Supervised Learning - Assignment 5

April 17, 2024

Simone Vaccari
915222

Annalisa Di Pasquali
858848

Abstract

The aim of this assignment is to present the results obtained from the classification of the images of the CIFAR-10 dataset using Convolutional Neural Networks, discuss the model's performances, and present the conclusions drawn from the analysis of the results.

1 Description of the dataset

The CIFAR-10 dataset is a collection of 60,000 color images, each of size 32x32 pixels, divided into 10 classes. These classes include common objects such as airplanes, cars, birds, cats, and more. Each class contains 6,000 images. CIFAR-10 is commonly used as a benchmark dataset for image classification tasks in machine learning and computer vision research due to its diversity, standardization, and manageable size.

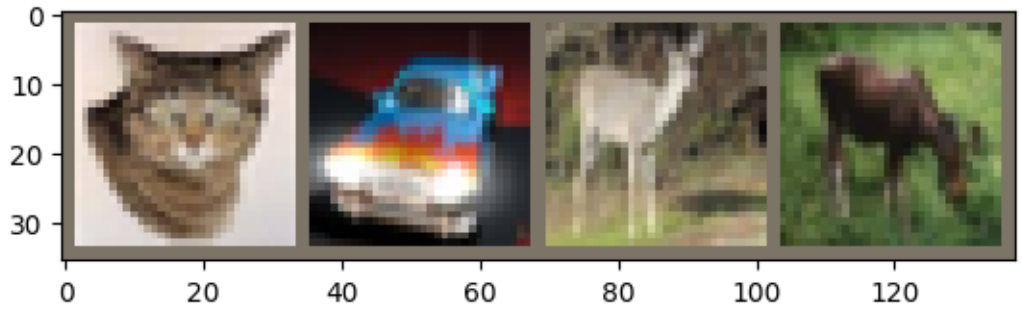

Figure 1: Example of images of the CIFAR-10 dataset

2 Implementation

In order to perform our image classification task we used Convolutional Neural Networks (CNNs), a particular type of deep neural networks that consist of multiple consecutive layers of feature extractors, and that are widely applied in computer vision.
The typical architecture of a CNN consists of two main parts:

- Convolutional layers, in which the inputs, the outputs, and the weights are arranged into volumes;
- Fully-connected layers, like the ones of a traditional MLP, in which the information is organized in the form of a 1-D array. Typically the last layer works as the predictor;

Other types of layers that might be present in such a network are spatial pooling layers and local response normalization layers.
The number of layers of each type, along with the number of neurons in each layer, can vary, and the various architectures will lead to different results. During our experiments we have defined CNNs with different parameters.

First, we tested various configurations of convolutional layers and fully-connected layers, exploring how adding or removing layers can affect the network in order to maximize its performance. After several attempts we decided for the following architecture:

- **Input Layer**: the input to the network is an RGB image (3 color channels).
- **Convolutional Layers**:
  - `conv1`: First convolutional layer with 3 input channels, 32 output channels, a kernel size of 3×3, and padding of 1.
  - `conv2`: Second convolutional layer with 32 input channels (from `conv1`), 64 output channels, a kernel size of 3×3, and padding of 1.
  - `conv3`: Third convolutional layer with 64 input channels (from `conv2`), 128 output channels, a kernel size of 3×3, and padding of 1.
  - `conv4`: Fourth convolutional layer with 128 input channels (from `conv3`), 128 output channels, a kernel size of 3×3, and padding of 1.
  - `conv5`: Fifth convolutional layer with 128 input channels (from `conv4`), 256 output channels, a kernel size of 3×3, and padding of 1.
  - `conv6`: Sixth convolutional layer with 256 input channels (from `conv5`), 256 output channels, a kernel size of 3×3, and padding of 1.

  For every layer the stride was 1 by default.
- **Pooling Layers**:
  - `pool`: Pooling layer with a kernel size of 2×2 and a stride of 2. This layer is applied after each pair of convolutional layers, reducing the spatial dimensions of the image.
- **Fully-Connected (fc) Layers**:
  - `fc1`: First fully-connected layer that takes input from the flattened convolutional layers (with a size of $256 \times 4 \times 4$) and produces 1024 outputs.
  - `fc2`: Second fully-connected layer with 1024 inputs and 512 outputs.
  - `fc3`: Output fully-connected layer with 512 inputs and the number of classes as output (defined by `nClasses`, defaulting to 10 for CIFAR-10).

The convolutional layers progressively extract features from the images, while the fully-connected layers combine these features to generate the final class prediction.

Once the architecture of our network had been defined, we focused on modifying other parameters, such as the type of **spatial pooling** or the **activation function** used within the network's layers. Among the available options, we decided to use the RReLU activation function, which applies the randomized leaky rectified linear unit function.

We also worked on the parameters that characterize the training process, such as:
- The **Loss Function**, by using the `CrossEntropyLoss`, which is suitable for multi-class classification tasks.
- The **Optimizer**, by using either `optim.Adam` or `optim.SGD` (Stochastic Gradient Descent), and trying different values for the learning rate.

The different models were trained, monitoring their progress at regular intervals, and saving the loss and accuracy values at every epoch to compare the various attempts.

Finally, the different networks were tested on a fraction of the dataset not used for training, consisting of 10,000 images (1,000 for each class).

3 Results

In order to compare the performances of the different models we set the number of training epochs to five.

In Table 1 we report the different accuracies on the testing set, obtained by varying the above-mentioned hyperparameters.

Activation Functions	Pooling	Optimizer	Learning Rate	Accuracy
GeLU/ReLU	Max	SGD	0,001	77%
GeLU/ReLU	Average	SGD	0,001	63%
GeLU	Max	SGD	0,001	70%
Leaky ReLU	Max	SGD	0,001	77%
Leaky ReLU	Max	SGD	0,0006	76%
Leaky ReLU	Average	SGD	0,0006	73%
Leaky ReLU	Max	Adam	0,001	15%
Leaky ReLU	Max	Adam	0,0001	79%
RReLU	Max	SGD	0,001	79%
RReLU	Average	SGD	0,001	79%
RReLU	Max	SGD	0,002	78%
RReLU	Max	Adam	0,0001	80%
RReLU	Average	Adam	0,0001	73%

Table 1: Table of results, with the best configuration highlighted in blue

In particular:

- Different activation functions were explored, including ReLU, Leaky ReLU, and variants such as Randomized Leaky ReLU (RReLU) and Gaussian Error Linear Unit (GeLU). These activation functions were applied to the convolutional and fully-connected layers to introduce non-linearity into the model. Variations in the activation function impacted the model's ability to capture patterns in the data.
  We can observe that in our case **RReLU** yields in general better results.
- For the pooling layers, max pooling and average pooling were applied: the first is focused on preserving the most prominent features by selecting the maximum values within each region, while the second computes the average intensity across the region, resulting in smoother down-sampled feature maps.
  It can be noted that by passing from max to average pooling the accuracy decreases, thus leading us to adopt the **Max Pooling** technique.
- Two different optimization algorithms were employed to update the model parameters during training: SGD with momentum always equal to 0.9 and the Adam optimizer. Both were able to achieve respectable performances, however some changes in the learning rate needed to be made.

- The learning rate, the hyperparameter controlling the step size during optimization, was varied across different experiments. It can be observed that with the SGD optimizer a learning rate of 0,001 generally leads to good results, while when applying the Adam optimizer it needs to be lowered significantly to achieve a comparable result in the same number of training epochs.

As shown in blue in Table 1, the best performance we were able to achieve for the CIFAR-10 dataset is an accuracy of 80%, obtained with a model that involves RReLu activation functions, max pooling, and the Adam optimizer with a learning rate of 0.0001. For this model, we decided to increase the number of training epochs from five to ten. Figure 2 shows how the loss and the accuracy vary with the different training epochs: the fact that the loss keeps decreasing while the accuracy increases tells us that the model is indeed learning.

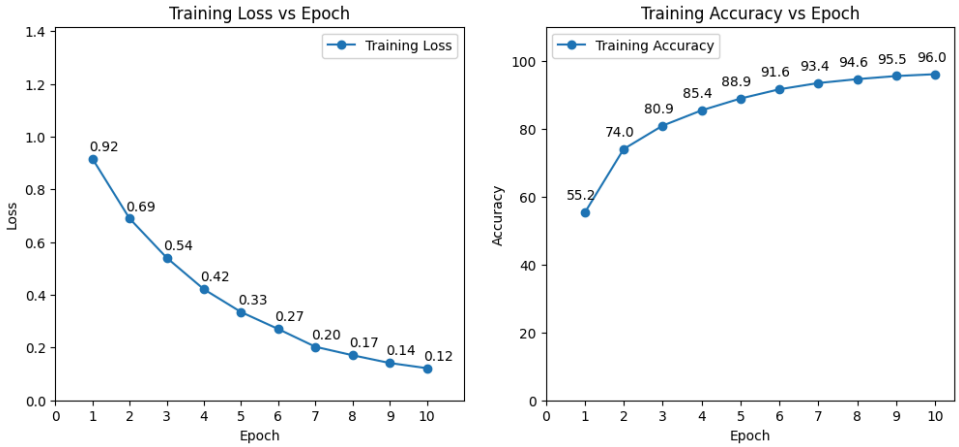

Figure 2: Training Loss and Training Accuracy vs Epoch

In this case the accuracy scored by the model was around 82%, indicating that its predictions are slightly more accurate, but not significantly, and for computational reasons one might consider sufficient training for five epochs only.
To visualize how the model performed on the individual classes we computed the accuracy class by class, as can be seen in Figure 3.

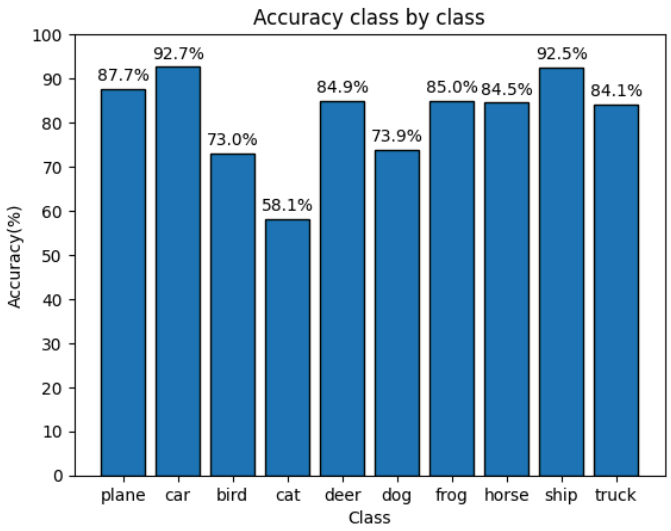

Figure 3: Accuracy for each class on the testing set

In order to provide a more comprehensive visualization of the model's classification performance across the different classes in the dataset, in Figure 4 we reported the confusion matrix, a matrix in which each row represents the actual class label (the ground-truth), and each column the predicted class label. The diagonal elements of the matrix correspond to correctly classified instances, while off-diagonal elements represent misclassifications. By examining the values in the matrix, we can assess the model's ability to accurately classify each class and identify any patterns of confusion between classes.

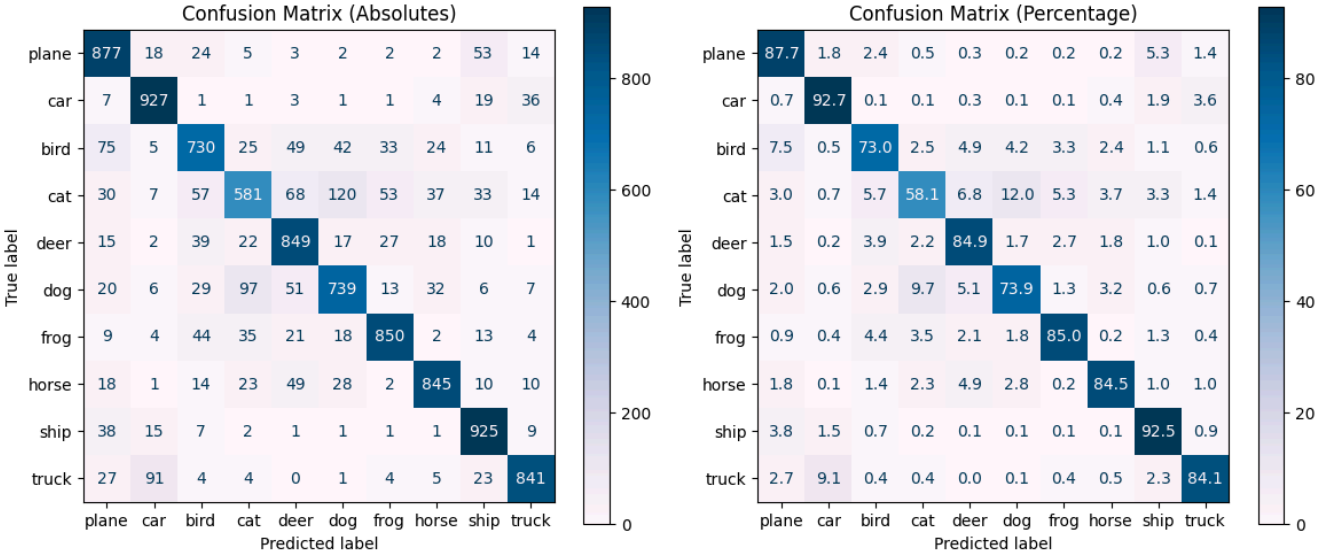

Figure 4: Confusion Matrix with absolute values (left) and percentages (right)

After looking at the confusion matrix, we decided to plot some of the images that were incorrectly classified by the model, in particular focusing on the most frequent errors made by our network. The highest percentages of misclassifications involve dogs being classified as cats (Figure 5), or viceversa (Figure 6).

Figure 5: 'Cat' images classified as 'Dog'

Figure 6: ‘Dog’ images classified as ‘Cat’

By looking at the images we might conclude that the model gets confused in particular when the animal is not shown in its entire figure, or by the fact that the background is often similar in both dogs and cats images.

Another example of quite frequent misclassification is shown in Figure 7 and it involves birds being classified as planes.

Figure 7: ‘Bird’ images classified as ‘Plane’

This is probably mostly due to the common blue background of the sky that appears in many bird and plane images and to the fact that the two subjects have similar features, such as colors or wings for example.

Lastly, the network made some mistakes in classifying trucks. The wrong label assigned more often in this case was ‘Car’. Once again, the main reason could be that both cars and trucks have similar features (wheels, windshields, colors), along with the fact that in some cases the dimensions of the two subjects in the images are similar, e.g. a car that takes up most of the image space or a truck that occupies only a fraction of it, as in the first or second image in Figure 8.

Figure 8: ‘Truck’ images classified as ‘Car’

4 Conclusions

The Convolutional Neural Network we defined to solve the CIFAR-10 classification task is able to achieve very promising results. Our experiments showed the critical importance of delineating the

right structure of the network and of finding the right hyperparameters that characterize it.
The misclassification analysis conducted in the end aimed to identify instances where the model's predictions deviated from the ground truth labels. This qualitative assessment facilitated a deeper understanding of the model's performance and could be useful to identify potential areas for improvement.