*Proceedings of the International Conference on*
*Information Technologies (InfoTech-2017)*
*20-21 September 2017, Bulgaria*

# MOBILE APPLICATION FOR PATTERN RECOGNITION BASED ON MACHINE LEARNING

**Desislava Ivanova, Vladimir Kadurin**

*Technical University of Sofia, Faculty of Applied Mathematics and Informatics*
*e-mails: d_ivanova@tu-sofia.bg, vladimirkadurin@gmail.com*
*Bulgaria*

**Abstract:** The paper presents the development of mobile application based on artificial intelligence. The functionality of the proposed mobile application is to recognize handwritten objects (letters and digits). For the development of the mobile application, the Artificial Neural Network (ANN) is used. The application is implemented on iOS platform, written in programming language Swift.
**Key words:** Mobile Application, Pattern Recognition, Artificial Neural Network (ANN), iOS.

## 1. INTRODUCTION

Artificial intelligence is very popular nowadays. Terms like artificial intelligence are very common. Artificial intelligence is not a new science. It appears in Greek mythology, showing the human desire for human alike machines, which can react autonomously to outside stimuli. But it is fair to be said that artificial intelligence begins with the invention of the first computer in 20th century [1, 2].

Artificial intelligence is very broad term, which includes many areas. One of them is machine learning. It is a science for algorithms, which can learn from different data sets and to make conclusion based on them. Machine learning has three main categories:

- ✓ *Supervised Learning* - algorithms learn from samples, which we prepare in advance. Then they use these samples as analogy for evaluation of new unknown data.
- ✓ *Unsupervised Learning* - algorithms learn from random data, which is not prepared in advance, and discover their own structure and meaning of the data.

✓ *Reinforcement Learning* - *a*lgorithms interact with dynamic environment, and the goal is to achieve certain goal without someone explicitly giving instructions.

There are also different approaches for solving these problems. They all offer different time-complexities and computational performance. Some of the popular ones are Decision Tree Learning (DTL), Association Rule Learning (ARL), Artificial Neural Networks (ANN), Support Vector Machines (SVM), Bayesian Networks (BN) and others [3, 5, 6].

The paper proposed the development of mobile application for pattern recognition based on artificial intelligence. The approach that is chosen for the development of mobile application is ANN. This model is inspired by the biological neural networks. Its structure is consisted of interconnected artificial neurons, which process the computations. Its main advantage is neural networks can model complex relationships between inputs and outputs to find patterns in big data sets or capture statistical structure in unknown data sets. They require complex and time-consuming training which is done with the process of backpropagation [7, 8].

The mobile application is implemented on iOS platform and it is written in programming language Swift. The main functionality of the mobile application is to recognize handwritten objects (letters and digits). The mobile application is trained in advance with the MNIST data set for handwritten digits.

The next sections of the papers will present the mobile application architecture, used techniques and the functionality of the mobile application. Finally, the experimental results are presented and analysed.

## 2. MOBILE APPLICATION FOR PATTERN RECOGNITION

The developed application put its focus on one of the most useful branches of artificial intelligence knows as machine learning. This is study for training computers to take actions without being specifically programmed to do it. If used correctly, it can perform tasks, which would be impossible or unrealistic, using standard programing techniques.

If we take for an example an application, which recognizes handwritten letters on sheet of paper: there is a possibility to create rules for classification of each different character. That would include extracting information from the image, their individual processing and development of extremely complicated mathematical model, which associates the darkness and position of each pixel to each letter.

It is clear that such approach is unpractical and almost impossible. Best case scenario is to end up with thousands lines of unpredictable code for recognition of precisely defined handwriting. The developed application shows how easier and better such functionality can be and just with several lines of code, using machine learning and requiring exactly zero explicit rules to be written by the developer. Here are some possible applications for it: handwriting recognition, gesture recognition,

face recognition, navigational systems, song recognition, speech recognition, gaming artificial intelligence, weather forecasting and fraud detection.

### 2.1. Mobile Application Architecture

The mobile application is consisted of two different parts, Fig 1. The first and most important part is the core business logic – this is implementation of the ANN and all the algorithms for training and digit recognition. It is designed with flexibility in mind and depends on a lot of dynamic parameters which can be used for tuning and optimization. It is completely separated and independent of the UI part which offers the graphic interface for the user.

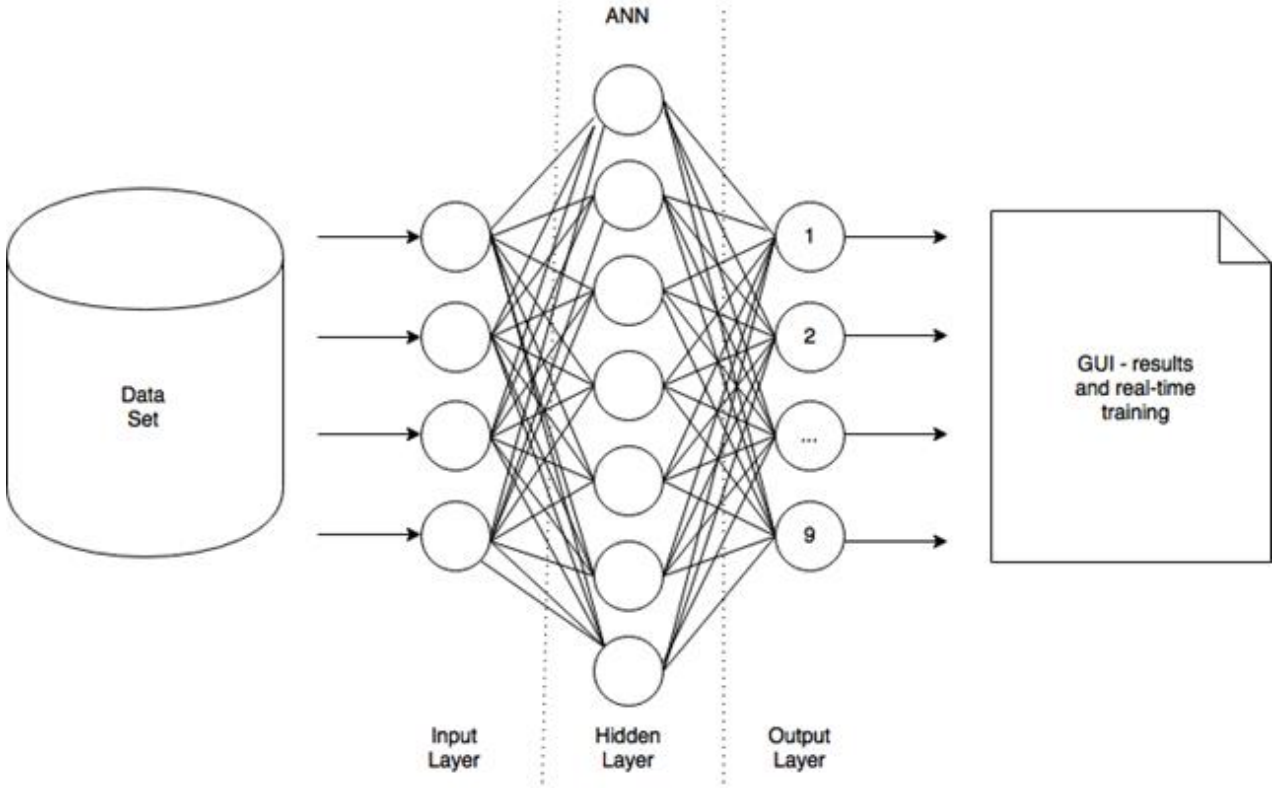


*Fig. 1. Mobile Application Architecture*

That architecture offers the possibility of taking the core functionality and using it for completely different problem solving. As the ANN is completely independent of the training data set, it can be trained for different problems. The only prerequisite is the prepared data set with carefully chosen samples.

The other part of the application is the UI layer which just gives user-friendly interface for using the neural network. It also visually represents the results of the recognition and the process of learning in real-time (regression). The module of the training sets is also separated and gives the flexibility for changing the training data very easy and without breaking the existing functionality.

### 2.1. Used Techniques and Tools

The application uses special algorithm knows as ANN. This technique falls under the Supervised Learning category as it requires pre-training of the network. The neural network consists of layers. Data flows from the input layer to the output layer. Each layer has different number of elements called neurons. Each neuron from previous layer is connected with every neuron from the next layer. This is called fully connected network. The input layer has the same number of neurons as the number of pixels of the input image. The output layer has 10 neurons - one for each digit from 0 to 9. Each connection between neurons has a weight, Fig. 1. The process of training adjusts these weights so when we give the network an image of 5 for example it will output 5. The mechanism for training is backpropagation. This is a process of gradually adjusting the weights while the desired accuracy is achieved.

The application has two parts. First one shows how a neural network is trained in real time using backpropagation. The real visualization of the mobile application presents two lines with different colours. The green one represents the target function and the red line the current state of the learning process. Over time both lines start to match and the reason for this is that the output error gradually decreases. What it really happens is for each X position the neural network tries to "guess" the respective Y value. It outputs some result which has some percent of error. Here comes the backpropagation which takes the result, compares it with the target value and propagates back the error, adjusting the weights. Target values are never saved, they are just used for computing the current error. It is possible in some cases for the training to stop. This is normal and is called local minima. This can be prevented by adjusting the learning rate and momentum factor but this requires a lot of tests.

The second part of the application recognizes handwritten digits. Users can write digits on the screen with their finger and the application tries to recognize them. The writing itself is rasterized and its resolution is reduced to 28 by 28 pixels. This is done so that we can decrease the size of the data given as input to the neural network. The less data we input, the smallest is the chance for error.

The platform for which the application is developed is iOS. It offers a lot of build-in core functionality for convolutional computations which are used for the process of backpropagation. The platform also lets developers use the GPU directly which makes the performance a lot better if optimized carefully.

The programming language of choice is Swift. It is faster and much more suitable for rapid prototyping development than other programming languages like Objective-C for example. It is also open-source which means that with the right frameworks it can easily be used on other platforms too which makes porting the existing functionality much easier.

## 3. EXPERIMENTAL RESULTS AND ANALYSIS

### 3.1. Experimental Datasets

The mobile application is trained in advance with the MNIST data set for handwritten digits [4]. The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centred in a fixed-size image. The MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. NIST originally designated SD-3 as their training set and SD-1 as their test set. However, SD-3 is much cleaner and easier to recognize than SD-1. The reason for this can be found on the fact that SD-3 was collected among Census Bureau employees, while SD-1 was collected among high-school students. Drawing sensible conclusions from learning experiments requires that the result be independent of the choice of training set and test among the complete set of samples. Therefore it was necessary to build a new database by mixing NIST's datasets. The MNIST training set is composed of 30,000 patterns from SD-3 and 30,000 patterns from SD-1. Our test set was composed of 5,000 patterns from SD-3 and 5,000 patterns from SD-1. The 60,000 pattern training set contained examples from approximately 250 writers. It is made sure that the sets of writers of the training set and test set were disjoint. The neural network can easily be trained to recognize letter too if you train it with good enough data set. For every written digit to the screen, network is forced to classify it to one of the 10 outputs (the digits from 0 to 9). Training is stopped after achieving 98% accuracy in the recognition.

### 3.2. Experimental Results and Analysis

First part of the application shows in real-time how the training of the neural network is done. As this process has to be relatively fast, the chosen training model is sinusoid. The weights of the neural network are randomly picked numbers close to zero. The application has two different lines: one (green) shows the training model itself and the other (red) shows the current state of the trained application. When the training is initiated the red line starts to curve and mimic the shape of the training model line. At first this becomes more visible close to the Y axis because the activation function is sigmoid and its shape resembles the shape of the sinusoid. That way the initial weights which are calculated during the first passes of the neural network, are very close to the sample ones and the error is small. This accelerates the process of backpropagation and just after several iterations some of the neurons have accurate enough weights.

After more time the learning curve of the network becomes very close the goal. There is even possibility this to happen first on the left or the right side, but it is most probable this to happen first on just one of the sides. The reason for this is that the

process of backpropagation first starts to get closer to the right values only on one of the network sides. After the error there is minimized, the neural network starts to update the weight more aggressively on the other side too.

It is possible for the network to first have weights which output very high error. In most cases reduction of the error is smooth and can be observed. This is exactly the purpose of the back propagation process - smooth reduction of the error delta.

The tests show that the learning curve never exactly match training model. The reason for this is that sinusoid has very sharp edges at the top and bottom and the rate of learning does not let the curve to ever be so sharp. As the rate is relatively high, with each iteration the learning algorithm always misses the sinusoid edges. This can be fixed with decreasing the learning rate but we risk making the learning process a lot more difficult and slower because the calculations will be much more as the dots will be more too. Also this can lead to over-fitting. The other worse thing is to pick learning rate which is too sparse - the training process itself will be faster but the neural network may never reach the desired acceptable error. That is why it is best to find the appropriate learning rate for the exact problem which has to be solved.

Sometimes very sharp change of direction of the learning curve can be observed. The reason for this is that the process of backpropagation is doing changes to already trained neurons so that the adjacent neurons can improve their performance. This causes the sharp amplitudes but the weights recover to their normal values very quickly (the whip effect). In other words, the network is making serious changes to already trained part of it, so that the error can be even in the whole network. Overly big errors in certain places are adjusted trough distribution over the whole neural network.

If the training model is simple, the sinusoid is a smooth curve, some anomalies to the learning curve can be observed during the process of training. The reason for this is that the learning rate is too small for the model. The neural network is trying too hard to reduce the error and this is impossible with the current model. This is the perfect example of overfitting.

It is good to be mentioned that the training process is stopped when the error value reaches 2%. This is chosen on purpose and can vary. Main criteria for choosing error percent are the size of the data set, the complexity of the training model and the problem which has to be solved. If the data set is too big and the problem does not require too much accuracy - error percent do not have to be so high. In another cases though like the recognition of road signs in the self-driving cars, the error has to be as low as possible because the reliability has to be very high.

The second part of the application is an input form for writing digits which the neural net-work tries to recognize. The network is pre-trained with the MNIST set, which includes around 70 000 samples of handwritten digits from which 60 000 are used for the training itself and the rest 10 000 for the verification of the end results. As the samples are digits which are handwritten on a piece of paper, it is possible for the network not to recognize so well the digits written with a finger on the screen. The reason is pretty simple - the shape of the digits is different from the training set.

Despite that the tests show very good results which exceed 90% accuracy in most cases.

After the digit has been written, it is rasterized and its resolution is decreased. The main reason for this is to decrease the size of the input data which will also decrease computational count. A minimum in which the digit is still readable has to be found. I found through many tests that a good resolution is 28 by 28 pixels. This counts for 784 input and 10 output neurons (digits from 0 to 9). If the image resolution is higher, we have to increase the size of the training data. These two factors would increase the degree of accuracy, but the training time will be longer. Current implementation has three layers - input, hidden and output. They are enough for achieving the goal accuracy with this resolution. The neural network can be relatively easy trained to recognize letters by only increasing the count of the output neurons. The only thing needed is a good training set.

The process of preparing the training data set has to be done on a separate machine which has better performance and computational capacity. Hardware of current mobile devices is still not powerful enough for good results to be achieved. External tools have to be used for the preparation of the training data - TensorFlow or Caffe. Custom solutions can be developed too. For this research TensorFlow framework has been used.

Recognition process takes less than a second to output a result. The reason for this is that the structure of the network is carefully implemented. As a whole the setup of the network characteristics is extremely important step for the solution of every machine learning problem. It is possible to use very good algorithm for learning, but if the characteristics are not picked according to the cur-rent data set, results will never be accurate enough.

Very important problem to solve for the recognition of any object is its absolute position - what happens if the number is written bottoms up. There several solution for this problem. The current implementation uses the so called "recognition grid". After every written digit, an evaluation of the density and distribution of the pixels is made and only the square including the digit is taken. This of course is done with a degree of assumption and can decrease the overall recognition accuracy. After the square with the digit is defined, it is given to the network in several different positions. Each output result is additionally evaluated to find the most accurate one. Test show that the neural network continues to give good results but they can be compared with cases when the digit is written in the correct direction.

## 4. CONCLUSION

This paper presents the main rules for implementing an mobile application, which uses machine learning. Current implementation includes training for recognition of handwritten digits but it can easily be trained to recognize letters too. Overall tests show that the current implementation achieves around 90-95% accuracy, Fig. 2. For better accuracy more training sets of data have to be prepared and it is also

important for them to be relevant. Part of the reason for the percent of errors is that the training set includes samples of digits written by pen on a sheet of paper and the application lets you draw digits directly on the screen with a finger. The chosen technique is ANN. Its structure has 3 layers - input, hidden and output layers. Neurons count is optimized for the current implementation for recognizing handwritten digits. Backpropagation is used for the training. The data set is prepared in advance with external tool called Tensor-Flow. Programming language is Swift and the platform is iOS. This environment gives enough performance for the training itself and then for the usage of the neural network.

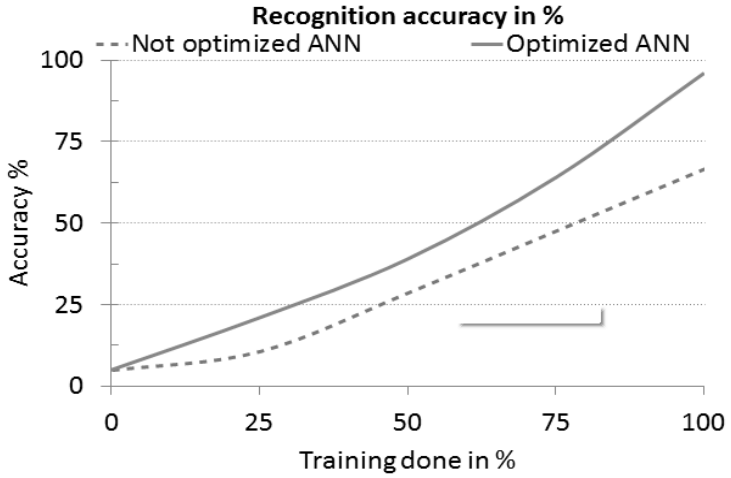


*Fig. 2. Experimental Results*

But for the preparation of the data, a more powerful computer configuration is needed. With the current pace of evolution of mobile devices, it won't be very long until the whole process of machine learning could be executed on them.

## REFERENCES

[1] James McCaffrey (**2012**). Neural Network Back-Propagation for Programmers, *MSDN Magazine.*

[2] Taranjit Kaur (**2012**). Implementation of Backpropagation Algorithm: A Neural Network Approach for Pattern Recognition, International Journal of Engineering Research and Development, ISSN: 2278-067X, Volume 1, Issue 5 (June 2012), pp.30-37

[3] Jürgen Schmidhuber (**2015**). Deep learning in neural networks: An overview, Elsevier, Volume 61, pp. 85–117.

[4] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, *The MNIST database* (*available at: http://yann.lecun.com/exdb/mnist/).*

[5] Sarah Guido, Andreas Müller (**2016**), Introduction to Machine Learning with Python, *A Guide for Data Scientists, O'Really.*

[6] Kasabov, N. (**2017**) From Multilayer Perceptrons and Neuro-Fuzzy Systems to Deep Learning Machines: Which Method to Use? – A Survey. International Journal on Information Technologies and Security, ISSN 1313-8251, No. 2 (vol. 9), pp. 3-24.

[7] https://developer.apple.com/reference/accelerate

[8] https://developer.apple.com/videos/play/wwdc2016/715/