Laboratorio di Analisi Numerica - Esercizio 7 & 9

Fusar Bassini Leonardo, Galimberti Chiara

June 14, 2025

1 NN in differential problems

Abstract

This work investigates neural networks (NNs) for solving non-linear second-order ordinary differential equations (ODEs). We focused on generalization, data augmentation, denoising, and compression.

Initially, NNs struggled to generalize beyond training data. However, data augmentation significantly improved generalization, suggesting a profound potential to alter computational paradigms for numerical simulations. We demonstrate this by training NNs on deliberately sparse datasets and achieving robust solutions.

Our approach involved: (1) generating data via numerical solvers (e.g., SciPy's solve_ivp), (2) training a fully-connected NN to map parameters to ODE solutions, (3) inferring solutions for unseen parameters, and (4) evaluating performance against ground truth. We applied this to the **Damped Pendulum**, **Van der Pol**, and **Rayleigh** equations.

Results show that while NNs struggle with extrapolation without augmentation, even minimal augmented data drastically improves performance. We also present preliminary findings on data compression (up to 10× reduction) and denoising capabilities. This study provides insights into NN strengths and limitations for ODEs, complementing traditional numerical methods.

1.1 Introduction

The integration of deep learning into differential equation (DE) solutions, notably via Physics-Informed Neural Networks (PINNs) [1], leverages NNs' universal approximation properties to solve complex systems. This approach offers flexibility for nonlinearities, irregular geometries, and high-dimensional problems, where traditional numerical methods (e.g., finite difference or finite element) often struggle.

Unlike classical machine learning tasks like classification, NNs for DEs involve an "inverse" paradigm: approximating a function $y(t)$ that satisfies a DE and its boundary/initial conditions over a continuous domain. This presents unique challenges and opportunities.

Our experimental study focuses on three distinct second-order non-linear ODEs, chosen for their complexity and physical relevance, allowing us to test generalization, data augmentation, data denoising, and data compression. Our general operative process for each case involved:

1. **Data Generation**: Employ a numerical solver (e.g., SciPy's 'solve_ivp') to generate a dataset of parameter values and corresponding ODE solutions.

2. **Neural Network Training**: Train a fully-connected NN to map parameters to complete solution functions, minimizing a loss function based on the DE residual and initial conditions.

3. **Solution Inference**: Use the trained NN to map novel, unseen parameter values to approximate solution functions.

4. **Performance Evaluation**: Compare NN solutions against ground truth, assessing accuracy, efficiency, and ability to capture qualitative features.

We have chosen a really simple net structure for the NN, following the thought that to have useful applications the NN required must at least be simple for simple equations. We used three hidden layers with 128 neurons each for our fully connected NN. The specific ODEs studied are:

1. **Damped Pendulum Equation**: A classic non-linear oscillator where solutions converge to zero, serving as a benchmark for capturing long-term damping.

2. **Van der Pol Equation**: Models self-sustained oscillations (limit cycles), a greater challenge requiring precise learning of non-linear damping. Solutions initially close may diverge significantly.

3. **Rayleigh Equation**: Similar to Van der Pol, also modeling self-sustained oscillations but with a different non-linear damping structure, allowing comparative study of NN adaptation to varied non-linearity forms.

We defer detailed discussion on certain methodological choices (e.g., specific parameters α, β within our chosen norm). Our results aim to provide insights into the strengths and limitations of NNs for solving ODEs, with comparisons to traditional numerical methods.

1.2 Generalization

Generalization is critical for robust machine learning in scientific computing. For NNs solving ODEs, it means accurately predicting solutions for **parameter values not encountered during training**. An effective model infers underlying relationships and extrapolates knowledge to new scenarios.

We assessed generalization using fully-connected NNs on second-order non-linear ODEs of the form

$$\Phi(t, y(t), y'(t), y''(t)) = 0,$$

with initial conditions $y(c) = g_0$ and $y'(c) = g_1$. Here, $c \in [0, a]$ for some $a > 0$. We trained the NN on a specific set of c values and evaluated performance on c values **outside the training range/distribution**.

We trained the network for 200 c values in $[0, 2]$. While the NN's accuracy within this range (Fig. 1, upper left/right) is satisfactory, its performance outside this range (Fig. 1, lower left/right) is poor, even for the simple damped pendulum equation.

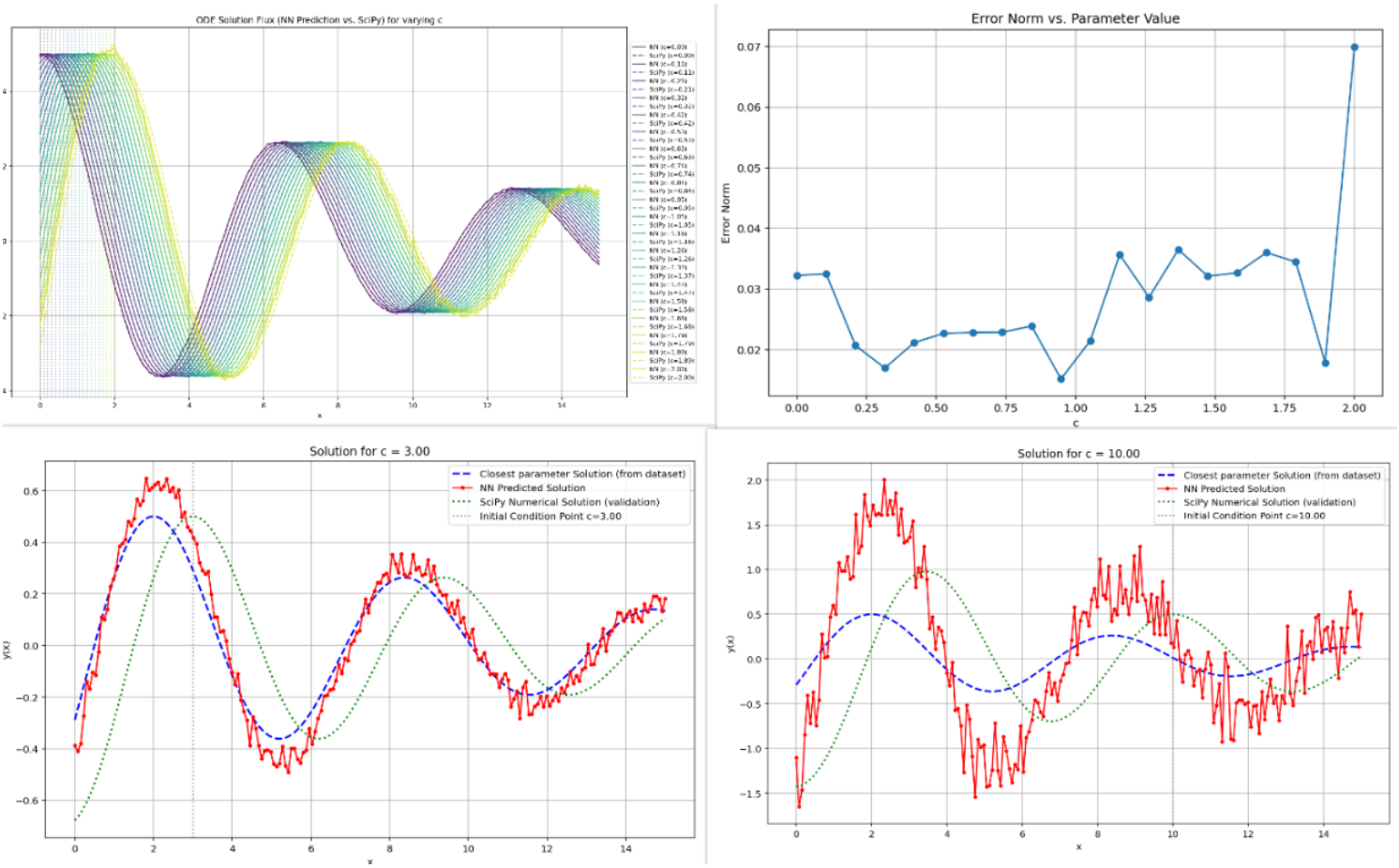


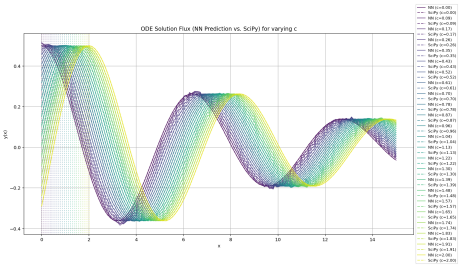
Figure 1: In order, flux of the mapping given by the NN, error norm for the values of c in the training range, solutions for two values outside the training range.

1.3 Data Augmentation

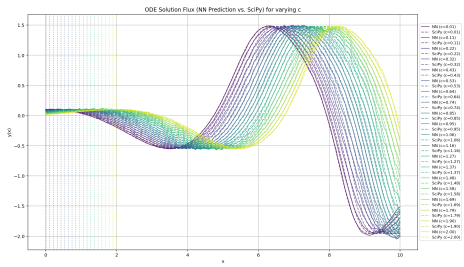
Limited generalization of NNs on sparse ODE datasets was a significant challenge. To overcome this, we explored **data augmentation**. In this context, it expands the effective training dataset by generating new, plausible data from existing limited sets. Our hypothesis: increasing data diversity, even synthetically, yields more robust and generalizable representations of the differential operator.

Our objective was to train the NN on a deliberately **small set of parameter values**, then assess its ability to "augment" this dataset by producing accurate solutions for parameters not explicitly in the training set, but inferred through interpolation or extrapolation. This goes beyond simple generalization; it tests the NN's capacity to autonomously infer ODE behavior for a continuous parameter spectrum from minimal examples. Success here would significantly reduce the computational cost of generating extensive training data via traditional solvers, accelerating numerical simulations.

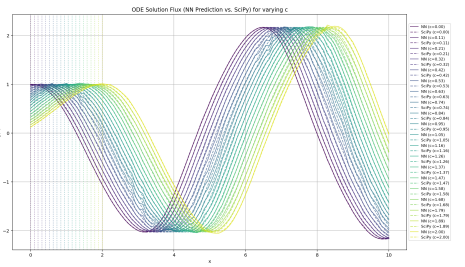
We used an extremely limited training set (only 8 samples per ODE). Despite this, we achieved limited error in most cases (our error norm is always $\geq L^2$-norm). Empirically, we found that wider parameter ranges or higher rates of change/non-linearity of the solution increased the need for accurate flux approximation (i.e., lower error norm).



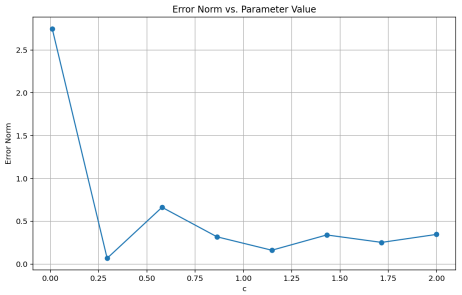
(a) Flux of NN solution of pendulum equation



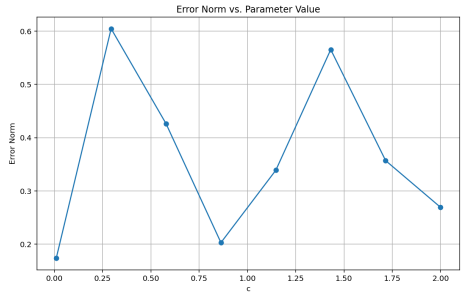
(b) Flux of NN solution of Van Der Pol equation



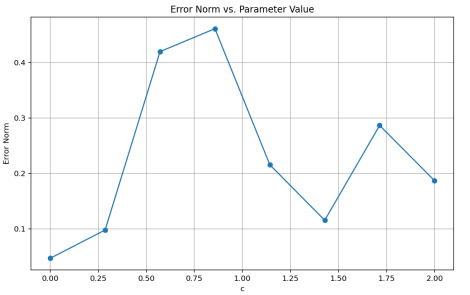
(c) Flux of NN solution of Rayleigh equation



(d) Error norm of NN solution of pendulum equation



(e) Error norm of NN solution of Van Der Pol equation



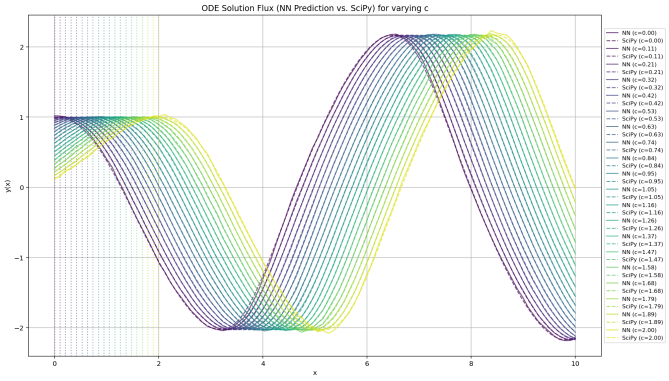
(f) Error norm of NN solution of Van Der Pol equation

1.4 Data Compression and Data Denoising

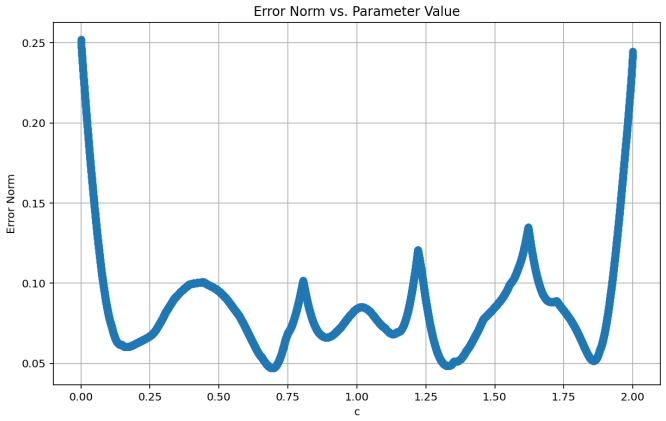
Beyond generalization and data augmentation, NNs offer potential for **data compression** and **data denoising** in solving DEs.

NNs show promising data compression, potentially reducing storage and transmission needs for simulation data by up to 10×. This is achieved by learning solution manifolds with fewer parameters than traditional methods, which could transform data handling in scientific computing.

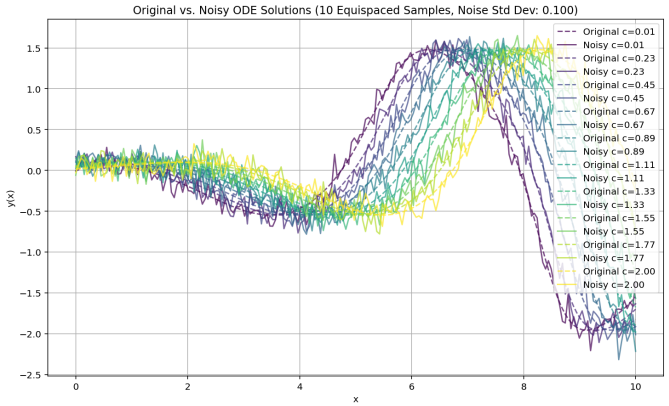
Additionally, NNs exhibit a basic denoising effect, filtering noise from input data due to their inherent regularization and ability to learn smooth functions. This suggests their potential as a denoising tool for noisy experimental data in DE models. While preliminary, these findings highlight significant research opportunities to explore compression and denoising mechanisms, their optimality, and applicability across diverse scenarios, pointing to advancements in computational efficiency and data quality.



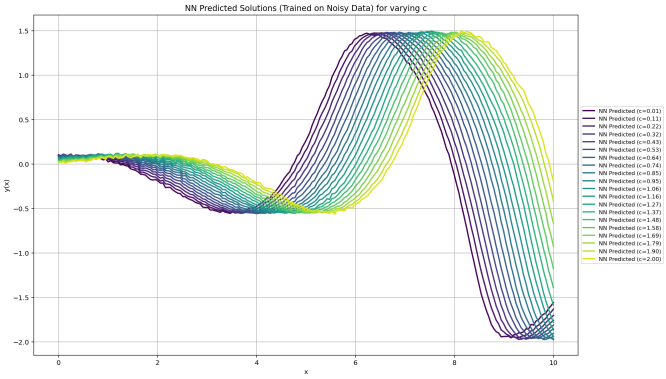
(a) Data reconstruction by NN



(b) Error in the data reconstruction by NN, note that the accuracy degrades towards the edges of the dominion



(a) Noised data fed in the training



(b) Flux of NN solutions, note the denoising effect

2 Multiclass Image Classification of Fruits-360 Dataset

Abstract

This exercise addresses the challenge of multiclass image classification using neural networks, specifically focusing on the Fruits-360 dataset [2]. We implemented and evaluated two distinct neural network architectures: a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN). Performance is evaluated by varying hyperparameters, primarily the number of layers and epochs, across two experimental settings: classifying different types of fruits and classifying different varieties of the same fruit. Our results indicate varying degrees of accuracy for both architectures under different configurations, highlighting the strengths and weaknesses of each for this specific image classification task.

2.1 Introduction

The dataset is made of RGB images of fruits and vegetables, designed for classification tasks, each with dimensions of 100×100 pixels; the dataset is structured with separate training and testing partitions, where each subfolder represents a distinct class. Our objective is to develop a classification system capable of accurately recognizing and categorizing these images.

To make this possible, we investigate two fundamental types of neural network architectures:

- **Multilayer Perceptron (MLP)**: often referred to as a Fully Connected Neural Network, the MLP processes input images by flattening them into a single vector. Each neuron in one layer is connected to every neuron in the subsequent layer, making it suitable for learning complex non-linear relationships.
- **Convolutional Neural Network (CNN)**: CNNs are specifically designed for image data, employing convolutional layers to automatically and adaptively learn the main features from input images. This makes them particularly effective for tasks like object recognition and classification.

We carried out two main experimental setups to study the performance of these two networks:

1. **First experiment: Different types of fruits**: This experiment focuses on the general multiclass classification problem where the network must distinguish between broadly different categories of fruits (e.g., Apple, Banana, Orange).
2. **Second experiment: Different varieties of the same fruit**: This experiment challenges the networks to differentiate between slightly variations within a single fruit type (e.g., Apple Braeburn, Apple Golden, Apple Red Delicious).

For each experiment and network type, we varied key hyperparameters, specifically the number of hidden layers and the training epochs, to observe their impact on classification accuracy.

2.2 Experimental Results

2.2.1 Different Types of Fruits

Multilayer Perceptron (MLP)

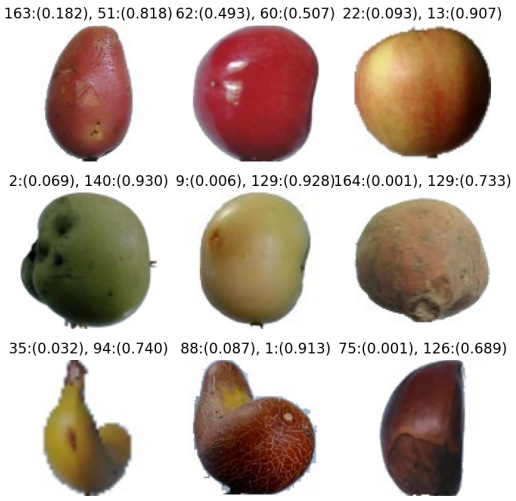
- 30 classes:
  - Epoch = 10: Three-layer network: 97% accuracy; Four-layer network: 91% accuracy;
  - Epoch = 15: Four-layer network: 93% accuracy;
- 60 classes: epoch = 10: Three-layer network: 97% accuracy;
- all 201 classes: epoch = 10: Three-layer network: 89.77% accuracy;

Starting with 30 different classes, the MLP achieved a high accuracy with a three linear layer architecture and 10 epochs. Interestingly, increasing the number of linear layers to four resulted in a decrease of the accuracy, suggesting potential overfitting or increased difficulty in training deeper MLPs on this task without further hyperparameter tuning (e.g., regularization). Increasing the training epochs value to 15 for the four-layer network improved its accuracy, indicating that deeper networks might require more training iterations to converge effectively.
Since it has the best accuracy, we tested the three linear network first on 60 classes, on which the accuracy remain stable, and then on all the classes of the dataset, with a decrease in accuracy.

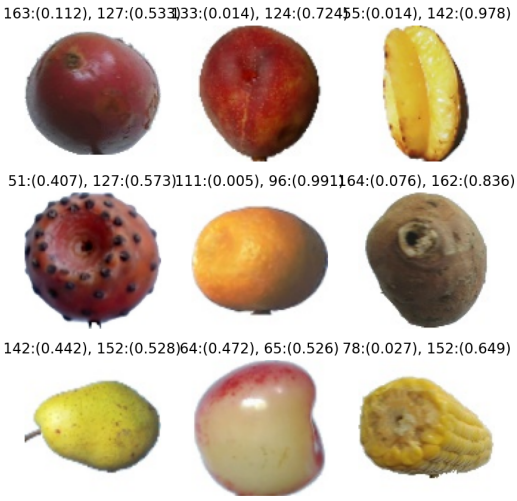
Convolutional Neural Network (CNN)

- 30 classes:
  - Epoch = 10: Two-layer network: 90% accuracy; Three-layer network: 80% accuracy;
  - Epoch = 15: Three-layer network: 78% accuracy;
- 60 classes: epoch = 10: Three-layer network: 92% accuracy;
- all 201 classes: epoch = 10: Three-layer network: 96.54% accuracy;

Instead, the CNN's initial performance, with a two convolutional layer basic configuration at 10 epochs, was lower than that of the MLP; with basic configuration we mean: kernel size at 3, padding at 1 and stride at 1. A three convolutional layer CNN at 10 epochs saw a significant drop in accuracy. Surprisingly, extending training to 15 epochs for the three-layer CNN doesn't improve its accuracy. This unexpected behaviour for the CNN, especially the decrease with more epochs, could indicate issues with basic parameter choices (e.g., learning rate, optimizer) or a need for more sophisticated regularization techniques to prevent overfitting.
As before, since it has the best accuracy, we tested the two-layer network first on 60 classes, on which the accuracy remain stable, and then on all the classes of the dataset, with an increase in accuracy, which made the CNN had a better result than the MLP.



(a) 201 classes: some MLP errors



(b) 201 classes: some CNN errors

2.2.2 Different Varieties of the Same Fruit

Multilayer Perceptron (MLP)

- **Apples :** 30 classes:
  - Epoch = 10: Two-layer network: 96% accuracy; Three-layer network: 96.5% accuracy;
  - Epoch = 15: Three-layer network: 96% accuracy;
- **Tomatoes :** 19 classes:
  - Epoch = 10: Three-layer network: 99.69% accuracy;

The MLP demonstrated consistent high performance. With 30 classes of apples, from the two-layer network at 10 epochs to the three-layer network at 15 epochs it maintained a strong 96% accuracy. This indicates that MLPs are robust for this slightly classification.
This is confirmed using the smaller dataset of tomatoes, with 19 classes, on which the MLP reach almost a perfect accuracy potentially due to the relatively consistent visual features within fruit varieties that can be captured even without spatial hierarchies, or because the dataset's variations are distinguishable in a flattened vector space.

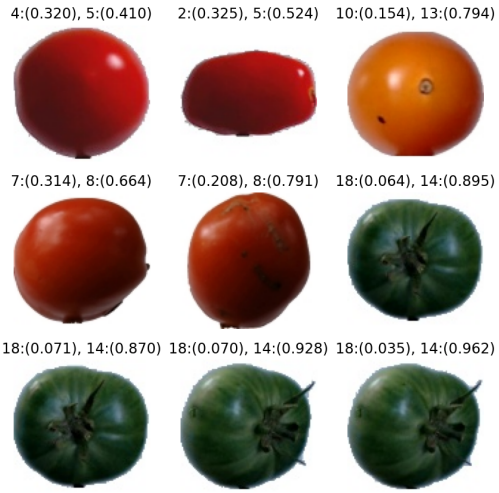
Convolutional Neural Network (CNN)

- **Apples :** 30 classes:
  - Epoch = 10: One-layer network: 91% accuracy; Two-layer network: 95% accuracy;
  - Epoch = 15: Three-layer network: 95% accuracy;
- **Tomatoes :** 19 classes:

– Epoch = 10: Two-layer network: 96% accuracy;

In the apples case, the CNN's performance was also strong. It maintained almost the same accuracy (slightly lower) compared to the MLP, apart from the minimal one-layer case. This shows that with a slightly deeper architecture (two or three layers), the CNN's ability to extract thin spatial features makes it highly effective for distinguishing subtle differences between fruit varieties.
However it's accuracy does not improve much reducing the number of classes; indeed it remains almost the same testing the network on the tomato dataset.

(a) apples classes: some MLP errors



tomatoes classes: some CNN errors

References

[1] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707.

[2] Horea Miron. (2017). Fruits 360: A Dataset of Fresh and Rotten Fruits. Kaggle. https://www.kaggle.com/datasets/moltean/fruits