Problem 1: Noisy Birds Classification

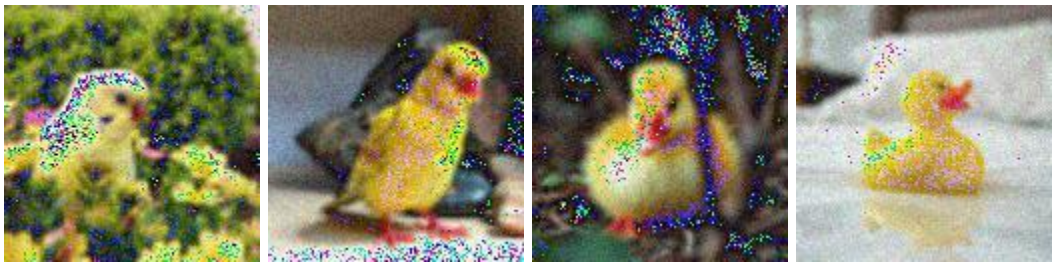
Introduction:

Image classification is the task of assigning a label to an image from predefined categories. It is commonly used in applications like object detection and medical imaging. Traditionally, models are trained with labeled datasets, where each image has an associated class label.

However, collecting labeled data can be time-consuming. In contrast, unlabeled data—images without labels—are often abundant. To leverage this, techniques like semi-supervised learning combine a small labeled dataset with a large amount of unlabeled data, improving model performance and reducing the need for extensive labeling.

Dataset:

The dataset used in this problem consists of bird images categorized into four classes: budgie, rubber duck, canary, and duckling. Each image has dimensions of 128x128 pixels and is stored in its respective class folder. The dataset contains a total of 1,358 images, with around 40 labeled images per class, while the remaining images are unlabeled and stored in the "unlabeled" folder. The labeled data provides ground truth for training, but the large portion of unlabeled data introduces additional complexity.
To further increase the challenge, noise has been added to each sample. Noise removal algorithms are not an option, as the test set, which is also noisy, remains inaccessible.
Here you can see a sample from each class:



Problem Statement:

The objective is to develop a model that can achieve optimal results. The challenge lies in effectively utilizing the unlabeled data to improve classification accuracy.

Evaluation

Your model is evaluated on a test set which also consists of four distinct classes, and all samples are labeled, with no unlabeled data in the test set. This balanced distribution across classes ensures a fair assessment of the model's performance. The performance of your model is measured in Accuracy.

Submission

You are required to submit both the model weights and the code that defines the model architecture. Zip both files into a single archive. The Python file containing the model architecture should be named **model.py**, and you must include any necessary library imports in your code. Ensure that neither of these files is placed in any subdirectories within the zip file.

```
import zipfile
torch.save(model.state_dict(),'model.pth')
with zipfile.ZipFile('submission.zip', 'w') as zipf:
    zipf.write('model.pth')
    zipf.write('model.py')
```

Limitations:

- The model must be implemented using **torch** and **torchvision** only (no other deep learning libraries are allowed for the model architecture).
- The class for the model must be named **Model**, and participants **should not change this name**.
- The model size should not exceed 70 MB.
- The input of the model should have the following dimensions: (Batch_sizex128x128x3)
- The output should be in the shape of (Batch_size x 4)
- You can use any predefined/pre-trained model or design your own model.
- **Your architecture code must not require any downloading from the judge server.**

Version: 1.4