Digits Classification using IoT and Edge AI

Zain Amir, Zaman
Department of Electrical and Computer Engineering
Technical University of Munich
Munich, Germany
zainamir.zaman@tum.de

***Abstract*—This paper presents a project in which a resource-constrained device was used to run a convolutional neural network for classification of digits. A custom dataset was created to train the model, which achieved a validation accuracy of 94.94%. When tested on the original MNIST training and test sets, the accuracy was 74.12% and 74.76% respectively. The model was deployed on an ESP32-CAM to perform real-time inference (< 0.11 seconds) on images captured by an attached camera. IoT functionality was also incorporated to allow the device to communicate with a NODE-RED server over WiFi using the MQTT protocol over secure WebSockets.**

***Index Terms*—Digits classification, Artificial Intelligence of Things (AIoT), MQTT.**

I. Introduction

The increasing computational power of microcontrollers has made it possible to run machine learning models on inexpensive devices. Since many development boards such as the ESP32 and the Arduino Nano 33 BLE come equipped with WiFi capabilities, it is now possible than ever to combine the benefits of on-board AI inference and IoT technologies. Running AI on the IoT device itself means that the IoT network does not need high bandwidth to transmit input data to the server for computation such as images and videos that have high memory occupancy. Also, server computers tend to have high processing capabilities, making them ideal for running complex machine learning models. While this generally results in better accuracy, these computers require huge amounts of energy. About 80% [1] of the world's energy is produced from fossil fuels. Since microcontrollers require only a small fraction of the energy needed by server computers, AI on the edge can reduce energy consumption and, hence, carbon emissions for cleaner environment.

II. Dataset creation

A. Preparation and preprocessing

For training the model, a dataset of images of handwritten digits from zero to nine was collected by 30 people. Each person wrote a digit ten times each, amounting to 3,000 images. Each person was responsible for preprocessing the dataset according to the way the original MNIST dataset was processed [2]. Fig. 1 shows raw and processed samples from my contribution to the collective dataset. OpenCV [3] in Python was used to perform the MNIST-style normalisation depicted in the figure, which can be broken down into the following steps:

1) Apply a thresholding function to remove the background of the digit,
2) crop the image to extract the pixels containing the digit,
3) resize the cropped digit to scale the longest dimension of the digit to 20 pixels, and finally,
4) find the centroid of the image and use it to position the digit at the centre of a 28x28 canvas.

However, it was observed that not everyone processed the images according to the above specifications. Therefore, a second preprocessing function was necessary to standardise the collective dataset. Thresholding was re-applied to remove artifacts with low colour intensities that were observed in some samples. To smoothen the edges of the digits, the images were upsized to 30x30 pixels and then downsized back to 28x28 to allow the interpolation algorithm to re-introduce smooth edges.

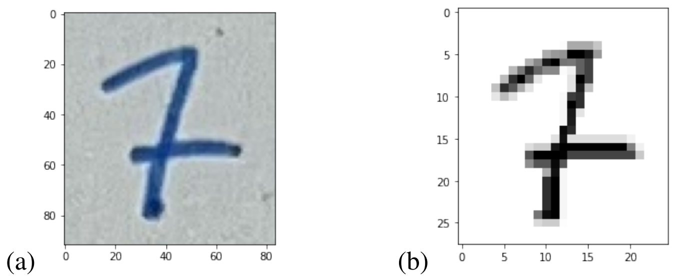

Fig. 1. (a) Raw sample from the dataset captured using a camera, (b) after both stages of MNIST-style normalisation.

B. Augmentation

A dataset consisting of only 3,000 images was insufficient to achieve reasonable accuracy from a deep learning neural network. Therefore, the existing samples were replicated and processed to produce 'new' samples that could be added to the dataset. An algorithm was written to perform random rotation from -15° to +15° and random increase in contrast on the replicated image to increase the dataset size and diversity.

III. Machine learning models

Three models were tested before deployment on the edge device.

A. Support Vector Machine (SVM)

SciKit-Learn was used to train an SVM classifier. 20,000 training samples and 4,000 validation samples were provided to the model.

B. Neural Network

A neural network with one hidden layer consisting of 20 activation units and a softmax output layer of 10 activation units was designed using Google's Tensorflow/Keras library. The size of the training samples was 47,984 and that of the validation set was 4,799 (10% validation-train split). The number of activation units in the hidden layer were deliberately kept low to keep the number of parameters at 15,910 so that the model could be deployed on the edge device if necessary.

C. Convolutional Neural Network

A convolutional neural network is similar to an ordinary neural network except that it has a convolutional layer at its input. The convolutional layer acts as an automatic feature extractor, which is why it is specialised for image recognition tasks such as digits classification. The model is shown in Fig. 2. The number of units in the convolutional layer and hidden layer were chosen to keep the number of parameters as low as possible. The first Conv2D layer performs the convolution operation which is responsible for feature extraction. The max pool layer downsizes the images produced by the convolution operation (in order to make the model computationally feasible). The next fully connected layer learns from the features generated by the first two layers. The dropout layer performs regularisation to prevent overfitting. The final layer uses the regularised outputs of the previous layer to generate the probabilities that the image belongs to each class. The class with the highest probability is chosen to be the prediction of the model. The same number of training and validation samples were used as in the case of training the neural network. To train the model, 10 iterations (epochs) were performed. Further iterations were avoided as it was noted that the test accuracy suffered as the model possibly began to overfit the training data.

IV. Results

The results of the training phase are shown in Table I. Alongside the validation accuracy, the models were tested on the training and test samples of the original MNIST dataset to check the models generalised or not. It is clear that the convolutional neural network was the best choice. The SVM peformed badly as it was clearly underfitting. The neural network showed respectable validation accuracy, but it failed to generalise to the MNIST dataset.

The convolutional neural network showed a validation accuracy of 94.6%, with test accuracy of around 74%. While the test accuracy could had been increased further by increasing the number of units in the convolutional and hidden layers, this would not have been possible without drastically increasing the number of parameters and, hence, the risk of a memory overflow in the edge device. In addition, Fig. 3 shows the confusion matrices for the model when tested on the training and test samples of the MNIST dataset. The digits zero, one, three, four, five, and six are classified with great accuracy. It also shows some shortcomings as the model tends to be confused between samples of ones and sevens and samples of sevens and nines.

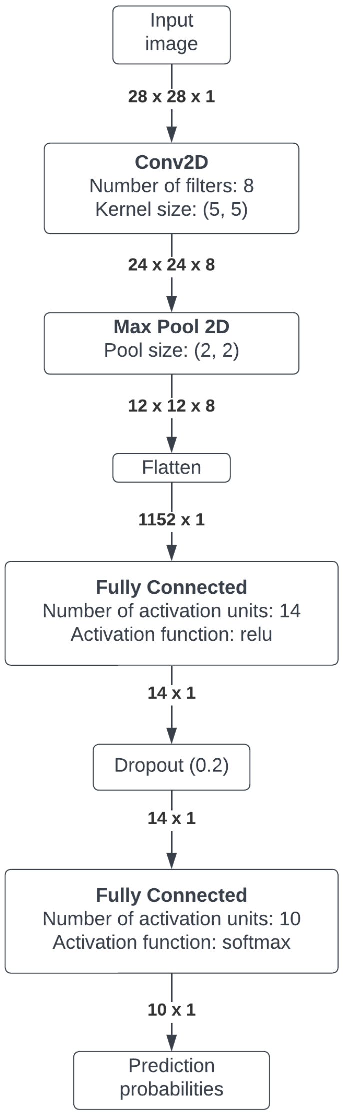

Fig. 2. Layers in the convolutional neural network.

V. Deployment on edge device

A. Hardware setup

An ESP32-CAM as seen in Fig. 4(a) was chosen as the edge device for this project as the board features a built-in WiFi module and an attachable OV2640 camera. Fig. 4(b) shows the setup used to test the system. Digits were drawn on the tablet device and the ESP32-CAM was positioned directly in front to capture images with the camera.

B. Camera capture techniques

It was necessary to use efficient capture techniques to keep the memory footprint as low as possible as the device needed

TABLE I
ACCURACY RESULTS

Model	Validation accuracy	Test accuracy on MNIST training samples (60,000)	Test accuracy on MNIST test samples (10,000)
SVM	67.25%	44.29%	44.77%
NN	87.21%	61.56%	62.44%
CNN	94.94%	74.12%	74.76%

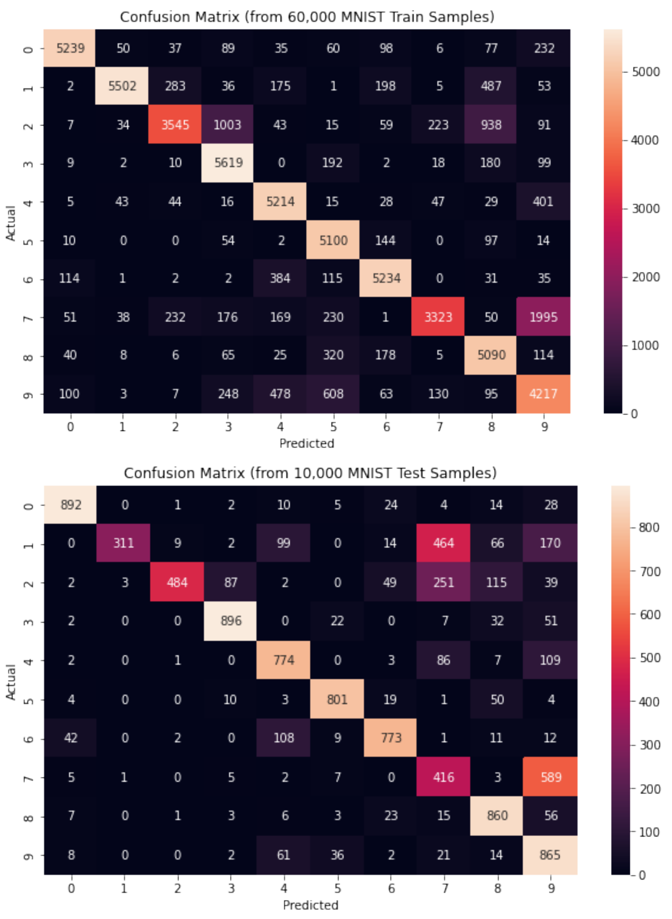

Fig. 3. Confusion matrices for the CNN model tested on the MNIST dataset.

memory for performing AI inference. To achieve this, the resolution of the camera was set to 96x96 pixels. Pixels were stored in grayscale format as colour was not necessary in classifying digits and it reduced the number of colour channels from three to one.

The raw capture had to preprocessed before feeding it to the model. As image processing libraries were not available for the ESP32-CAM, algorithms were written to achieve this. Firstly, a thresholding function was applied to get rid of the background and to extract dark foreground elements (the digits in this case). Next, the 96x96 capture had to be downsized to 28x28 as the CNN model accepted an input of $28 \times 28 = 784$ features. To keep the processing complexity low, a simple algorithm was designed. Every third pixel from the raw image was extracted to perform downsizing (as $\text{int}(96/28) = \text{int}(3.43) = 3$). Furthermore, it was observed that the inference performed poorly unless the digit was centered in the image. Therefore, an algorithm was created to center the digit inside the 28x28 image. The dimensions of the digit were found and used to fill the central part of the image.

C. On-device inference

The CNN model trained in TensorFlow had to be exported as a C array to be used for inference on the ESP32-CAM. Tinymlgen [4] was used to generate the array containing the weights of the trained model. The generated array had 68,552 weights.

Next, a custom library (EloquentTinyML [5]) was used to simplify the deployment of TensorFlow Lite on the ESP32-CAM. A header file was created to store the weights array and the library was initialised with the weights to begin inference. There was no need to specify which TensorFlow operations were required to perform inference, which is the case when TensorFlow Lite is directly used. The average inference time was 0.109 seconds (calculated from 10 inferences).

D. Communication with NODE-RED server over WiFi and MQTT

NODE-RED [6] is an open-source flow-based visual programming tool developed by IBM for setting up IoT communication. NODE-RED was used to control the ESP32-CAM via MQTT. The Aedes [7] plug-in was used to create a broker in a NODE-RED flow. A message injection node sent the string ”Recognize” to a local publisher node which published

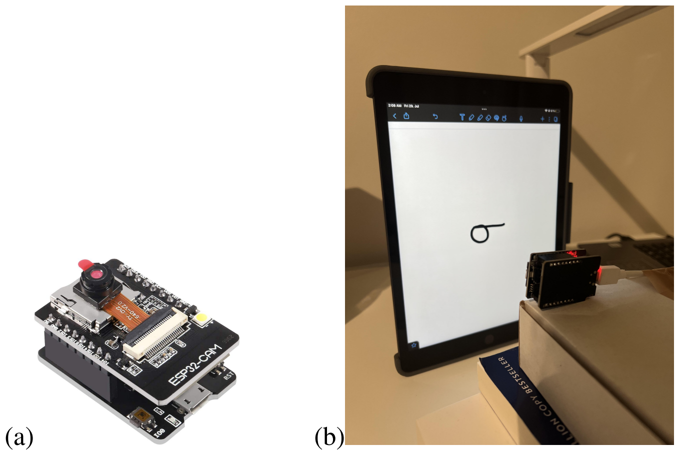

Fig. 4. (a) ESP32-CAM device with camera module and built-in WiFi, (b) Experimental setup.

the message to the broker over the topic ”COMMAND”. Upon receiving the string ”Recognize” from the broker, the ESP32-CAM took a picture, preprocessed it, and performed inference to generate the prediction result and probability of the predicted class. The two values were encoded in a JSON string and sent to the broker over the topic ”RESULT”. A local subscriber node subscribed over the topic ”RESULT” received JSON string and parsed it to get the prediction result and probability.

Furthermore, WebSockets were used as the medium of communication as they enable full-duplex communication, which can reduce traffic through a single communication channel. As they use the TCP protocol, WebSockets also allow for reliable communication. To ensure secure communication, SSL security was used.

VI. Conclusion

In this project, an end-to-end AIoT solution was successfully built to identify digits through on-device machine learning inference. It was demonstrated that combining by AI and IoT technologies, the entire IoT system gained numerous advantages. Firstly, the system was able to cut down on network bandwidth utilisation. Without on-board AI, the AI processing would have to be carried out at the server. This would mean that the ESP32-CAM would have to send a picture to the server for each classification, thereby increasing the data transfer requirements of the network. But with on-board inference, the device only sent the results of the classification which consisted of only two strings.

A major challenge of AIoT is the deployment of machine learning models onto resource-constrained devices. With limited memory and compute power, it is difficult to perform inference without significantly reducing the number of parameters in the model that tends to deteriorates performance. In this project, the number of activation units in the fully connected layer and the number of filters in the convolutional layer of the CNN had to be reduced until the exported model could fit into the memory of the device. This resulted in a model that was underfitting the training data. Another issue was the extensive amounts of memory management that was needed to prevent a memory overflow. Further work needs to be done to efficiently allow quantisation of AI models with minimal performance loss.

References

[1] EESI (Environmental and Energy Study Institute): https://www.eesi.org/topics/fossil-fuels/description#:~:text=Overview, percent%20of%20the%20world’s%20energy.

[2] MNIST dataset by Yann LeCun, Corinna Cortes, Christopher J.C. Burges: http://yann.lecun.com/exdb/mnist/

[3] OpenCV: https://opencv.org/

[4] Tinymlgen: https://github.com/eloquentarduino/tinymlgen

[5] EloquentTinyML library: https://github.com/eloquentarduino/EloquentTinyML

[6] NODE-RED by IBM: https://nodered.org/

[7] Aedes MQTT broker: https://flows.nodered.org/node/node-red-contrib-aedes