Braille Detection and Translation Project: First-Term Highlights

Ali Elneklawy, Nada Gamal El-Dien, Nada Tarek

February 2024

1 Introduction

The first week started with an introduction to the project. Several methods to approach the project were introduced, including deep learning techniques such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), as well as image processing techniques.

The team was split into two groups; one group approaching the project using deep learning techniques and the other group using image processing techniques. We decided to go with the deep learning techniques using CNNs for image recognition and RNNs for voice generation.

2 Dataset

The dataset we used was found on kaggle, a popular website used in data science and machine learning [here].

- The dataset contains images for the 26 characters.
- Each character has 60 examples.
- The images are man-made not scanned.

(a) Letter B (b) Letter M

Figure 1: Example images from the dataset

- Each image is a 28x28 image in BW Scale.
- Images are noisy.
- Data augmentation process includes whs - width height shift, rot - Rotation, dim - brightness).

The Braille images are preprocessed to ensure consistent size and normalization. The images were first resized to 224 * 224 to be consistent with the model requirements. The images were then normalized by dividing the entire image array by 255.0. Two examples from the dataset is shown in Fig.1.

3 Models

We experimented with various pretrained model architectures, including *VGG16* and *DenseNet201*. The primary criterion for assessing the models was accuracy, given the balanced nature of the dataset. Precision and recall metrics were employed for the final model evaluation. The *DenseNet201* model outperformed the VGG16 model, achieving 95% accuracy compared to the latter's 75%. Consequently, we opted

to proceed with the *DenseNet201* model for additional analysis. Additionally, a custom CNN model was employed, demonstrating a comparable accuracy of 95% to that of *DenseNet201*.

3.1 DenseNet201 Model

The proposed model employs the DenseNet201 architecture, a convolutional neural network (CNN) known for its parameter efficiency and ability to handle complex tasks. The original DenseNet201 consists of 201 layers, with the final layer responsible for classification. For this project, the final layer is removed, and only the last 30 layers are trainable. This approach allows the model to leverage the pre-learned features from the original DenseNet201 while adapting to the specific task of Braille translation.

Since we had prior knowledge that the images we are going to scan will not be rotated, we trained two DenseNet201 models. The first model was trained on the entire dataset including the rotated images [Model A]. We then removed the rotated images from the dataset and trained the model again [Model B]. Interestingly, model 'B' exhibited better performance than model 'A'.

3.1.1 Model A

We first made the last 30 layers trainable so that the model adapts to the new dataset as follows,

```
1
2 base_model = DenseNet201(weights='imagenet', include_top=False,
      input_shape=(224, 224, 3))
3
4 for layer in base_model.layers:
5     layer.trainable = False
6
```

```
for layer in base_model.layers[-30:]:
    layer.trainable = True
```

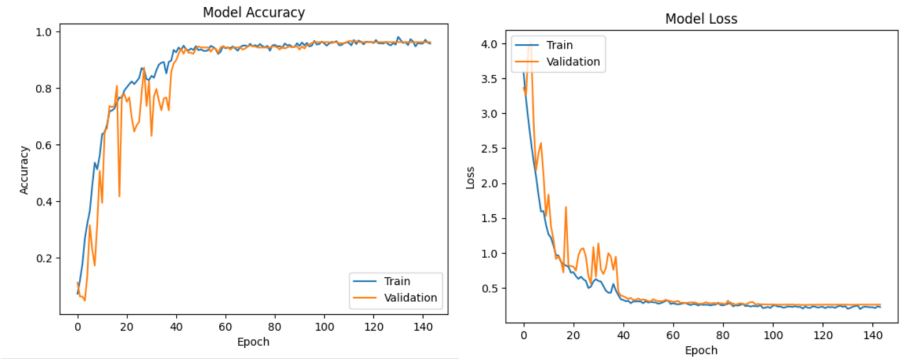
The model was then trained on the training set and evaluated on the test set. Table 1 shows the classification report of model 'A' on the test set. Overall accuracy is 95%.

Class	Precision	Recall	F1-Score	Support
A	0.90	1.00	0.95	9
B	1.00	1.00	1.00	9
C	1.00	0.78	0.88	9
D	0.90	1.00	0.95	9
E	0.89	0.89	0.89	9
F	0.82	1.00	0.90	9
G	1.00	1.00	1.00	9
H	1.00	0.89	0.94	9
I	0.80	0.89	0.84	9
J	1.00	1.00	1.00	9
K	0.90	1.00	0.95	9
L	0.90	1.00	0.95	9
M	1.00	1.00	1.00	9
N	1.00	1.00	1.00	9
O	1.00	0.89	0.94	9
P	1.00	0.89	0.94	9
Q	1.00	1.00	1.00	9
R	1.00	0.78	0.88	9
S	1.00	1.00	1.00	9
T	0.89	0.89	0.89	9
U	1.00	0.89	0.94	9
V	0.90	1.00	0.95	9
W	1.00	0.89	0.94	9
X	1.00	1.00	1.00	9
Y	1.00	1.00	1.00	9
Z	0.90	1.00	0.95	9

Table 1: Classification Metrics

Model loss and model accuracy are shown in Fig.2.

The confusion matrix shown in Fig.3 shows exactly which letters the model has mistaken. Most of the mistakes that the model made were on the rotated images where the rotations has confused the letters with each other. For example, Fig.4



(a) Model accuracy (b) Model loss

Figure 2: Model loss and model accuracy

shows the letters 'B' and 'I'. Letter 'I' can be easily confused with the rotated version of letter 'B'.

3.1.2 Model B

For this model, we removed all the rotated images from the dataset. We then performed data augmentation to increase the number of the images to prevent overfitting. This model achieved a significaltly better performance with 99.8% train accuracy and 98.3% test accuracy.

4 CNN Model

This model is a convolutional neural network (CNN) designed using the Keras library with a sequential architecture. Here is a breakdown of its structure:

Convolutional Layers:

- The first convolutional layer has 64 filters, a kernel size of (5, 5), 'same' padding, ReLU activation, and L2 reg-

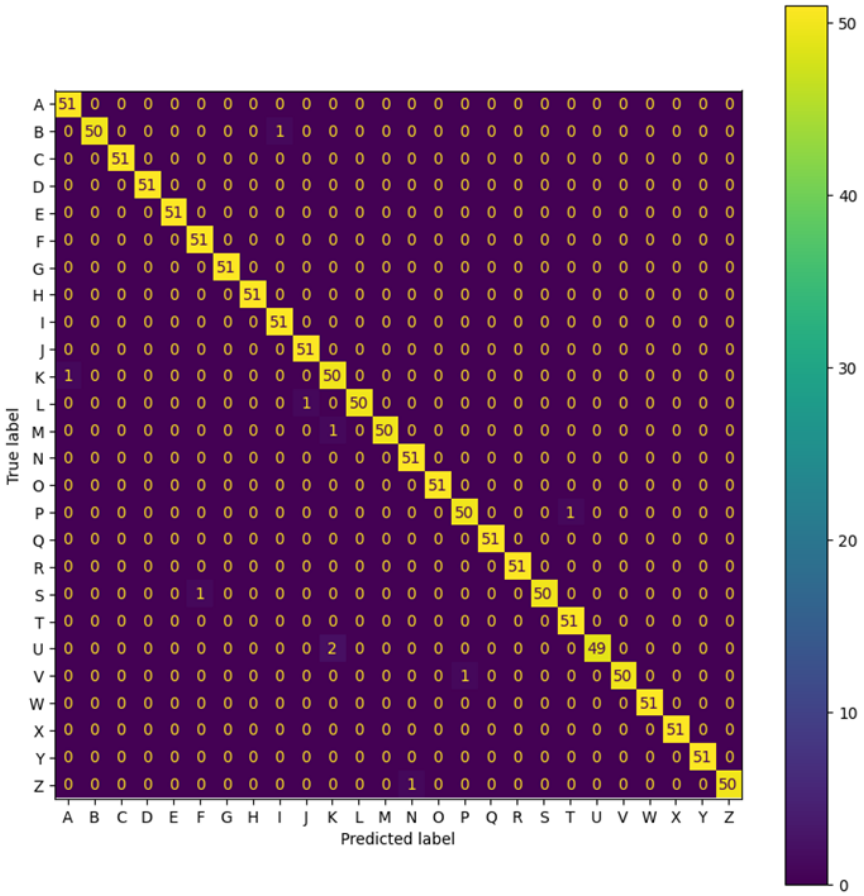


Figure 3: Confusion matrix

(a) Letter B (b) Letter I

Figure 4: Letter 'I' can be confused with the rotated version of letter 'B'

ularization with a regularization parameter of 0.0015.

- Batch normalization follows the first convolutional layer.
- The second convolutional layer has 64 filters, a kernel size of (3, 3), 'same' padding, ReLU activation, and L2 regularization with a regularization parameter of 0.0015.
- Batch normalization follows the second convolutional layer.
- Max pooling is applied after the second convolutional layer.

Additional Convolutional Layers and Pooling:

- Two more pairs of convolutional layers followed by batch normalization and max pooling. The third pair uses 64 filters, a kernel size of (3, 3), 'same' padding, ReLU activation, and L2 regularization with a regularization parameter of 0.0013.

Flattening and Dense Layers:

- The model then flattens the output and passes it through a dense layer with 576 units, ReLU activation, and L2 regularization with a regularization parameter of 0.0011.
- Batch normalization follows the dense layer, and dropout with a rate of 0.6 is applied.

Additional Dense Layers:

- Another dense layer with 288 units, ReLU activation, and L2 regularization with a regularization parameter of 0.0011.
- Batch normalization follows the second dense layer, and dropout with a rate of 0.4 is applied.

Output Layer:

- The final dense layer has 26 units with softmax activation, representing the number of classes in the output.

This architecture incorporates convolutional layers for feature extraction, max pooling for down-sampling, and dense layers for classification. Batch normalization helps stabilize and accelerate training, while dropout is employed for regularization, reducing overfitting during training. L2 regularization is applied to the kernel weights in convolutional and dense layers to control model complexity. The model's output is a probability distribution over 26 classes, making it suitable for a multi-class classification task.

5 Other Findings

Discovered models on Roboflow designed for character detection within a page. These models provide character labels along with the corresponding coordinates of their bounding boxes on the page. Unfortunately, direct access to the model code is restricted, but it is feasible to test the models with any provided image. Additionally, identified annotated datasets comprising over 2000 images; however, these datasets exhibit noise and do not meet our desired lighting standards.

```
model = keras.Sequential([
    keras.layers.Conv2D(filters=64, kernel_size=(5, 5), padding='same', activation='relu', kernel_
regularizer=regularizers.l2(0.0015)),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu', kernel_
regularizer=regularizers.l2(0.0015)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu', kernel_
regularizer=regularizers.l2(0.0013)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu', kernel_
regularizer=regularizers.l2(0.0013)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPooling2D(),
    keras.layers.Flatten(),
    keras.layers.Dense(units=576, activation="relu", kernel_regularizer=regularizers.l2(0.0011)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.6),  # Add dropout with a dropout rate of 0.5
    keras.layers.Dense(units=288, activation="relu", kernel_regularizer=regularizers.l2(0.0011)),
    keras.layers.BatchNormalization(),
    keras.layers.Dropout(0.4),  # Add dropout with a dropout rate of 0.5
    keras.layers.Dense(units=26, activation="softmax")  # output layer
])
```

Figure 5: CNN Model Architecture

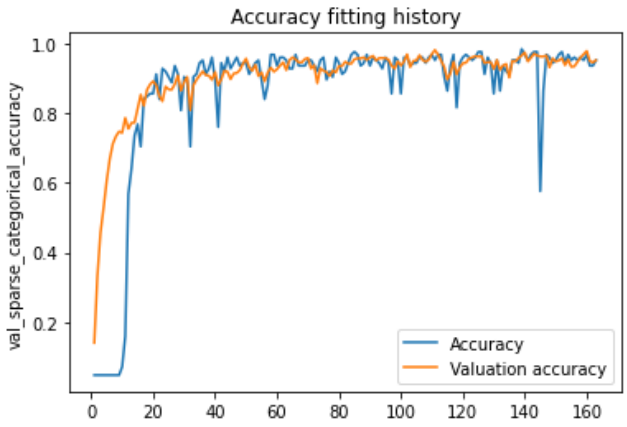


Figure 6: CNN model accuracy fitting history over 163 epochs

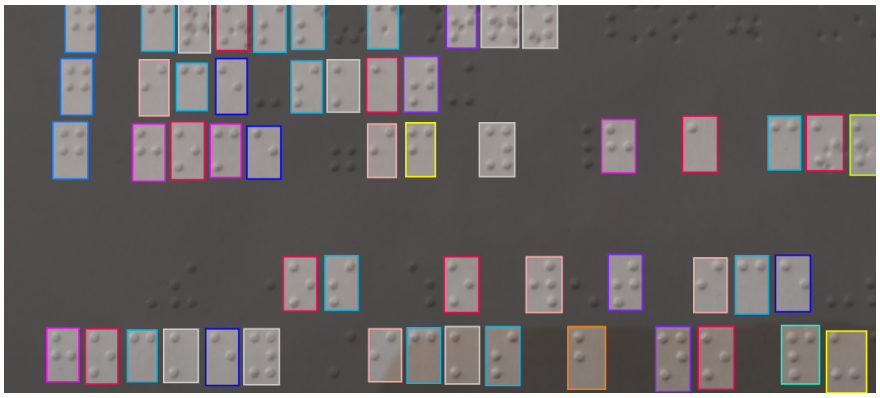


Figure 7: Sample of annotated dataset found on roboflow

Consequently, the accuracy of these models is not dependable.

Upon reflection, it was recognized that creating a custom annotated dataset could be a viable solution. Utilizing an object detection algorithm such as YOLO (You Only Look Once) may lead to a higher level of accuracy and better alignment with our specific requirements.