Advanced Machine Learning: Street signs image recognition using Neural Networks

Nicoli Leal[1 [2021272755]] and Stefano Trenti[2 [2023252452]]

University of Coimbra, PT

Abstract. The project aimed to explore Artificial Neural Networks (ANNs) for image classification, focusing on traffic sign images. Two ANN models, Multi-Layer Perceptrons (MLP) and Convolutional Neural Networks (CNNs), were developed and assessed with varying architectures and parameters. The dataset underwent preprocessing and augmentation to enhance diversity. Despite challenges related to overfitting, CNNs demonstrated superior generalization capabilities, achieving higher accuracies. The best-performing CNN model achieved a test accuracy of 93.4% and a Kaggle score of 0.71, showcasing their effectiveness for image classification tasks.

Keywords: Machine Learning · Convolutional Neural Networks · Neural Networks · Multilayer Perceptron · Image Recognition.

1 Introduction

This report presents the main methods, results, and analysis for the first practical assignment within the scope of the Advanced Machine Learning course. Despite the various methods available for implementing machine learning, this assignment focuses on exploring Artificial Neural Networks (ANNs) for image classification, specifically traffic sign images.

Therefore, the primary goal is to examine the dataset and develop a machine learning method utilizing neural networks as models capable of conducting supervised image classification on the dataset. For this purpose, we will use the dataset provided by the responsible Professor. Here will be discussed two ANNs models, Multi-Layer Perceptrons (MLP) and Convolutional Neural Networks (CNNs), with different layer architectures, loss functions, optimizers, and hyperparameters. The models will be evaluate and discussed later in this report.

2 Experimental Setup

2.1 Dataset Description

The dataset utilized in this project is derived from the "Traffic Sign Dataset", originally comprising 54 distinct classes. However, for the purposes of this study, only 9 classes will be analyzed. These classes are as follows:

- 6 - Speed limit (70km/h).
- 12 - Don't go left or right
- 13 - Don't go right
- 24 - Go right
- 37 - Children crossing
- 38 - Dangerous curve to the right
- 39 - Dangerous curve to the left
- 44 - Go left or straight
- 50 - Fences

The images within the dataset have a resolution of 75x75 pixels and are organized into two main folders: "Train" and "Test". The "Test" folder, containing 310 unlabeled images, was specifically allocated for evaluating the performance of the models on Kaggle . On the other hand, the "Train" folder, comprising 518 images, is subdivided into folders corresponding to each class. These images were utilized for training and validating the models developed.

Given the restricted size of our dataset one of the most relevant challenges will be that of over-fitting. To combat this a pre-processing and data augmentation pipeline was implemented. Pre-processing of the

images was executed using the torchvision and rembg libraries, involving as a first step background removal and then image resizing to make all images have a consistent size of 75x75 pixels. The background removal step was implemented in a function as a custom "transform" so that it could be seamlessly implemented into the `torch.utils.data.DataLoader` pipeline. This was necessary as these image processing steps must be applied to the images that we need to predict even after the model is trained.

After these steps the dataset was pre-processed and ready to undergo data augmentation. Four different transformation functions were implemented: RandomPerspective(), RandomAffine(), ColorJitter(), RandomAdjustSharpness(). This was done in an attempt to increase variety in our training data, and was shown to have a positive impact on the final results.

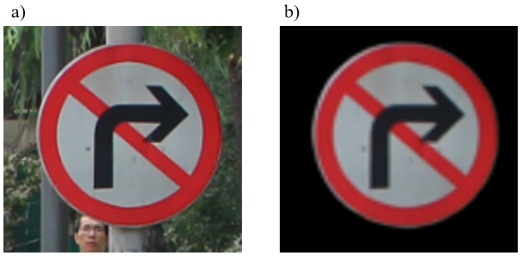

Fig. 1: (a) Original Image; (b) Image with background removed.

An enlargement factor was also defined to control the size of the final available dataset. Different values were tested but the best compromise in terms of results and processing time was achieved with the augmented dataset having 6 times the original size with a final number of 3108 available images for the CNN case and 2 times for the MLP resulting in 1036 images.

Fig. 2: Examples of augmented images

The dataset was partitioned into training, validation, and test subsets. The test dataset served as an initial evaluation to gauge the model's generalization performance before testing with the images from the "Test" folder on Kaggle. A split ratio of 0.4 was applied for this purpose.

For both MLP and CNN, the initial model, which will be introduced in the following section, was evaluated using the original image, normalized image, and normalized image with no background. The findings revealed a significant improvement when utilizing normalized images with background removed. Therefore, the presented results are based on those.

2.2 Model Design and Evaluation

Different combinations of hyperparameters and architectures were tested and filtered based on their impact on the results, which involved experimentation and evaluation across Multilayer Perceptron (MLP) and Convolutional Neural Network (CNN) models. Considering an initial model as a simple architecture, employing Cross Entropy as the loss function, Adam Optimizer for parameter adjustment, and ReLU activation functions, comprising fewer layers and standard hyperparameters tailored to the task, the chosen

approach entailed iteratively examining more complex models and diverse hyperparameter configurations to assess their impact on the outcomes. Regularization Techniques were utilized in different configurations in order to see how well it was improving the generalization of the model, i.e. Dropouts, Batch Normalization (CNN), and L2 Regularization (MLP).

For Multilayer Perceptron architectures, we devised four distinct designs. By factoring in the various combinations of hyperparameters, we generated a total of 32 models for each architecture (refer to Tables 1). As for the Convolutional Neural Network, we initially explored 72 potential configurations(refer to Table 2) across 16 different architectures. These configurations varied in the number of convolutional layers, ranging from two to eight, and included differing combinations of pooling layers, dropout rates, batch normalization, and fully connected layers. However, these options underwent further refinement during the early stages of our analysis and will not be mentioned in this report. Only those deemed noteworthy are presented in the next section. The results, and the details of the process, will be discussed in the next section.

Multi-Layer Perceptrons

Hyperparameters	Values
BATCH_SIZE	32, 64
Folds	10
Learning Rate	0.001, 0.001
L2 Regularization	0.01, 0.001
Epochs	50, 100
Architecture	**Choices**
Nº of hidden layers	2, 4
Size of hidden layers	64, 128
Dropout rate	none, 0.1

Table 1: Hyperparameters and Architectures for MLP

Convolutional Neural Network

Hyperparameters	Values
BATCH_SIZE	16, 32, 64
Folds	5, 10
Learning Rate	0.01, 0.001, 0.0001
Epochs	30, 50, 100
Dropout rate	0.2, 0.5

Table 2: Hyperparameters for MLP

3 Experimental results

3.1 MLP

The table below presents a selection of achieved results, comprising 7 out of 80 trained models. These selections were made to better illustrate the influence of various hyperparameters and architectures on the outcomes. Each model underwent testing via a 10-fold cross-validation process, with one model trained for each fold. Subsequently, the model exhibiting the highest validation accuracy across the folds was chosen for testing on the independent test set.

From the results, we can observe that the best test accuracy achieved was 73.9%, obtained with a combination of factors including a batch size of 32, 100 epochs, 10 folds, a dropout rate of 0.1, a learning rate of 0.0001, L2 regularization of 0.01, four hidden layers, each with a size of 128 neurons. Interestingly, the number of hidden layers seems to play a significant role. The dropout rate and L2 regularization also appeared to impact the model's performance.

Considering the four tested architectures as follows: (v1) featuring two hidden layers with Dropout implemented between them; (v2) comprising two hidden layers without dropout; (v3) incorporating four hidden layers with dropout; and (v4) utilizing four hidden layers without dropout. Knowing that the size of the architecture was considered as a hyperparameter (subject to change), the average accuracy across all possible combinations for each architecture is: 60%, 64%, 54%, 62%, respectively

Hyperparameters	Enlarge Factor	1	1	1	1	1	1	1
	BATCH_SIZE	64	64	64	32	32	32	32
	Epochs	50	50	50	50	100	100	100
	Folds	10	10	10	10	10	10	10
	Dropout rate	0.1	0.1	-	-	-	-	-
	Learning rate	0.001	0.001	0.001	0.001	0.001	0.0001	0.0001
	L2 Regularization (λ)	0.01	0.01	0.01	0.01	0.01	0.001	0.01
MLP	Nº of Hidden layers	2	4	4	2	4	4	4
	Size of the hidden layers	128	128	128	64	64	128	128
Evaluation								
Test Accuracy (%)		60.6	57.4	65.8	68.1	72.2	64.9	73.9

Table 3: Main Results for MLP models

From the analysis provided, it's evident that different architectural configurations yielded varying average accuracies. Architecture v2, comprising two hidden layers without dropout, achieved the highest average accuracy of 0.64. Conversely, architecture v3, featuring four hidden layers with dropout, had the lowest average accuracy of 0.54. Interestingly, the inclusion of dropout seemed to negatively impact the performance of architectures v1 and v3 compared to their dropout-free counterparts (v2 and v4, respectively), possibly due to redundancy with L2 regularization. Additionally, it's noteworthy that the configuration with the highest accuracy was observed in the model with four hidden layers and no dropout. However, the model with two hidden layers consistently displayed greater stability and higher average accuracy across all hyperparameter combinations. This suggests that the number of hidden layers may play a crucial role in determining the model's performance and stability.

The graph below (figure 3) demonstrates the impact of optimizers, loss functions, and activation functions on the final model's accuracy. Notably, utilizing ReLU activation with Cross Entropy loss and RMSprop optimizer resulted in the highest accuracy, peaking at 61.8%. Close behind is the combination of Cross Entropy loss and Adam optimizer, our initial preference, which achieved an accuracy of 60.4%. However, performance notably declined with alternative activation and loss functions. These findings emphasize the crucial importance of selecting appropriate activation and loss functions, along with optimizers, to enhance the performance of the MLP model in image classification tasks.

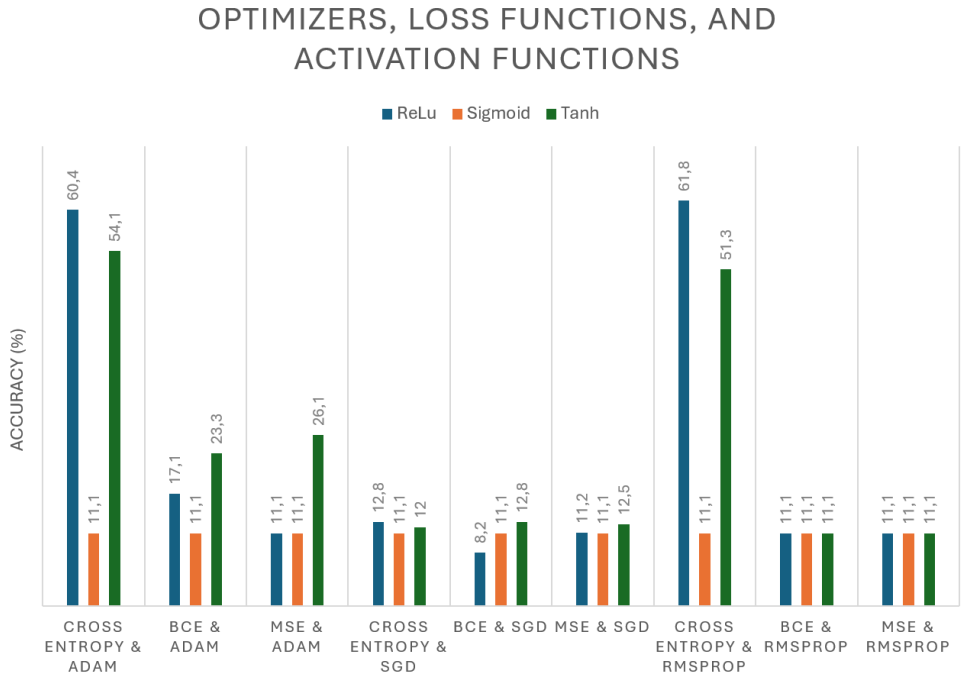

Fig. 3: Optimizers, loss functions, and activation functions for MLP

3.2 CNN

The table below presents a selection of achieved results, highlighting 5 selected from over 150 trained models. As in the preceding section, these models were specifically chosen to illustrate the impact of various architectures and hyperparameters.

Hyperparameters	Enlarge factor	5	5	5	5	5
	BATCH_SIZE	32	32	64	32	32
	Folds	10	5	5	10	10
	Epochs	50	50	20	30	30
	Dropout rate	0.5	0.5	-	0.5	-
	Learning rate	0.001	0.001	0.001	0.0001	0.001
Models		V2	V4	V6	V6_d	V6_BN
Evaluation						
Test Accuracy (%)		90.7	92.6	93.4	92.6	97.3
Kaggle Score		0.62	0.65	0.71	0.67	0.67

Table 4: Main Results for CNN models

- V2 Model: This model comprises two convolutional layers with batch normalization applied between them. The first convolutional layer has an output channel of 32, while the second has 16. Additionally, a MaxPooling layer with a kernel size of 2 follows the first convolutional layer. The model includes a single fully connected layer (excluding the output layer) with 128 neurons, and a dropout rate of 0.5 is before the output layer.
- V4 Model: In contrast, the V4 model features four convolutional layers with batch normalization between each layer. The output channels for these layers are 64, 32, 16, and 8, respectively. Similar to the V2 model, a MaxPooling layer with a kernel size of 2 follows the first convolutional layers. The model then incorporates two fully connected layers (excluding the output layer) with 256 and 128 neurons, respectively. A dropout rate of 0.5 is applied between these fully connected layers to prevent overfitting.
- V6 Model: This model features six convolutional layers with batch normalization applied between each layer. The output channels for these layers are 16, 32, 64, 64, 32, and 16, respectively. Three MaxPooling layers with a kernel size of 2x2 are inserted after the first, third, and last convolutional layers. Additionally, a fully connected layer with 120 neurons precedes the output layer.
- V6_d Model: In comparison to the V6 model, a dropout layer with a rate of 0.5 is introduced before the output layer in this model.
- V6_BN Model: Similar to the V6 model, this architecture also comprises six convolutional layers with batch normalization between each layer. However, the output channels for these layers are 20, 18, 16, 14, and 8, respectively.

In all models, the convolutional layers maintain the input image dimensions with kernel sizes of 3x3 and padding of 1. Following each convolutional layer, a ReLU activation function is applied to introduce non-linearities to the network.

Overall, the findings suggest that models with deeper architectures and batch normalization tend to yield superior performance, particularly when utilizing smaller batch sizes and sufficient training epochs. Notably, V6 achieved an accuracy of 93.4% on the test set with a Kaggle score of 0.71, while V6_BN attained a test set accuracy of 97.3% with a Kaggle score of 0.67. Interestingly, V6_d, incorporating a dropout rate of 0.5, demonstrated comparable performance. This underlines the minimal impact of dropout regularization on model performance in this context, contrary to expectations. Except for the case where the learning rate was set to 0.01, resulting in poor performance, the remaining values appeared suitable for the task, with negligible differences observed among them.

Although there are subtle differences among the presented models, they all exhibit signs of overfitting. This tendency can likely be attributed to the limited diversity within the images from the "Train" folder, leading to high accuracy during training-validation splits but comparatively lower performance when tested on images from the "Test" folder, as evaluated on Kaggle. Despite the preprocessing steps implemented partially mitigate this issue, they do not eliminate the problem.

In addition to the tests outlined previously, we also experimented with implementing a voting system, which aggregated predictions from the top-performing models across various architectures. This approach was motivated by the recognition that models of differing complexities may capture distinct features, and by combining their diverse perspectives, we aimed to enhance overall performance and generalization capabilities. The results, however, did not show any significant improvement.

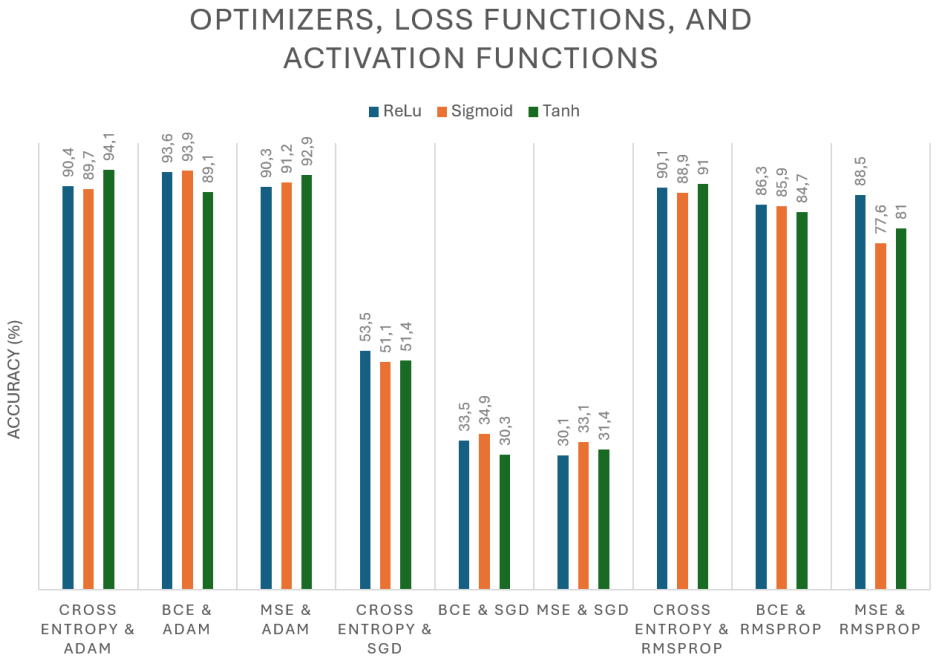

Fig. 4: Optimizers, loss functions, and activation functions for CNN

The graph above (figure 4) demonstrates the impact of optimizers, loss functions, and activation functions on the final model's accuracy. It is evident that the choice of loss function and optimizer significantly impacts the model's performance. For instance, when paired with the Adam optimizer, the Cross Entropy loss function consistently yielded high accuracies across all activation functions, with Tanh activation achieving the highest accuracy of 94.1%. The initial model defined, using Adam, Cross Entropy and ReLu, also achieved a notable result with 90.4% of accuracy. On the other hand, using the SGD optimizer generally resulted in lower accuracies across all loss functions and activation functions, indicating challenges with convergence.

4 Discussion and Conclusion

Multilayer Perceptrons are frequently utilized in image classification tasks, serving as classifiers following image feature extraction. Typically, images undergo preprocessing and are flattened into vector representations, with each element representing a pixel or image feature. However, MLPs face limitations in capturing spatial information within images, prompting the widespread adoption of convolutional neural networks (CNNs) for such tasks.

Unlike MLPs, CNNs leverage convolutional and pooling layers, enabling direct learning of hierarchical feature representations from raw pixel data. This architecture not only reduces parameters but also ensures translational invariance, rendering CNNs more robust and effective for image classification. Consequently, while MLPs yield expected results, CNNs demonstrate significantly improved performance due to their inherent ability to extract features without explicit feature engineering during preprocessing.

The results for both MLP and CNN were as expected. The convolutional neural networks tested showed much better performance for the proposed task than the MLP, despite differences in model complexity and hyperparameters. The issues regarding the model's generalization capability can primarily be attributed to the lack of variety in the images of the dataset used for training. Effective normalization techniques did not appear to have a significant impact, either positively or negatively, on this matter.

5 References

1. Battiti, R., Colla, A.M.: Democracy in neural nets: Voting schemes for classification (1994)
2. Lu, J., Huang, Y., Xu, Y. : Image Classification Using Multilayer Perceptron.
3. Wang, B.: Research on the Optimal Machine Learning Classifier for Traffic Signs (2022)
4. REMBG, Daniel Gatis, Github repository (2024), `https://github.com/danielgatis/rembg`. Last accessed 01 April 2024