Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
Laboratório Maxwell de Micro-ondas e Eletromagnetismo Aplicado
Anais do Workshop de Micro-ondas ISSN 2675-4460

Image Identification and Classification through Convolutional Neural Networks

João Gabriel de Souza Bitencourt[1], Arnaldo de Carvalho Júnior[1], Walter Augusto Varella
[1] Federal Institute of Sao Paulo – Cubatao Campus, joao.bitencourt@aluno.ifsp.edu.br, [adecarvalhojr , varella] @ifsp.edu.br

Abstract - In this work, a convolutional neural network (CNN) was developed for image classification using the CIFAR-10 database. The database contains 60,000 color images distributed in 10 classes, 50,000 for training and 10,000 for testing. The network architecture was composed of multiple convolutional layers, followed by pooling and fully connected layers. The objective is to highlight relevant features of the images. The model was trained using the backpropagation technique with gradient descent optimization. Results demonstrated the effectiveness of CNN for the image classification task, with an accuracy of 99% achieved in the test set. The results indicate that the proposed approach is efficient for classifying images from the CIFAR-10 database, comparable to other techniques in the literature.

Keywords: Convolutional Neural Networks, Machine Learning, CIFAR10, Google Colab, Python

Introduction

An artificial neural network (ANN) is a machine learning (ML) model inspired by the functioning of the human brain, composed of artificial neurons organized in layers [1]. The ANN is splitted in the following main layers: the input layer, which receives the raw data (such as pixels of an image), the hidden layers, which process the data and learn patterns by applying mathematical operations, and the output layer, which provides the result as the classification of an image. The neurons in each layer perform an operation on the inputs, multiplying them by weights, adding these values and applying an activation function that defines whether the neuron will be "activated" or not. Activation functions such as sigmoid, hyperbolic tangent, rectified linear unit (ReLU), and Leaky-ReLU introduce nonlinearity, making the network capable to learn complex patterns [1].

In the *forward propagation process*, the input data is passed through the layers of the network. In each layer, operations performed on the neurons adjust the data until a final output is generated. The output is then compared with the expected (true) output, and the difference between them is calculated by a cost function, which quantifies the network's prediction error. Neural network learning occurs when this error is minimized over several iterations of the training process. To do this, the network uses the *backpropagation method*, in which the error is propagated back through the layers and the neuron weights are adjusted using the gradient descent method, which seeks to gradually reduce the cost function [1]. Figure 1 shows the structure of a neural network.

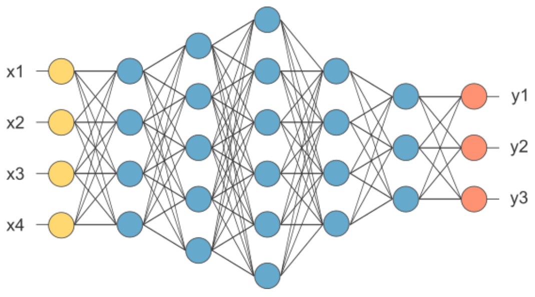


Figure 1 – Representation of a Neural Network. [1]

During training, the model goes through multiple iterations over the dataset, called epochs. At each epoch, weights of the ANN are adjusted based on the calculated error, learning to map the input data to the correct outputs. This continuous adjustment process allows the ANN to learn increasingly detailed patterns. The learning process brings the ability for the ANN to work with new and unseen data [2].

There are several types of neural networks, some of the most common being: the simple *perceptron*, which is the most basic form of neural network, with a single layer of neurons; *feedforward neural networks*, where data flows in a single direction, from input to output; *convolutional neural networks* (CNNs), which are specialized in image processing, applying convolution operations to detect visual patterns in different regions of the image; and r*ecurrent neural networks* (RNNs), used to process sequential data, such as text or time series, because they maintain information about previous states. In short, neural networks are powerful tools for finding complex patterns in data, adjusting their parameters through the training process and minimizing the error of their predictions to perform tasks such as classification, recognition and prediction with high efficiency [1].

Neural networks belong to the machine learning layer *within* the broader field of artificial intelligence (AI), specifically the subfield known as deep learning . Within the AI hierarchy, this is an area focused on algorithms and techniques that allow machines to learn from data, without needing programmed instructions for each specific task. Machine learning is a subfield of AI that encompasses different learning approaches, where artificial neural networks stand out for their power to identify complex patterns [2].

When these networks have many layers, they fall into the category of deep learning (DL). These deep ANNs. DL is particularly effective at handling large volumes of data and solve sophisticated problems such as natural language processing (NLP) and image recognition. These tasks requires to model complex and abstract patterns. Thus, neural networks sit at the intersection of machine learning and deep learning, offering efficient solutions to challenging problems in AI [1].

A. Convolutional Neural Networks

CNN is a type of ANN designed to handle grid-like structured data, as images. CNNs are proven to be used for computer vision, image classification and object detection. The reason is because CNNs are effective in extract and detect visual patterns. Figure 2 shows the structure of a CNN.

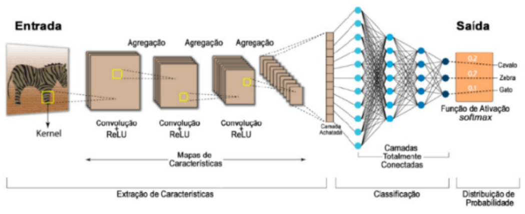


Figure 2 – Image classification by a CNN.

The core element of CNNs is the convolution operation, in which small filters are applied to blocks of an image to extract local features, such as edges and textures. These filters scan the image in different regions and generate a feature map that represents the presence of these patterns. After convolution, the data passes through an activation function, usually the ReLU, which introduces nonlinearity [4], allowing the network to learn more complex patterns. Then, pooling layers, such as Max Pooling, are used to reduce the dimensionality of the feature maps, summarizing information and making the model more efficient, as well as robust to small variations in the data. In the final part of the CNN, the features extracted by the convolutional layers are "flattened" and passed to fully connected layers, where each neuron is connected to all neurons in the previous layer. The *softmax* or *sigmoid* activation function is used to generate the final output of the network, which can be a class or a score.

CNN is trained using backpropagation, iteratively adjusting the weights of the filters and connections to minimize the cost function. Techniques such as dropout and regularization help to avoid overfitting and improve the generalization ability of the network. CNNs have proven to be extremely powerful in computer vision tasks due to their ability to automatically capture complex visual patterns, while reducing computational complexity through pooling and other techniques [2]. Figure 3 shows an example of layer-by-layer image analysis performed by a CNN.

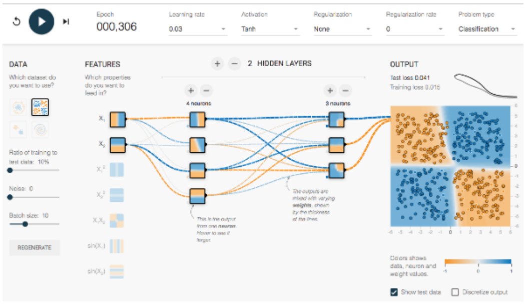


Figure 3 – Layer-wise image analysis, performed by a CNN. Available at https://blog.tadtarget.com/o-que-e-redes-neurais/

B. Database

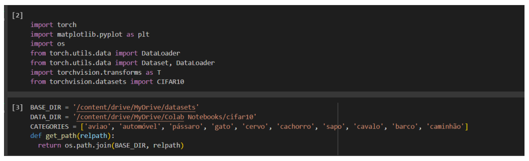
The success of an image recognition AI project begins with the analysis, selection, processing and filtering of the database to be used in training the CNN [3].

CNN datasets can be easily found online on data science platforms such as Kaggle [3]. Kaggle offers a wide variety of datasets that can be used to train CNNs, including textual data for natural language processing tasks, financial time series, weather data, and more. These datasets are available in ready-to-process formats, allowing researchers and developers to efficiently test and improve their CNN networks. In addition to Kaggle, other sources such as academic repositories, research organizations, or open-source platforms such as the UCI Machine Learning Repository also provide rich and varied datasets for training CNNs on different types of tasks [4].

DEVELOPMENT

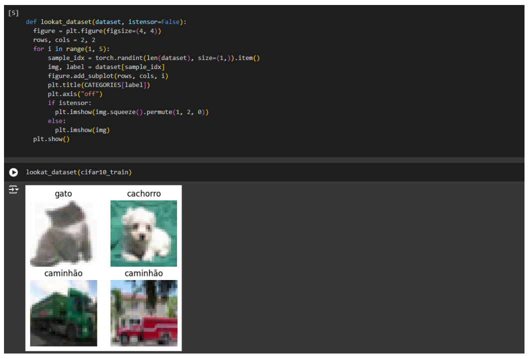
For development, the CIFAR-10 database was used, which consists of 60,000 color images in 10 different classes, with 6,000 images per class, provided by the *Canada Institute for* Advanced Research [5]. This database is widely used for image classification problems and was chosen for its moderate complexity and wide use in CNN network *benchmarks*. The construction of the algorithm and the training of the network were developed directly in Google Colab, a platform that facilitates the execution of Python code with access to computational resources such as GPUs, essential for accelerating the training of deep learning models.

The entire CNN framework was implemented in Python, using popular machine learning libraries such as PyTorch, TorchVision, and Matplotlib. Figures 4 and 5 represent the applications of the aforementioned libraries.



```
[2]
    import torch
    import matplotlib.pyplot as plt
    import os
    from torch.utils.data import DataLoader
    from torch.utils.data import Dataset, DataLoader
    import torchvision.transforms as T
    from torchvision.datasets import CIFAR10

[3] BASE_DIR = '/content/drive/MyDrive/datasets'
    DATA_DIR = '/content/drive/MyDrive/Colab Notebooks/cifar10'
    CATEGORIES = ['aviao', 'automóvel', 'pássaro', 'gato', 'cervo', 'cachorro', 'sapo', 'cavalo', 'barco', 'caminhão']
    def get_path(relpath):
      return os.path.join(BASE_DIR, relpath)
```

Figure 4 – Importing libraries and creating a class



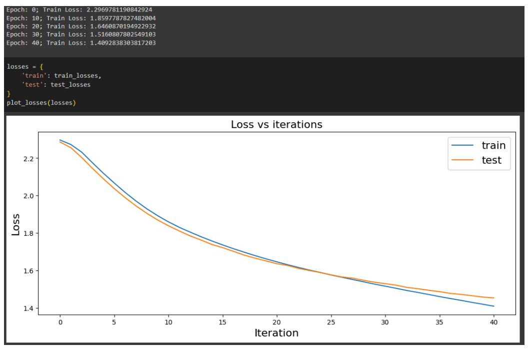
```
[5]
    def lookat_dataset(dataset, istensor=False):
      figure = plt.figure(figsize=(4, 4))
      rows, cols = 2, 2
      for i in range(1, 5):
          sample_idx = torch.randint(len(dataset), size=(1,)).item()
          img, label = dataset[sample_idx]
          figure.add_subplot(rows, cols, i)
          plt.title(CATEGORIES[label])
          plt.axis("off")
          if istensor:
            plt.imshow(img.squeeze().permute(1, 2, 0))
          else:
            plt.imshow(img)
      plt.show()

    lookat_dataset(cifar10_train)
```

Figure 5 – predefined sampling of 4 classes

The network was built with several interconnected layers, using the ReLU activation function in each intermediate layer and ending with a layer designed to classify the images. The input images were flattened *to* 3 red-green-blue (RGB) channels and 32x32 pixels to a single dimension before passing through the *multi-layer perceptron* (MLP) layers. The images were normalized, and the data was divided into training and testing sets to evaluate the model's performance. The model was trained using the categorical *cross-entropy cost function* and the *Adam optimizer* , with multiple epochs to ensure that the network adequately learned the characteristics of the different CIFAR-10 classes. Dataloaders were used to compile the images into batches*,* so that 64 images could be loaded at once for training.

In the code, the loss function used is *Cross-Entropy Loss*, which is ideal for classification tasks because it compares the model's predictions with the correct classes, penalizing incorrect predictions and encouraging the model to assign a higher probability to the correct class. The optimizer is *stochastic gradient descent (SGD)*, which adjusts the model's weights based on the gradient of the loss function. The *model.parameters* () *parameter* passes the model's weights and biases to the optimizer, while the learning rate (lr=1e-3) controls the magnitude of these updates, determining how much the weights change at each iteration to minimize loss and improve model performance. In this case, the iterations will have a learning rate of 0.001, resulting in an optimization of the learning result, causing few images to be lost during the process.

Later, we introduced code that defines the training and validation process of a machine learning model in PyTorch, as well as plotting losses *throughout* the training. The *train function* executes the training process: for each batch of images and labels from the *dataloader*, it moves the data to the device (such as a GPU), resets the gradients, makes predictions, calculates the loss using the provided loss function, performs backpropagation (*loss.backward* ()), and adjusts the model weights with the optimizer (*optimizer.step* ()), accumulating the total loss. The *validate function* is similar, but serves to evaluate the model, without updating the weights, only accumulating the loss and with *torch.no_grad* ( ) to disable the calculation of gradients, speeding up the process. The final loop executes the training and validation for several epochs (41 in this case), storing the losses of each epoch in lists for training and validation. These losses are then plotted by the *plot_losses* function, which generates a graph showing the evolution of the loss over time to evaluate the model's performance in different iterations, as shown in Figure 6.



```
Epoch: 0; Train Loss: 2.2969781190842924
Epoch: 10; Train Loss: 1.8597787827482004
Epoch: 20; Train Loss: 1.6460870194922932
Epoch: 30; Train Loss: 1.5160807802549103
Epoch: 40; Train Loss: 1.4092838303817203
```

```
losses = {
    'train': train_losses,
    'test': test_losses
}
plot_losses(losses)
```

Figure 6 – graph relating Losses vs MLP Iterations

Next, a code was applied that implements functions to evaluate the performance of a classification model in PyTorch, using the confusion matrix and accuracy per class.

The *make_confusion_matrix function* creates a confusion matrix, which compares the model's predictions to the true labels for each example in the dataset, accumulating the hit and miss counts into a matrix of dimensions [n_classes x n_classes]. The *evaluate_accuracy function* uses this confusion matrix to calculate and print the individual accuracy for each class, as well as the model's overall accuracy, i.e. the percentage of correct predictions across all classes. Finally, the *test function* also evaluates the model's performance, but without using the confusion matrix directly; it counts the correct and total predictions for each class and calculates the accuracy in a similar way, also printing the per-class and overall accuracy. Both functions help measure how well the model is classifying the data in terms of hits and misses per class, providing a more detailed view of the model's performance beyond just overall accuracy.

The code developed generates a graphical visualization of the confusion matrix as a *heatmap* using the *seaborn* and *matplotlib libraries*. First, the confusion matrix is calculated by the *evaluate_accuracy function*, which compares the model's predictions to the true labels for each class in the test set. The resulting matrix is then converted to a Python list and passed to the *sn.heatmap function*, which creates the graph. The *heatmap* shows the counts of correct and incorrect predictions across classes, with annotations enabled (*annot=True*) to display the values in the cells. The graph size is adjusted to 12x12 inches, and the font size of the annotations is set to make the numbers within the graph easier to read. This *heatmap* provides a clear and visual way to understand the model's performance across different classes, highlighting the classification hits and misses, as shown in Figure 7.

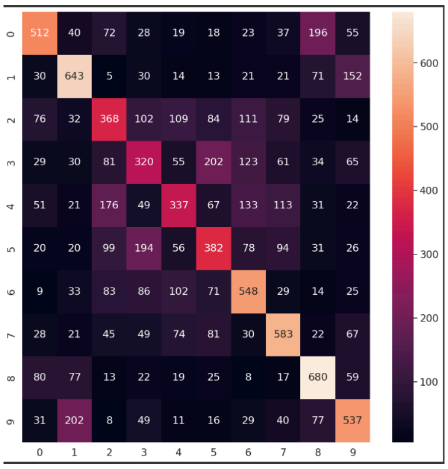


Figure 7 – *Heatmap* of the applied database

A. Applying a CNN

A CNN using the PyTorch library was used to perform the image classification task. The network is defined by the *ConvolutionalModel class*, which inherits from *nn.Module* , the base class of all neural networks in PyTorch. The model has two main parts: a sequence of convolutional layers (using *nn.Conv 2d* and *nn.MaxPool2d*) and a sequence of fully connected layers (also known as " *fully connected* "). The convolutional layers serve to extract relevant features from the input images, while the fully connected layers perform the classification task itself.

The *__init_ _(*) constructor, the first convolutional layer takes a three-channel input (representing an RGB image) and uses 16 filters (or feature maps) with a kernel size of 3x3. The ReLU activation operator is used to introduce non-linearities into the model, followed by a *max pooling layer*, which reduces the dimensions of the input. The second convolutional layer uses 32 filters and follows the same pattern, with another *max pooling layer*. After the convolutional layers, the image is "flattened" (with *torch.flatten*) and passes through two linear layers, the first reducing to 256 neurons and the second to 10 outputs, which presumably correspond to the 10 output classes.

Below the model definition, the code performs the configuration for CNN training. The loss function and optimizer are the same as those used previously in MLP, with a learning rate (lr) of 0.001.

The training loop iterates for 51 epochs, using the *train*( ) and *validate*() *functions* to train and validate the model, respectively. After each epoch, the training loss (TL) is stored, and every 10 epochs, the code prints the current loss. At the end, the TL and test are stored in a dictionary called *conv_losses, which is then passed to the plot_losses* ( ) function, which generates a graph comparing the losses over the epochs, as shown in Figure 8.

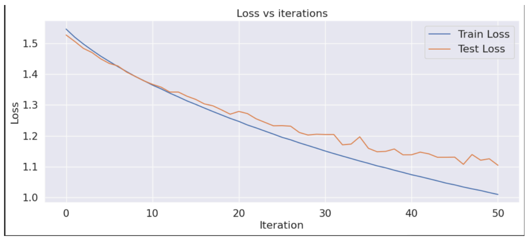
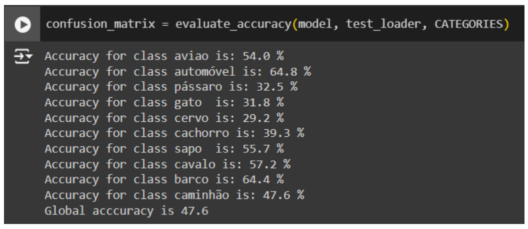


Figure 8 – CNN Losses vs Iterations Graph

RESULTS AND DISCUSSIONS

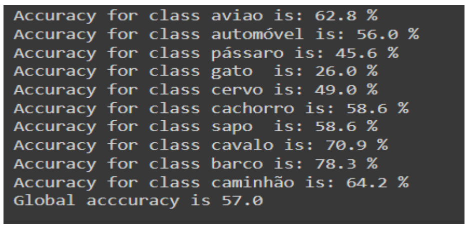
The results obtained with MLP showed a variation in accuracy between the different classes, as shown in Figure 9. The "car" class obtained the highest accuracy, with 64.8%, followed by "boat" with 64.4%, while "horse" and "airplane" also performed relatively well, with 57.2% and 54%, respectively. However, some classes performed much lower, such as "bird", with only 32.5%, and "cat", with 31.8%. The "deer" and "dog" classes also performed below 40%, with 29.2% and 39.3% accuracy, respectively. The "truck" class presented an accuracy of 47.6%, which coincides with the overall accuracy of the model, which was 47.6%.



```
confusion_matrix = evaluate_accuracy(model, test_loader, CATEGORIES)

Accuracy for class aviao is: 54.0 %
Accuracy for class automóvel is: 64.8 %
Accuracy for class pássaro is: 32.5 %
Accuracy for class gato  is: 31.8 %
Accuracy for class cervo is: 29.2 %
Accuracy for class cachorro is: 39.3 %
Accuracy for class sapo  is: 55.7 %
Accuracy for class cavalo is: 57.2 %
Accuracy for class barco is: 64.4 %
Accuracy for class caminhão is: 47.6 %
Global acccuracy is 47.6
```

Figure 9 – MLP accuracy result

The results indicate that although MLP performs satisfactorily in some classes, there are areas that need adjustments, especially for classes with low accuracy, such as “cat”, “bird” and “deer”. In the “*Train Loss*” category, the result obtained was 1.4% [10]. After a new attempt, the CNN results showed an overall improvement compared to MLP, with an overall accuracy of 57.0%, as shown in Figure 10.

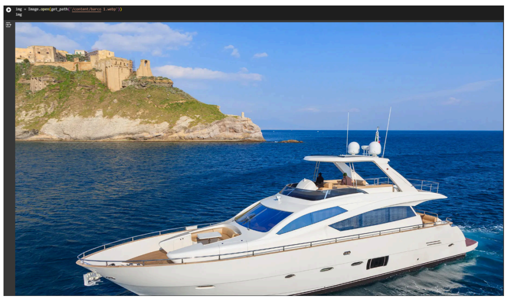


```
Accuracy for class aviao is: 62.8 %
Accuracy for class automóvel is: 56.0 %
Accuracy for class pássaro is: 45.6 %
Accuracy for class gato  is: 26.0 %
Accuracy for class cervo is: 49.0 %
Accuracy for class cachorro is: 58.6 %
Accuracy for class sapo  is: 58.6 %
Accuracy for class cavalo is: 70.9 %
Accuracy for class barco is: 78.3 %
Accuracy for class caminhão is: 64.2 %
Global acccuracy is 57.0
```

Figure 10 – CNN accuracy results

The “boat” class performed best, achieving 78.3%, followed by “horse” with 70.9%. Other classes, such as “airplane”, “truck” and “dog”, also showed reasonable accuracy, with 62.8%, 64.2% and 58.6%, respectively. However, there are still classes with lower performance, such as “cat”, which had the lowest accuracy, with 26.0%, and “bird”, with 45.6%. The “car” and “deer” classes were around 50%, with 56.0% and 49.0%, respectively. These results indicate that CNN is more effective in several classes compared to MLP, especially for “boat”, “horse” and “airplane”, but there is still room for improvement in classes such as “cat” and “bird”.

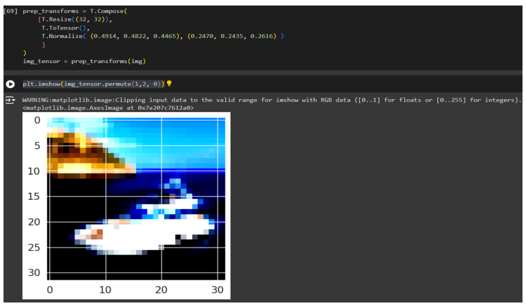
To test the accuracy of the CNN, a test image of a boat, as shown in Figure 11, was used to evaluate its performance.



```
img = Image.open(get_path('/content/barco 1.webp'))
img
```

Figure 11 – Test image used [11]

The test image of a boat was transformed before being fed into the CNN using a series of preprocessing steps. First, the image was resized to 32x32 pixels with *T.Resize*, then converted to a tensor with *T.ToTensor*, and normalized to specific mean and standard deviation values for each color channel (red, green, and blue) using *T.Normalize*. After these transformations, the resulting tensor was visualized with *plt.imshow*, adjusting the dimensions to the appropriate format with permute(1, 2, 0). These transformations ensure that the image is in the correct format for the model to process, standardizing the data to improve the network's accuracy, as shown in Figure 12.



```
[69] prep_transforms = T.Compose(
         [T.Resize((32, 32)),
          T.ToTensor(),
          T.Normalize( (0.4914, 0.4822, 0.4465), (0.2470, 0.2435, 0.2616) )
          ]
     )
     img_tensor = prep_transforms(img)

plt.imshow(img_tensor.permute(1,2, 0))

WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
<matplotlib.image.AxesImage at 0x7e207c7612a0>
```

Figure 12 – Image generated after pre-processing phase.

After preparing the image for the model, it was adjusted to simulate a batch, as the model expects. The model was placed in evaluation mode, that is, ready to make predictions without training interference. The image was then passed through the CNN, which generated an output indicating the probability of it belonging to each category. These probabilities were converted to percentages, and associated with each class, such as "airplane", "dog", "boat", etc. This data was organized in a table and, finally, visualized in a horizontal bar chart, showing the model's confidence in its classification for each of the analyzed categories, as shown in Figure 13 [13].

The results can be seen as shown in Figure 13. The results showed that the CNN identified the image with high confidence in the "barco" (boat, in light brown) class, with a score of 95.28%, which confirms that the model correctly recognized the category of the image tested. The other classes had significantly lower scores, with "automóvel” (car, in orange) at 2.59% and "avião" (airplane in dark blue) at 1.90%, while the others, such as “pássaro” (bird, in green), "gato" (cat, in red), "cachorro" (dog, in brown), "cavalo" (horse, in grey), "cervo” (deer, in purple), "sapo" (frog, in pink), and “caminhão” (truck, in light blue) were practically not considered relevant by the model, with scores close to zero. This demonstrates that the model was very assertive in identifying the image as belonging to the "boat" class, highlighting the effectiveness of the CNN in this specific test.

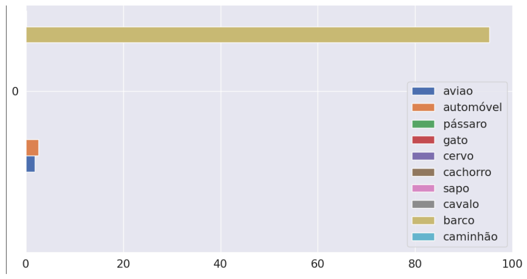


Figure 13 – Results presented by CNN

CONCLUSION

The objective of this research was achieved and a CNN developed was able to classify different types of vehicles. Testing other images on the CNN can provide a broader view of the model’s performance in different categories. By inserting images from other classes, such as cars, planes or animals, it is possible to observe how the CNN behaves and which classes it can identify with greater or lesser confidence. This process helps to assess whether the model is well generalized to different types of data or whether it has difficulty distinguishing certain categories. Furthermore, by comparing the predictions for different images, we can identify possible adjustments to the model or to the transformations applied to the images that could improve its accuracy.

REFERENCES

[1] RAUBER, T. W. Artificial neural networks. Federal University of Espírito Santo, v. 29, 2005.A.

[2] CARVALHO, A. Artificial Neural Networks: Algorithms Artificial neural networks: powerful algorithms for AI and ML applications (PORTUGUESE). EAILab, April 2024. Available at: https://eailab.labmax.org/2024/04/03/artificial-neural-networks-powerful-algorithms-for-ai-and-ml-applications/. Accessed in Oct 07, 2024.

[3] CARVALHO, A. Access Datasets for AI Projects (PORTUGUESE). EAILab, June 2024. Available at: https://eailab.labmax.org/2024/06/17/datasets-de-acesso-livre-para-projetos-de-ia/. Accessed in Oct 07, 2024.

[4] CARVALHO, A. et al. Model reference control by recurrent neural network built with paraconsistent neurons for trajectory tracking of a rotary inverted pendulum, Applied Soft Computing, 2022, 109927, ISSN 1568-4946, DOI: 10.1016/j.asoc.2022.109927.

[5] KRIZHEVSKY, A.; NAIR, V.; HINTON, G. The Cifar-10 dataset. Computer Science, Univeristy of Toronto, 2024. Available at: https://www.cs.toronto.edu/~kriz/cifar.html. Accessed on Oct 07, 2024.