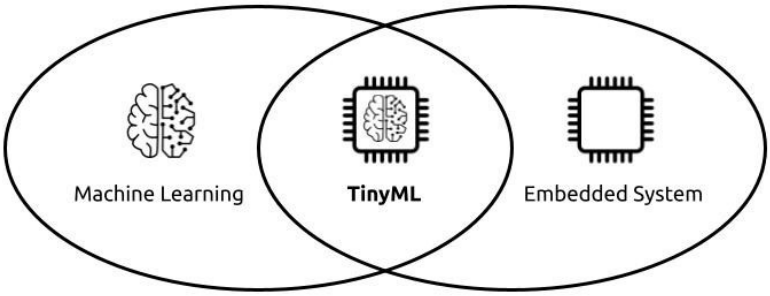
Alma Mater Studiorum · University of Bologna

3 CFU Project work : Extension of handwritten recognition with TinyML technique on ESP32 device[1]



IoT course

Davide Angelani(davide.angelani@studio.unibo.it)
Salvatore Pisciotta (salvatore.pisciotta2@studio.unibo.it)

February 15, 2023

[1]The code for the project is publicly available on GitHub

Contents

1 Introduction

Machine learning and deep learning have become indispensable part of the existing technological domain. Edge computing and Internet of Things (IoT) together presents a new opportunity to imply machine learning techniques and deep learning models at the resource constrained by embedded devices at the edge of the network. Generally, machine learning and deep learning models requires enormous amount of power to predict a scenario. In last paper, we presented an application about the potentiality of TinyML in the context of handwritten recognition challenge using an ESP32 device. In this paper, we want to propose a possible extension of the project in order to make the device able to be used by users to create long sentences. So we will not only make prediction made on single letters captured by the device but also some make some inference on the word that the user is writing and the next word that the user could write next to the predicted one

2 Project Architectures

2.1 Project Description

During recent years, traditional recognition technologies have indeed evolved; however, there are still difficulties when reading cursive, non-constrained handwriting, etc. Breakthrough was possible only after Machine Intelligence was introduced. Recent advancements in Deep Learning such as the advent of transformer architectures have fast-tracked our progress in cracking handwritten text recognition [8] but the are still some challenges on this task and one of this is to reduce as much as possible the delay that it is present between the end of the transcription of the character and the prediction itself. In last experiment we proposed the idea to make inference directly on a micro controller that is electronically linked with the input source of the character and obtain the prediction of it thanks to the TensorFlowLite [6] library and its compatibility with the ESP32 microcontroller. Despite the interesting results, the device in this configuration lacks of usability, especially from a point of view of the user experience. In this paper we propose an extension of the last process, integrating new features that could make the device more user friendly. In particular, since the device capacity is almost full filled by the handcharacter recognition model, we rely on the use of another device with much greater memory capacity and computational capabilities. To deal with the interaction with these two devices we developed a communication protocol between them.

2.2 Technologies

2.2.1 Hardware

In this section we recall the characteristics of the device presented in the last proposal:

- *ESP32 Microcontroller* : based on a dual-core processor Tensilica LX6 that can be individually controlled, their clock frequency is adjustable from 80 MHz to 240 MHz, 520kB of RAM is included. The module can communicate wirelessly over WiFi 2.4GHz and Bluetooth 4.0 thanks to the integrated antenna
- *Touchscreen display* : a ESP32 compatible HiLetgo 2.8" ILI9341 TFT RGB LCD Screen Display Touch Panel SPI Serial 240x320 pixel.

The two components are connected using a breadboard as represented in 2.1:

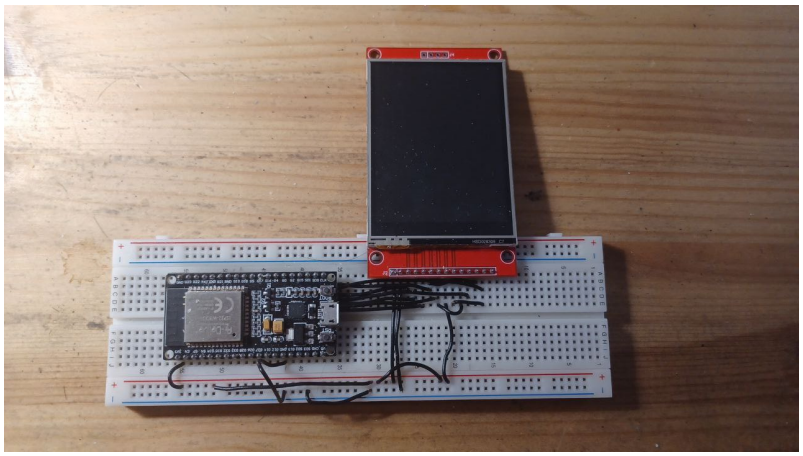


Figure 2.1: ESP32 with display connected

On the server side, we decide to use our computer device, but in general every component that is able to store the models for the text predictions and compatible with the communication protocol selected.

2.2.2 Software

The project was divided in two main parts:

- **Model definition and train part** : performed with Jupyter Notebooks running on Google Colab. All this part was implemented in Python.
- **Inference part** : developed in C++ both for backend and frontend part
- **Server communication part**: Flask that is a famous web framework that works with Python language, so compatible with the models definition and libraries.

With this extension, we add a third part, the inference made by the server. Every time a condition on the input text is met, the server will do the correct inference to make and pass the prediction through the communication protocol.

Furthermore we recall that on the device side, we dealt with models import and inference using TensorflowLite libraries in C++, while for for the compression and quantization of the model we rely on TensorflowLite [6] methods in Python language. On the contrary, in order to deal with the models that has the aim to make inference on the current and next word prediction, presented in 3 we used Tensorflow [1] and Pytorch libraries in Python language.

3 Project Implementation

3.1 Next word predictor

In order to predict the next word in the sentence, we use **GPT-2** pretrained model introduced in [5]. The selection of this model was done in order to guarantee a good level of performances on the next word prediction, with a word based model approach. During our experiments we noticed that the pre-trained model was enough to create acceptable results in the prediction of common English words, that is the main field of application of this device, that can be intended as a touchscreen keyboard for common sentence production. Indeed, word-based models can't generate out-of-vocabulary (OOV) words, they are more complex and resource demanding. But they can learn syntactically and grammatically correct sentences and are more robust than character-based ones.

3.2 Current word predictor

For the prediction of the the word that the user is composing, the we will call *current word*, we decide to take a model that is able to make inference not at word level but at character level, once character at the time. The main reason is that we want to make predictions every time a new character is inserted, like in modern smartphones input text systems, so we cannot use a word model prediction, otherwise we have to wait until the end of the word transcription. The proposed model for the character level word prediction is a simple model that is composed by 3 layers:

- Embedding layer
- GRU layer
- Dense layer

The embedding layer is the part of the network that has the goal to encode each character as an embedding vector, in order to allow the model to distinguish between every different character. It can be imposed as trainable or not. Our design choice, considering also the not complexity of the model is to keep it trainable.

The GRU (Gatered Recurrent Unit) layer, introduced by [3], is the most important component of the network. It is a type of Recurrent Neural Network (RNN) that, in certain cases, has advantages over the other well known long short term memory (LSTM). In general it has been shown in the literature that, GRU uses less memory and is faster than LSTM, but losing some accuracy when using datasets with longer sequences of texts.In 3.1 it is possible to see the stucture of a GRU.

Since our training is done not on sentences but on words, since we just want to predict single words, we decide to keep the GRU instead of the LSTM.

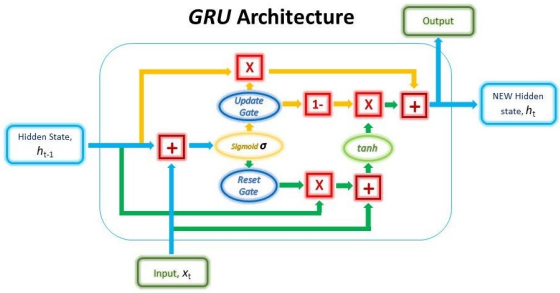


Figure 3.1: Architecture GRU

The final Dense layer is added in order to make the final step of prediction, giving in output logits for each character.

3.3 Dataset

The main dataset used for the training of the current word predictor is a dataset publicy available on Digital Commons @ Michigan Tech site. This is a dataset thinked for train chatbots and in particular, we use one part of the android dataset that is composed by conversations in which the most frequent English words are used in. The words of the dataset, before being passed to the network were pre processed in the following way:

- Removal of every possible symbol and spaces presented
- Removal of stop words composed by only one letter

3.4 Training

Training is a highly demanding task in terms of both computing and memory usage, even with our not so high complex models, so it is important to use properly the available resources. For that purpose, we train on one GPU provided by Google Colab services. In particular all the models where trained follow this configuration of hyper parameters empirically decided:

- *Epochs* : 30
- *Optimizer* : Adam
- *Learning rate* : 0.001
- *Batch size* : 64

The training of the current word prediction model was done for 30 epochs of training. Due to the not so complex model, the training was really fast and efficient so we were able to use bigger vocabularies with respect to the one used in the early stages of the experiments.

3.4.1 Loss function

The loss selected for the training is the **sparse categorical crossentropy**. This choice was made since there are an high number of classes (27 classes, 28 counting possible <unk> (unknown) symbols) and we directly use the vector of the probabilities to extract the predictions.

3.5 Communication protocol

For the communication part the ESP32 device will act as a client while the computer device will act as the server.

Client side (ESP32) To enable the communication between the ESP32 and the computer we first needed to connect the ESP32 to the WiFi to enable the communication with other devices over the internet. The connection is handled by the *WiFi.h* library and the WiFi module already included in the ESP32 device as presented in 2.2.1. Then we need to actual send request and receive response to the server via HTTP. This is done by the *HTTPClient.h* library.

Server side (Computer) The implementation of the server was done using *Flask* which will expose two endpoints.

- *current_word prediction*: returns the predicted the current word that the user is writing
- *next_word prediction*: returns the predicted the next word in the sentence

3.6 Application

The application started from the base of the other experiment where we had most of the area of the touchscreen dedicated to the handcharacter input. On the down part we have the history of the character inserted and we also insert other **buttons** for the predictions that can be selected. Two buttons are filled with the next predicted word. The full-fill of them is triggered by the insertion of a space character in the text field, this as a consequence will call the next word prediction module described in 3.1 that returns the two words with the highest probability. Regarding the prediction of the word that the user is writing, the insertion of every character will call the inference of the model presented in 3.2 passing as input the sequence of character and the suggestion that will appear in the screen is the word with the highest probability. All these proposed words can be accepted by clicking on them, or ignored going on writing in the drawing area of the touch screen.

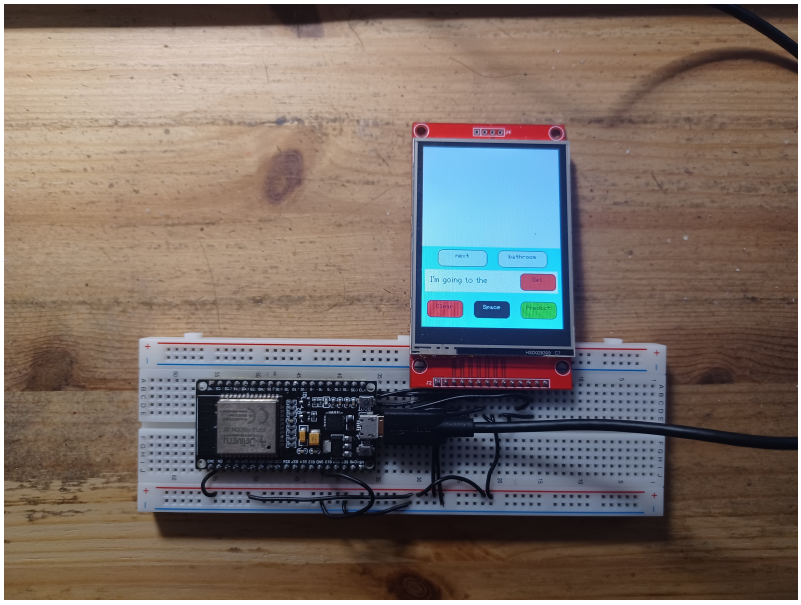


Figure 3.2: Graphic interface of the device predicting the next word

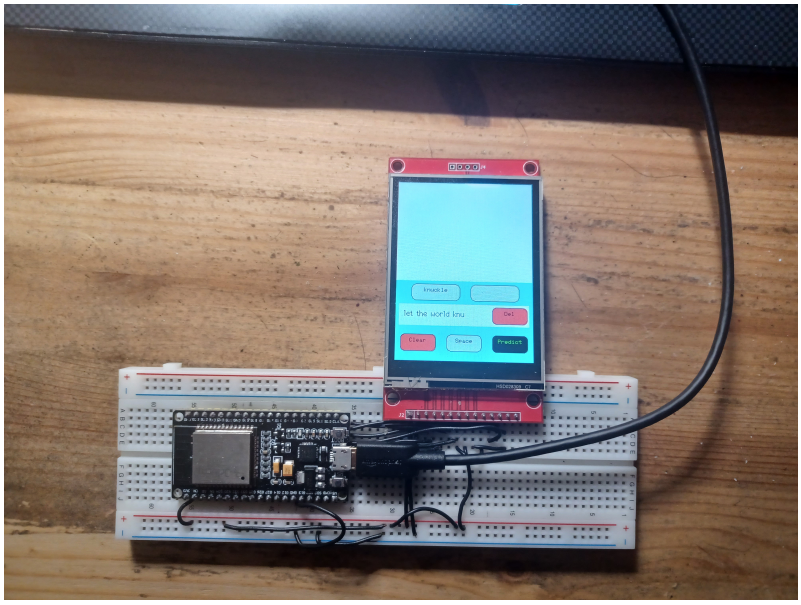


Figure 3.3: Graphic interface of the device predicting the current word

4 Analysis of the results of handcharacter prediction

In the last paper, we presented the results of the prediction made by the device. In this section we will make a comparison between our results and the state of the art:

4.1 Metrics

In order to evaluate results produced by different models' predictions on the test set of images, it has been used three metrics and that are the most used at the state of art [4]

- *Precision*

$$\frac{TP}{TP + FP}$$

- *Recall*

$$\frac{TP}{TP + FN}$$

- *Accuracy*

$$\frac{TP + TN}{TP + TN + FP + FN}$$

4.2 Results

The results that can be seen in the table 4.1 are the final values of every metric computed on the predictions on the test set of the merged dataset, a dataset composed by the MNIST alphanumerical characters and some other characters handcrafted by us. We report only the Modified LeNet-5 model since it was the most promising configuration of the network.

	Test		
	Accuracy	**Precision**	**Recall**
Modified LeNet-5	0.73	0.76	0.73

Table 4.1: Test results of every model

Looking at the state of the art, there are several surveys on the hand character recognition challenge in the literature. Generally they report impressive results, regarding the LeNet5 network, that are almost always over the 95% of accuracy. But generally these surveys are done with high computational capabilities models that are not being part of a quantization and compression models. To take into account the quantization, other studies try to achieve better compression of model in order not to limit too much the performances. The papers proposed by [7] and [2] promise to maintain the same performances above mentioned of LeNet-5 also in compressed methods. For example in the table below it is reported the different accuracy on validation sets for different types of compressions applied on LeNet-5 network, proposed by [2]. We can see that the accuracy obtained by our compressed model is in line with the values reported.

Method	val. error [%]
Original	0.8
VNQ (no P&Q)	0.67
VNQ + P&Q	0.73
VNQ + P&Q (random init.)	0.73
Deep Compression (P&Q)	0.74
Soft weight-sharing (P&Q)	0.97
Sparse VD (P)	0.75
Bayesian Comp. (P&Q)	1.0
Structured BP (P)	0.86

Figure 4.1: Validation accuracy of LeNet5 compressed with different compression methods

The use of these compression techniques can be a good starting point for the future, even if in the mentioned papers , but also others, do not mention the exactly quantity of memory to contain the models, so there will be the necessity to investigate if these methodologies are able to reduce enough the model to fit in our ESP32 device.

5 Conclusions and future works

The possible and potential applications related to the presented device are really a lot. In this experiment we proposed only two major improvements, but once established the connection with a server that is able to receive the input data of an user, it is possible to add whatever feature it can be inserted. For example in order to improve even more the usability of the device, that has been improved in this project, an additional function able to correct possible mistake can be done. Furthermore, introducing a collection of the data kept by the server can be a good added feature from the server side. In this way, it is possible to adapt the models to the writing style of a person from a semantic point of view. This of course will require a more complex back end system that periodically update the model methods calling the training on the different model involved in the overall prediction pipeline.

Bibliography

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. "Variational network quantization". In: *International Conference on Learning Representations*. 2018.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. 2014. DOI: 10.48550/ARXIV.1412.3555. URL: https://arxiv.org/abs/1412.3555.

[4] Angelo Monteux. *Metrics for semantic segmentation*. URL: https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html.

[5] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners". In: (2019).

[6] Li Shuangfeng. "TensorFlow Lite: On-Device Machine Learning Framework". In: *Journal of Computer Research and Development* 57.9, 1839 (2020), p. 1839. DOI: 10.7544/issn1000-1239.2020.20200291. URL: https://crad.ict.ac.cn/EN/abstract/article_4251.shtml.

[7] Xuefu Sui, Qunbo Lv, Yang Bai, Baoyu Zhu, Liangjie Zhi, Yuanbo Yang, and Zheng Tan. "A Hardware-Friendly Low-Bit Power-of-Two Quantization Method for CNNs and Its FPGA Implementation". In: *Sensors* 22.17 (2022). ISSN: 1424-8220. DOI: 10.3390/s22176618. URL: https://www.mdpi.com/1424-8220/22/17/6618.

[8] Christoph Wick, Jochen Zöllner, and Tobias Grüning. "Transformer for Handwritten Text Recognition Using Bidirectional Post-decoding". In: *Document Analysis and Recognition – ICDAR 2021*. Springer International Publishing, 2021, pp. 112–126. DOI: 10.1007/978-3-030-86334-0_8. URL: https://doi.org/10.1007/978-3-030-86334-0_8.