Group Members: Morathi Mnkandla, Ali Shan Ahad

Class: Deep Learning ITAI 2376

October 18 2025

L04 Lab ITAI 2376

During this lab, we gained experience with Convolutional Neural Networks (CNNs) by building and training a model to classify handwritten digits from the MNIST dataset. While we had some prior understanding of basic neural networks, working directly with CNN layers helped us connect theoretical ideas to real implementation. Going step by step—from data preprocessing to training and evaluation—helped us understand how each layer and parameter affects the final model performance.

Learning Insights

We learned that CNNs are designed to handle image data by automatically detecting features like edges, textures, and shapes. The **Conv2D layers** were central to this process. We experimented with the kernel size and number of filters, realizing that filters control how many unique features the model learns, while the kernel size determines the receptive field of each filter. Keeping a **(3,3)** kernel size and increasing filters to **64** helped improve accuracy without slowing training too much. We saw how these convolutional layers captured meaningful spatial patterns that dense layers alone could not.

The **MaxPooling2D layers** were also important for simplifying feature maps. We noticed that pooling reduced the size of the data while keeping the most relevant information. When we tried reducing pooling, the model took longer to train and slightly overfitted. This experiment confirmed that pooling helps improve computational efficiency and generalization.

We also revisited the concept of **one-hot encoding** when preparing the labels. Representing each digit as a binary vector ensured that the network treated each class independently rather than assuming any numerical relationship between digits. This step clarified how categorical data must be formatted for multi-class classification.

The **Flatten layer** acted as a bridge between the convolutional layers and the dense output layer. It transformed the 2D feature maps into a 1D vector so the model could interpret learned features for classification. Without this layer, the dense network wouldn't be able to process the extracted spatial features.

When we reached model compilation, we selected **categorical_crossentropy** as the loss function and **Adam** as the optimizer. We chose categorical cross-entropy because it measures how well the model predicts probability distributions across multiple classes. Adam was ideal because it adapts learning rates automatically, making training more stable and efficient. We also used **accuracy** as the metric, since our main goal was to correctly classify digits.

Challenges and Growth

One of our main challenges was tuning the **batch size** and **number of epochs**. We initially used small batch sizes and fewer epochs, but this led to underfitting—the model didn't generalize well. After experimenting, we found that a **batch size of 128** and **15 epochs**

achieved the best balance between speed and accuracy. Increasing epochs too much risked overfitting, while too few limited learning. Watching the validation accuracy stabilize around the tenth epoch showed us when to stop training.

We also encountered difficulties in managing overfitting when experimenting with deeper models. Adding too many convolutional layers increased accuracy on the training set but slightly decreased test accuracy. Including a **Dropout layer (0.5)** was an effective solution, it forced the model to rely less on specific neurons and improved generalization. This was a good demonstration of how regularization works in practice.

Evaluating the model on the test set was one of the most satisfying parts. Our final **test accuracy was 0.9920 (99.2%)**, which indicated that the network generalized well to unseen data. Comparing the training, validation, and test accuracy confirmed that the model was balanced and not overfitting. It was rewarding to see how the steps we took in preprocessing, architecture design, and training optimization all contributed to this strong performance.

Throughout the lab, we made good use of TensorFlow and Keras documentation to understand layer parameters and training behavior. We also observed how normalization—dividing pixel values by 255—significantly stabilized learning. These resources helped us troubleshoot issues without relying on trial and error alone.

Personal Development

This lab deepened our understanding of **deep learning architecture** and how CNNs learn from visual data. Before this, CNNs felt abstract, but now we clearly see how each layer contributes to extracting and processing information. The Conv2D layers act as feature detectors, the pooling layers simplify these features, and the dense layers make final classification decisions based on them.

We were also surprised by how efficient CNNs are when trained on normalized and properly preprocessed data. Seeing the training accuracy rapidly improve after normalization emphasized how crucial data preparation is in deep learning workflows.

Moving forward, I want to explore more advanced CNN architectures such as LeNet-5, AlexNet, and VGG. We are also interested in learning about transfer learning, where pre-trained models can be fine-tuned on new datasets. These techniques could help us tackle more complex visual tasks without starting from scratch.

If we were to improve this lab, we would visualize intermediate **feature maps** to see how the CNN perceives different layers of abstraction. Understanding what each filter “sees” would make our interpretation of CNNs more intuitive and explainable.

Overall, this lab gave us confidence in implementing and understanding CNNs. We learned to interpret accuracy metrics, tune hyperparameters, and appreciate how layer design impacts

model performance. Achieving a near-perfect accuracy on MNIST felt like a meaningful milestone and strengthened our foundation for future deep learning projects.