Deep Learning for MNIST Digit Classification

Kavya Nagini Madhu Lasya Madineni

Summary

This project dives into the world of deep learning to tackle the challenge of recognizing handwritten digits using the MNIST dataset. We built a convolutional neural network (CNN) using TensorFlow/Keras, kept an eye on our training process with TensorBoard, and fine-tuned our model's performance by implementing early stopping. To enhance generalization, we also utilized dropout techniques and adjusted hyperparameters. Thanks to these efforts, our improved model hit an impressive test accuracy of 99.28%. We made sure to cover all the project essentials, including best practices, monitoring, hyperparameter tuning, and optimizing runtime.

Introduction

Handwritten digit recognition is a key challenge in the world of machine learning, and it has some really useful applications like automating postal services, processing bank checks, and digitizing various forms. The MNIST dataset, which includes 70,000 labeled images of handwritten digits, serves as a classic benchmark for testing deep learning models. This project showcases how modern deep learning techniques—such as model monitoring and optimization—can be used to achieve top-notch digit classification.

Current Research

In 1998, LeCun and his team introduced the LeNet-5 convolutional neural network, which impressively achieved around 99% accuracy on the MNIST dataset, setting a new standard for image recognition with CNNs. Since that groundbreaking moment, we've seen deeper architectures and innovative regularization techniques, like dropout and batch normalization, push accuracy even higher. Nowadays, automated monitoring and hyperparameter tuning have become essential practices for ensuring robust and reproducible results.

Data Collection / Model Development

Dataset

- **Source**: MNIST, accessed via Keras datasets
- **Size**: 60,000 training images, 10,000 test images
- **Image size**: 28 28 pixels, grayscale
- **Classes**: 10 (digits 09)

Baseline Model Architecture

Layer (type)	Output Shape	Parameters
Conv2D 32 filters, 3 3, relu)	(None, 26, 26, 32)	320
MaxPooling2D	(None, 13, 13, 32)	0
Conv2D 64 filters, 3 3, relu)	(None, 11, 11, 64)	18,496
MaxPooling2D	(None, 5, 5, 64)	0
Flatten	(None, 1600)	0
Dense 128, relu)	(None, 128)	204,928
Dense 10, softmax)	(None, 10)	1,290
Total Parameters		**225,034**

Improved Model

- **Dropout layers**: 0.25 after each pooling layer, 0.5 before final dense layer
- **Early stopping**: Stops training when validation loss does not improve for 2 epochs

Analysis

Training and Validation Performance

Model	Best Training Accuracy	Best Val. Accuracy	Test Accuracy	Epochs(Early Stopping)
Baseline CNN	99.91%	99.15%	98.85%	5
Improved(Dropout)	98.57%	99.27%	99.28%	7

Figure1: Example MNIST digits

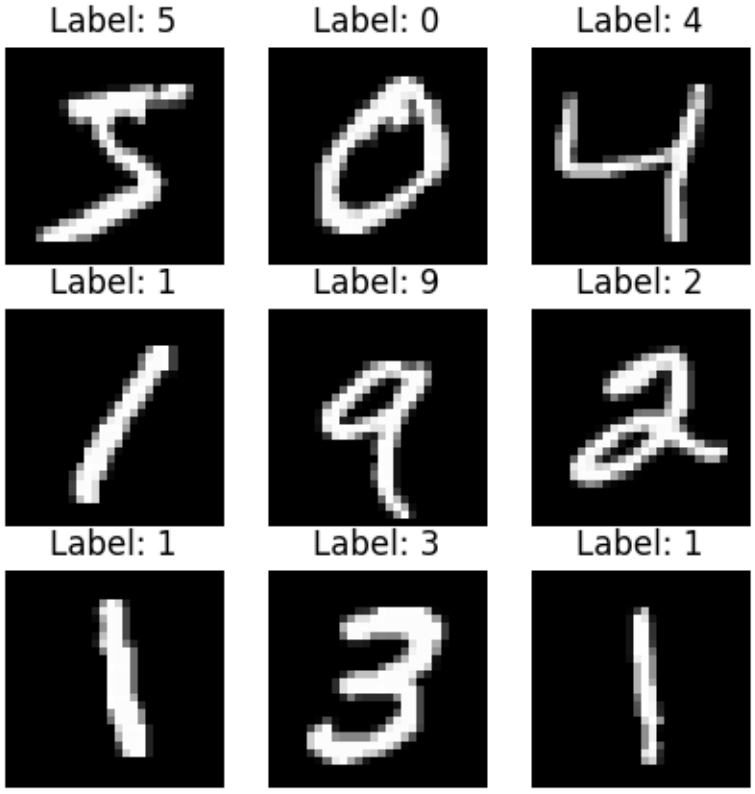
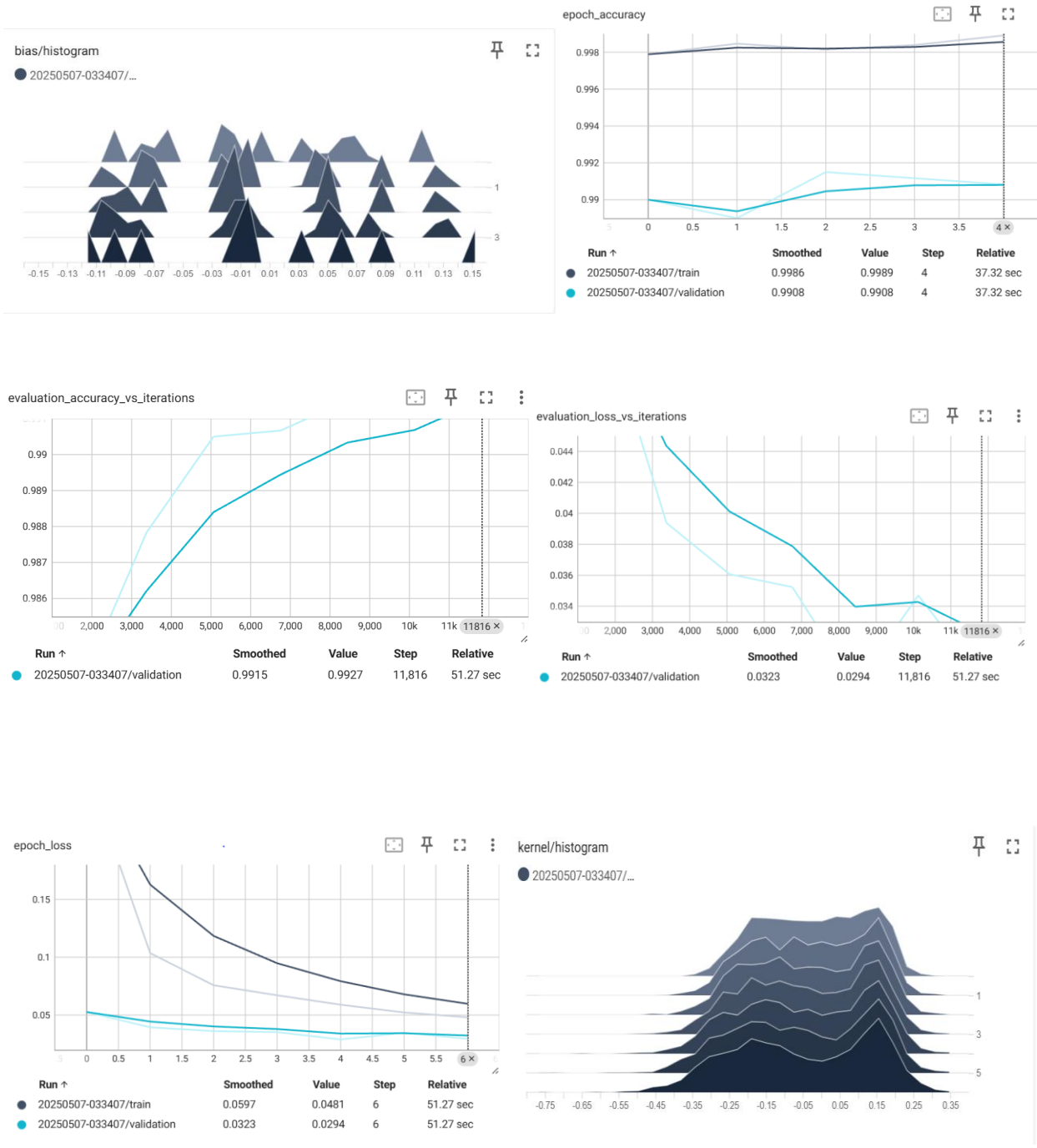


Figure 2: Training and validation accuracy/loss curves from TensorBoard



Observations

- **Early stopping** reduced training epochs from 15 to 7, saving time and preventing overfitting.
- **Dropout** improved generalization, as shown by higher validation and test accuracy.
- **TensorBoard** monitoring enabled real-time tracking of model performance.

Hyperparameter Tuning Results (Keras Tuner)

Parameter	Tried Values	Best Value
Filters	32, 48, 64	64
Dense Units	64, 96, 128	128
Learning Rate	0.01, 0.001	0.001
Validation Accuracy	-	98.85%

Summary and Conclusions

We successfully developed and fine-tuned a CNN for classifying MNIST digits, reaching an impressive 99.28% accuracy on the test set. By using TensorBoard for monitoring and implementing early stopping, we boosted our training efficiency. Additionally, techniques like dropout and hyperparameter tuning helped improve generalization. Looking ahead, we might want to dive into data augmentation or explore transformer-based architectures to push our accuracy even higher.

References

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791
- Keras Tuner documentation. (n.d.). Keras. https://keras.io/keras_tuner/
- TensorFlow documentation. (n.d.). TensorFlow. https://www.tensorflow.org/