MDPI

Article

Image Classification - CIFAR

Alberto Ramos [1,†,‡], Nuno Azevedo [2,‡] [2,*]

[1] Alberto Ramos; 1221165@isep.ipp.pt
[2] Nuno Azevedo; 1221167@isep.ipp.pt
[†] These authors contributed equally to this work.

Abstract: This work compares deep learning and classical machine learning methods for image classification on CIFAR-10 and CIFAR-100. Two CNN's of different complexity were implemented and regularized. Classical classifiers were trained on features extracted from the CNN's, with and without PCA. Results show that the complex CNN achieved good accuracy on CIFAR-10 but struggled on CIFAR-100, highlighting the challenge of this dataset. Classical classifiers performed worse than the neural networks' fully connected layers, but improved with better features and PCA, which also reduced training time. Pre-trained models set a clear state-of-the-art reference. The study confirms the importance of model complexity and feature quality, and shows that even standard datasets like CIFAR-100 remain difficult for custom models.

Keywords: image classification; convolutional neural networks; CIFAR-10; CIFAR-100; deep learning; classical classifiers; feature extraction; PCA; regularization; PyTorch

1. Introduction

This work is part of the Applied Artificial Intelligence course, taught in the context of the Bachelor's Degree in Electrical and Computer Engineering , and focuses on the exploration and comparison of different approaches to image classification using machine learning techniques. By employing the CIFAR datasets, widely recognised in academic research and commonly used as benchmarks for evaluating model performance, this project aims to implement, train, and evaluate a variety of models, analysing the impact of different preprocessing strategies, architectural choices, and training configurations.

The methodology adopted is experimental and iterative, emphasising the practical application of supervised learning. Multiple model types and training strategies are tested and compared to assess their relative performance, with particular attention given to the role of hyperparameters, data augmentation, and network depth or complexity. The goal is to understand how each design choice influences classification accuracy and model generalisation.

Image recognition, the broader field to which this project belongs, is a central task in computer vision. It involves identifying and classifying objects or patterns within digital images and has become increasingly relevant in areas such as autonomous vehicles, medical imaging, industrial inspection, and security systems. Advances in deep learning and the availability of large-scale labelled datasets have led to significant improvements in model accuracy and robustness in recent years.

This report documents the entire process, from data preparation to model evaluation, and provides a critical analysis of the results obtained. It is intended to offer a practical and comprehensive overview of the key concepts and challenges involved in building image recognition systems using modern machine learning techniques.

Received:
Revised:
Accepted:
Published:

Citation: Lastname, F.; Lastname, F.; Lastname, F. Title. *IAAPLI* **2025**, *1*, 0.

2. Tools and Materials

2.1. The CIFAR Datasets

The CIFAR-10 and CIFAR-100 datasets (Canadian Institute for Advanced Research) are among the most widely used resources in the field of computer vision, particularly for image classification tasks. Both datasets were developed by researchers at the University of Toronto, most notably Alex Krizhevsky and Geoffrey Hinton, with the aim of providing standardized, accessible, and sufficiently challenging databases for training and evaluating deep learning algorithms, especially CNN's [2]. These datasets were derived from the larger Tiny Images dataset, developed at MIT [3].

Both CIFAR-10 and CIFAR-100 consist of 60,000 color images with a resolution of 32×32 pixels in RGB format (three channels: red, green, and blue). The low resolution enables efficient use in computationally constrained environments without compromising the visual diversity of the data. In both cases, the datasets are split into two subsets: 50,000 images for training and 10,000 for testing, ensuring a clear separation between the learning and evaluation phases [2].

The primary difference between the two lies in the granularity and complexity of the classification task:

- **CIFAR-10** contains 10 distinct classes, with 6,000 images per class. These categories are broad and well-defined, covering objects such as *airplane*, *automobiles* and *trucks*. The balanced distribution of images across classes is detailed in Table 1, and it facilitates direct performance comparisons across different models.
- **CIFAR-100**, on the other hand, features a significantly more detailed structure, with 100 classes and 600 images per class. These are grouped into 20 superclasses, which cluster semantically related categories such as vehicles, animals, instruments, or plants. Table 2 presents this hierarchical structure, showing the relationship between superclasses and their corresponding classes, along with the respective number of training and test images.

Both datasets are fully compatible with modern machine learning libraries such as `TensorFlow` and `PyTorch`, and are frequently used as benchmarks in academic research and the development of new neural network architectures.

Table 1. CIFAR-10: Image distribution by class in training and test sets.

Class	Test Images	Train Images	Total Images
airplanes	1.000	5.000	6.000
automobiles	1.000	5.000	6.000
cats	1.000	5.000	6.000
cats	1.000	5.000	6.000
deer	1.000	5.000	6.000
dogs	1.000	5.000	6.000
frogs	1.000	5.000	6.000
horses	1.000	5.000	6.000
ships	1.000	5.000	6.000
trucks	1.000	5.000	6.000
Total	**10.000**	**50.000**	**60.000**

Table 2. CIFAR-100: Image distribution by superclass and class.

Superclass	Class	Test Images	Train Images	Total
aquatic mammals	beaver, dolphin, otter, seal, whale	100	500	600
fish	aquarium_fish, flatfish, ray, shark, trout	100	500	600
flowers	orchids, poppies, roses, sunflowers, tulips	100	500	600
food containers	bottles, bowls, cans, cups, plates	100	500	600
fruit and vegetables	apples, mushrooms, oranges, pears, sweet_peppers	100	500	600
household electrical	clock, keyboard, lamp, telephone, television	100	500	600
household furniture	bed, chair, couch, table, wardrobe	100	500	600
insects	bee, beetle, butterfly, caterpillar, cockroach	100	500	600
large carnivores	bear, leopard, lion, tiger, wolf	100	500	600
large man-made	bridge, castle, house, road, skyscraper	100	500	600
large natural	cloud, forest, mountain, plain, sea	100	500	600
people	baby, boy, girl, man, woman	100	500	600
reptiles	crocodile, dinosaur, lizard, snake, turtle	100	500	600
small mammals	hamster, mouse, rabbit, shrew, squirrel	100	500	600
trees	maple_tree, oak_tree, palm_tree, pine_tree, willow_tree	100	500	600
vehicles 1	bicycle, bus, motorcycle, pickup_truck, train	100	500	600
vehicles 2	lawn_mower, rocket, streetcar, tank, tractor	100	500	600
medium-sized mammals	fox, porcupine, possum, raccoon, skunk	100	500	600
non-insect invertebrates	crab, lobster, snail, spider, worm	100	500	600
small man-made	baby, bottle, bowl, can, clock	100	500	600
Total		**10.000**	**50.000**	**60.000**

* Each superclass contains five distinct classes, with the values presented per class.

2.2. Libraries Used

This project relied on a set of open-source libraries widely used in the fields of data science and machine learning, with the objective of facilitating the loading, processing, training, evaluation, and visualization of classification models applied to the CIFAR-10 and CIFAR-100 datasets. The main libraries and modules employed are described below.

`Matplotlib` was used to create graphical visualizations that enabled monitoring of the models' evolution throughout the training process and comparison of their performance. This library proved particularly useful for representing metrics such as accuracy, loss per epoch, and performance comparisons across different classifiers, providing a clear and informative visual analysis of the results.

`Scikit-learn` played a central role in the construction and evaluation of traditional classification models. With a broad collection of ready-to-use algorithms, it allowed the implementation of models in a straightforward and efficient manner. In addition to model creation, `Scikit-learn` was also employed in their evaluation, particularly through the computation of prediction accuracy. Furthermore, it was used for applying dimensionality reduction techniques such as PCA, improving data interpretability and, in some cases, computational performance.

`PyTorch` was the library chosen for the implementation and training of convolutional neural networks. It is a powerful and flexible deep learning framework widely adopted in research and development. This library enabled the definition of network architectures, activation functions, loss functions, and optimization procedures. Its support for GPU computation significantly accelerated model training. Additionally, tools from `PyTorch` were used for efficient data loading in batches and for the creation of custom datasets adapted to the structure of the CIFAR datasets.

Complementing `PyTorch`, the `Torchvision` library provided essential tools for image preprocessing. Transformations such as normalization and data augmentation were applied with the goal of improving the generalization capabilities of the models and simulating variations in the input data.

Finally, `Pickle` was used for loading the CIFAR datasets, which are stored in binary format. This library supports serialization and deserialization of Python objects, making it particularly useful for reading the original dataset files and converting them into manipulable structures within the project environment.

3. State of the Art

Over the past decades, the CIFAR-10 and CIFAR-100 datasets have become fundamental benchmarks for evaluating image classification algorithms. The current consensus in the literature is clear: CNN's continue to lead in performance on these datasets and are widely adopted as the standard of excellence in top-tier conferences such as *CVPR*, *NeurIPS*, and *ICML*. Although traditional methods like logistic regression, SVM's, k-NN, or Random Forests still appear in the literature, they are typically used as baselines or in scenarios with severely limited computational resources. Their performance, while competitive in some domains, quickly saturates when faced with the complexity and high dimensionality of images in CIFAR.

Several CNN-based architectures have demonstrated state-of-the-art results on these benchmarks, as summarized in Table 3. Notable examples include `TResNet` [10], designed for GPU efficiency, and `EfficientNet` [12], which employs a compound scaling method to balance network width, depth, and resolution. Among classical models, `ResNet` [13] introduced the revolutionary concept of residual connections, enabling the effective training of very deep networks. Variants such as `ResNet-18` are still widely used today due to their balance of performance and efficiency.

Other influential models include `WideResNet` [14], which favors wider layers over deeper ones to improve learning dynamics, and `DenseNet` [15], which encourages feature reuse through dense connections between layers. Despite being compact, these networks consistently outperform traditional methods. Lightweight CNN's can already surpass 80% accuracy on CIFAR-10, while non-deep approaches like SVM or k-NN rarely exceed 14% [16].

In summary, although traditional classifiers still hold relevance in didactic contexts or where interpretability is key, it is clear that convolutional models continue to define the state of the art in natural image recognition. Their performance far surpasses that of shallow methods, making them the preferred choice in most modern classification systems.

Table 3. Accuracy of state-of-the-art models on CIFAR-10 and CIFAR-100.

Model	CIFAR-10 Accuracy	CIFAR-100 Accuracy
TResNet-XL	99.0%	91.5%
EfficientNet-B7	–	91.7%
ResNet-164 v2	93.4%	–
WideResNet-28-10	96.2%	81.5%
DenseNet-BC	92.9%	72.2%

4. Data Preprocessing

For model development, the CIFAR-10 and CIFAR-100 datasets were manually downloaded from the official author's website[1], rather than using the automatic loaders provided by libraries such as `torchvision`. This approach allowed for more direct control over the data preparation pipeline and facilitated the introduction of an explicit validation stage, which is not included by default in the datasets.

[1] https://www.cs.toronto.edu/~kriz/cifar.html

In the case of CIFAR-10, which is distributed across five training batches and one test batch, one of the training batches was set aside exclusively for validation (16.5% of the total data). The remaining four batches were used for training, while the test batch remained unchanged. For CIFAR-100, whose training data is provided as a single file, an 80/20 split was applied to create separate training and validation sets.

The files were loaded using the `pickle` library, which handles the serialization and deserialization of Python objects. Each file represents a dictionary containing image data (as NumPy arrays) and their corresponding labels. After loading, the data was converted into three-dimensional NumPy arrays with shape (N, 3, 32, 32), preserving the RGB color channels and original image dimensions. Figure 1 provides examples of preprocessed images and their associated class labels, illustrating the pixel-based structure of the dataset.

To enable the use of data augmentation transformations provided by `torchvision`, the image arrays were converted to `PIL.Image` format during the data loading process. This step is required because most of the augmentation operations in PyTorch expect input images in this format.

A custom class derived from `torch.utils.data.Dataset` was implemented to enable compatibility with PyTorch. This allowed seamless integration with the `DataLoader` interface, as well as the application of real-time preprocessing transformations during training.

This type of manual pipeline is common in projects that require greater flexibility in handling input data and follows practices frequently recommended in official PyTorch documentation [9].

(**a**) Example of the class *cat* (**b**) Example of the class *truck*

Figure 1. Examples from the dataset used in the project. (**a**) Image corresponding to the class *cat*. (**b**) Image corresponding to the class *truck*.

5. Development

The main objective of this work was to compare different machine learning approaches and techniques, assessing their effectiveness in the context of image classification. In particular, CNN's were implemented from scratch, with explicit definition of all layers. Throughout the project, the complexity of the architectures was progressively increased in order to investigate the impact of architectural changes on model performance. Anticipating the risk of *overfitting*, especially in simpler models, various regularization techniques were introduced, such as data augmentation and dropout, aiming to improve generalization. The performance of the developed CNN's was then compared to that of traditional machine learning classifiers, validating the advantages of deep learning methods in this domain.

Finally, the implemented CNN's were also compared to state-of-the-art pre-trained models, allowing for a critical analysis of their relative performance and complexity.

5.1. CNN Architectures

Convolutional Neural Networks (CNN's) are a class of deep learning models particularly well-suited for image data, due to their ability to extract and learn spatial hierarchies of features directly from the raw pixels. In this project, the implemented CNN architectures incorporate a variety of commonly used layers. `Conv2d` layers apply a set of learnable filters over local regions of the input image to extract low-level and high-level features. `ReLU` (Rectified Linear Unit) activation functions introduce non-linearity, allowing the network to model complex patterns. `BatchNorm2d` layers normalize intermediate activations, accelerating training and improving stability. `MaxPool2d` layers reduce the spatial dimensions of the feature maps, contributing to translation invariance and helping to control overfitting. After convolutional processing, a `Flatten` layer reshapes the multi-dimensional feature maps into a 1D vector, which is then passed through one or more `Linear` (fully connected) layers that perform the final classification. In deeper architectures, `Dropout` is also employed as a regularization technique that randomly deactivates neurons during training, further mitigating overfitting. These components, when combined, form the basis of powerful CNN architectures capable of learning from complex visual data [11].

The `SimpleCNN` model was implemented as a lightweight baseline to assess the capabilities of a minimal convolutional architecture. It comprises two convolutional layers with a small number of filters, each followed by `ReLU` activation and `MaxPool2d` operations, progressively reducing the spatial resolution while increasing feature abstraction. After the convolutional stages, the output is flattened and passed through two fully connected layers, culminating in a final output layer that maps to the number of target classes. This compact design offers a useful reference point for evaluating the effect of increased model depth and complexity. The full structure is depicted in Figure 2.

In contrast, the `ComplexCNN` model was implemented to explore a deeper and more regularized architecture. It consists of three convolutional blocks, each containing two convolutional layers with `BatchNorm2d` and `ReLU` activations, followed by `MaxPool2d` layers to downsample the spatial dimensions. The feature extractor is followed by a three-layer fully connected head, with `Dropout` regularization applied after each hidden layer to mitigate overfitting. The final layer outputs class probabilities for the task at hand. This more complex architecture enables the model to learn hierarchical and abstract features with greater representational power. The full architecture is shown in Figure 3.

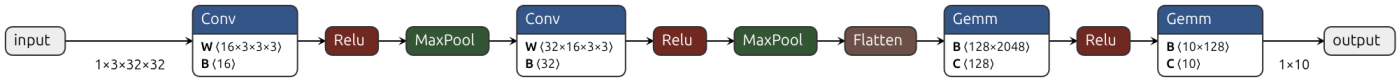


Figure 2. Simple CNN Architecture

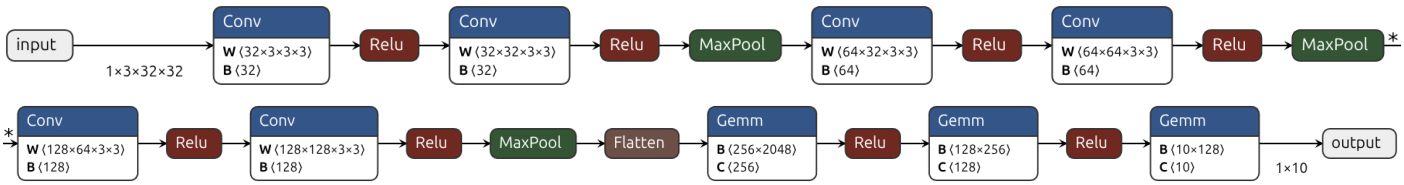


Figure 3. Complex CNN Architecture
* The figure is divided into 2 parts

5.2. Training Parameters and Overfitting Mitigation

The training process was designed to be efficient, reproducible, and consistent across all experiments. The input data, already preprocessed and organized into training, validation, and test sets as described in section 4, was passed to a custom dataset class implementing the `torch.utils.data.Dataset` interface. This enabled the integration of real-time transformations during training and evaluation.

During training, the data was loaded in mini-batches of size 32, with `shuffle` enabled to improve the statistical diversity of each batch. A supervised learning approach was employed, with the model trained for a predefined number of epochs using standard mini-batch gradient descent. The loss function used was `CrossEntropyLoss`, which is appropriate for multi-class classification tasks. Model parameters were updated using the `Adam` optimizer, a widely used adaptive optimization algorithm that tries to ensure stable and efficient convergence.

Model performance was evaluated after each epoch using the validation set, allowing for the monitoring of learning progress and early detection of overfitting. Upon completion of training, the model was evaluated on an independent test set to estimate its generalization performance.

5.2.1. Overfitting and Mitigation Strategies

Due to the relatively low number of parameters in the implemented convolutional networks, overfitting was expected to be a significant concern. To address this, multiple regularization techniques were incorporated into both the training process and model architecture.

- **Data Augmentation:** To increase the effective size and diversity of the training set, a set of stochastic image transformations was applied at training time. These included random cropping with padding and horizontal flipping. Such augmentations introduce variability and help prevent the model from overfitting to specific training examples [17].
- **Dropout:** Dropout layers were added after certain convolutional and fully connected layers. This technique randomly disables a subset of neurons during training, promoting redundancy in feature representations and reducing co-adaptation.
- **Batch Normalization:** In deeper architectures, batch normalization layers were included after convolutional operations. These layers normalize activations across mini-batches, stabilizing training dynamics and serving as an implicit form of regularization.
- **Weight Decay:** L2 regularization was introduced through the optimizer's weight decay parameter. This penalizes large parameter values and encourages simpler, more generalizable solutions.

Together, these mitigation strategies were intended to improve generalization and reduce the performance gap between training and validation phases.

5.3. Classical Classifiers

To complement the CNN's developed in this work, a set of classical machine learning classifiers was applied to the deep features extracted from the trained CNN models. This methodology allows for the evaluation of the discriminative capacity of the learned features, independently of the CNN's final classification layer. Moreover, by providing the same feature representation to multiple classifiers, it becomes possible to assess how much the performance depends on the classifier itself versus the quality of the learned features.

In addition, Principal Component Analysis (PCA) was applied as a dimensionality reduction technique to the extracted features prior to classification. PCA transforms the

feature space by projecting it onto a smaller set of orthogonal components that capture the maximum variance in the data. This reduces feature dimensionality and potential noise, which can improve classifier efficiency and generalization.

The classifiers selected represent a variety of modelling assumptions and complexities, enabling a broad evaluation of the feature space.

Logistic Regression

Logistic Regression is a linear classification model that estimates the probability of class membership through the logistic function. Despite its simplicity, it is widely used due to its interpretability and efficiency.

Its inclusion in this context allows to assess whether the feature space produced by the CNN's is linearly separable. If good performance is achieved with this model, it suggests that the CNN was able to learn features that are well-organized in terms of class boundaries.

Random Forest

Random Forest is an ensemble method composed of multiple decision trees trained on random data and feature subsets. It combines the outputs of individual trees to improve generalization and reduce overfitting.

Its use here is justified by its robustness and ability to handle complex, non-linear patterns without requiring extensive hyperparameter tuning. Applying it to deep features enables a comparison with simpler linear models and an evaluation of the richness of the learned representations.

K-Nearest Neighbors (KNN)

KNN is a non-parametric method that classifies a sample based on the majority label among its K nearest training examples. It relies on a distance metric (typically Euclidean) to define proximity in the feature space.

This classifier is particularly well-suited for evaluating the geometric structure of the deep feature space. If CNN's group semantically similar inputs close together, KNN can perform well, despite not involving a learned decision function.

Support Vector Machine (SVM)

The Support Vector Machine constructs decision boundaries that maximize the margin between classes. It supports both linear and non-linear classification through kernel functions.

Due to the high dimensionality of deep features, SVM's are well-suited for this context. Their ability to find robust separating hyperplanes makes them a strong benchmark for evaluating the separability of the learned embeddings, especially with non-linear kernels like RBF (Radial Basis Function).

Multi-Layer Perceptron (MLP)

The MLP is a fully connected feedforward neural network that uses non-linear activation functions to model complex decision boundaries. It is trained using backpropagation and offers more flexibility than traditional classifiers.

Applying an MLP to deep features helps assess whether additional non-linearity beyond the CNN improves performance by capturing patterns that simpler models might miss

5.4. Pre-trained Models

For the CIFAR-10 dataset, pre-trained convolutional models made available in the `PyTorch_CIFAR10` repository by Huy V. Phan [1] were used. This collection includes well-

established architectures such as `ResNet-18`, `DenseNet-121`, and `VGG-19`, all previously trained on the same dataset. These models provide a solid performance reference, leveraging prior training to perform classification directly on the test set.

The use of these pre-trained models proved effective both in saving computational resources and in establishing a robust baseline using state-of-the-art architectures.

6. Results and Discussion

To ensure a fair and consistent comparison between all classifiers and architectures evaluated, the same training configuration was maintained throughout the experiments. All models were trained for **200 epochs**, with no use of early stopping. This choice ensured that any signs of overfitting could emerge naturally, offering insights into the generalization capacity of each technique.

The optimizer used was Adam, with a learning rate fixed at 0.001, and mini-batches of size 32. The loss function adopted was cross-entropy, suitable for multi-class classification tasks.

This uniform setup guarantees that observed differences in performance arise from the models and feature representations themselves, rather than from inconsistencies in the training process.

The results are organized into three main sections corresponding to the simple CNN, the complex CNN, and the pre-trained models. Within each CNN section, results are presented separately for each overfitting correction technique and dataset (CIFAR-10 and CIFAR-100). The pre-trained models are evaluated only on CIFAR-10 due to the availability of pre-trained weights.

6.1. Simple CNN

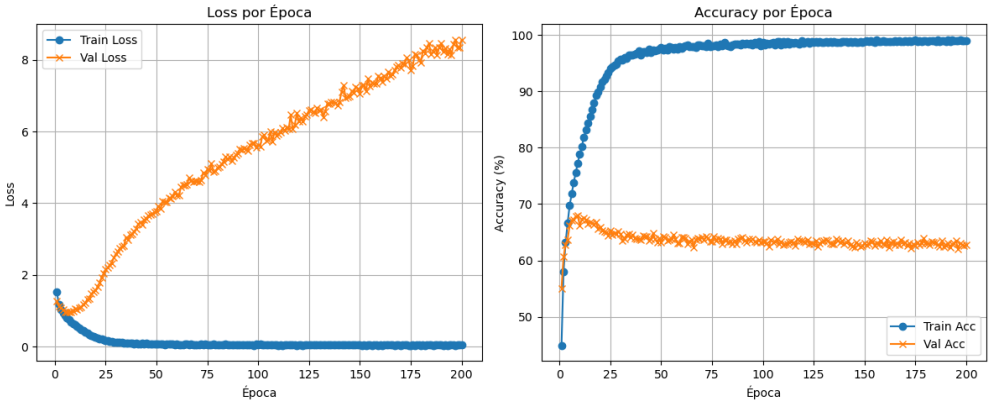
Figure 4(**a**) to Figure 4(**d**) present the training and validation loss and accuracy curves over 200 epochs for the simple CNN, both without and with overfitting correction techniques, on the CIFAR-10 and CIFAR-100 datasets.

Without regularization (Figures 4(**a**) and 4(**c**)), a clear overfitting phenomenon is observed. For CIFAR-10, the training loss quickly drops to near zero, while the validation loss initially decreases but then rises continuously, exceeding 8 by the end of training. The training accuracy reaches 99%, while validation accuracy peaks around 67% before declining to about 63%.

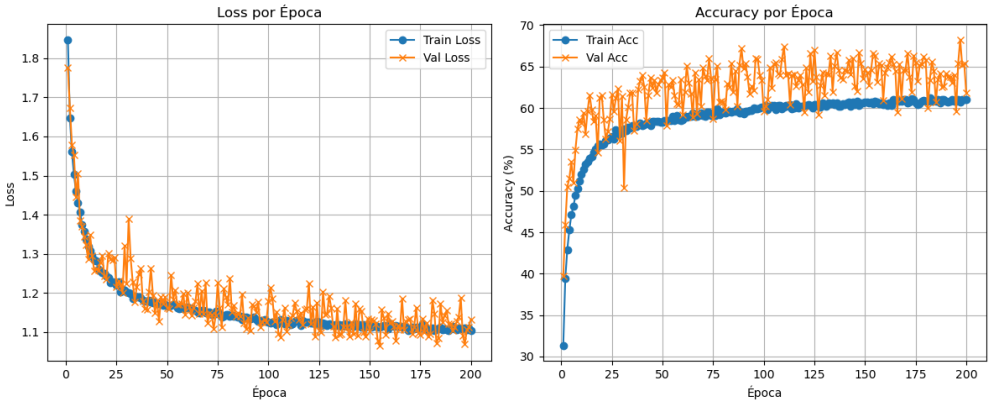
The effect is even more pronounced on CIFAR-100, where training loss again approaches zero, but validation loss escalates above 22. Training accuracy nearly hits 100%, while validation accuracy peaks at 33% before falling to approximately 26%. This demonstrates the model's limited generalization capacity, especially with the added complexity of CIFAR-100.

To address overfitting, regularization methods were applied: dropout with a probability of 0.45, weight decay of 7.5×10^{-4}, and data augmentation via random cropping and horizontal flipping. The impact of these techniques is visible in Figures 4(**b**) and 4(**d**). For CIFAR-10, both training and validation losses stabilize around 1.1, with training accuracy at 60% and validation accuracy fluctuating between 60% and 67%, indicating improved generalization.

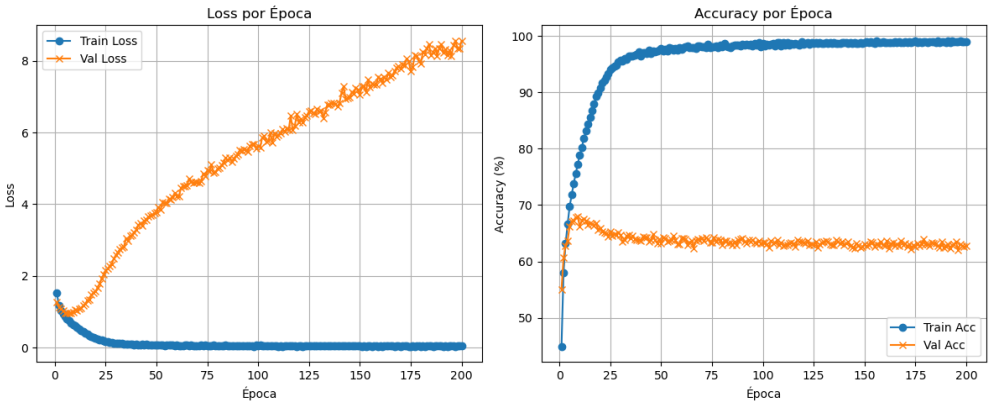
Similarly, in CIFAR-100, both losses converge near 3.0–3.2, and training and validation accuracies stay close, between 25% and 32%, showing that the regularized model avoids memorization and generalizes better despite the dataset's difficulty.



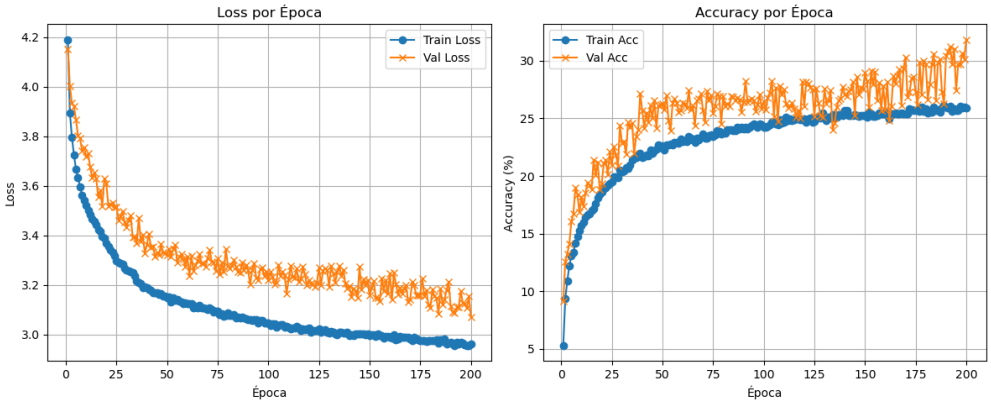
(**a**) Loss and Accuracy during training and validation on CIFAR-10 without regularization.



(**b**) Loss and Accuracy during training and validation on CIFAR-10 with regularization.



(**c**) Loss and Accuracy during training and validation on CIFAR-100 without regularization.



(**d**) Loss and Accuracy during training and validation on CIFAR-100 with regularization.

Figure 4. Training and validation loss and accuracy curves for the simple CNN on CIFAR-10 and CIFAR-100

Table 4 summarizes the final training and validation metrics for all four scenarios discussed, providing a concise overview of the model's behavior with and without regularization on each dataset.

Despite the improvements brought by regularization, the overall results remain modest, especially on CIFAR-100. This outcome highlights the necessity for more complex models capable of capturing richer representations and handling higher dataset complexity.

Table 4. Summary of final training and validation results for the simple CNN after 200 epochs.

Dataset	Regularization	Train Accuracy (%)	Val Accuracy (%)	Train Loss	Val Loss
CIFAR-10	No	∼99	∼63	∼0	∼8.5
CIFAR-10	Yes	∼60	60–67	∼1.1	∼1.1
CIFAR-100	No	∼99	∼26	∼0	∼22
CIFAR-100	Yes	∼25	25–32	∼3.0	∼3.2

6.2. Classical Classifiers with Simple CNN

The traditional classifiers introduced in subsection 5.3 were trained using the features extracted from the simple CNN. Each was evaluated with and without the application of PCA, retaining 95% of the original variance. This process resulted in the automatic selection of 48 components for CIFAR-10 and 51 for CIFAR-100.

Table 5 summarizes the accuracy and training time for each configuration, while Figures 5(**a**) to 5(**d**) provide a graphical overview of the results.

In the case of CIFAR-10, the highest accuracy (0.46) was obtained without PCA, and PCA reduced training times considerably, particularly for SVM, whose time dropped from 220 to 53 seconds,with only marginal changes in accuracy. The MLP also maintained strong performance while reducing training time by nearly 30%.

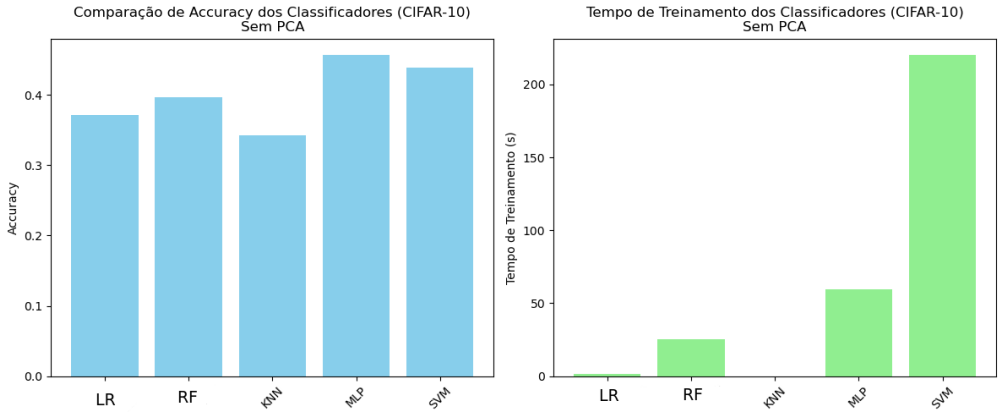
The CIFAR-100 dataset posed a greater challenge due to its increased class complexity. Accuracy scores were generally lower across all models. Still, both MLP and SVM showed relatively strong results, particularly with PCA applied, reaching 0.20 and 0.22 accuracy, respectively. Notably, PCA not only reduced training times significantly but also led to a substantial improvement in SVM's performance.

Further insights can be drawn from the confusion matrices for CIFAR-10, included in Appendix A. The MLP and SVM were notably effective at classifying visually distinct categories such as *ship* (654 correct predictions for MLP), *frog*, and *horse*, but struggled with semantically similar ones like *cat* vs. *dog*, and *automobile* vs *truck*. KNN showed strong bias toward certain classes, such as overpredicting *airplane*, while Logistic Regression demonstrated widespread misclassification in ambiguous categories. Confusion matrices were not included for CIFAR-100 due to the large number of classes, which made visualization and interpretation impractical.

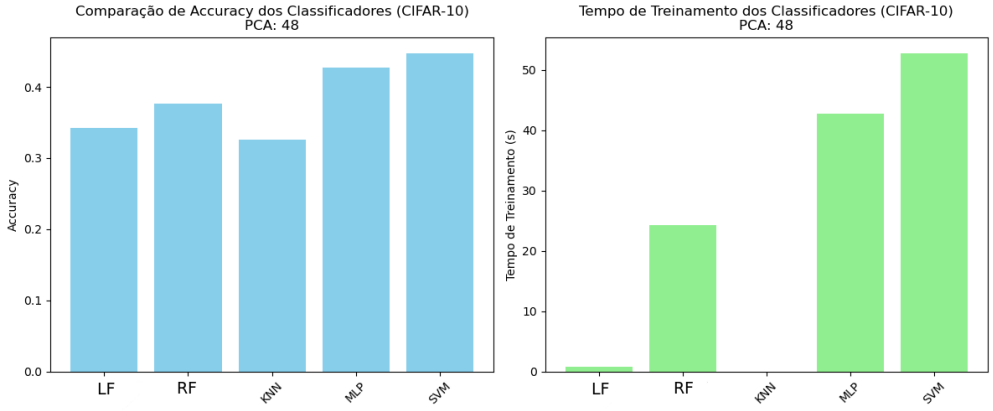
Table 5. Accuracy and training time of traditional classifiers, with and without PCA.

Classifier	CIFAR-10		CIFAR-10 + PCA		CIFAR-100		CIFAR-100 + PCA	
	Acc.	Time	Acc.	Time	Acc.	Time	Acc.	Time
Logistic Regression	0.37	2	0.34	1	0.10	2	0.11	2
Random Forest	0.40	26	0.38	24	0.16	58	0.16	77
KNN	0.34	0	0.32	0	0.13	0	0.15	0
MLP	0.46	60	0.43	42	0.19	102	0.20	98
SVM	0.44	220	0.45	53	0.16	104	0.22	58

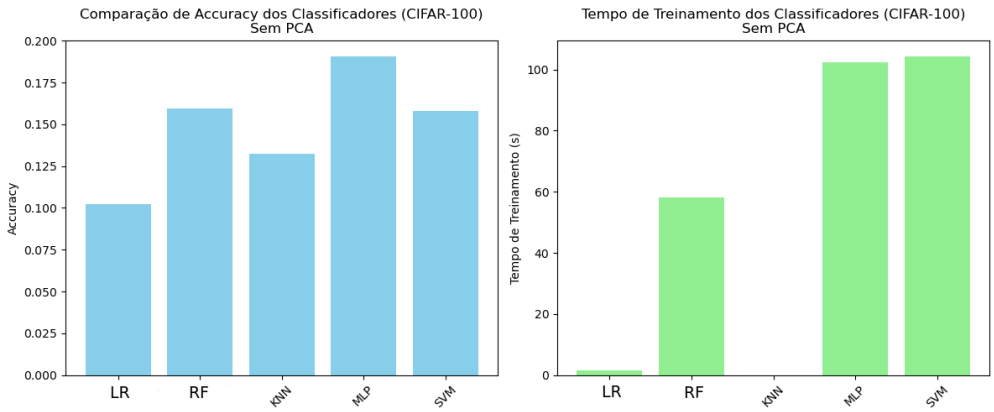
* Accuracy rounded to two decimal places; training time in seconds.



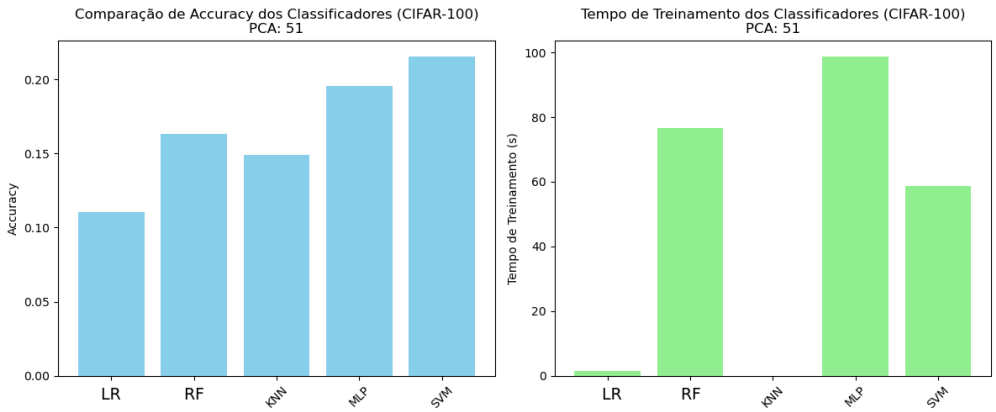
(**a**) Accuracy and training time of classifiers on CIFAR-10 without PCA.



(**b**) Accuracy and training time of classifiers on CIFAR-10 with PCA.



(**c**) Accuracy and training time of classifiers on CIFAR-100 without PCA.



(**d**) Accuracy and training time of classifiers on CIFAR-100 with PCA.

Figure 5. Accuracy and training time of classifiers using CNN features, with and without PCA

Overall, the results indicate that the most complex classifiers are better equipped to leverage the high-level representations extracted by the CNN, while PCA proves to be a valuable strategy for reducing computational cost with minimal impact on predictive performance.

6.3. Complex CNN

To improve upon the results obtained with the simple CNN, a more complex convolutional neural network architecture was designed by adding additional convolutional and fully connected layers. The rationale behind increasing the model depth was to enable the network to capture richer and more abstract feature representations, potentially leading to better performance on the challenging CIFAR-10 and CIFAR-100 datasets. The training methodology remained consistent with the previous experiments.

The performance was monitored through the evolution of training and validation loss and accuracy, and the final metrics are summarized to provide a clear comparison between different configurations.

Table Table 6 presents the final training and validation results for the complex CNN across CIFAR-10 and CIFAR-100 datasets, with and without overfitting correction techniques. These values were derived from the loss and accuracy curves shown in Figures 6(**a**) to 6(**d**).

For CIFAR-10 without regularization (Figure 6(**a**)), the training accuracy reached approximately 100%, with a training loss near zero. The validation accuracy was about 82%, while the validation loss remained around 1.8. This suggests the model effectively learned the training data and generalized reasonably well to the validation set, although some overfitting may be present given the gap between training and validation loss.

In contrast, when applying overfitting correction on CIFAR-10 (Figure 6(**c**)), training accuracy dropped to around 75% with a corresponding training loss near 0.7. The validation accuracy was slightly lower, approximately 80%, with a validation loss also around 0.7, indicating improved generalization and reduced overfitting effects.

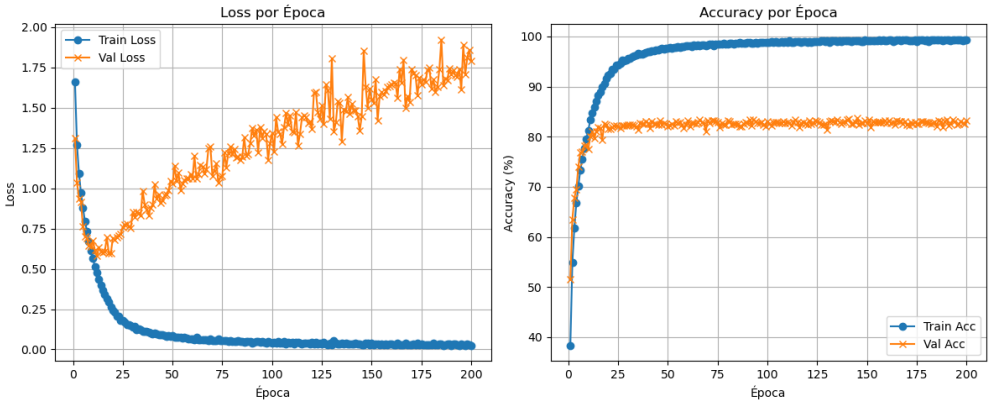
For CIFAR-100 without regularization (Figure 6(**b**)), the model achieved about 80% training accuracy with training loss near 0.75, but validation accuracy was only around 40%, with validation loss exceeding 5. This large gap between training and validation metrics indicates pronounced overfitting.

When overfitting correction techniques were applied on CIFAR-100 (Figure 6(**d**)), the model showed substantially lower training performance, with approximately 32% training accuracy and a high training loss near 2.5. The validation accuracy remained low at about 36%, and validation loss ranged between 2.5 and 3.0, reflecting the difficulty of the dataset and the strong regularization limiting model capacity.

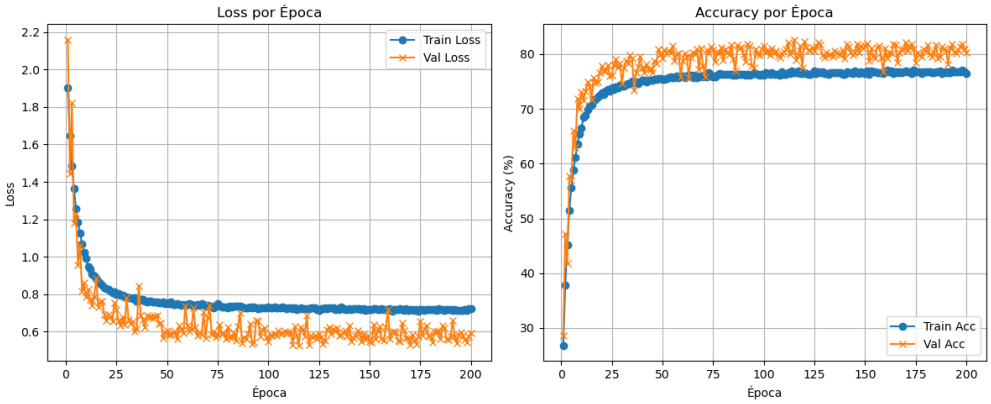
These results highlight the trade-offs between model complexity, overfitting correction, and dataset difficulty, illustrating that regularization can improve generalization at the expense of training accuracy, especially on more complex datasets.

Table 6. Summary of final training and validation results for the complex CNN after 200 epochs.

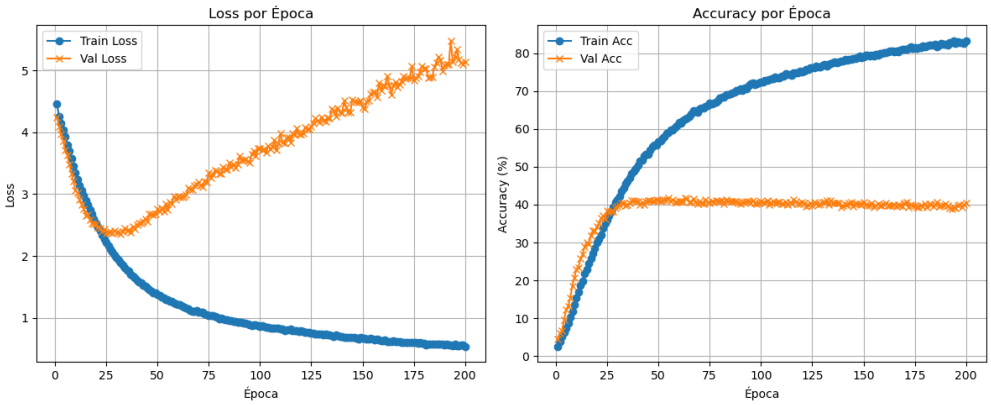
Dataset	Regularization	Train Accuracy (%)	Val Accuracy (%)	Train Loss	Val Loss
CIFAR-10	No	∼100	∼82	∼0.0	∼1.8
CIFAR-10	Yes	∼75	∼80	∼0.7	∼0.7
CIFAR-100	No	∼80	∼40	∼0.75	>5
CIFAR-100	Yes	∼32	∼36	∼2.5	2.5–3.0



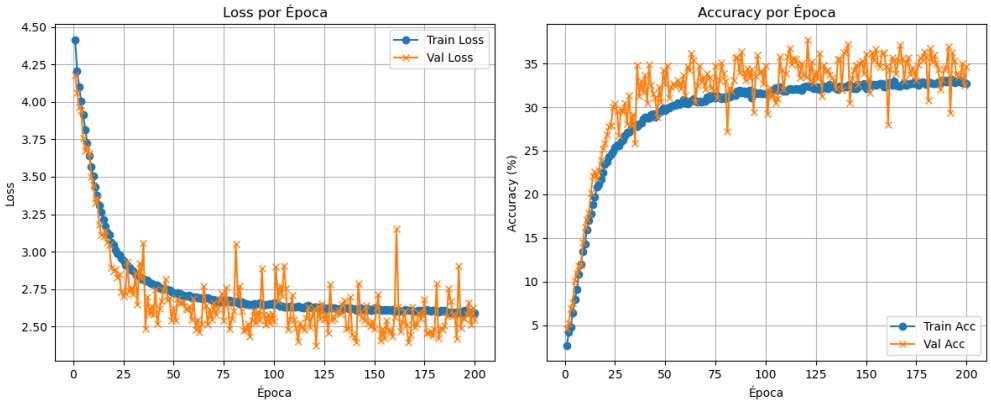
(**a**) Loss and Accuracy during training and validation on CIFAR-10 without regularization.



(**b**) Loss and Accuracy during training and validation on CIFAR-10 with regularization.



(**c**) Loss and Accuracy during training and validation on CIFAR-100 without regularization.



(**d**) Loss and Accuracy during training and validation on CIFAR-100 with regularization.

Figure 6. Training and validation loss and accuracy curves for the simple CNN on CIFAR-10 and CIFAR-100

6.4. Classical Classifiers with Complex CNN

Following the analysis conducted using the simple CNN, the same set of traditional classifiers introduced in subsection 5.3 was applied to the features extracted from the more complex CNN. As before, the classifiers were evaluated both with and without the application of PCA, retaining 95% of the original variance. This resulted in 54 components for CIFAR-10 and 65 for CIFAR-100.

Table 7 summarizes the accuracy and training time for each classifier, and Figures 7(**a**) to 7(**d**) illustrate these results graphically.

On CIFAR-10, the complex CNN provided higher-quality features, leading to improved performance across all classifiers. SVM and MLP were the most effective, reaching accuracies of 0.79 and 0.78 respectively. The application of PCA significantly reduced training time, particularly for SVM (from 64 to 32 seconds), without compromising performance.

A more detailed analysis of class-wise performance can be found in the confusion matrices provided in Appendix B. As with the simple CNN, classifiers such as MLP and SVM showed high accuracy in visually distinctive categories like *airplane*, *ship*, and *truck*, but struggled with visually or semantically similar categories, especially *cat* vs. *dog*, and *automobile* vs. *truck*. Logistic Regression and KNN exhibited more confusion, particularly in ambiguous categories.

For CIFAR-100, results remained lower overall due to the increased class complexity. SVM and MLP continued to outperform the other methods, achieving accuracies of 0.39 and 0.35 respectively with PCA. Once again, PCA proved effective in reducing computational cost, especially for SVM.

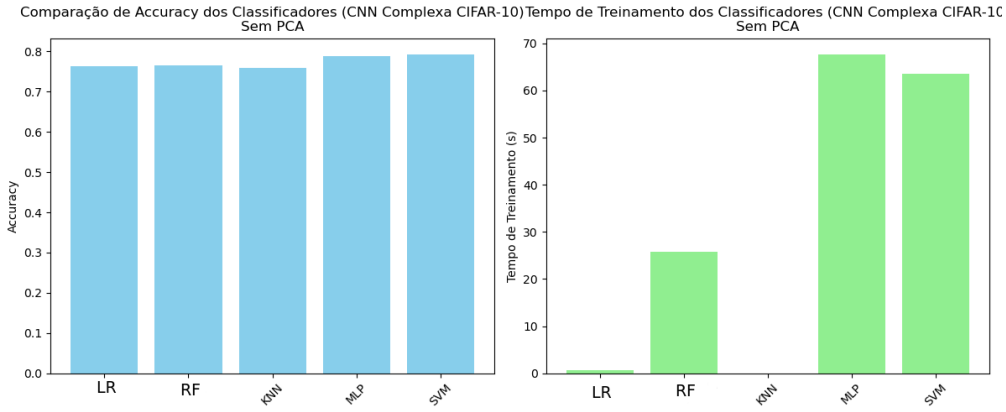
Confusion matrices were not included for CIFAR-100 due to the large number of classes, which made visualization and interpretation impractical.

Table 7. Accuracy and training time of traditional classifiers using features from the complex CNN, for CIFAR-10 and CIFAR-100, with and without PCA.

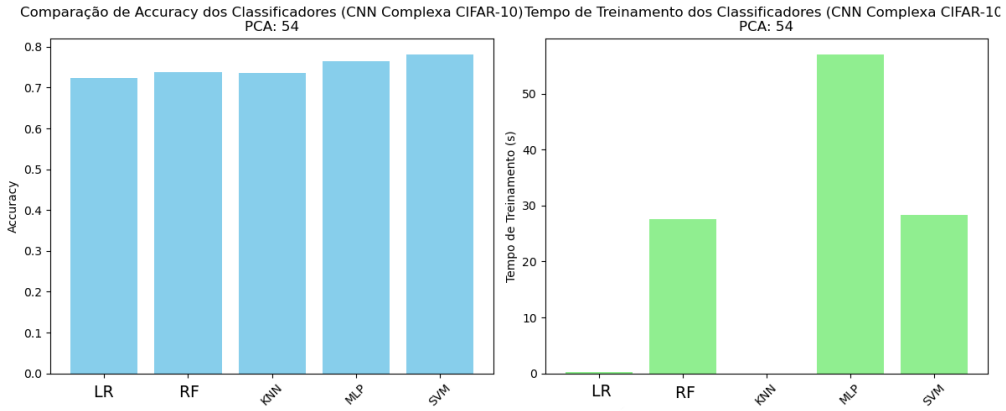
Classifier	**CIFAR-10**		**CIFAR-10 + PCA**		**CIFAR-100**		**CIFAR-100 + PCA**	
	Acc.	**Time**	**Acc.**	**Time**	**Acc.**	**Time**	**Acc.**	**Time**
Logistic Regression	0.76	1	0.75	1	0.29	2	0.29	2
Random Forest	0.76	26	0.75	33	0.30	64	0.29	84
KNN	0.76	0	0.75	0	0.28	0	0.28	0
MLP	0.78	68	0.78	54	0.36	105	0.35	87
SVM	0.79	64	0.79	32	0.37	58	0.39	48

* Accuracy rounded to two decimal places; training time in seconds.

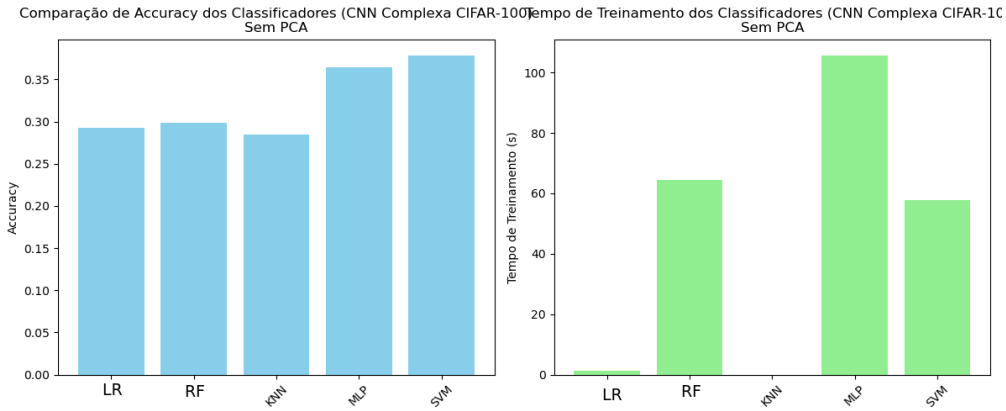
Overall, the results suggest that the additional depth and capacity of the complex CNN contribute to higher-quality feature representations, which are more effectively leveraged by advanced classifiers such as SVM and MLP. PCA remains a valuable tool for reducing computational cost without significantly impacting classification performance.



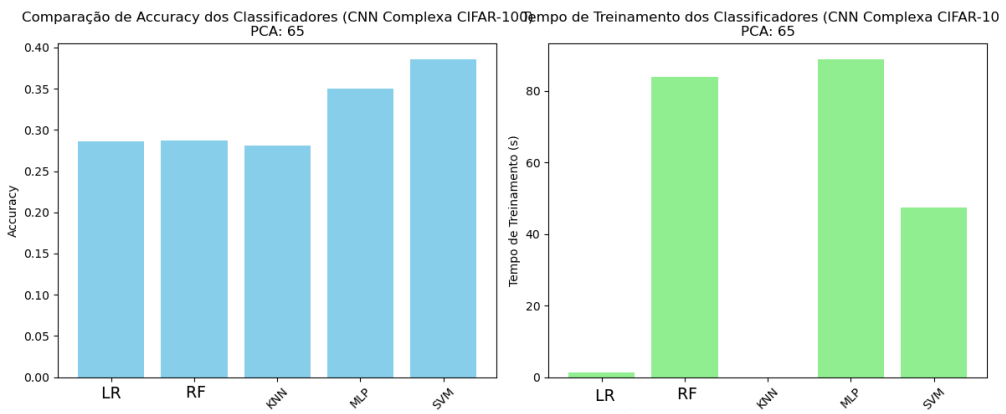
(**a**) Accuracy and training time of classifiers on CIFAR-10 without PCA.



(**b**) Accuracy and training time of classifiers on CIFAR-10 with PCA.



(**c**) Accuracy and training time of classifiers on CIFAR-100 without PCA.



(**d**) Accuracy and training time of classifiers on CIFAR-100 with PCA.

Figure 7. Accuracy and training time of classifiers using CNN features, with and without PCA.

6.5. Pre-trained Models

The evaluation of pre-trained models on CIFAR-10 yielded high classification accuracy across all tested architectures. Table 8 presents the accuracy scores obtained by each model.

Table 8. Accuracy of pre-trained models on CIFAR-10.

Model	Accuracy
ResNet-18	93.08%
DenseNet-121	94.06%
VGG-19	93.35%

* Accuracy values are reported as percentages.

`DenseNet-121` achieved the highest performance, reaching an accuracy of 94.06%. `ResNet-18` and `VGG-19` followed closely, with 93.08% and 93.35%, respectively. While the differences are relatively small, they reflect the architectural advantages of more recent models, such as `DenseNet's` dense connectivity and feature reuse.

Overall, all models achieved state-of-the-art performance on CIFAR-10, establishing strong baselines for comparison.

7. Final Discussion

The results obtained throughout this work provide a comprehensive perspective on the challenges and limitations inherent to image classification, even when using datasets as established as CIFAR-10 and CIFAR-100. Despite the adoption of best practices in data preprocessing, regularization, and model design, the implemented CNN's did not reach state-of-the-art performance, especially on CIFAR-100, where the highest validation accuracy remained well below the benchmarks reported in the literature. This outcome highlights the substantial complexity of the task: even "basic" datasets like CIFAR-100 present significant obstacles due to their high intra-class variability and the large number of categories, demanding models with greater representational capacity and more sophisticated training strategies.

On CIFAR-10, the results were more encouraging. The simple CNN, when regularized, achieved validation accuracies in the 60–67% range, while the complex CNN, with appropriate overfitting mitigation, reached approximately 80%. These values, although distant from the 90%+ accuracy of pre-trained models, are notable given the relatively modest depth and parameter count of the custom architectures. The observed gap between training and validation performance in the absence of regularization further reinforces the importance of techniques such as data augmentation, dropout, and weight decay in promoting generalization.

The experiments with classical classifiers corroborate the trends described in the state-of-the-art section. When applied directly to deep features extracted from the CNN's, traditional models like Logistic Regression, Random Forest, KNN, SVM, and MLP consistently underperformed relative to the fully connected layers of the neural networks. Their accuracy was particularly limited when using features from the simpler CNN, but improved markedly as the complexity of the feature extractor increased. This demonstrates that the discriminative power of classical classifiers is heavily dependent on the quality of the input representation, a finding that aligns with the broader consensus in the literature.

The application of PCA as a dimensionality reduction technique proved to be a valuable strategy for managing computational resources. In almost all scenarios, PCA significantly reduced training times, especially for computationally intensive models like SVM, without causing a substantial drop in accuracy. In some cases, such as SVM on CIFAR-100, PCA even led to improved performance, likely due to noise reduction and better feature

compactness. This suggests that, in practical settings, dimensionality reduction can be leveraged to balance efficiency and predictive power.

Finally, the evaluation of pre-trained models on CIFAR-10 established a clear upper bound for performance, with `DenseNet-121`, `ResNet-18`, and `VGG-19` all achieving accuracies above 93%. The substantial gap between these results and those obtained with custom CNN's underscores the impact of architectural advances, large-scale training, and hyperparameter optimization in achieving state-of-the-art results.

In summary, this work demonstrates that while deep learning models, especially CNN's, are essential for tackling image classification tasks, achieving top-tier performance requires not only careful model design and regularization, but also access to advanced architectures and extensive computational resources. The experiments reinforce the importance of feature quality for downstream classifiers and highlight PCA as a practical tool for resource management. Above all, the persistent difficulty of CIFAR-100 serves as a reminder that even "basic" benchmarks remain challenging and continue to drive progress in the field.

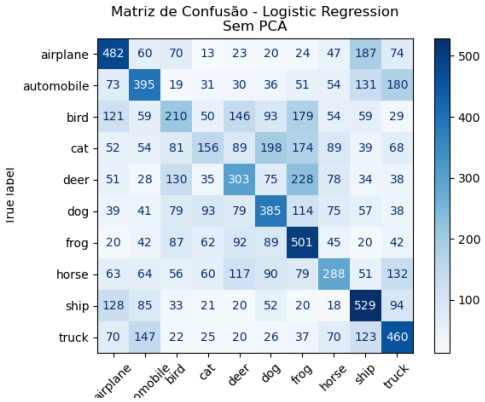
Abbreviations

The following abbreviations are used in this manuscript:

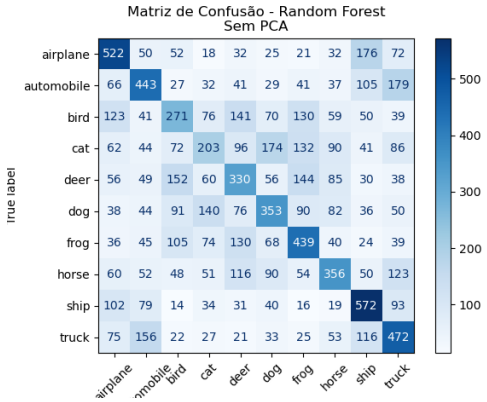
RGB	Red Green Blue
CNN	Convolutional Neural Network
CVPR	Conference on Computer Vision and Pattern Recognition
SVM	Support Vector Machine
KNN	K-Nearest Neighbors
MLP	Multi-Layer Perceptron
PCA	Principal Component Analysis
GPU	Graphics Processing Unit
ReLU	Rectified Linear Unit
L2	Euclidean Norm (used in regularization)
RBF	Radial Basis Function

Appendix A. Confusion Matrices of Classifiers (Simple CNN)

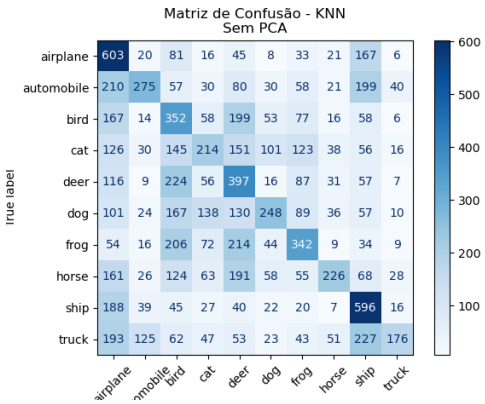
This appendix presents the confusion matrices obtained for the classical classifiers evaluated on the CIFAR-10 dataset without PCA, using features extracted from the simple CNN. These matrices provide a detailed view of the classification performance across individual classes, highlighting which categories are more frequently confused and which are better distinguished by each model. This additional information complements the quantitative analysis presented in subsection 6.2, offering a more nuanced understanding of each classifier's behaviour.



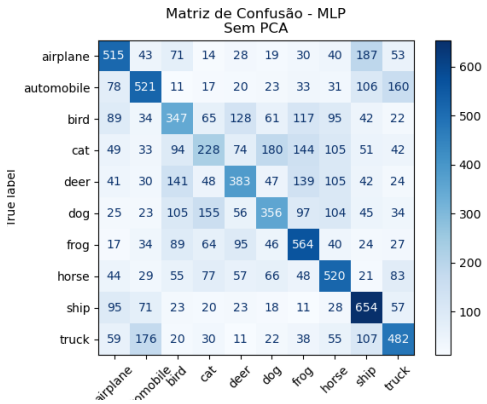
(**a**) Confusion matrix of Logistic Regression.



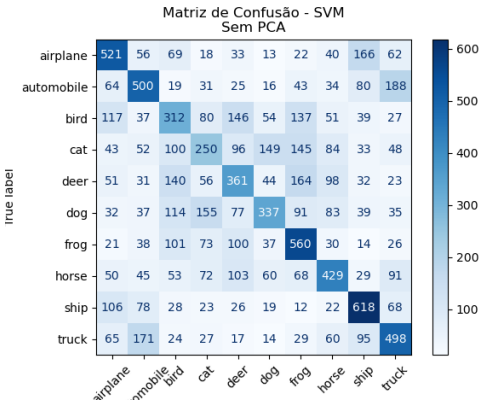
(**b**) Confusion matrix of Random Forest.



(**c**) Confusion matrix of KNN.



(**d**) Confusion matrix of MLP.

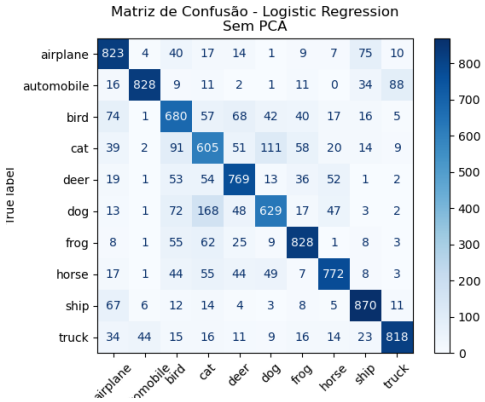


(**e**) Confusion matrix of SVM.

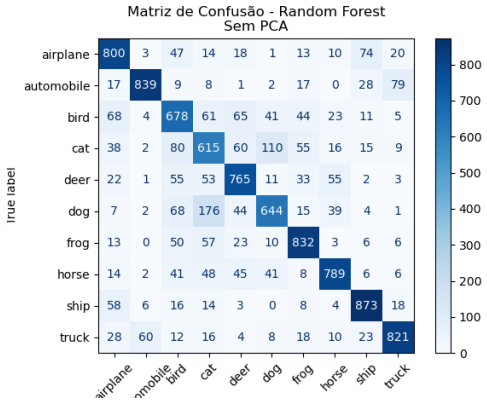
Figure A1. Confusion matrices for classical classifiers on CIFAR-10 without PCA.

Appendix B. Confusion Matrices of Classifiers (Complex CNN)

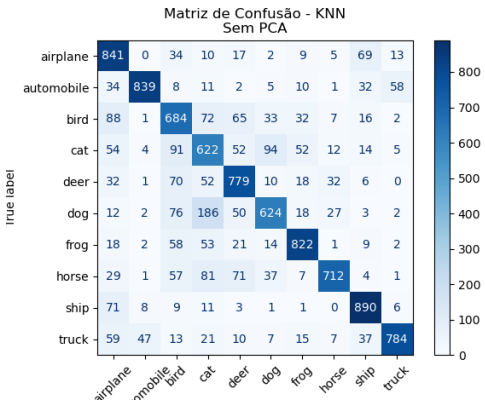
This appendix presents the confusion matrices obtained for the classical classifiers evaluated on the CIFAR-10 dataset without PCA, using features extracted from the complex CNN. These matrices provide a detailed view of the classification performance across individual classes, highlighting which categories are more frequently confused and which are better distinguished by each model. This additional information complements the quantitative analysis presented in subsection 6.4, offering a more nuanced understanding of each classifier's behaviour.



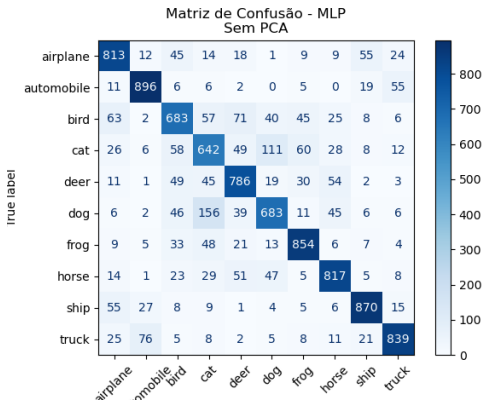
(**a**) Confusion matrix of Logistic Regression.



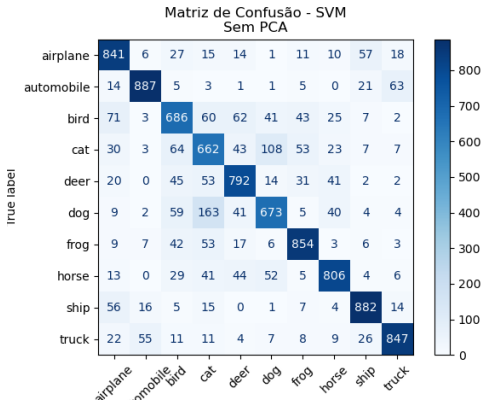
(**b**) Confusion matrix of Random Forest.



(**c**) Confusion matrix of KNN.



(**d**) Confusion matrix of MLP.



(**e**) Confusion matrix of SVM.

Figure A2. Confusion matrices for classical classifiers on CIFAR-10 without PCA.

References

1. Huy V. Phan. *PyTorch models trained on CIFAR-10 dataset*. GitHub repository. Available at: https://github.com/huyvnphan/PyTorch_CIFAR10. Accessed: May 31, 2025.
2. A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, University of Toronto, Technical Report, 2009. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf
3. A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large dataset for nonparametric object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008. doi: 10.1109/TPAMI.2008.128
4. J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. doi: 10.1109/MCSE.2007.55
5. F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
6. A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.
7. S. Marcel and Y. Rodriguez, "Torchvision the Machine Vision Package of Torch," in *Proceedings of the ACM International Conference on Multimedia*, 2010.
8. Python Software Foundation, "`pickle` — Python object serialization," https://docs.python.org/3/library/pickle.html, accessed May 2025.
9. PyTorch Contributors. *PyTorch Documentation: Data Loading and Processing Tutorial*. Available at: https://pytorch.org/tutorials/beginner/data_loading_tutorial.html (Accessed: 2025).
10. Ridnik, T., Lawen, H., Noy, A., Ben-Baruch, E., Sharir, G., and Friedman, I. *TResNet: High Performance GPU-Dedicated Architecture*. Available at: https://github.com/Alibaba-MIIL/TResNet (Accessed: 2025).
11. S. Singh, "Introduction to Convolutional Neural Networks," *GeeksforGeeks*, 2023. [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
12. Tan, M. and Le, Q. V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. arXiv preprint arXiv:1905.11946. Available at: https://arxiv.org/abs/1905.11946 (Accessed: 2025).
13. He, K., Zhang, X., Ren, S., and Sun, J. *Deep Residual Learning for Image Recognition*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.
14. Zagoruyko, S. and Komodakis, N. *Wide Residual Networks*. arXiv preprint arXiv:1605.07146. Available at: https://arxiv.org/abs/1605.07146 (Accessed: 2025).
15. Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. *Densely Connected Convolutional Networks*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
16. ResearchGate. *CNN vs SVM on CIFAR-10 Classification*. Available at: https://www.researchgate.net/publication/335264157 (Accessed: 2025).
17. C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019. doi: 10.1186/s40537-019-0197-0