LAB 5 REPORT

A PREPRINT

Daniele cecca
Artificial Intelligence for Science and Technology
Milano Bicocca University
Supervised Learning

April 21, 2024

1 Introduction

In this lab, we will study Convolutional Neural Networks (CNNs) for image classification using the CIFAR-10 dataset. Rather than sticking to a single CNN architecture, we aim to create and analyze multiple CNN configurations to discern their impact on classification performance. The CIFAR-10 dataset comprises 60,000 32x32 color images in 10 classes, with 6,000 images per class.

Objectives: Our primary objective is to conduct a systematic investigation into various CNN architectures, training methodologies, and optimization strategies to unravel their effectiveness in tackling the CIFAR-10 classification task. Through this exploration, we will perform:

- **Data Loading and Normalization:** Utilizing torchvision, we load and normalize the CIFAR-10 training and test datasets. This involves converting PIL images to PyTorch tensors and normalizing pixel values
- **CNN Architecture Definition:** Crafting multiple CNN architectures with distinct layer configurations, activation functions, and filter sizes to discern their influence on classification performance.
- **Loss Function** and Optimizer Selection: Employing different loss functions and optimizers to discern their impact on convergence speed and final accuracy.
- **Training and Evaluation:** Training each CNN configuration on the CIFAR-10 training dataset and evaluating its performance on the test dataset. We will analyze accuracy, loss curves, and potentially explore techniques like data augmentation to enhance generalization.
- **Error Analysis:** Delving into the misclassifications made by the best-performing CNN configurations to identify patterns and potential areas for improvement.

2 Network 1

This network is a simple convolutional neural network (CNN) architecture designed for image classification tasks, particularly suited for datasets like CIFAR-10. This network architecture is inspired by LeNet, one of the earliest and influential CNN architectures developed by Yann LeCun and others. Here's a breakdown of its components:

- **Convolutional Layers**:
  - `self.conv1`: The first convolutional layer with 3 input channels (representing RGB images), 6 output channels, and a kernel size of 5x5.
  - `self.conv2`: The second convolutional layer with 6 input channels (coming from the first convolutional layer), 16 output channels, and a kernel size of 5x5.
- **Pooling Layer**:
  - `self.pool`: Max pooling layer with a kernel size of 2x2. It's applied after each convolutional layer to reduce spatial dimensions.

- **Fully Connected Layers**:
  - `self.fc1`: The first fully-connected layer with 120 neurons.
  - `self.fc2`: The second fully-connected layer with 84 neurons.
  - `self.fc3`: The output layer with 10 neurons, corresponding to the 10 classes in CIFAR-10.
- **Forward Pass**:
  - The `forward` method defines the forward pass of the neural network. It applies convolution, activation (ReLU), and pooling operations sequentially, followed by flattening the output and passing it through fully connected layers.

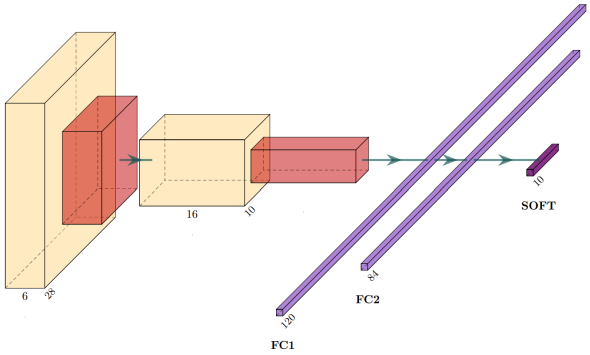
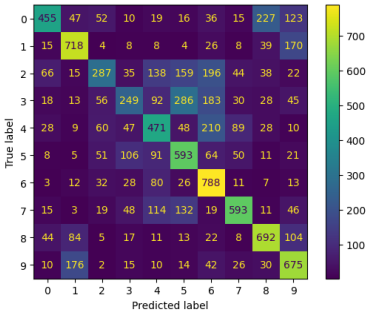
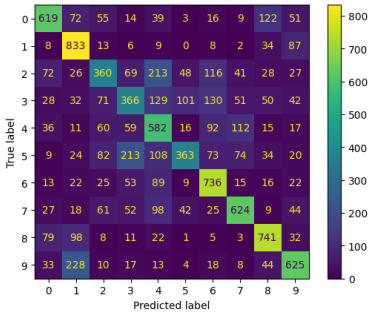


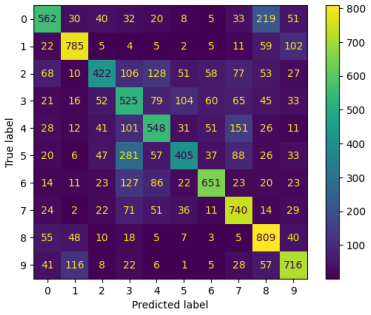
Figure 1: Network 1



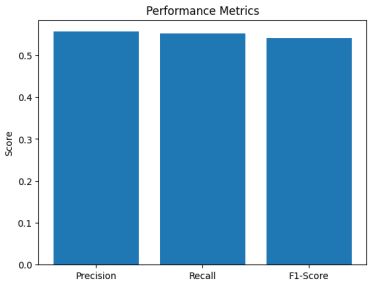
(a) Confusion Matrix 2 epochs



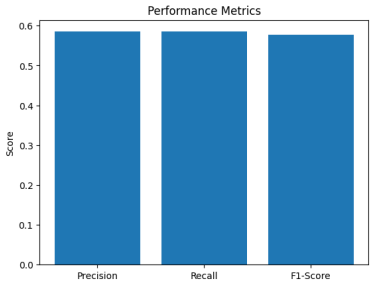
(b) Confusion matrix 4 epochs



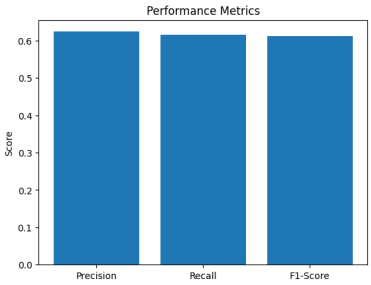
(c) Confusion Matrix 10 epochs



(a) Metrics 2 epochs



(b) Metrics 4 epochs

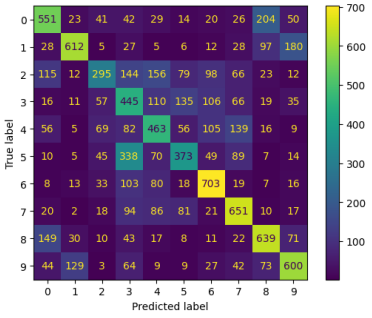


(c) Metrics 10 epochs

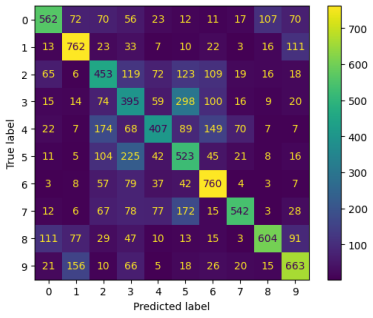
3 Network 2

This network is another convolutional neural network (CNN) architecture designed for image classification tasks. Here's a breakdown of its components:

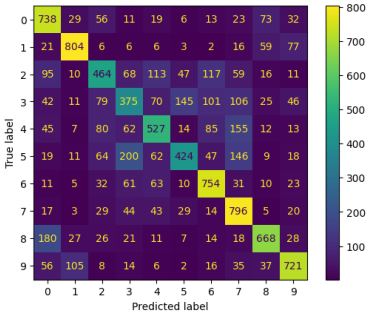
- **Convolutional Layers**:
  - `self.conv1`: The first convolutional layer with 3 input channels (representing RGB images), 6 output channels, and a kernel size of 5x5.
  - `self.conv2`: The second convolutional layer with 6 input channels (coming from the first convolutional layer), 16 output channels, and a kernel size of 3x3.
  - `self.conv3`: The third convolutional layer with 16 input channels (coming from the second convolutional layer), 32 output channels, and a kernel size of 3x3.
  - `self.conv4`: The fourth convolutional layer with 32 input channels (coming from the third convolutional layer), 32 output channels, and a kernel size of 3x3.
- **Pooling Layer**:
  - `self.pool`: Max pooling layer with a kernel size of 2x2. It's applied after the first and fourth convolutional layers to reduce spatial dimensions.
- **Fully Connected Layers**:
  - `self.fc1`: The first fully-connected layer with 150 neurons.
  - `self.fc2`: The second fully-connected layer with 100 neurons.
  - `self.fc3`: The output layer with 10 neurons, corresponding to the 10 classes in the classification task.
- **Forward Pass**:
  - The `forward` method defines the forward pass of the neural network. It applies convolution, activation (ReLU), and pooling operations sequentially, followed by flattening the output and passing it through fully connected layers.



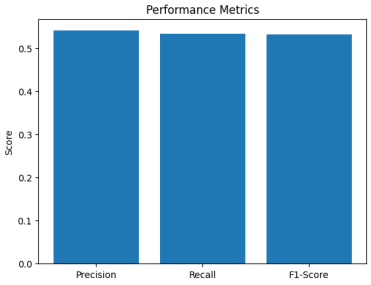
(a) Confusion Matrix 2 Epochs



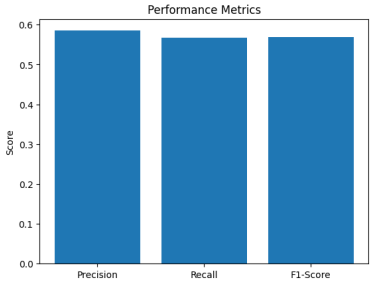
(b) Confusion Matrix 4 Epochs



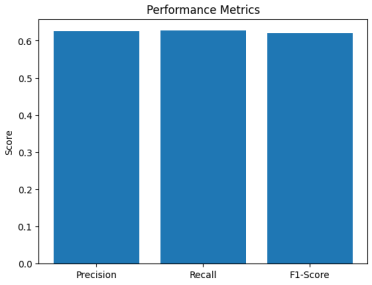
(c) Confusion Matrix 10 Epochs



(a) Metrics 2 Epochs



(b) Metrics 4 Epochs



(c) Metrics 10 Epochs

4 Network 3

This network is another convolutional neural network (CNN) architecture designed for image classification tasks. Here's a breakdown of its components:

- **Convolutional Layers**:
  - `self.conv1`: The first convolutional layer with 3 input channels (representing RGB images), 12 output channels, a kernel size of 5x5, and padding of 2 pixels.
  - `self.conv2`: The second convolutional layer with 12 input channels (coming from the first convolutional layer), 24 output channels, and a kernel size of 3x3.
  - `self.conv3`: The third convolutional layer with 24 input channels (coming from the second convolutional layer), 48 output channels, and a kernel size of 3x3.
  - `self.conv4`: The fourth convolutional layer with 48 input channels (coming from the third convolutional layer), 64 output channels, and a kernel size of 3x3.
- **Pooling Layer**:
  - `self.pool`: Max pooling layer with a kernel size of 2x2. It's applied after the first and fourth convolutional layers to reduce spatial dimensions.
- **Fully Connected Layers**:
  - `self.fc1`: The first fully-connected layer with 512 neurons.
  - `self.fc2`: The second fully-connected layer with 128 neurons.
  - `self.fc3`: The output layer with 10 neurons, corresponding to the 10 classes in the classification task.
- **Forward Pass**:
  - The `forward` method defines the forward pass of the neural network. It applies convolution, activation (ReLU), and pooling operations sequentially, followed by flattening the output and passing it through fully connected layers.

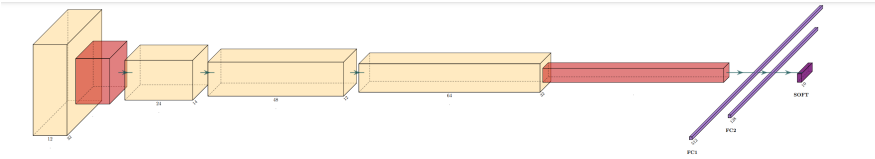
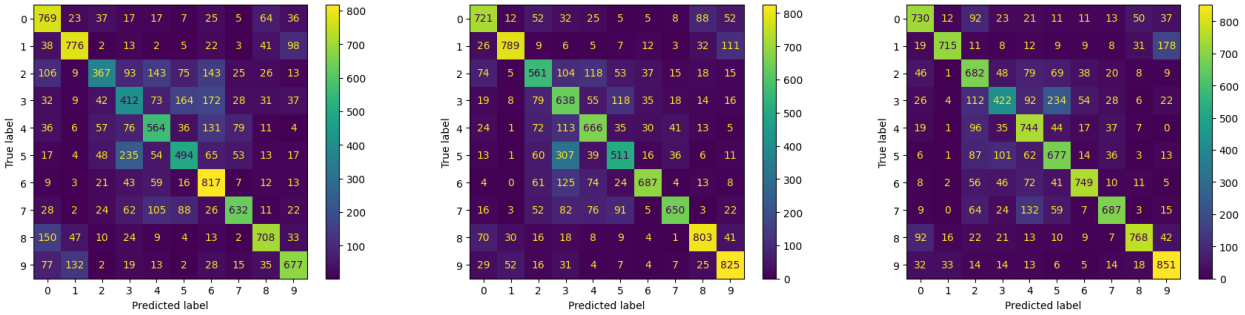
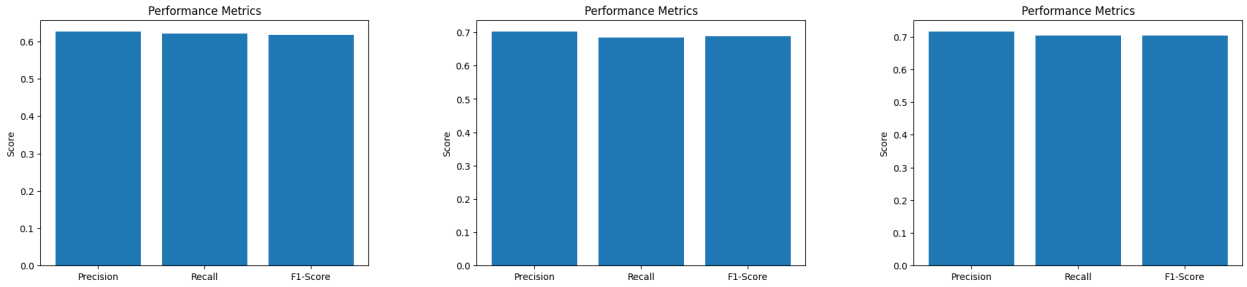


Figure 6: Network 3



(a) Confusion Matrix 2 Epochs (b) Confusion Matrix 4 Epochs (c) Confusion Matrix 10 Epochs



(a) Metrics 2 Epochs (b) Metrics 4 Epochs (c) Metrics 10 Epochs

5 Network 4

This network is another convolutional neural network (CNN) architecture designed for image classification tasks. Here's a breakdown of its components:

- **Convolutional Layers**:
  - `self.conv1`: The first convolutional layer with 3 input channels (representing RGB images), 12 output channels, a kernel size of 5x5, and padding of 2 pixels.
  - `self.conv2`: The second convolutional layer with 12 input channels (coming from the first convolutional layer), 24 output channels, and a kernel size of 3x3.
  - `self.conv3`: The third convolutional layer with 24 input channels (coming from the second convolutional layer), 48 output channels, and a kernel size of 3x3.
  - `self.conv4`: The fourth convolutional layer with 48 input channels (coming from the third convolutional layer), 64 output channels, and a kernel size of 3x3.
- **Pooling Layer**:
  - `self.pool`: Max pooling layer with a kernel size of 2x2. It's applied after the first and fourth convolutional layers to reduce spatial dimensions.
- **Fully Connected Layers**:
  - `self.fc1`: The first fully-connected layer with 512 neurons.
  - `self.fc2`: The second fully-connected layer with 128 neurons.
  - `self.fc3`: The output layer with 10 neurons, corresponding to the 10 classes in the classification task.
- **Forward Pass**:
  - The `forward` method defines the forward pass of the neural network. It applies convolution, activation (Leaky ReLU), and pooling operations sequentially, followed by flattening the output and passing it through fully connected layers.

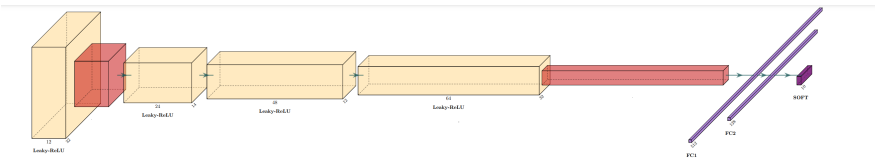
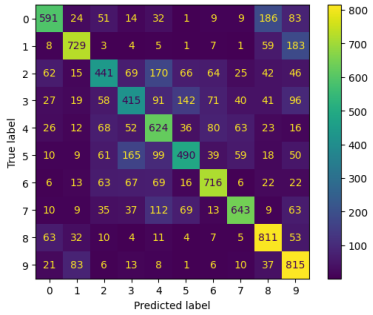
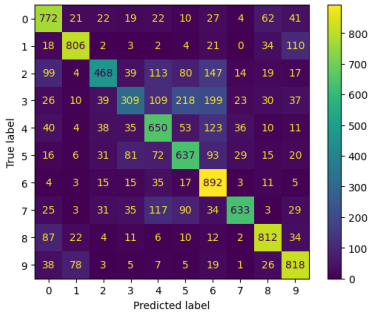


Figure 9: Network 4

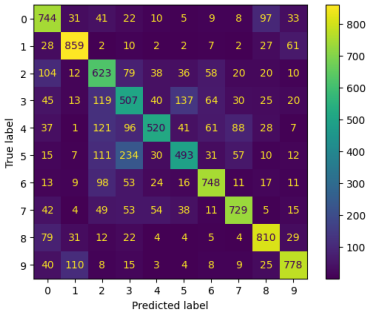
The Network 4 was also evaluated with other three experimental setups. In the following section we will have a look to the results:



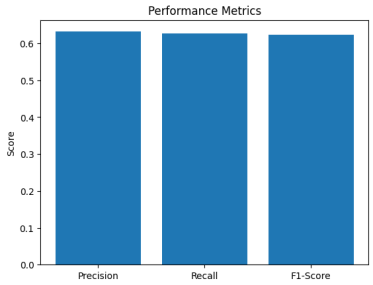
(a) Confusion Matrix 2 Epochs



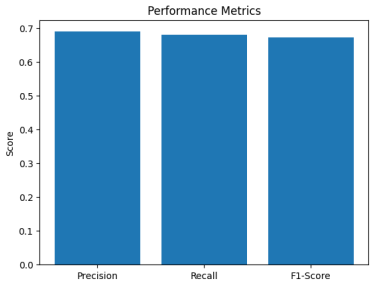
(b) Confusion Matrix 4 Epochs



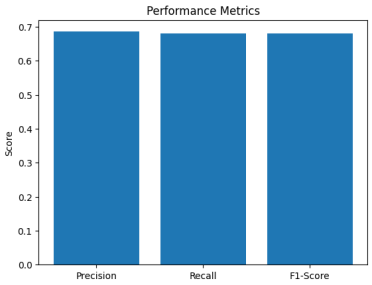
(c) Confusion Matrix 10 Epochs



(a) Metrics 2 Epochs



(b) Metrics 4 Epochs



(c) Metrics 10 Epochs

6 Network 5

This network is inspired by AlexNet, a pioneering convolutional neural network (CNN) architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Here's a breakdown of its components:

- **Upsampling Layer**:
  - `self.upsample`: Upsampling layer with a scale factor of 8 and bilinear interpolation. It's applied at the beginning of the network to increase the spatial resolution of the input.
- **Convolutional Layers**:
  - `self.conv1`: The first convolutional layer with 3 input channels (representing RGB images), 3 output channels, and a kernel size of 30x30.
  - `self.conv2`: The second convolutional layer with 3 input channels, 96 output channels, a kernel size of 11x11, and a stride of 4.
  - `self.conv3`: The third convolutional layer with 96 input channels, 256 output channels, a kernel size of 5x5, and padding of 2 pixels.
  - `self.conv4`: The fourth convolutional layer with 256 input channels, 384 output channels, a kernel size of 3x3, and padding of 1 pixel.
  - `self.conv5`: The fifth convolutional layer with 384 input channels, 384 output channels, a kernel size of 3x3, and padding of 1 pixel.
  - `self.conv6`: The sixth convolutional layer with 384 input channels, 256 output channels, a kernel size of 3x3, and padding of 1 pixel.
- **Pooling Layer**:
  - `self.pool`: Max pooling layer with a kernel size of 3x3 and a stride of 2. It's applied after the second and third convolutional layers to reduce spatial dimensions.
- **Dropout Layer**:
  - `self.drop`: Dropout layer with a dropout probability of 0.5. It's applied after the flattening operation.
- **Fully Connected Layers**:
  - `self.fc1`: The first fully-connected layer with 4096 neurons.

- `self.fc2`: The second fully-connected layer with 410 neurons.
- `self.fc3`: The output layer with 10 neurons, corresponding to the 10 classes in the classification task.

- **Forward Pass**:
  - The `forward` method defines the forward pass of the neural network. It applies upsampling, convolution, activation (ReLU), pooling, dropout, and fully connected operations sequentially.

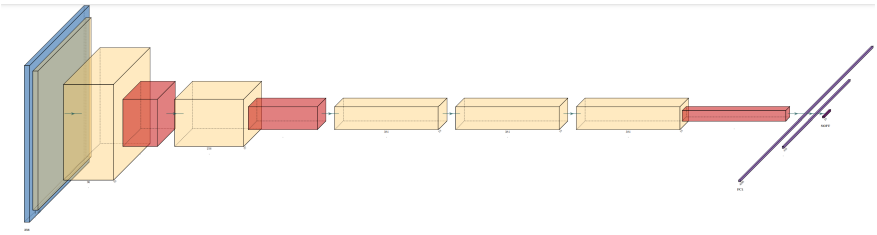
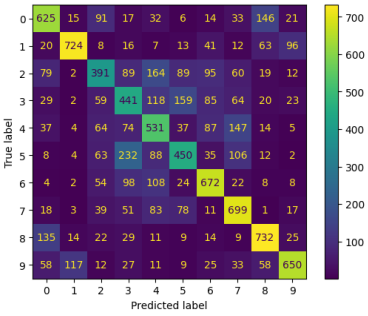
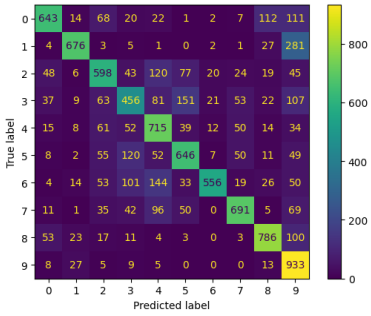


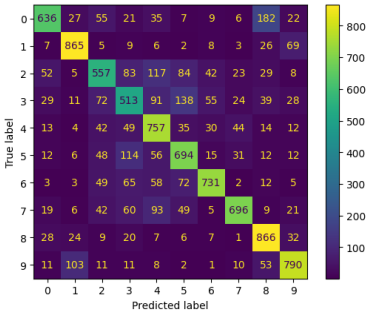
Figure 12: Network 5



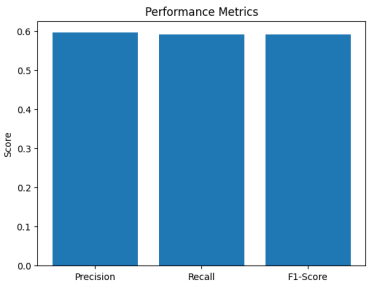
(a) Confusion Matrix 2 Epochs



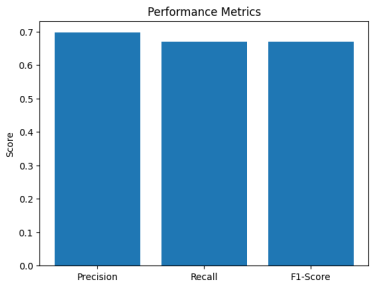
(b) Confusion Matrix 4 Epochs



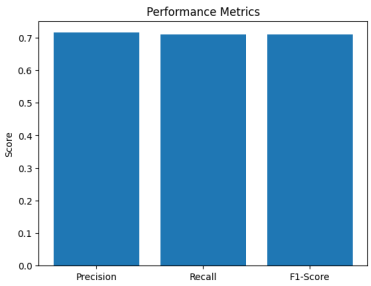
(c) Confusion Matrix 10 Epochs



(a) Metrics 2 Epochs



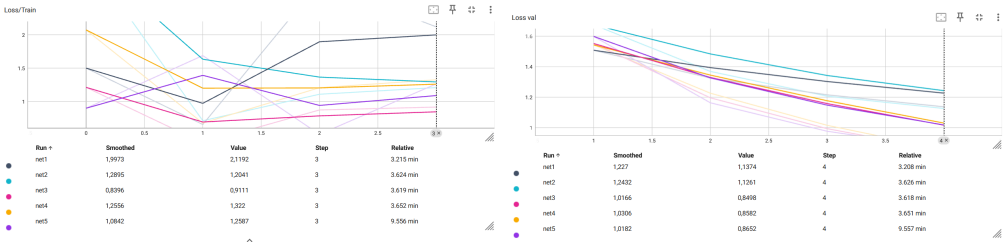
(b) Metrics 4 Epochs



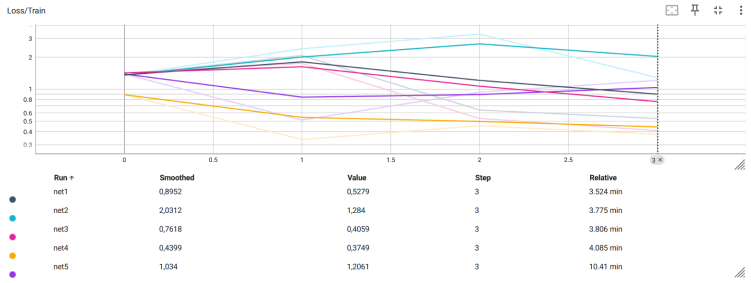
(c) Metrics 10 Epochs

7 Results

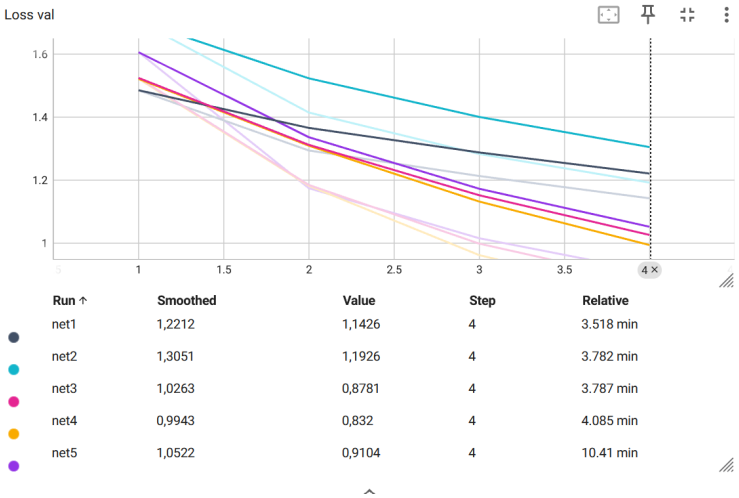
7.1 Results obtained with 2 Epochs



7.2 Results obtained with 4 Epochs

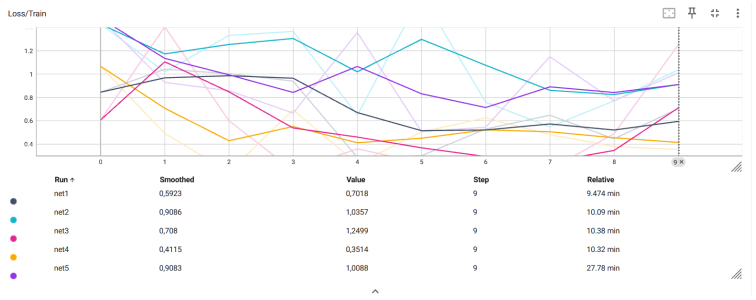


(a) Loss/Train delta

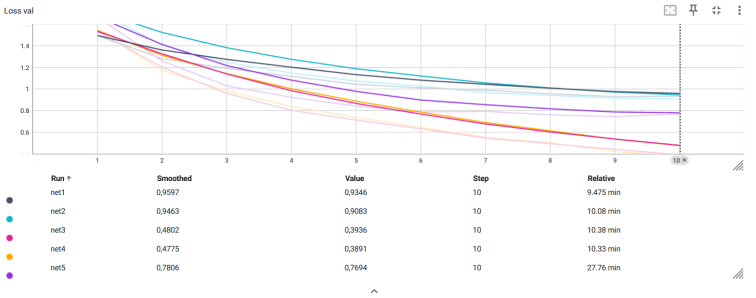


(b) Loss

7.3 Results obtained with 10 Epochs



(a) Loss/Train delta



(b) Loss

7.4 Accuracy

This are the accuracy scored by each CNN for 2, 4 and 10 Epochs:

CNNs Accuracy (2 Epochs)
58%
56%
68%
67%
67%

CNNs Accuracy (4 Epochs)
58%
56%
68%
67%
67%

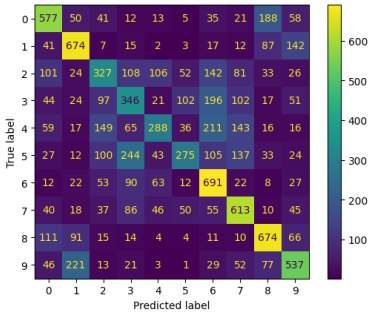
CNNs Accuracy (10 Epochs)
61%
62%
70%
68%
71%

8 Additional Experiments (1/2)

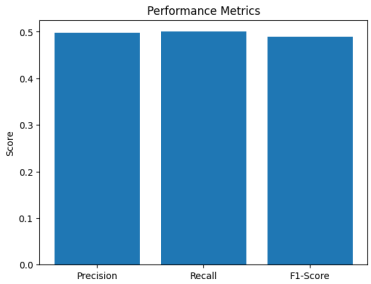
For experimental purposes, we also evaluated the networks (using 4 epochs) with various configurations. In particular, we explored changes involving the use of two different learning rates (0.0001 and 0.001, with momentum=0.9) and the ADAM optimizer.

8.1 Network 1

In the following, we present the results obtained by the networks, which were trained for 4 epochs using the aforementioned learning rate.

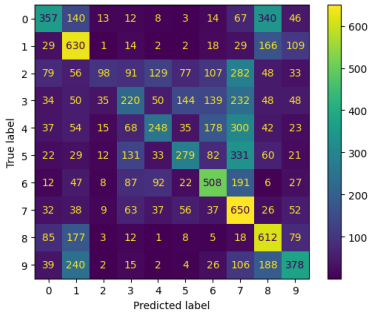


(a) Confusion Matrix

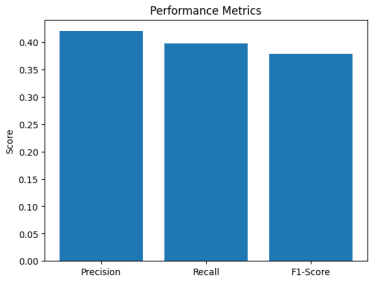


(b) Performance Matrix

8.2 Network 2

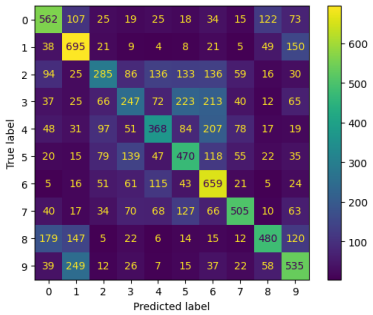


(a) Confusion Matrix

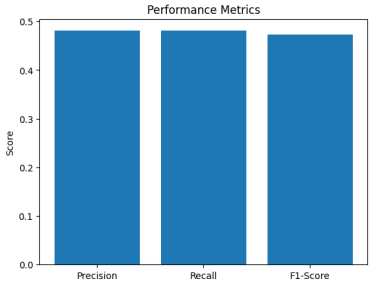


(b) Performance Matrix

8.3 Network 3

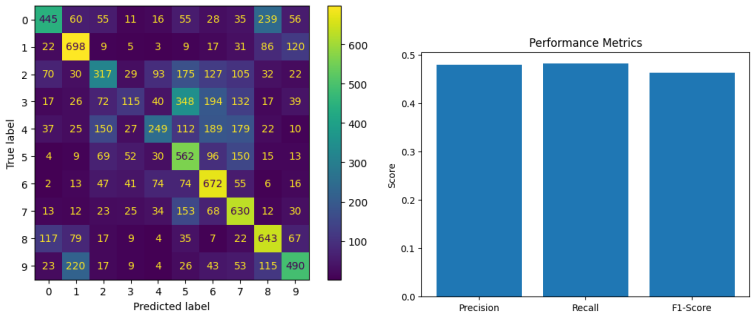


(a) Confusion Matrix



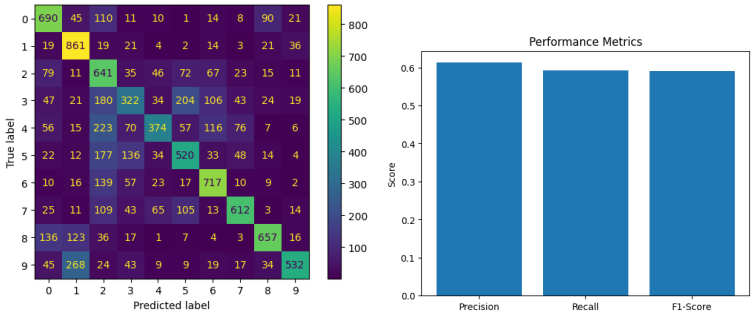
(b) Performance Matrix

8.4 Network 4



(a) Confusion Matrix (b) Performance Matrix

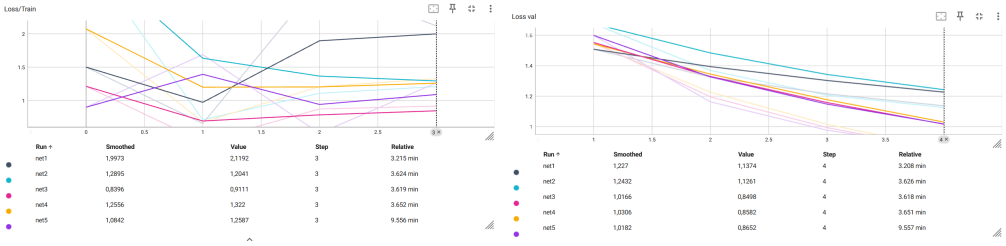
8.5 Network 5



(a) Confusion Matrix (b) Performance Matrix

9 Results

Here we have the results using a learning rate of 0.01

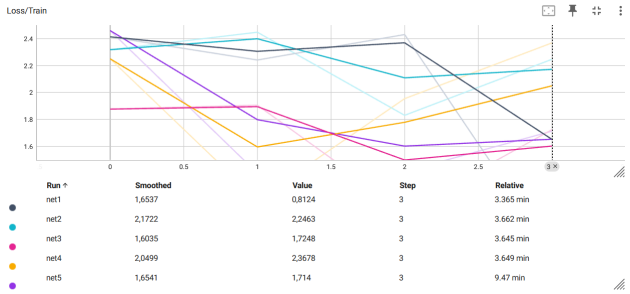


(a) Loss/Train Delta (b) Loss

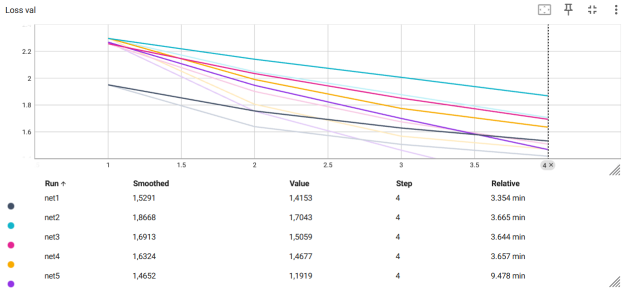
10 Further Additional Experiments

We employed further experiments, such as the use of 0.0001 as learning rate and the use of ADAM as optimizer. Just for a summarized view, we insert here the results obtained.

Results obtained using a learning rate of 0.0001:

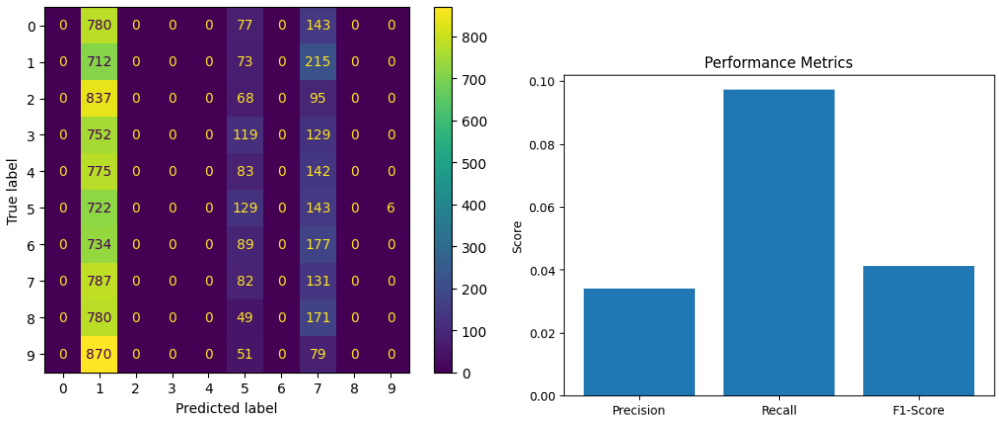


(a) Loss/Train Delta



(b) Loss

And results obtained using ADAM as optimizer and a learning rate of 0.01, those are just two samples of a confusion matrix and performance metrics of the best cnn.



11 Conclusion

In this study, we systematically explored the impact of varying network parameters on the classification performance of CNNs using the CIFAR-10 dataset. Our experiments used multiple configurations, focusing on different layer depths, activation functions, filter sizes, and training epochs. These modifications clarified key insights into the architecture's behavior and its sensitivity to specific changes.

The analysis revealed that deeper networks generally performed better, owing to their enhanced capability in capturing complex patterns in the data. However, this came with the caveat of increased computational complexity and the potential for overfitting, necessitating sophisticated regularization techniques like dropout. Our findings also underscored the critical nature of choosing the right optimizer and learning rate, as these significantly influenced the training dynamics and ultimate model accuracy.

Particularly, the use of a lower learning rate tended to stabilize training but required more epochs to converge to a similar performance, highlighting a trade-off between training time and stability. The error analysis, supported by confusion matrices, indicated that most misclassifications occurred between visually similar classes, suggesting further areas for refinement in feature extraction and classification layers.

This exploration not only advances our understanding of CNN architectures but also serves as a foundational guide for future research and practical applications in image classification tasks.

References

[1] Training a Classifier *Pytorch website*. Available at: `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`. Accessed 18 Apr. 2024.

[2] How to use TensorBoard with PyTorch *Pytorch website*. Available at: `https://pytorch.org/tutorials/recipes/recipes/tensorboard_with_pytorch.html`. Accessed 17 Apr. 2024.

[3] AlexNet Architecture *Wikipedia*. Available at: `https://it.wikipedia.org/wiki/AlexNet`. Accessed 17 Apr. 2024.