CIFAR10 Image Classification With Convolution Neural Networks

Davide Vettore 868855
Alessio De Luca 919790

1. Objective:

The objective of this activity is to train a Convolution Neural Network (CNN) with various configurations of network parameters and analyzing the impact on classification performance. This involves experimenting with various spatial pooling layers, exploring different non-linear activation functions, optimizer parameters, and observing the impact of epoch variations. We are going to analyze the results by showing how the training accuracy and loss varies in the different epochs and how this translates in the test accuracy. Furthermore, we'll delve into identifying which classes are prone to misclassification and provide illustrative examples.

2. Introduction:

We're utilizing data sourced from the CIFAR-10 dataset, which is a subset of the extensive 80-million Tiny Images dataset. CIFAR-10 comprises 60,000 32x32 color images categorized into 10 classes (*plane, car, bird, cat, deer, dog, frog, horse, ship, truck*), each containing 6,000 images. The dataset is divided into 50,000 training images and 10,000 test images. Notably, the classes are entirely mutually exclusive.

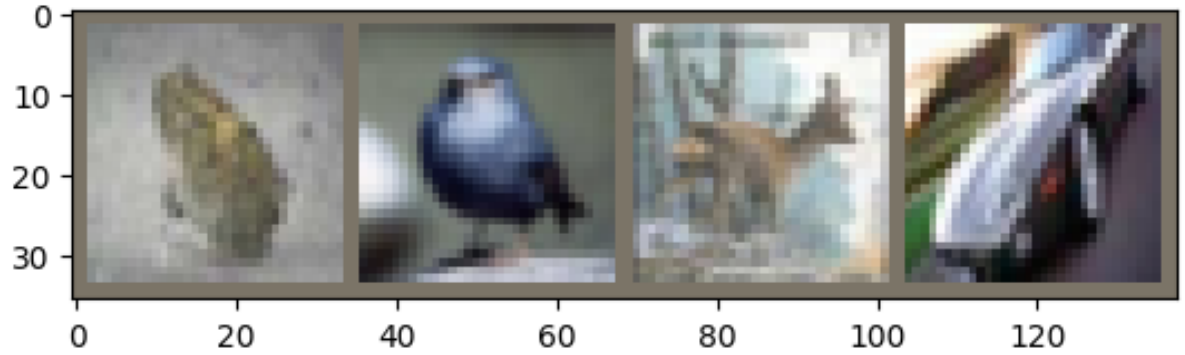

Figure 1: Random batch (four elements) extracted from the training set, the labels are respectively *frog, bird, deer* and *car*.

Convolutional neural networks (CNNs) are biologically-inspired architectures that are currently the state-of-the-art in computer vision tasks. By employing convolution layers, pooling layers, and fully connected layers, CNNs effectively extract hierarchical features, enabling robust interpretation of visual data. This hierarchical learning process facilitates the detection of complex shapes, textures, and spatial relationships, resulting in enhanced object recognition and classification accuracy.

3. Procedure:

Initially, we define the Convolutional Neural Network architecture. The main structure will remain the same while we tweak certain parameters such as the spatial pooling layers and the non-linear activation functions. Subsequently, we'll also vary the training procedure by experimenting with different training optimizers, adjusting learning rates, and exploring variations in the number of training epochs.

The network starts with an *input layer* designed to receive images with 3 RGB channels. Following this, three successive *convolutional layers* employ a cascade of filters to extract progressively intricate features, yielding 32, 64, and 128 feature maps, respectively. Each convolutional layer is followed by an *activation function*, injecting non-linearity into the network to capture more intricate relationships within the data. *Max pooling layers*, with a 2x2 kernel and a stride of 2, serve to downsample the feature maps, effectively reducing spatial dimensions by half. This process aids in preserving essential information while mitigating computational complexity. Following three pooling layers, the output is flattened and passed through three *fully connected layers*. These layers act as powerful classifiers, mapping learned features to the respective class labels. With carefully chosen configurations, the first fully connected layer receives 12844 input neurons and outputs 1024 neurons, while the second one further reduces dimensions to 256 neurons.

The *output layer* comprises 10 neurons, each representing the probability of an input image belonging to one of the 10 classes. Activation functions, applied after each fully connected layer except for the output layer, introduce non-linearity into the network, enhancing its capacity to model complex relationships within the data.

We'll be adjusting the following network parameters:

- **Non-linear activation functions**: We'll experiment with Hyperbolic Tangent (TanH), Rectified Linear Unit (ReLU), an alternative Leaky ReLU, and the Gaussian Error Linear Unit (GELU).
- **Pooling function**: We'll explore both 2D average pooling and max pooling.

During the training phase, we'll test both the Stochastic Gradient Descent (SDG) optimizer and the Adam optimizer. While keeping the optimizer fixed, we'll vary the learning rate across 0.01, 0.001, and 0.0001 to find a balance between fast learning and stability.

4. Results:

Various configurations of the CNN were tested, we reported some of them. In Table 1 we can observe how the test accuracy varies while exploring different non-linear activation functions.

Convolution Functions	Pooling Functions	Non-linear activation functions	Loss function	Optimizer	Learning Rate	N° of Epochs	Test Accuracy
Conv2d	MaxPool2d	Tanh	CrossEntropyLoss	SGD	0.001	5	72.81%
Conv2d	MaxPool2d	ReLU	CrossEntropyLoss	SGD	0.001	5	73.35%
Conv2d	MaxPool2d	Leaky ReLU	CrossEntropyLoss	SGD	0.001	5	74.22%
Conv2d	MaxPool2d	GeLU	CrossEntropyLoss	SGD	0.001	5	75.55%

Table 1: Variation in performance while trying multiple non-linear activation function.

Convolution Functions	Pooling Functions	Non-linear activation functions	Loss function	Optimizer	Learning Rate	N° of Epochs	Test Accuracy
Conv2d	AvgPool2d	Tanh	CrossEntropyLoss	SGD	0.001	5	46.55%
Conv2d	AvgPool2d	ReLU	CrossEntropyLoss	SGD	0.001	5	70.88%
Conv2d	AvgPool2d	Leaky ReLU	CrossEntropyLoss	SGD	0.001	5	64.20%
Conv2d	AvgPool2d	GeLU	CrossEntropyLoss	SGD	0.001	5	74.08%

Table 2: Variation in performance while changing pooling function to *AvgPool2d*.

In Table 2 we observe how the performances decrease when we apply 2D average-pooling operations instead of max-pooling. Similarly, with fixed learning rate, we observe how Adam optimizer usually leads to lower accuracies (the best performing network achieves only 73.0% accuracy). Adam optimizer shows comparable performances only when the learning rate is adjusted to 0.0001 (best performing network achieves 75.27% accuracy). Such a low learning rate, however, does not improve SDG optimizer test accuracy.

Ultimately, the best result was given by a CNN designed with GELU activation functions and max pooling layers, trained with SDG optimizer and learning rate set to 0.001. This CNN allowed us to obtain an accuracy of 75.55%. We also tried training it with 10 epochs: the training loss and accuracy were better, as we can see in Figure 2, however, the accuracy on the test data dropped from 75.55% to just 74.20%. This probably happened because the model overfitted the training data.

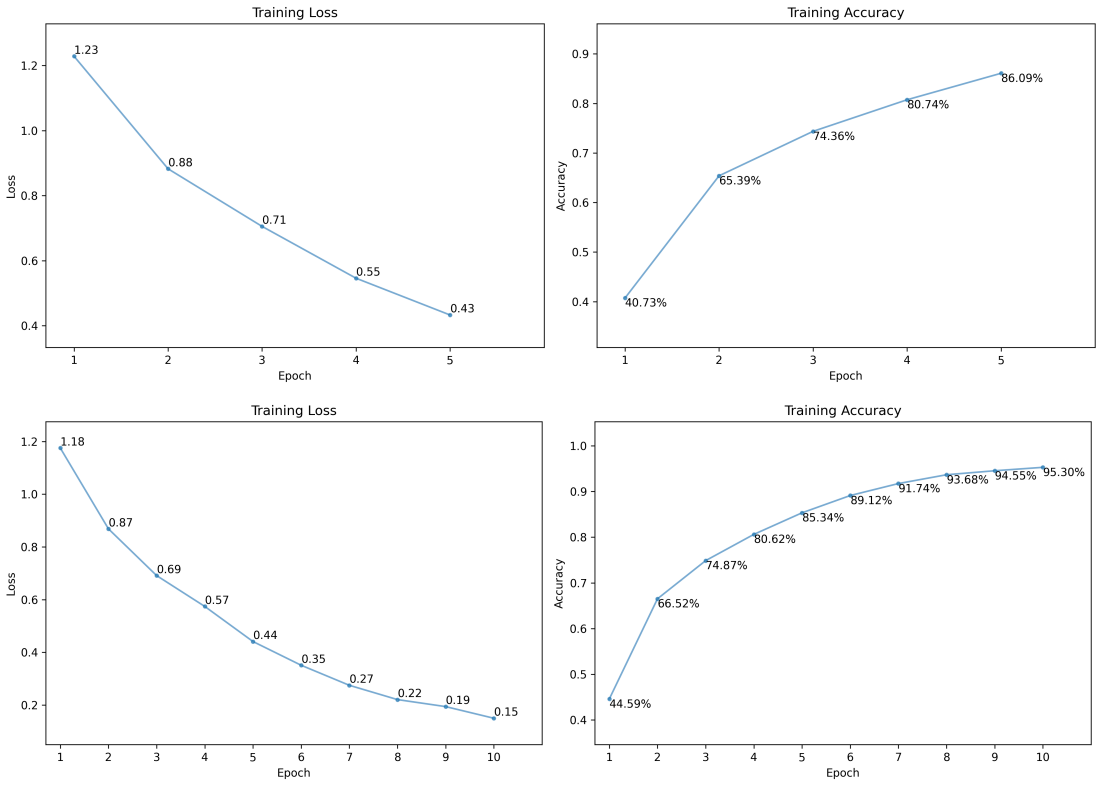

Figure 2: Best performing network configuration's training Loss and Accuracy with increasing epochs.

In Figure 3, we can observe the best performing CNN trained over five epochs absolute and percentage confusion matrices. On the principal diagonal the correct predictions are shown, while the misclassified ones are in every other allocation.

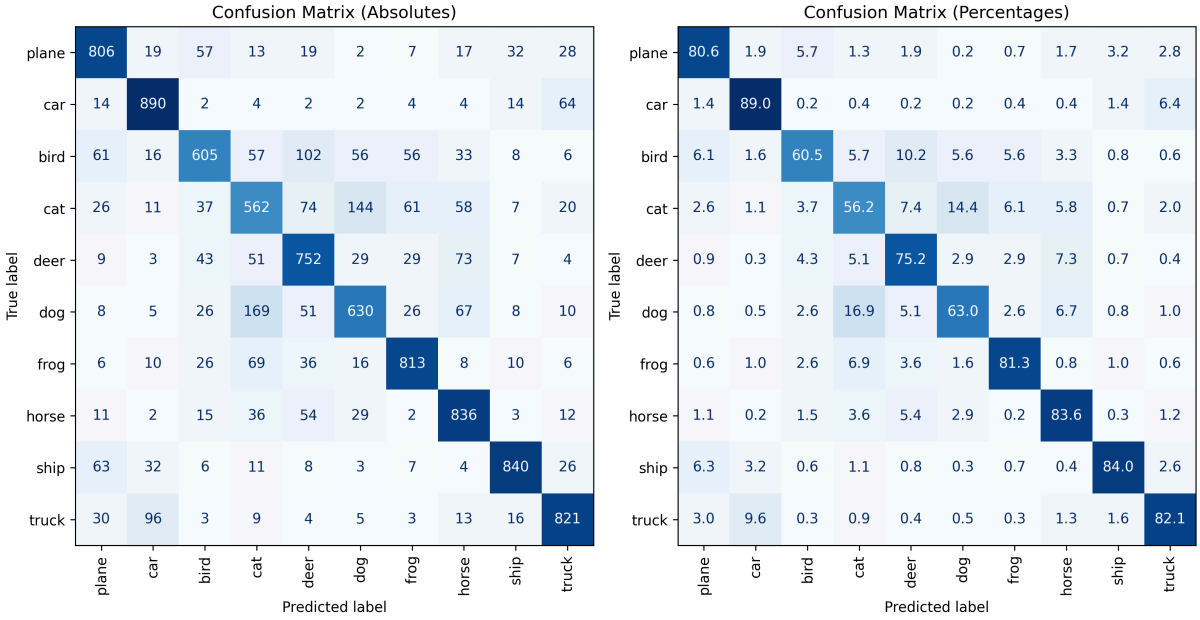

Figure 3: Absolute and percentage confusion matrices.

In our investigation of the CIFAR-10 image classification results, we're focusing on pairs of classes that exhibit approximately 10% misclassification rate. By analyzing the confusion matrix, we aim to identify and understand the patterns contributing to this higher rate of misclassification. The most ocurring misclassifications are:

- *bird* as *deer*
- *cat* as *dog*
- *dog* as *cat*
- *truck* as *car*

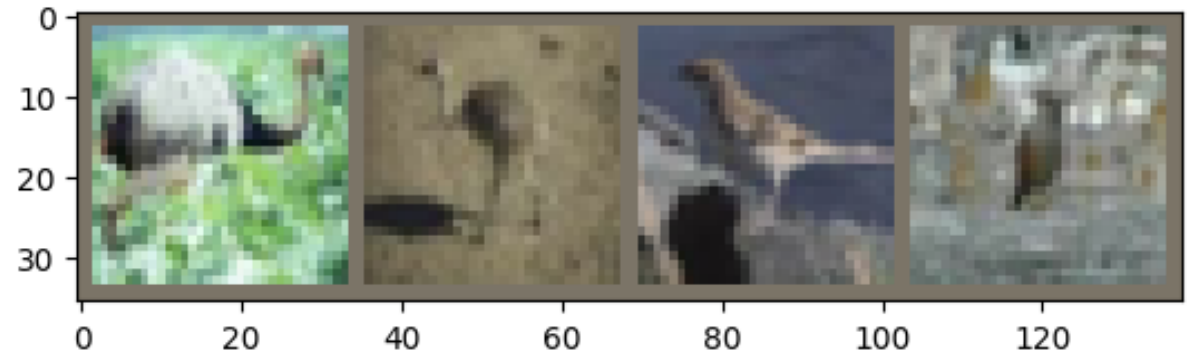

Figure 4: *bird* images misclassified as *deer*.

In Figure 4, we observe instances where birds are erroneously classified as deers. Although initially unconventional, this phenomenon becomes comprehensible after closer examination of the images: misclassified birds consistently appear grounded rather than in flight. This propensity may induce the model to classify them as deer by detecting background features resembling those present in deers photographs.

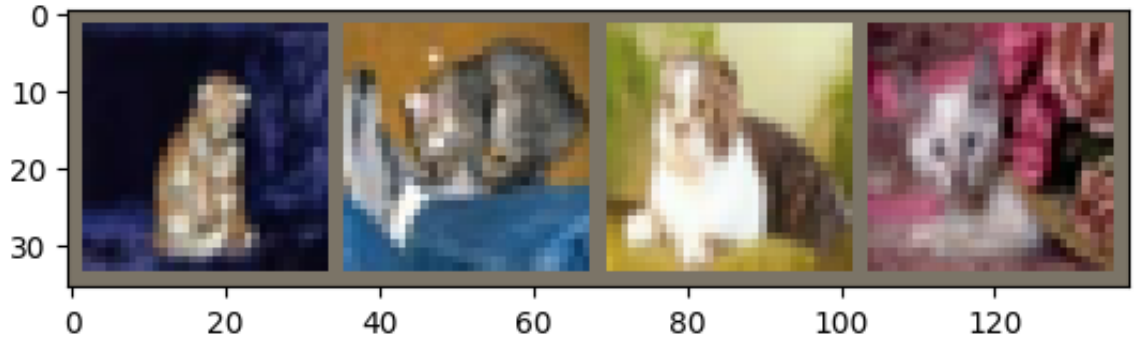

Figure 5: *cat* images misclassified as *dog*.

Figure 5 and Figure 6 show cats and dogs getting misclassified, particularly when the size of the animals is comparable or when they are in a laid down position. Additionally, background elements can contribute to misclassification, as many images are captured within similar environments, such as household settings or outdoor landscapes like grassy areas.

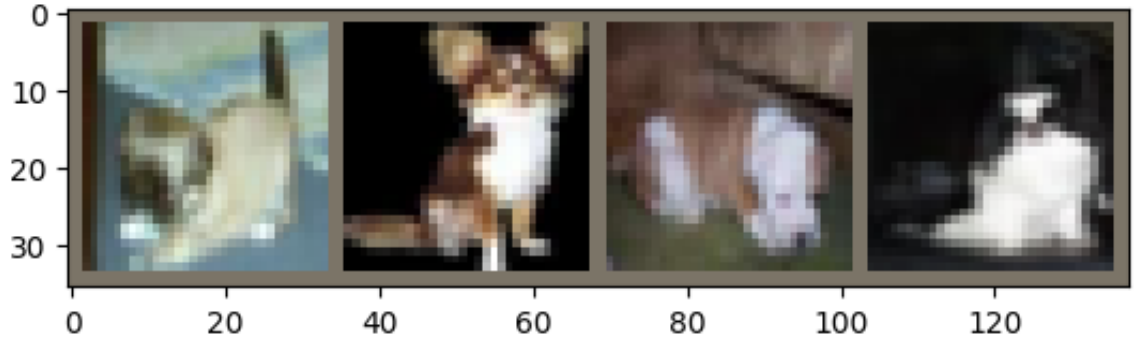

Figure 6: *dog* images misclassified as *cat*.

In Figure 7, misclassifications occur between trucks and cars, likely attributable to the close resemblance between these vehicle types. Shared characteristics such as wheels, mirrors, and headlights contribute to this confusion. Additionally, the absence of a clear delineation between cars and SUVs within our image dataset further complicates classification efforts.

Figure 7: *truck* images misclassified as *car*.

5. Conclusions:

In conclusion, our exploration of CIFAR-10 image classification using Convolutional Neural Networks yielded valuable insights. After defining our CNN architecture, through systematic experimentation with various network parameters, activation functions, pooling techniques, and optimizers, we identified the optimal configuration that maximized classification test accuracy.

Moreover, our analysis of confusion matrices revealed common misclassifications between classes sharing visual similarities or contextual features. Instances such as birds being misclassified as deers and cats mistaken for dogs highlighted the challenges in distinguishing subtle visual patterns, excpetially when similar backgrounds are involved.

These discoveries underscore the importance of refining network architectures and training strategies to enhance classification accuracy and address real-world challenges in image recognition tasks.