Maria Jose Viveros Riquelme

Professor Patricia McManus

ITAI 1378

September 25, 2025

Reflection Journal 3: Chihuahua or Muffin with CNN

The challenge of classifying Chihuahuas and Muffins is a common exercise to demonstrate the capabilities of deep learning models in recognizing visually similar images. The primary objective of this laboratory was to implement and train a Convolutional Neural Network, a type of deep learning model specifically designed to process grid-like data, such as images, by applying convolutional filters that learn spatial hierarchies of features (Goodfellow, Bengio, and Courville). Nevertheless, the *traditional neural network (NN)* used in the previous workshop, which flattened the image inputs and relied solely on fully connected layers, the *CNN* preserves the spatial structure of images and automatically extracts hierarchical features through convolution and pooling layers. This architecture enabled the *CNN* to achieve higher accuracy and better generalization, while being more parameter-efficient for image data (Goodfellow, Bengio, and Courville). Although training per epoch was slightly longer due to convolution operations, the *CNN* required fewer epochs to reach optimal performance, making it overall more effective than the traditional *NN* for this image classification task.

The algorithm of Chihuahua or Muffin with CNN had six phases: setup and imports, data preparation, model definition, training setup, model training, and model evaluation. During the setup and *import phase,* the algorithm begins by installing *PyTorch* and *torchvision* libraries to prepare the environment for deep learning tasks. PyTorch is used for training neural networks, while torchvision is used for managing datasets and transformations. Then, it installs *DataLoader* for efficient batching, Matplotlib for visualization, and *tqdm* for progress tracking. Finally, the algorithm checks if a GPU with CUDA support is available; if so, it assigns computations to the GPU (Graphics Processing Unit) to speed up processing, otherwise it defaults to the CPU (Central Processing Unit) for execution.

During the data preparation phase, the program begins by verifying the dataset structure, which is divided into two main folders: *train* and *validation*. The training set includes 65 Chihuahua images and 55 Muffin images, while the validation set contains 17 Chihuahua images and 13 Muffin images. Subsequently, the images are resized to 128×128 pixels to standardize their dimensions before training, using a configuration with three convolutional layers, which are sufficient to capture basic features such as edges and simple textures. However, when the network is expanded to five convolutional layers, the input image size is increased to 254×254 pixels. This change occurs because deeper architectures require larger input dimensions to preserve sufficient spatial information across layers (Goodfellow, Bengio, and Courville; PyTorch Documentation). The program uses PyTorch and Torchvision for dataset handling, while the *dataloader* organizes the images into batches of 32. Finally, *Matplotlib* and *tqdm* are utilized to visualize results, monitor progress during training, and prepare the data in tensor format for efficient model training and evaluation.

In the model definition phase, the model is built using *PyTorch* as a convolutional neural network (CNN) to classify Chihuahuas and Muffins. Initially, the architecture consisted of three *convolutional layers;* subsequently, two more layers were added, enabling the network to extract more complex hierarchical features, ranging from edges and textures to abstract shapes (PyTorch Documentation). The *nn.Conv2d* layer performs 2D convolution on inputs, extracting spatial features from images to help the

neural network learn patterns (PyTorch Documentation). In other words, the *nn.Conv2d* layer increases the number of filters progressively 32→64→128→256→512 while maintaining the input channels at three for *RGB* (Red, Green, and Blue) images, with a kernel size of 3 and padding of 1 to preserve spatial dimensions. The *nn.MaxPool2d (Kernel_size=2, stride=2)* es the spatial resolution after each convolution, balancing efficiency with feature retention. The classifier section utilizes a *linear layer* (flattened_size→512), *ReLU activation*, *Dropout* (0.5) to mitigate overfitting, and a final *linear layer* to produce two-class outputs. Finally, the model runs via CUDA on CPU, and the function *torchsummary.summary* displays the layer details, output shapes, and parameter counts.

In the training setup phase, the model's ability to handle different optimizers was demonstrated. The loss function was defined as *nn.CrossEntropyLoss()*, a standard choice for classification tasks. The first optimizer used in testing was *Adam* with a *learning rate* (*lr*) of 0.001, a versatile choice suitable for general-purpose training. Other tests included *Adam* with a higher *learning rate (lr)* of 0.05, which can cause instability but was still manageable, *Stochastic Gradient Descent (SGD)* with a very small *learning rate (lr)* of 0.0001 often combined with *momentum (0.9)* to accelerate convergence, and finally *AdamW* with a l*earning rate (lr)* of 0.0005 and weight decay of 0.01 to add *L2 regularization* and prevent overfitting. Each optimizer has different parameters because they implement distinct strategies for updating weights efficiently, controlling the step size, and managing overfitting, making these choices necessary for stable and practical model training (DataCamp).

The model training utilizes the *train_model function* to train a neural network for a specified number of epochs, iterating over both the training and validation phases. For each epoch, it sets the model to training mode during training and evaluation mode during validation, computes the forward pass, calculates the loss using *criterion,* and updates the weights using *optimizer* only in the training phase. Originally, the model was trained for 10 epochs; however, during testing, the number of epochs was increased to 20 and 30. Increasing epochs caused the training process to slow down significantly without improving accuracy, showing that for this model, the optimal number of epochs remains 10. Longer training did not provide better performance and only increased processing time.

Finally, in the model evaluation phase, the algorithm starts with the function *evaluate_model* to test a trained neural network on a validation dataset. The model is set to evaluation mode, and gradients are disabled to speed up computation and save memory. For each batch of inputs, the model generates predictions, which are collected alongside the true labels. After evaluating all data, the validation accuracy is calculated by comparing the predicted labels with the true labels. Additionally, the *plot_results* function visualizes a subset of validation images, showing each image with its predicted and true label, colored green if correct and red if incorrect. In the first test, the model achieved a high accuracy of 0.9667. However, in three subsequent tests, the accuracy varied between 0.5667 and 0.9333, as summarized in Table 1.

Table 1. Tests and results (own elaboration)

Test	Epoch	Learning Rate	Optimizer	Layers	Image Size	Accuracy
0	10	0.001	Adam	3	128 x 128	0.9667
1	30	0.05	Adam	4	128 x 128	0.5667
2	30	0.0001	SGD (m =0.9)	4	128 x 128	0.7000
3	10	0.001	AdamW(wd= 0.01)	5	256 x 256	0.9333

The table illustrates that model performance is susceptible to hyperparameters and architecture. Test 0 achieved the highest accuracy (0.9667) using *Adam* with a moderate learning rate (0.001), 3 layers, and 128×128 images, indicating that simpler architectures and moderate learning rates work well for small datasets. Increasing the learning rate to 0.05 (Test 1) resulted in unstable training and a sharp drop in accuracy (0.5667), whereas using *SGD* with a very small learning rate (Test 2) improved stability but still underperformed (0.7000). *AdamW* with weight decay, 5 layers, and larger images (Test 3) achieved 0.9333, indicating that regularization and deeper networks are beneficial, but the benefit is marginal for small datasets. Consequently, this experiment demonstrates that very large learning rates destabilize the model, as they result in excessively large weight updates during training. Instead of gradually moving toward the minimum of the loss function, the optimizer overshoots it, causing the loss to fluctuate or even diverge. Overall, for small datasets, the best model is Test 0: *Adam* optimizer, moderate learning rate, fewer layers, and standard image size, as it achieves the highest accuracy with minimal complexity.

In real-world applications, CNN-based image classification models are widely used in facial recognition technology, which has become a crucial component in security and identification systems. For example, in England, facial recognition is deployed in airports, train stations, and public spaces to identify individuals, prevent crime, and enhance public safety. These models can also be used in access control systems, such as unlocking devices or securing facilities, and in law enforcement to assist investigations. The technology demonstrates how CNNs can process complex visual data to recognize subtle features, providing fast and automated identification that would be challenging for humans to perform manually. However, these tools can have limitations—sometimes a person could be misidentified as a giant muffin! (a light-hearted way to illustrate classification errors).

Ethical issues surrounding the deployment of facial recognition models raise serious concerns. Privacy is a key issue, as people can be monitored without explicit consent, thereby risking violations of their civil rights. Biases in the training data can lead to misidentification, particularly for specific demographic groups, resulting in unfair treatment or false accusations. Moreover, transparency and accountability are essential, as officials and developers need to ensure that system decisions can be explained and verified. Responsible use requires strict protections, informed consent when possible, and ongoing oversight to prevent abuse and safeguard individual rights.

References

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, https://mitpress.mit.edu/9780262035613/deep-learning/, Accessed 21 Sept. 2025

DataCamp. *"AdamW Optimizer in PyTorch Tutorial." DataCamp*, 18 Apr. 2024, https://www.datacamp.com/tutorial/adamw-optimizer-in-pytorch. Accessed 23 Sept. 2025.

"torch.nn.Conv2d." *PyTorch Documentation*, PyTorch Foundation, 2025, https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html. Accessed 23 Sept. 2025

torch.nn.CrossEntropyLoss." *PyTorch Documentation*, PyTorch Foundation, 2025, https://docs.pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Accessed 21 Sept. 2025.

"torch.optim.Adam." *PyTorch Documentation*, PyTorch Foundation, 2025, https://docs.pytorch.org/docs/stable/generated/torch.optim.Adam.html. Accessed 22 Sept. 2025.

"torch.optim.AdamW." *PyTorch Documentation*, PyTorch Foundation, 2025, https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html. Accessed 23 Sept. 2025.