Convolutional Neural Network: Digit Recognizer

Patrick Redington

Bellevue University

DSC680: Applied Data Science

Professor: Amirfarrokh Iranitalab

Convolutional Neural Network: Digit Recognizer

Introduction

The primary objective of this project is to utilize the "Keras" package to construct a convolutional neural network capable of accurately recognizing handwritten numbers. In today's digitized landscape, the ability to effectively process and identify handwritten digits is crucial across various industries, spanning financial document processing, postal services, and automated form filling. Traditional digit recognition methods often lack accuracy and scalability, particularly when confronted with handwritten characters. Leveraging Convolutional Neural Networks (CNNs) for digit recognition offers a transformative solution to this challenge.

One of my key objectives in undertaking this project is to engage in learning and skill development pertaining to the implementation of CNNs, Keras, and Kaggle Accelerators as well. By immersing myself in this endeavor, I aim to gain practical experience in developing advanced neural network models, thereby enhancing my expertise in machine learning and deep learning technologies. This objective not only aligns with my personal goal of continuous growth but is also a steppingstone in my path to being competitive in the Harmful Brain Activity Classification competition. Where the knowledge of implementing CNNs is vital to being able to handle the exponentially larger datasets and complex images.

Data

The dataset utilized in this project originates from Kaggle and is derived from the MNIST public dataset, a benchmark dataset widely used for handwritten digit recognition tasks. Comprising two CSV files, train.csv and test.csv, these dataset offers a collection of gray-scale images depicting hand-drawn digits ranging from zero through nine. The training dataset contains 42,000 images, while the

testing/submission dataset contains 28,000 images. Each image is represented as a 28x28x1 matrix of pixels, totaling 784 pixels per image. The pixel values range from 0 to 255, with higher values indicating darker shades. A snapshot of the training dataset can be seen in figure 1 below.

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 1: MINST Training Dataset

The data was loaded into the notebook and inspected for null values or corrupted images, which were not found. Subsequently, the images were normalized by dividing the pixel values by 255 to ensure none exceeded one, facilitating faster convergence during training. The datasets were then reshaped to conform to the format of 28x28x1. Following this, the labels were one-hot encoded to render them compatible with the neural network. Figure 2 displays the images and their corresponding labels. The training set was subsequently split into training and validation datasets using a 90/10 ratio.

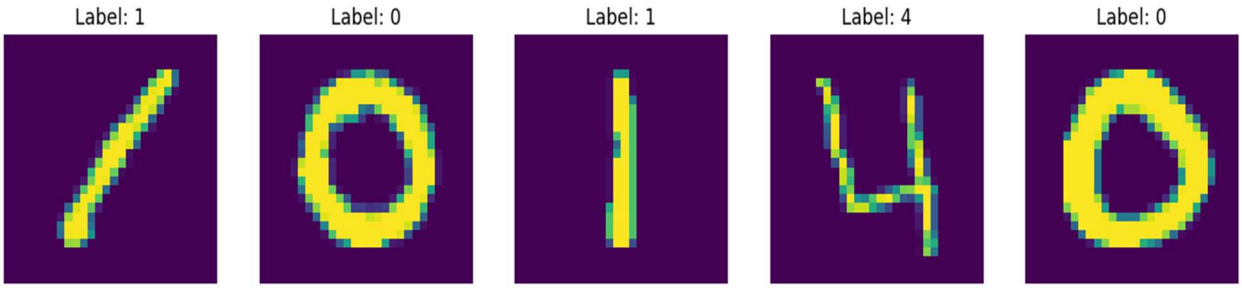

Figure 2: Labels and Images

Data augmentation techniques were then employed to generate additional images for the purpose of training. This was facilitated by Keras's "ImageDataGenerator" which was configured with specific augmentation parameters. These parameters included rotation_range,

zoom_range, width_shift_range, height_shift_range, horizontal_flip, and vertical_flip. By fitting this data generator to the training dataset, it applied these augmentation techniques to create variations of the original images. Enriching the training dataset with diverse examples to enhance the model's robustness, and reducing concerns of overfitting.

Methods

The Convolutional Neural Network (CNN) model was chosen due to its effectiveness in image recognition tasks, particularly for handwritten digit classification. The model architecture consisted of multiple convolutional layers followed by max-pooling layers to extract and down sample features from the input images. Dropout layers were incorporated to prevent overfitting by randomly deactivating neurons during training. The fully connected dense layers at the end of the network were responsible for classification. The model was trained using the Adam optimizer, and the categorical cross-entropy loss function. The training process was expedited by leveraging Kaggle's GPU accelerators, capitalizing on their parallel processing capabilities to significantly reduce training time. This enabled efficient exploration of various model architectures and hyperparameters, accelerating the optimization process.

To optimize parameters and assess the CNN model's performance, accuracy was designated as the primary metric, measuring the proportion of correctly classified images out of the total number of images in the dataset. Additionally, the loss function was monitored to gauge model convergence during training. Visualizations depicting these metrics can be observed in the figure below. To complement the evaluation process, a learning rate reduction

strategy was implemented using “ReduceLROnPlateau”, which dynamically adjusts the learning rate based on the validation accuracy.

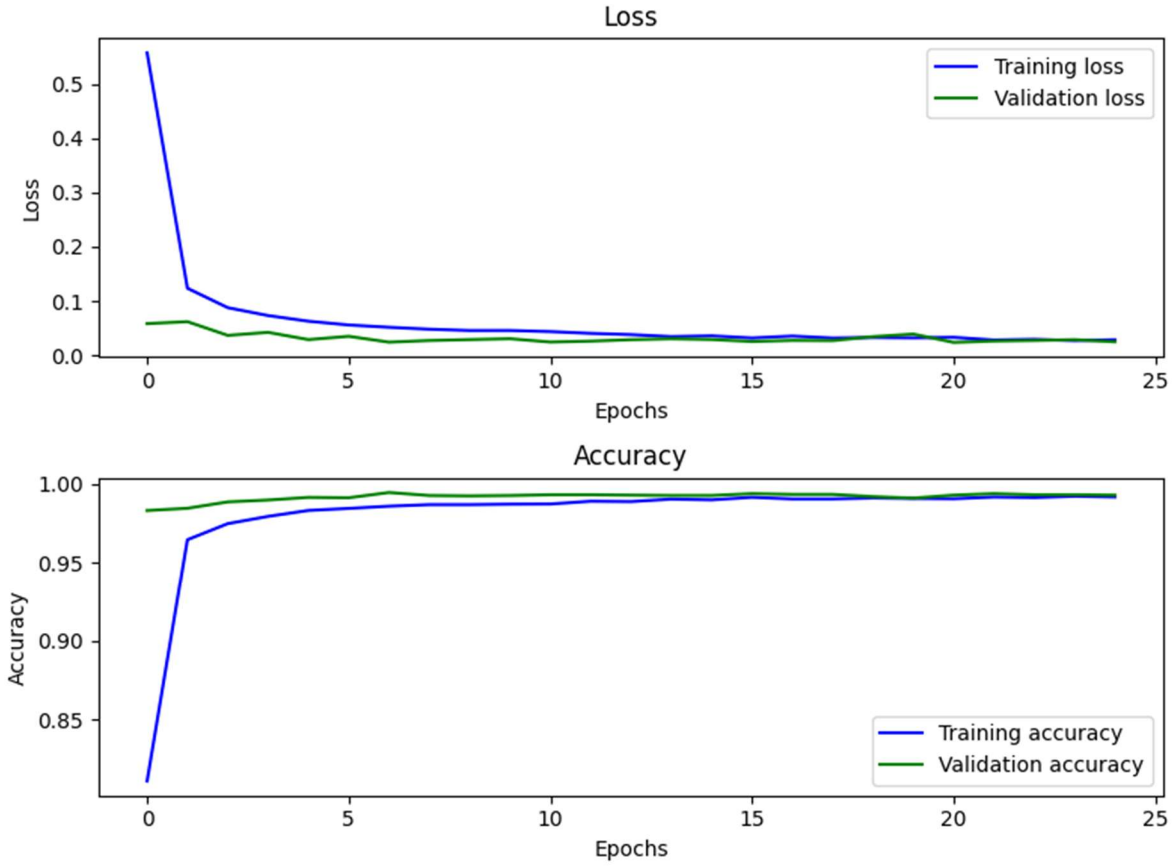

Figure 3: Model Training Accuracy and Loss By Epoch

Analysis and Optimization

Through iterative experimentation, I analyzed various parameters of the model and data generator to optimize the model's training inputs and performance. This involved exploring different data manipulation techniques, including flipping the numbers, adjusting rotation, zoom, and image shifts. Additionally, variations in the training and testing split were tested, with the optimal ratio determined to be 90/10%. These explorations aimed to enhance the model's ability to generalize to unseen data and improve overall performance.

Furthermore, extensive testing was conducted on variations in the number of layers in the CNN and their hyperparameters to identify the ideal architecture for maximizing model performance. A configuration of three layers, each with different numbers of filters and kernel sizes, was determined to be optimal. Finally, the impact of epochs and batch sizes on model performance was examined. These parameters were fine-tuned to strike a balance between training efficiency and model accuracy. The analysis and optimization of these parameters culminated in the development of an effective CNN model for digit recognition tasks.

Conclusion

In conclusion, the development and implementation of the Convolutional Neural Network (CNN) model for digit recognition yielded promising results. With an accuracy score of 99.303% on the test dataset, the model demonstrated its effectiveness in accurately classifying handwritten digits. This performance showcases the potential of deep learning techniques in tackling image recognition tasks with high precision. Additionally, the model's competitive performance, securing 155th place out of 1611 teams in the competition, highlights its ability to deliver meaningful results within a diverse range of approaches. However, further analysis reveals areas for improvement, particularly in correctly identifying the digits 9 and 4, where the model achieves only 99% accuracy shown in the normalized confusion matrix below. Despite these areas of improvement, this endeavor serves as a steppingstone towards harnessing the potential of CNNs and Keras in image recognition.

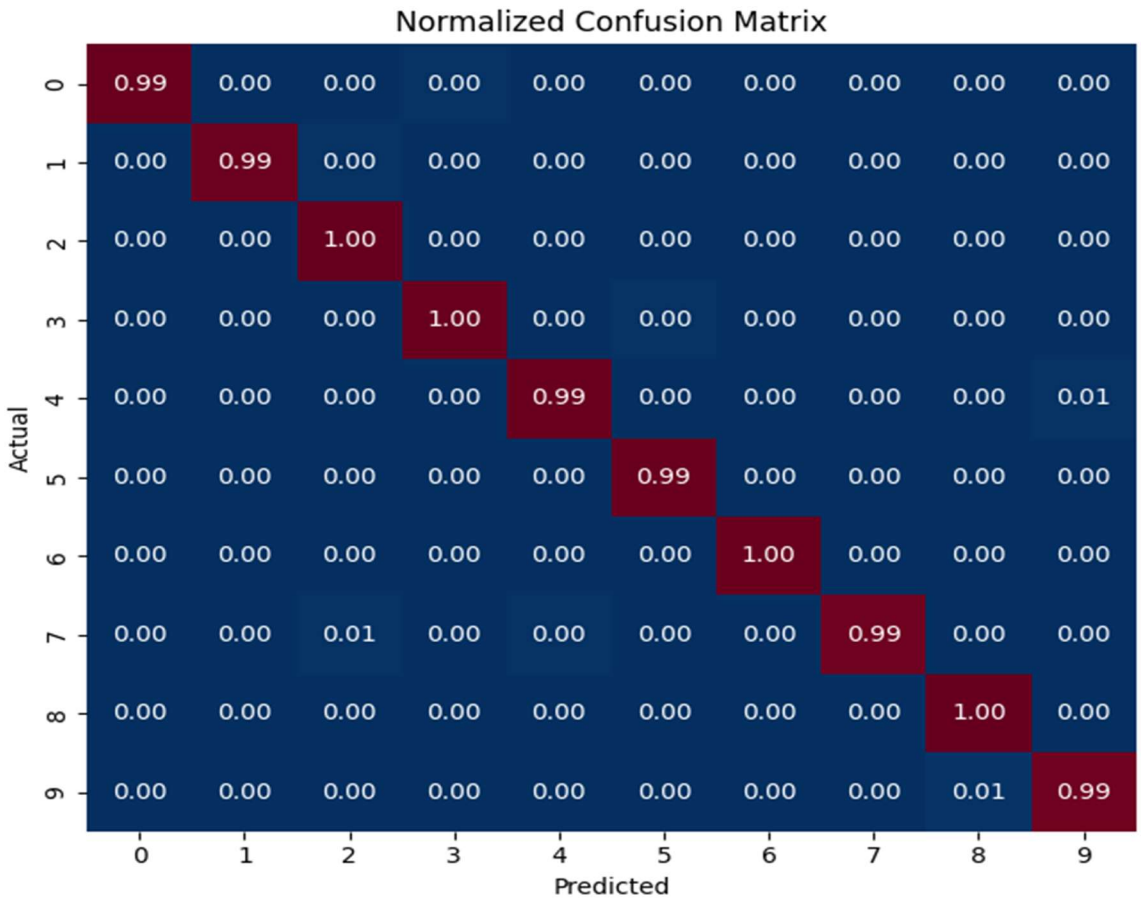

Figure 4: Normalized Confusion Matix of Validation Dataset

For the next steps, I will focus on optimizing data augmentation techniques and fine-tuning model parameters to potentially achieve higher accuracy. This will involve experimenting with various augmentation strategies such as rotation angles, zoom levels, and image shifts, while also adjusting parameters like learning rates and batch sizes to refine the model's performance. However, my main priority is to apply the knowledge I gained about CNNs and Keras from this project to the Harmful Brain Activity Classification competition. With a better understanding of CNNs and how to implement models using Keras, I will be better equipped to tackle the complexity of the HMS competition.

Ethical Considerations

The development and deployment of a digit recognizer model presents few immediate ethical concerns. However, it's essential to ensure the transparency of the model's outcomes. This includes addressing any biases present in the training data or the model itself to prevent disproportionate outcomes. Additionally, measures should be taken to monitor the model's performance post-deployment and address any issues that may arise. By adhering to these ethical principles, we can ensure that the digit recognizer model serves its intended purpose ethically and responsibly.

References

1. Digit recognizer. Kaggle. (n.d.). https://www.kaggle.com/competitions/digit-recognizer
2. Team, K. (n.d.). Keras Documentation: Keras 3 API documentation. https://keras.io/api/
3. Tf.keras.preprocessing.image.imagedatagenerator ： tensorflow V2.15.0.POST1. TensorFlow. (n.d.). https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
4. Wikimedia Foundation. (2024, January 27). MNIST database. Wikipedia. https://en.wikipedia.org/wiki/MNIST_database#:~:text=The%20MNIST%20database%20contains%2060%2C000,taken%20from%20NIST's%20testing%20dataset.

Appendix

A. Data Augmentation Techniques:

Data augmentation was performed using the "ImageDataGenerator" from Keras. The following augmentation techniques were applied:

- Rotation range: 15 degrees
- Zoom range: 0.15
- Width shift range: 0.15
- Height shift range: 0.15
- Horizontal flip: True
- Vertical flip: False

B. Model Training Parameters:

- The model was trained using the following parameters:
- Epochs: 25
- Batch size: 128
- Learning rate reduction: ReduceLROnPlateau with a patience of 3, a factor of 0.5, and a minimum learning rate of 0.00001.

C. CNN Model Architecture:

The Convolutional Neural Network (CNN) model architecture used in this project is outlined below:

```
#Creates the CNN model
model = Sequential()

#Creates "conv2d", "maxpool", and "dropout", and layers
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.15))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.15))

model.add(Conv2D(filters = 128, kernel_size = (6,6),padding = 'Same',
                 activation ='relu'))
model.add(Conv2D(filters = 128, kernel_size = (6,6),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.15))


model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```