PenToPrint with Arduino

Team Members: Yuyang Jin, Vishnu Pradheep Raveendran, Sofia Boselli Graf

Overview of the Project

The primary objective of this project is to implement handwriting text recognition on the Arduino platform using a deep neural network. However, certain components of the network, such as the Convolutional Neural Network, demand significant computational power, posing a challenge for the hardware's capabilities. To address this limitation, we initially conducted pre-training using the IAM dataset on more powerful computers. Subsequently, we transitioned to the Arduino for the second phase of training, focusing on the dense layers. To enhance the model's performance, we incorporated L2-regularization, dropout techniques, and image pre-processing. The efficacy of the trained model was evaluated using a set of forty test images, resulting in a final accuracy of 0.77.

Plan, Tasks, and Responsibilities

The workload is shared in the team equally. The main subtasks are listed below.

I. Task 1: Data Collection

During the initial training phase, certain considerations, such as computational power and dataset size, are not critical, as the training takes place on a computer. A widely adopted option for this phase is the IAM dataset, comprising 115,320 words scribed by 657 distinct authors. A selection of samples is presented below (Fig. 1).

Fig. 1: Example of the IAM dataset

For on-device training, we utilized a self-generated dataset comprising 490 words authored by group members. The deliberate choice to keep the dataset relatively modest was made in consideration of the hardware limitations. A selection of sample entries is provided below (Fig. 2).

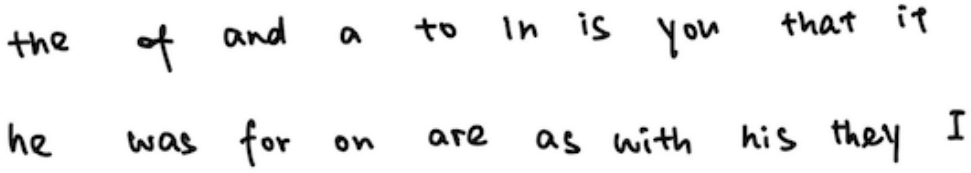

Fig. 2: Self-created dataset

II. Task 3: Data Pre-processing

As depicted earlier, the dataset contains images with varying sizes. To standardize the input shape, we developed a padding algorithm that adjusts the width to 32 pixels and the height to 128 pixels, ensuring uniformity. It is implemented on both datasets and the test images. Some images after padding is presented below (Fig. 3)

Fig. 3: Post-Processed images

Furthermore, it is noteworthy that during application testing, the input images typically encompass entire sentences or multiple words. Consequently, it becomes essential to crop all the words before applying padding. This procedure is imperative for both the self-generated dataset and the test images. There is no need to clip the images for the IAM dataset since all the words in it are isolated.

III. Task 4: Training On PC

A neural network based on CNN and Fully-Connected was created to obtain a feature vector for each input image (which will be used in the Arduino for final training and prediction). The model is composed of 3 2D-convolutional layers with Max-Pool and 3 Dense final layers (The final layer has 20 nodes which represent the 20 classes used for training this network, the feature vector is taken from the dense layer before this one) . The exact composition can be observed in Fig. 4. The Network is trained with every available image in the defined 20 classes divided into Training and Validation. An accuracy of 97% is obtained on the training dataset and 89% in the Validation after 10 epochs.

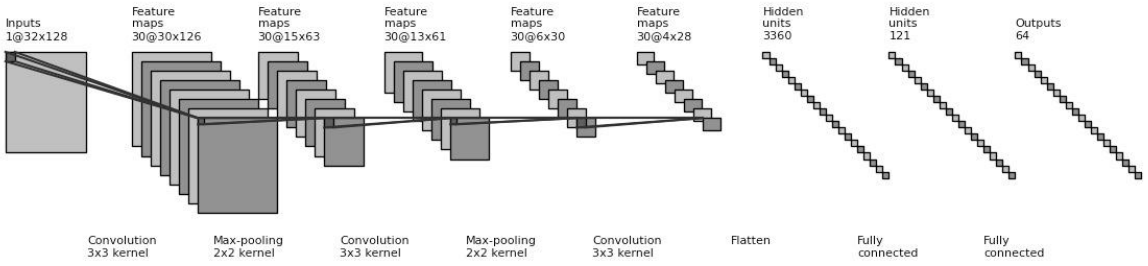

Fig. 4: Composition of PC Neural Network

IV. Task 2: On device training

The model, developed using images from the IAM dataset, is trained employing convolutional neural networks (CNN) and dense layers. After this training phase, it is tested with a separate dataset of handwritten images, generating predicted outputs. These outputs are then used as a training dataset for an Arduino. This Arduino training involves a dense layer Neural Network, written in C, featuring one hidden layer with 30 nodes. The training process on the Arduino includes forward propagation, backpropagation, and the use of regularization techniques such as dropout and L2 regularization. Additionally, the network is configured to train for 20 classes.

Dropout:

In the dropout method, we randomly deactivate hidden nodes during training. A new random set of nodes is selected for every pattern during the backpropagation procedure. The dropout method prevents hidden nodes from collaborating excessively, which focuses on details in the training dataset (overfitting). By using dropout, we encourage the network not to rely too heavily on individual neurons. The dropout method is implemented in the forward propagation function.

L2 Regularization:

This involves the smoothing of the output function, allowing it to have curvature and preventing wild oscillations (stops wild oscillations). The formula

$$L(\tilde{\omega}, D) = E(\tilde{\omega}, D) + \lambda/2 \sum_i \tilde{\omega}^2_i$$

represents this concept. Here, $\lambda/2 \sum_i \tilde{\omega}^2_i$ is the L2 regularization term, λ is the regularization strength, $E(\tilde{\omega}, D)$ is the cross entropy error and $\tilde{\omega}$ represents the weights. In L2 regularization, the term added to the loss function is proportional to the square of the weights. During backpropagation in a neural network, L2 regularization adjusts the weights by subtracting a term proportional to the current weight value itself. This occurs because the derivative of the square of the weights with respect to the weights (which is used in gradient descent) is linear in the weights.

V. Task 5: Evaluation and Results

We reserved 50 words in the self-generated dataset for the test set. Upon conducting on-device training for 11 epochs, the test accuracy reached 0.77. Attempts to further increase the training epochs resulted in a decline in both validation accuracy and test accuracy, with noticeable fluctuations, indicative of potential overfitting. Consequently, we implemented early stopping at the 11th epoch. The image below illustrates the training accuracy, validation accuracy, and test accuracy. Additionally, we provide printed predictions alongside true labels to enhance the clarity of the experimental results.

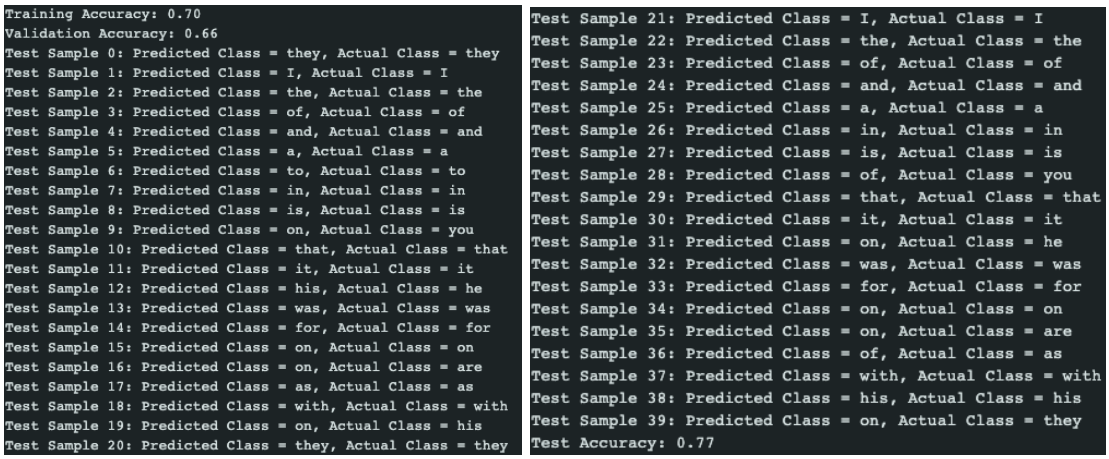

Fig. 5 Evaluation and results of the trained model

Deviations from the Initial Plan

From the original plan, the following things were changed.

Firstly, the work established to obtain a grade 5 was not done due to lack of time.
Secondly, the camera module of the arduino was not used in the final iteration of the project. Some pictures were taken and the results were very bad, the color was distorted and the quality was very poor. As a replacement, the written text was written on an ipad and the “images” are screen shots from the device which would be the same as taking pictures from a high-quality camera.

Finally, the project idea was to identify at least a hundred different words. The project started in this way but when the results were passed into the arduino a big amount of data needed to be used and this caused the arduino to freeze. This data corresponded to the amount of features (which needed to be greater than 100), the size of the hidden layer (again, greater than 100 nodes) and the amount of data points (training, validation and testing) which needed to be big enough to have at least a few samples of each word. The Arduino freezes during the forward propagation phase, before it even enters the backpropagation stage for training. Therefore, even though we have implemented weight pruning, it couldn't prevent freezing, as weight pruning occurs during training. Instead of weight pruning, we implemented dropout and L2 regularization, which increased the accuracy. When the arduino froze it was difficult to “revive it” which included unplugging the arduino, waiting and changing USB ports several times before it was able to receive a new code upload. For this reason the 100 classes were lowered to 20 classes which correspond to the 20 most-used words in the English language.

Links to the Final Outcome

- Video Demo: https://youtu.be/vcx4aTHGlco
- Git Repository: https://github.com/batudrsnn/HandWritingDetection.git