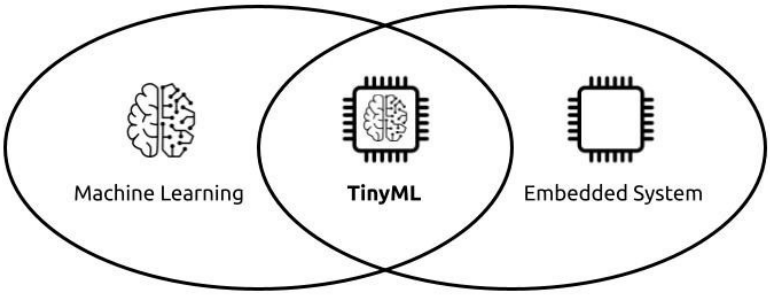
Alma Mater Studiorum · University of Bologna

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)

September 14, 2022

[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. The aim of TinyML paradigm is to shift from traditional high-end systems to low-end clients. The challenges doing such transition are in particularly to maintain the accuracy of learning models, provide train-to-deploy facility in resource frugal tiny edge devices, optimizing processing capacity, and improving reliability. In this paper, we present an application about such possibilities for TinyML in the context of handwritten recognition challenge. We firstly, present our project. Secondly, we list the tool sets for supporting of our experiment and methodologies. Thirdly, we will present the implementation of it focusing on the process of dataset creation and model implementation and quantization. Finally, after having shown the training results we will do an analysis of the results we obtained and identify key challenges and prescribe a future roadmap for this challenge applied in the TinyML context.

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 [6] 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 this paper, we propose 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. This is possible thanks to the Tiny ML technologies available and in particular thanks to the TensorFlowLite [5] library and its compatibility with the ESP32 microcontroller.

2.2 Technologies

2.2.1 Hardware

The hardware used for this project is a device composed by:

- *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:

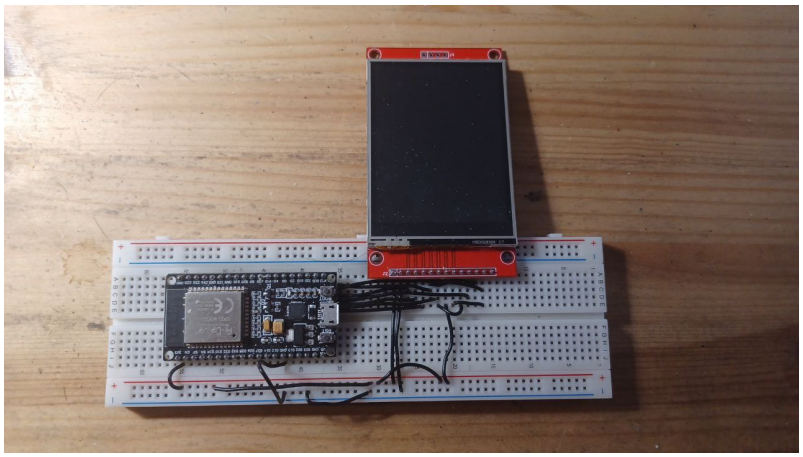


Figure 2.1: ESP32 with display connected

2.2.2 Software

The project is 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

The libraries used to deal with the models presented in 3 are the one of Tensorflow [1]. For the compression and quantization of the model we rely on TensorflowLite [5] methods in Python language, that allow to pass a Keras implemented model and through a function, it returns a lite version of the model that can be deployed on the microcontroller device. Specifically the *fullintegerquantization* method was the one that provides the highest percentage of reduction. Then, on the device side, we dealt with models import and inference using TensorflowLite libraries in C++.

2.3 Dataset

The main dataset used for this project is the dataset provided by [2] for a character recognition task where each instance is a pair compose by a lowercase or uppercase alphanumerical character linked with the correspondent label, in particular the dataset has been divided in training set, validation set and test set. All the characters were equally distributed in all the subsets. The final dataset is structured as follow:

- *Training Set* of 2914 pairs (approximately 85% of dataset size)
- *Validation Set* 310 of pars (approximately 10% of dataset size)
- *Test Set* 186 of pairs (approximately 5% of dataset size)

For experimental purpose, in order to have an overview of how the model could behave with different data, we also provide two further datasets. The first one is a dataset created by us composed by 1984 alpha numeric characters and labels collected manually from the ESP32 device touchscreen divided as follow:

- *Training Set* of 1674 pairs (approximately 85% of dataset size)
- *Validation Set* 186 of pairs (approximately 10% of dataset size)
- *Test Set* 124 of pairs (approximately 5% of dataset size)

The second one was created as the integration of the main dataset and the one created by us as:

- *Training Set* of 4588 pairs (approximately 85% of dataset size)
- *Validation Set* 558 of pairs (approximately 10% of dataset size)
- *Test Set* 248 of char pairs (approximately 5% of dataset size)

2.4 Preprocessing

The first preprocessing operation that has been done is the conversion of the images from RGB to greyscale images in order to simplify the representation of the images for the model. Then it has been necessary to resize the images since the dataset provides 1200x900 pixels images but during the experiments, with this resolution the model size was too big to be quantize and fitted in the ESP32 device. Following this idea, the images have been resized to 28x28 and 23x30, depending on the model used as explained in 3.1, these dimensions were empirically founded making some experiments with different sizes. After the resize, we also normalize the range of the values in image matrices with the purpose, again, to simplify the input and model complexity. Regarding the labels, we mapped each alpha numerical character into a integer number.

3 Project Implementation

3.1 Models

The baseline architecture for our task is the famous LeNet5 [3] model, that is one of the first architecture to use convolutional layers in order to solve Computer Vision tasks and since it was one of the first model in the literature its overall structure is not so complex so it was a good candidate for our project purposes. Despite this trivial structure, this architecture is a good candidate for our task.

3.1.1 LeNet-5

LeNet5 has 5 layers with learnable parameters. It has three sets of convolution layers combined with average pooling layers. After the convolution and average pooling layers, we have two fully connected layers. At last, a Softmax classifier which classifies the images into respective class. In our case the input to this model is a 28 X 28 grayscale image hence the number of channels is one, while the output is a vector of size 62 (the number of the alphanumeric characters).

In 3.1 it is represented the general architecture of the Lenet-5 model:

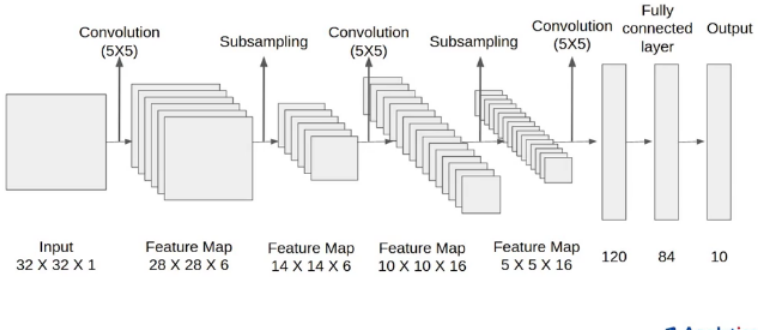


Figure 3.1: Architecture of LeNet-5

3.1.2 Modified LeNet-5

In order to investigate the behaviour of other networks we tried to look in the literature for other models that could fit in our ESP32 based device but all of them, even after the quantization, required too much memory space with respect to the one available. So the solution that we thought was to modify some parts of the LeNet-5 and provide to it images with different input shapes. In particular we changed the architecture of the model converting the last convolutional layer into a dense layer and use as input 23x30 grayscale normalized images.

In 3.2 it is possible to see the summary of this modified version of the LeNet5

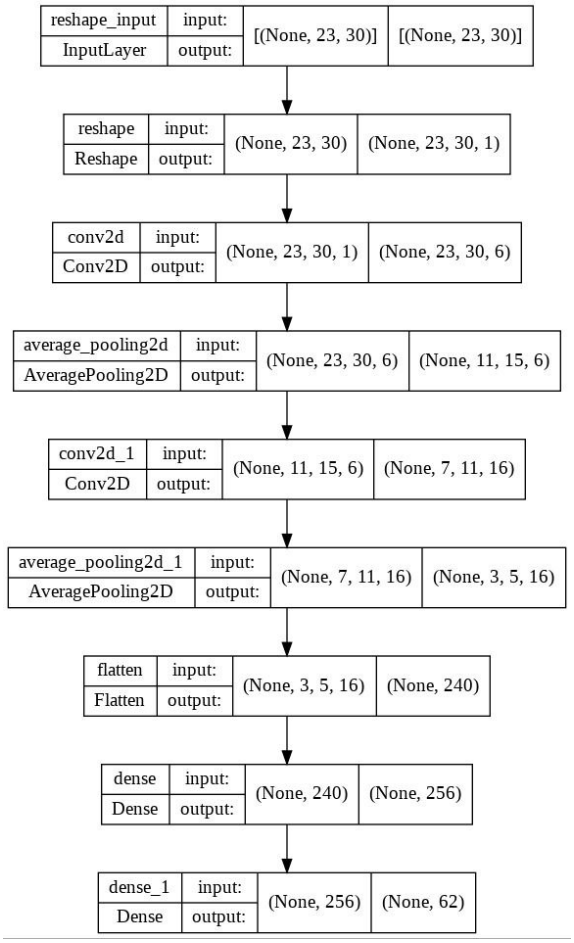


Figure 3.2: Architecture of our modification of LeNet-5

3.1.3 Activation functions

The activation function of the last layer in a convolutional neural network is the one in charge of classifying the category each image belongs to. Since the output should be a probability, softmax and sigmoid are the principal candidates.

For *multi class classification problem*, it is usually better to use a softmax function since there is only one correct class to assign.

Furthermore, after an inspection of the provided dataset, inspired also by the architecture of LeNet-5 it has been favored an output with softmax function.

3.1.4 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 (62 classes) and we do not rely on one-hot encoded representation of the output but we directly use the vector of the probabilities to extract the

prediction.

3.1.5 Regularization

For regularization purposes, in order to avoid overfitting situation, we decided to insert a **dropout layer** after the first Dense layer of every model and adopt an early stopping strategy during the training.

3.2 Training

3.2.1 Hardware used

Training is a highly demanding task in terms of both computing and memory usage, even wit 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 service. In particular all the models where trained follow this configuration of hyperparameters empirically decided:

- *Epochs* : 100
- *Optimizer* : Adam
- *Learning rate* : 0.001
- *Early stopping patience* : 10 monitoring the val_loss
- *Batch size* : 32
- *Dropout rate* : 0.4

3.2.2 Training results

In 3.1 are the summary of our trainings on the used models:

	Epochs			
	loss	**val_loss**	**accuracy**	**val_accuracy**
LeNet-5 baseline	0.488	0.591	0.559	0.559
LeNet-5 modified	0.525	0.618	0.591	0.591

Table 3.1: Train results of every model

3.3 Quantization

Starting from the implementation of the models, as already introduced in 2.2.2, with the conversion techniques implemented by tensorflow lite we want to reduce the dimension of the trained model to import it in the limited capacity memory of our device. Unfortunately, microcontrollers support only one method of quantization which is the *full integer quantization*. This quantization process consists in converting 32-bit floating-point numbers (such as weights and activation outputs) to 8-bit fixed-point numbers. Such approximation is done using the following formula:

$$int8\_value = \frac{real\_value}{scale} + zero\_point$$

where *zero_point* is the quantized value q corresponding to the real value 0 while the constant *scale* is an arbitrary positive real number, typically represented in software as a floating point quantity.

3.4 Application

We create an application that allows to take the input and collect the prediction of every single letter in order to compose a word, shown on the display with a graphic interface, shown in 3.3 with three buttons each one associated to an action. In order to make the prediction based on the input taken from the touchscreen, we implement in the application the nearest neighbor algorithm for the downscaling of the image, that was previously converted from matrix, captured by the display that has to be used as input for the model prediction.

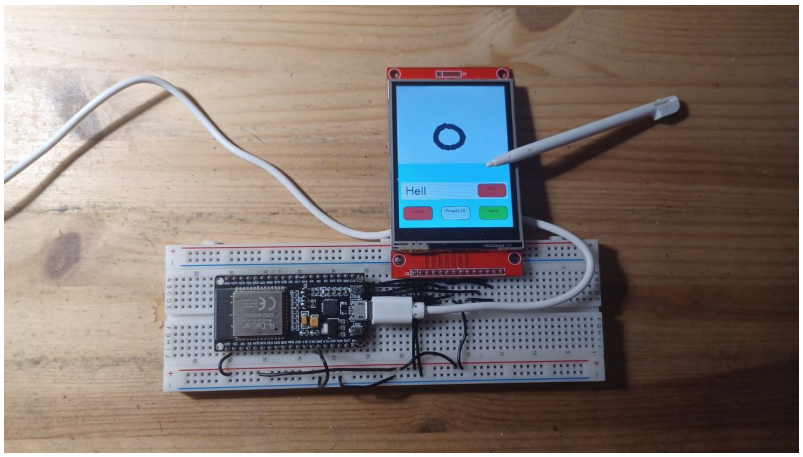


Figure 3.3: Graphic interface of the device

4 Results

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*
- *Recall*
- *Accuracy*

4.1.1 Precision

The precision effectively describes the purity of our positive classifications relative to the ground truth. Precision is calculated as the ratio between true positive and the sum between true positive and false positive results.

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

In this project it has been used the Keras Precision class passing the ground truths and prediction made on the test set.

4.1.2 Recall

The recall describes the completeness of our positive predictions relative to the ground truth. Recall is calculated as the ratio between true positive and the sum between true positive and false negative results.

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

Also in this case it has been used the Keras Recall class passing the ground truths and the predictions on the test set.

4.1.3 Accuracy

The accuracy describes the correctness of our predictions. Accuracy is calculated as the ratio between true predictions and the overall number of predictions.

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

Also in this case it has been used the Keras Accuracy class passing the ground truths and the predictions on the test set.

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.

	Test		
	Accuracy	**Precision**	**Recall**
LeNet-5	0.69	0.73	0.69
Modified LeNet-5	0.73	0.76	0.73

Table 4.1: Test results of every model

4.3 Comparison after quantization

Now we investigate the difference between the performance of the models with and without the quantization.

Dataset	**Model**	**Size (in kB)**	**Accuracy(%)**
Main dataset	not quantized lenet5	263.4	57.53
	quantized lenet5	74.92	58.06
	not quantized modified lenet5	317.45	47.29
	quantized modified lenet5	84.88	46.77
Handcrafted dataset	not quantized lenet5	263.4	87.9
	quantized lenet5	269.724	86.29
	not quantized modified lenet5	317.45	90.32
	quantized modified lenet5	84.88	90.32
Concatenation of both	not quantized lenet5	263.4	68.95
	quantized lenet5	269.724	68.95
	not quantized modified lenet5	317.45	72.58
	quantized modified lenet5	84.88	72.58

Table 4.2: model comparison

4.4 Analysis of the results

From the results that we have shown before we can do some analysis that we divide in three main points:

- **Quantized models performance are comparable with standard models one**: as we can see from the tables, we can appreciate that the quantized models, in our case, do not underperform with respect to the standard models. The underperformances, if present, are relatively small.
- **Models work better with extended datasets**: it can also be appreciated that both the models obtain better results on the dataset composed by the merge between handmade and main

datasets. The results shown really good performances in the handmade dataset but these results could be affected by overfitting due to the limited number of samples in the dataset.

- **Modified version of LeNet5 provides better results than the original one**

5 Conclusions and future works

From the results it is possible to see that the **modified LeNet-5** is the model that gives the best results with respect to the other one. Even if in their compressed versions the comparison of the results see the model **modified LeNet-5** overperform the other one. In order to improve the work that has been done in this project the first step could be do a further research in the literature in order to find some models that could be quantized and inserted inside the device. Of course most of the nowadays models requires machines with more computational capabilities than the ones that were used for this project. Moreover, it can be a good idea also to use different metrics in order to analyze in a more detailed way the results obtained by the various used models.

5.1 Future tasks

In order to use this device in a useful way we thought that a possible extension of this project could be to implement a communication system between the device and another one with more computational capabilities.In this way create a system where the device can collect an entire word through one-by-one character predictions. Then these words can be sent to the other device that will predict using another neural network model some parts, like the next word to insert, or the entire sentence that the user wants to write.

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] T. E. de Campos, B. R. Babu, and M. Varma. "Character recognition in natural images". In: *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*. 2009.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

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

[5] 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.

[6] 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.