs good enough to classify the desired data, rather than to output accurate values.

Always talking about evaluation parameters, it was observed, using the main application in preliminary session tests, that almost all the fails of the character recognition subsystem were due to the low output of the corresponding network. Considering only the results about the MSE and BitFail, it could be quite surprising. Trying to exploit this phenomenon, two types of measure have been defined: the *intra-class* MSE/BitFail (MSE_{IC}/$BitFail_{IC}$) and the *extra-class* MSE/BitFail (MSE_{EC}/$BitFail_{EC}$)[2]. The former is the MSE/BitFail reported by a network over the validation samples that it should recognize, i.e., the positive set; the latter is the error in all the other cases (the ones that it should reject), i.e., the negative set. As we will see in the next section, as the number of classes grows, the ratio between positive samples and the negatives diminishes (since one class is trained against all the others), and so, being the error measure averaged on the whole training set, the resulting networks will be more biased to reject true positive samples. This consideration led to give to the user the opportunity to counterbalance such issue. The most simple solution is to allow to replicate the positive training set to "equalize" such phenomenon. It can be done either by specifying manually how many times it is duplicated, or to allow the software to automatically determine such parameter, with the goal to ensure an equally divided data set (although this requirement could be obtained also manually, not always the class sizes are equal, so it has to be determined on a from time to time). Of course there are solutions that better address these issues (a good starting point is [18]). Nevertheless the simplicity of our solution allowed us to quickly develop the whole framework, and, as we will see later, it is effective in improving the performances. Moreover, the proposed solution is meant to be extensible, so such more advanced techniques can be later integrated.

The last issue to address is how to select the best classifier

[2]We will refer to the *global* MSE/BitFail as the MSE/BitFail mentioned until now, i.e., the ones on the entire validation set, in order to distinguish them from these new definitions.

among the set of the described configurations. This means to estimate the generalization ability of the trained systems, and the most simple way is using the *holdout* method [19], with the results on the validation set as an estimate of the real classification performances. It is well known that, with this technique, the main issue is how to properly split the whole data set, establishing the right proportions. In this respect there is no best choice, thus we have to make some trades-off: with a division of 90% for the training and 10% for the validation we privilege the learning process, in order to be sure to have well trained ANNs. On the other hand, the need for an accurate estimate of the generalization properties would suggest to reverse the proportions. Repeating the trainings and evaluations according to such scenario, we would expect a consistent performance drop. This could be due to a low generalization capability, but if verified across all the solutions, we could also hypothesize that the cause is the small training set. As already pointed out in [17], the training set should be made larger as long as the ANNs get more complex, and, in our case, having only the 10% could be insufficient. In conclusion, the problem is the holdout method itself, which is inadequate for our purposes; the alternatives [20] are the k-*cross-validation* [21] and the *bootstrap* method. In [22] those two techniques have been compared: the first one has in general a slightly higher variance, especially when k is small, while the second one has a low variance but an extremely large bias in some cases. It is also reported that stratification helps to obtain even better results in cross-validation, and the suggested method is thus the stratification ten-fold cross-validation.

The proposed framework is able to handle both the hold-out method, specifying the proportion between the training set and the validation one, and stratified cross-validation; the original image repository is automatically divided and converted to an intermediate representation, better suited to be processed in the training/evaluation steps. Even if cross-validation is more accurate, the holdout takes less time to train and evaluate ANNs: for this reason it has been extensively used during the initial development phase of the overall framework and to design new network configuration. In the end all tests have been repeated by means of the cross-validation process.

In the next section we will present how this proposed solution has been effectively used in our research project, where the ANN subsystem has to recognize character images, acquired by the input peripheral of our mobile reader device for blind people.

IV. Results

As already pointed out in Sec. I, the primary motivation to develop this framework has been the design and development of a mobile reader device for blind people. First of all we collected a set of more than 10000 images, with the same target application we were developing. Some images used for validation are shown in Fig. 2. These were subdivided in 85 classes, including the uppercase and lowercase characters of the entire Latin alphabet, numbers and other punctuations and symbols. This means that each ANN configuration will be made of 85 networks, each of which will be trained with about 10000 samples and evaluated on the same amount of data (according to the cross-validation method).

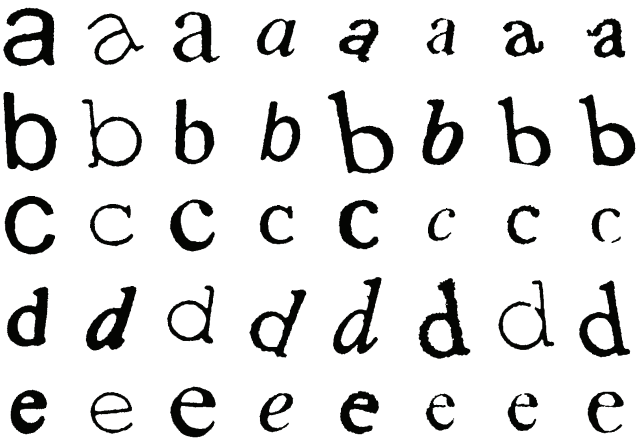

Fig. 2. Example images used for the ANNs validation process. They have been already binarized, segmented and rotated by the previous computation steps. All the images are real-world samples obtained with the same application we were developing.

During the setup and fine tune stage of the classifier system, we defined a first set of configurations (these are derived from [4][3]), trained and evaluated them, analyzed their results and then defined new, and possibly improved, solutions. We ended to have 36 different configurations. For practical reasons not all the possibility have been explored, and we stopped as soon we reached good performances, following the Occam's razor principle. In this paper not all the results will be reported, only the most meaningful. Notice that all the performance measures have been collected for all the configurations. Moreover, the same tests have been repeated with the holdout method (both 90%-training/10%-validation and 10%-training/90%-validation) and with the stratified cross-validation.

TABLE I
Run-time performance comparison of different ANN solution. All timing are in ms.

Name	Arithmetic	Time	Speedup
beta-full	fp	1268.5	—
beta-h64	fp	342.14	+270%
beta-h64	fxp	78.05	+338%

First of all we measured the run-time performances (shown in Table I) of the designed classifiers. It is important to remark that they have been obtained on the target platform, described previously, with the final application (the complete real-time reader engine) taking as input a set of test video files (representing real use cases of the whole appliance). Thus they reflect the average behavior of the overall character

[3]For practical reasons, the configurations discussed in the previous cited work have been renamed, so the `16x16` prefix has become `beta` and `8x8` has turned into `alpha`.

recognition subsystem, not only its core neural network part (although it is of course the primary contributor to such timings). It is easy to see that reducing the connections from a fully-connected ANNs (`beta-full`) to a partially-connected one (`beta-h64`) leads to a system about three times faster, and that switching from floating-point arithmetic (marked as "fp" in the table) to the fixed-point one ("fxp") the recognition engine gained a further more than three times speedup. In the end, the resulting system is more than sixteen time faster. As a matter of facts, `beta-h64` is not (as we will see) the configuration with the highest accuracy, nevertheless the run-time performances depend only upon the network topology, and the final chosen solution has the same layout as this one.

TABLE II
COMPARISON OF THE BITFAIL (ALONG WITH THE MSE) PARAMETER FOR THE BASIC ANN CONFIGURATIONS

Name	Input type	Total neurons	BitFail	MSE
beta	16x16-n	291	0.004973	0.000599
beta-h64	16x16-n	323	0.005595	0.000493
2Dn	16x16-n	388	0.007413	0.000687
2Dp	16x16-p	388	0.007425	0.001209

The next step has been to design two new basic configurations, named `2Dn` and `2Dp` (different only in the feature vector used as input), which should theoretically yield better performances. Along with the MSE we now measure also the BitFail, as described earlier. The results are shown in Table II. Even if the new solutions do not show any improvement, they have been used as the base for testing the modified training strategies and relationship between the fixed-point mode and the floating-point one.

Starting from such basic configurations, we implemented the possibility of replicating the positive training set either automatically or by a user-defined parameter. The expected result was the consistent reduction of the error ratio (both MSE and BitFail) between the intra- and extra-class one. In Table III the results are reported. The technique is effective, since the gap between the two errors is large for the starting configurations (`2Dn`, `beta` and `beta-h64`), but, as long we increase the Positive Set Multiplier (PSM), the ratio decreases, even if this works only to some reasonably extent (see `2Dn-x85-10k` vs `2Dn-x100-10k`). It is also noteworthy that this approach leads to a worse global BitFail. Mainly focusing on the positive outputs, the learning process will take "less care" of the results on the other input data. Despite this, looking for example at the two best configurations (`2Dn-xa-10k` and `2Dn-x85-10k`) respect to the "original" one (`2Dn`), we can see that we have obtained an improvement of one order of magnitude on the intra/extra class BitFail ratio, compared with a worsening of the global BitFail by a factor of 2.4–3.3. We have verified that MSE exhibits the same behavior.

The main task of this framework is the management of the arithmetic used by ANNs, for the run-time performance

TABLE III
RESULTS OBTAINED WITH THE REPLICATION OF THE POSITIVE SET DURING THE TRAINING

Name	PSM	Stop criterion	$\frac{BitFail_{IC}}{BitFail_{EC}}$	BitFail
2Dn-xa-10k	auto	Err	4.25	0.019132
2Dn-x85-10k	85	Err	4.26	0.020421
2Dn-x100-10k	100	Err	4.79	0.023967
2Dn-xa-bf1-10k	auto	BitFail	4.99	0.016461
2Dn-x50-10k	50	Err	6.01	0.018326
2Dn-x10-10k	10	Err	12.30	0.018280
2Dn	—	Err	40.47	0.007413
beta	—	Err	42.92	0.004973
2Dp	—	Err	50.32	0.007425
beta-h64	—	Err	69.31	0.005595

reason seen before. Thus we have to be sure that even if the training is performed with the floating-point mode, the behavior in the fixed-point one is the same. The first thing we did was to repeat the same tests already shown with the new arithmetic. We have seen (the results are omitted for brevity reason) that the same used technique of replicating part of the data set is still effective, even if not always as much as in the previous case. So we further investigated the relationship between the two modes by directly comparing the outputs, for every network and every validation sample. We measured the difference, both in terms of MSE and BitFail (i.e., the amount of difference between the corresponding outputs above a certain threshold). Results are shown in Table IV; notice that the ratio between the two BitFail (fixed- and floating-point) is near to 1.0, leading to conclude that the classification properties should be almost the same.

TABLE IV
RESULTS OF THE COMPARISON OF THE FLOATING-POINT MODE AND THE FIXED-POINT ONE

Name	MSE	BitFail	Average dec. point	$\frac{BitFail_{fxp}}{BitFail_{fp}}$
beta-h64	0.000036	0.009819	10.0	1.03
2Dp	0.000542	0.023552	7.7	1.12
2Dn-xa-10k	0.001666	0.055117	9.6	0.92
2Dn	0.003125	0.026016	8.1	1.52
2Dn-x50-10k	0.005290	0.108415	9.4	1.11
2Dn-x100-10k	0.005376	0.108611	9.4	0.67
2Dn-x85-10k	0.006873	0.104777	9.4	1.19
2Dn-x10-10k	0.007990	0.105905	8.9	1.31
2Dn-xa-bf1-10k	0.012345	0.064338	9.3	1.02

Until now, all the tests have been uniquely setup for the newly designed layout, but, since the implemented procedures have shown to be useful, it would be interesting to apply them also to the networks designed in the previous work. The results (see Table V) are again encouraging and more promising than the previous ones.

Despite all the tests done, the real goal we are interested to achieve is to have an accurate measure of the generalization ability of the proposed configurations of classifiers. As already pointed out, the stratified cross-validation is one of the most reliable method, and in Table VI the performances

TABLE V

RESULTS IMPROVEMENT OBTAINED FROM THE BETA-H64 BASE CONFIGURATION (FIXED-POINT MODE)

Name	PSM	Stop criterion	$\frac{BitFail_{IC}}{BitFail_{EC}}$	BitFail
beta-h64-x100-10k	100	Err	7.65	0.020421
beta-h64-x85-10k	85	Err	8.13	0.019558
beta-h64-xa-bf1-10k	auto	BitFail	9.81	0.011891
2Dn-x85-10k	85	Err	9.83	0.024289
beta-h64-x50-10k	50	Err	11.92	0.016864
beta-h64-x85-bf2-10k	85	BitFail	13.60	0.005618
beta-h64-x10-10k	10	Err	23.80	0.014861
beta-h64-10k	—	Err	54.19	0.014033
beta-h64	—	Err	110.44	0.005756

are reported (clearly with the fixed-point arithmetic that will be used in the target application) along with their confidence interval. According to the winner-takes-all approach we make distinction between wrong classifications and no classification at all. In our system, a successive stage will recover this last ones by means of an integrated spell-checker. It is noteworthy that the best configuration (`beta-h64-x85-bf2-10k`) is a consistent improvement of the basic one (`beta-h64`), and setting the BitFail as the training target along with the replication of the positive set have served to obtain such results.

TABLE VI

FINAL CLASSIFICATION RESULTS (%) OF THE BEST CONFIGURATIONS (RUNNING IN FIXED-POINT MODE) WITH THE STRATIFIED CROSS-VALIDATION (THE CONFIDENCE LEVEL IS 99%)

Name	Correct results	Confidence interval	Wrong results	No results
beta-h64-x85-bf2-10k	94.72	94.12–95.26	0.99	4.29
beta-h64-x50-10k	92.71	92.01–93.34	1.47	5.82
beta-h64-x85-10k	92.59	91.89–93.23	1.48	5.93
beta-h64-x100-10k	92.52	91.82–93.16	1.54	5.94
beta-h64-xa-bf1-10k	92.50	91.80–93.15	1.30	6.20
beta-h64-x10-10k	92.08	91.36–92.74	1.30	6.63
beta-h64-x85-bf2-1k	91.37	90.62–92.06	1.64	6.99
beta-h64	90.15	89.36–90.89	0.76	9.09

V. CONCLUSION AND FUTURE PERSPECTIVES

In this paper we presented the design and the development of a framework for designing and evaluating artificial neural network targeting the mobile world. It is a vertical solution, since it is able to cope with all the issues involved in such kind of projects: taking as input both the description of the desired solutions and data set, it divides and manipulates the training set, generates the networks, evaluates them before and after the conversion from the floating-point mode to the fixed-point one. The obtained results are encouraging, we have been able to improve the recognition rate from about 90% to about 95%. This value should be comparable with those of commercial available products. The developed artificial neural networks have been successfully integrated in mobile real-time reader device for visually impaired people, currently under test in collaboration with local associations for the blind.

Being based on open-source technologies, it has been possible to make it a cross-platform tool, easy to extend and to improve through a scripting language, while ensuring the same results and behavior on all the involved platforms. In this way the user can both design ANN systems to be later integrated in real-word applications (since they will be able to re-use and integrate the same code base), and to experiment new functionalities, by developing new ANN algorithms. In fact the proposed framework is only the starting point for a more complete system. There are several possibilities for improving the system, ranging from adding more powerful data processing routines for class-imbalanced problems (see [18]) to completely moving the ANNs training to new high-performance computing platforms (e.g. the new generation of GPUs, see [23]), or to add new classifier evaluation routines (such as the methods based on the ROC curve, see [24]). Currently we are working on these aspects, nevertheless the implementation of the optional replication of part of the data set constitutes the base for such improvements, and managing the same ANNs with two different arithmetics makes easier to interface with new computing models.

In the end, the usefulness of the proposed solution has been shown by developing the core part of a mobile real-time reader device for blind people. The resulting ANNs are at the base of the character recognition engine, which exhibits a high recognition rate while keeping low the resource need.

REFERENCES

[1] J. Linvill and J. Bliss, "A direct translation reading aid for the blind," *Proceedings of the IEEE*, vol. 54, no. 1, pp. 40–51, Jan. 1966.

[2] J. Bliss, "A relatively high-resolution reading aid for the blind," *Man-Machine Systems, IEEE Transactions on*, vol. 10, no. 1, pp. 1–9, March 1969.

[3] *kReader Mobile User Guide*, K-NFB Reading Tecnology Inc., 2008.

[4] P. Motto Ros and E. Pasero, "Artificial neural networks for real time reader devices," in *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, Aug. 2007, pp. 2442–2447.

[5] *Intel XScale Core Developer's Manual*, Intel, Jan. 2004.

[6] *Intel PXA27x Processor Family Developer's Manual*, Intel, Jan. 2006.

[7] *Intel Wireless MMX Technology Developer Guide*, Intel, Aug. 2002.

[8] J. Holt and T. Baker, "Back propagation simulations using limited precision calculations," *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 2, pp. 121–126, Jul. 1991.

[9] S. Draghici, "On the capabilities of neural networks using limited precision weights," *Neural Netw.*, vol. 15, no. 3, pp. 395–414, 2002.

[10] M. Riedmiller, "Rprop - description and implementation details," Institut fr Logik, Komplexitt und deduktionssysteme and University of Karlsruhe, Tech. Rep., Jan. 1994.

[11] D. Boling, *Programming Microsoft Windows CE*, 2nd ed. Microsoft Press, Jul. 2001.

[12] K. Beck, *Test-Driven Development By Example*. Addison Wesley, Nov. 2002.

[13] D. Astels, *Test-Driven Development: A Practical Guide*. Prentice Hall PTR, Jul, 2003.

[14] S. Nissen, "Implementation of a fast artificial neural network (FANN)," Department of Computer Science, University of Copenaghen (DIKU), Tech. Rep., Oct. 2003. [Online]. Available: http://leenissen.dk/fann/

[15] R. Ierusalimschy, *Programming in Lua*, 2nd ed. Lua.org, Mar. 2006.

[16] *The GNU Awk User's Guide*, Free Software Foundation, Inc., 2009.

[17] T. Kavzoglu, "Determining optimum structure for artificial neural networks," in *Proceedings of the 24 th Annual Technical Conference and Exhibition of the Remote Sensing Society*, 1999, pp. 675–682.

[18] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.
[19] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson Education (US), 2006.
[20] A. R. Webb, *Statistical Pattern Recognition*, 2nd ed. John Wiley and Sons Ltd, 2002.
[21] C. Schaffer, "Technical note: Selecting a classification method by cross-validation," *Mach. Learn.*, vol. 13, no. 1, pp. 135–143, 1993.
[22] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial intelligence (IJCAI)*. Morgan Kaufmann, 1995, pp. 1137–1143.
[23] K.-S. Oh and K. Jung, "Gpu implementation of neural networks," *Pattern Recognition*, vol. 37, no. 6, pp. 1311 – 1314, 2004.
[24] T. Fawcett, "An introduction to roc analysis," *Pattern Recogn. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.