# Bird Image Classification

**Sébastien Meyer**
Institut Polytechnique de Paris, France
sebastien.meyer@ip-paris.fr

## Abstract

*This report details the different approaches I have implemented in order to get the best possible score I could on the bird image classification competition. The main task was to predict the category of new, unseen birds based on training and validation sets which were given.*

## 1. Training my own Convolutional Network

The dataset is comprised of 1,082 training images, 103 validation images and 517 test images of various sizes. There are 20 different classes. In order to capitalize on what I learned about convolutional networks, I decided to build my own CNN.

Based on the proposed baseline, the architecture of my convolutional network was made up of two convolutional layers each having 32 out channels. In-between, I applied padding in order to retrieve feature maps of same sizes as input images, then I added a batch normalization layer, a ReLU activation function and a max pooling layer. After this convolutional part, there were two different fully connected layers separated with a ReLU activation function. The model was trained during 10 epochs, with a default stochastic gradient descent (SGD) optimizer with a learning rate of 0.01 and a momentum of 0.5.

After some trials, I decided to submit my first predictions. However, the results of my model were pretty bad. Indeed, I achieved an accuracy of 0.17419 on the leaderboard.

## 2. Transfer Learning of Vision Transformers

The *huggingface* Python package allows to easily retrieve pretrained models to train and use on new tasks. In particular, research teams from top companies such as Google and Facebook upload state-of-the-art models to the community. On the other hand, particularly in the field of image classification, transformers have proved to outperform classical convolutional models. Following this, I used the well-known *google/vit-base-patch16-224* Vision Transformer model for our task.

After retrieving the model, I used my own training loop to optimize the model. Most importantly, I kept using the SGD optimizer, but with a learning rate of 1e-4. Also, I had to drastically reduce the batch size in order for the data to fit on my own GPU. Simply by using transfer learning, my new submission would reach an accuracy of 0.61290 on the leaderboard, which was a great improvement of my last score. However, I noticed that the corresponding validation accuracy was 0.79612. Clearly, the transfered model was overfitting on both the training and validation sets.

## 3. Data Augmentation & Hyperparameter Tuning

On the one hand, I decided to leverage data augmentation in order to simulate more training samples. The first technique I used was color jitter, which randomly modifies images' channels to change their brightness, contrast and saturation. Then, random rotations and horizontal flips would be applied to the images. After some trials, it appeared that the color jitter technique always worsened my validation accuracy, so I dropped this technique.

On the other hand, hyperparameter tuning allows to find the best hyperparameters for the model. To this end, I used the *optuna* Python package to optimize the learning rate and momentum for the SGD optimizer as well as the scale and ratio parameters for the random resizing and cropping of input images, the degrees of random rotation and the probability of horizontally flipping the image. Finally, using all these techniques, I was able to reach an accuracy of 0.85806 on the leaderboard!

| Model | Accuracy |
|---|---|
| CNN | 0.17419 |
| Transfer learning: *google/vit-base-patch16-224* | 0.61290 |
| Transfer learning: *google/vit-base-patch16-224* + Data Augmentation + Hyperparameter Tuning | 0.85806 |