**Documentation: MNIST CNN Tutorial**

**Title**: CNN on MNIST – MecaNano Summer School Tutorial  
**Purpose**: Learn how to build and train a Convolutional Neural Network (CNN) to classify handwritten digits using the MNIST dataset.  
The notebook covers data loading, network design, training, validation, evaluation metrics, and predictions on custom images.

**1. Overview**

This notebook introduces a complete machine learning pipeline using PyTorch.  
The goal is to train a CNN on the MNIST dataset of handwritten digits and evaluate its performance using:

* Validation accuracy
* Loss trends
* Confusion matrices
* Sample predictions on user input

It ends with the capability to classify user-provided digit images, such as those drawn with a pen and scanned or photographed.

**2. Notebook Sections**

**Section 1: Imports**

This section loads the necessary Python libraries:

* torch, torchvision: neural networks and image datasets
* matplotlib, numpy: plotting and array handling
* scikit-learn: confusion matrix computation
* PIL: image loading and preprocessing

These libraries form the basis of model development, training, evaluation, and visualization.

**Section 2: Hyperparameters and Data Loading**

* Defines training settings: batch size, number of epochs, learning rate
* Loads the MNIST dataset from torchvision.datasets
* Normalizes the images using the dataset mean and standard deviation
* Splits training data into:
  + **Training set** (90%)
  + **Validation set** (10%)

Normalization is crucial for stable training and reproducibility.

**Section 3: CNN Definition**

A custom class CNN is defined using PyTorch’s nn.Module.  
Network structure:

* Conv2D (1 → 32) with 3×3 kernel
* Conv2D (32 → 64) with 3×3 kernel
* MaxPool2D with 2×2 window
* Dropout layers to reduce overfitting
* Flatten → Linear (9216 → 128)
* Linear (128 → 10) output logits for digit classification (0–9)

**Activation**: ReLU  
**Output**: raw scores passed to softmax by the loss function.

**Section 4: Training Loop**

Performs the core training:

* Model set to train() mode
* Batches passed through forward pass
* Loss computed using CrossEntropyLoss
* Gradient calculated via backpropagation
* Model updated using Adam optimizer

At each epoch:

* Average training loss is computed
* Validation loss and accuracy are measured by switching to eval() mode

Results are stored in:

* train\_losses
* val\_losses
* val\_accuracies

This allows tracking learning progress.

**Section 5: Loss and Accuracy Plot**

The notebook visualizes:

* Loss curves for training and validation
* Accuracy over epochs for validation

These plots help interpret:

* Underfitting (high loss, low accuracy)
* Overfitting (diverging curves)
* Training effectiveness and convergence

**Section 6: Confusion Matrix**

Generates a confusion matrix using scikit-learn:

* Compares predictions and ground truth from validation set
* Matrix cell [i, j]: number of true class i predicted as j

This is useful to:

* Identify classes that are systematically confused (e.g. 3 vs. 8)
* Evaluate class-level accuracy and errors

**Section 7: Prediction Examples**

Shows real examples from the validation set with model predictions:

* Displays 6 random images
* Titles show predicted class label

Aids in:

* Visual understanding of model behavior
* Spot-checking obvious successes or mistakes

**Section 8: User Image Prediction**

Provides a function: