25-Bird-Image-Classification

Rupak Neupane
Khwopa College of Engineering
neupanerupak7@gmail.com

1 Introduction

This repository contains the implementation of an image classification model of 25 categories of birds using PyTorch. The notebook includes data preparation, model training, and evaluation steps for an image classification task.

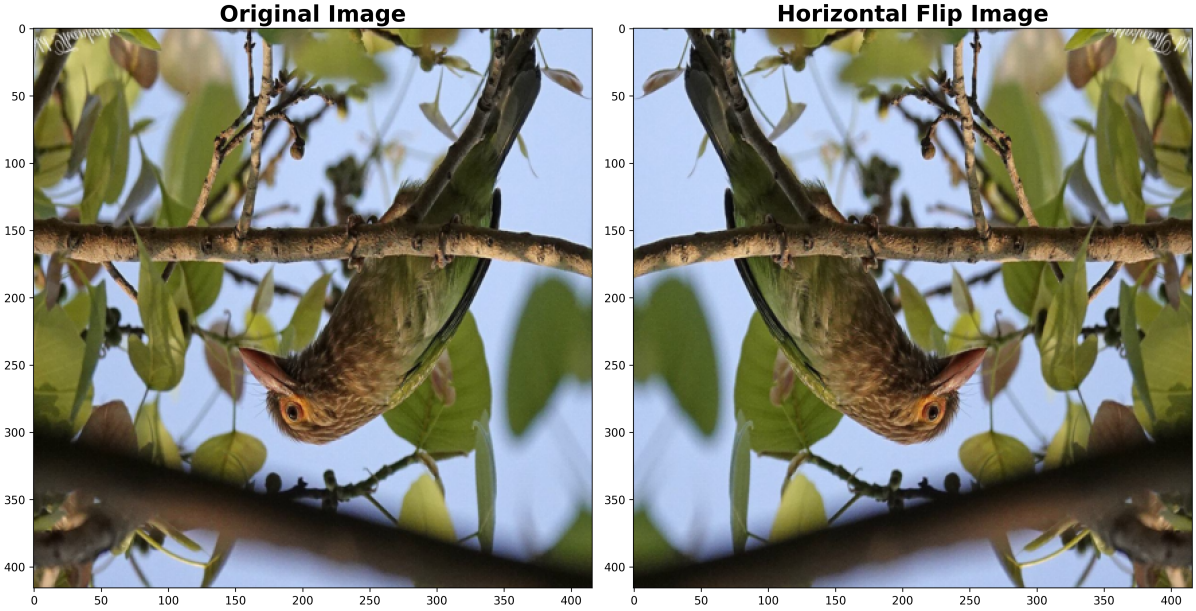
2 Requirements

- Python 3.10 or higher
- PyTorch 1.10 or higher
- torchvision
- tqdm
- numpy

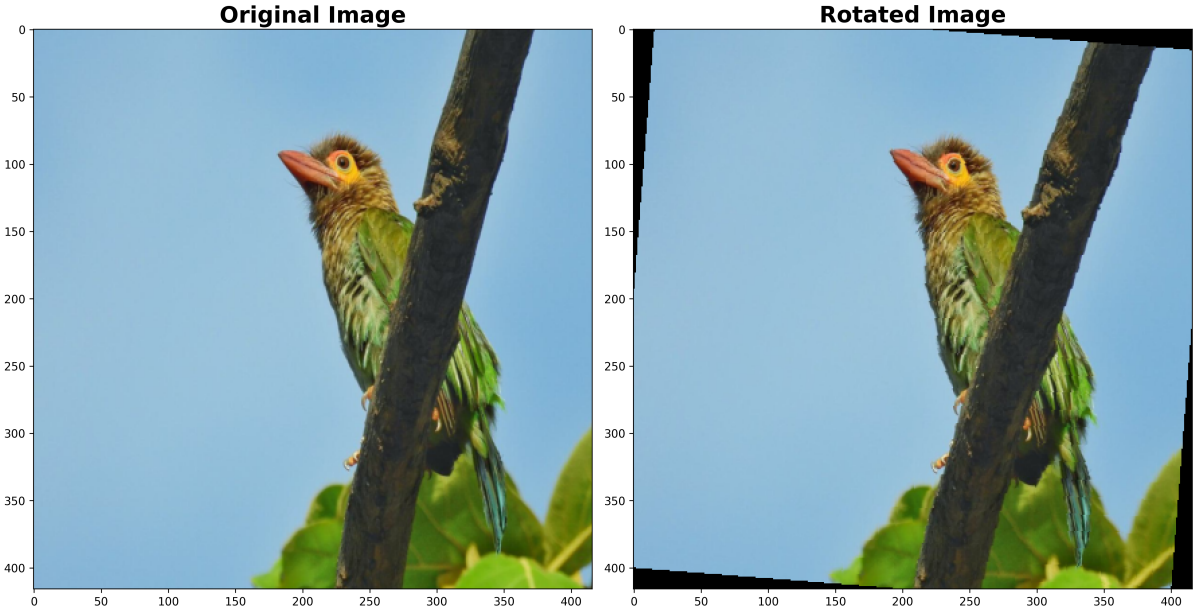
3 Data Preprocessing

3.1 Data Augmentation

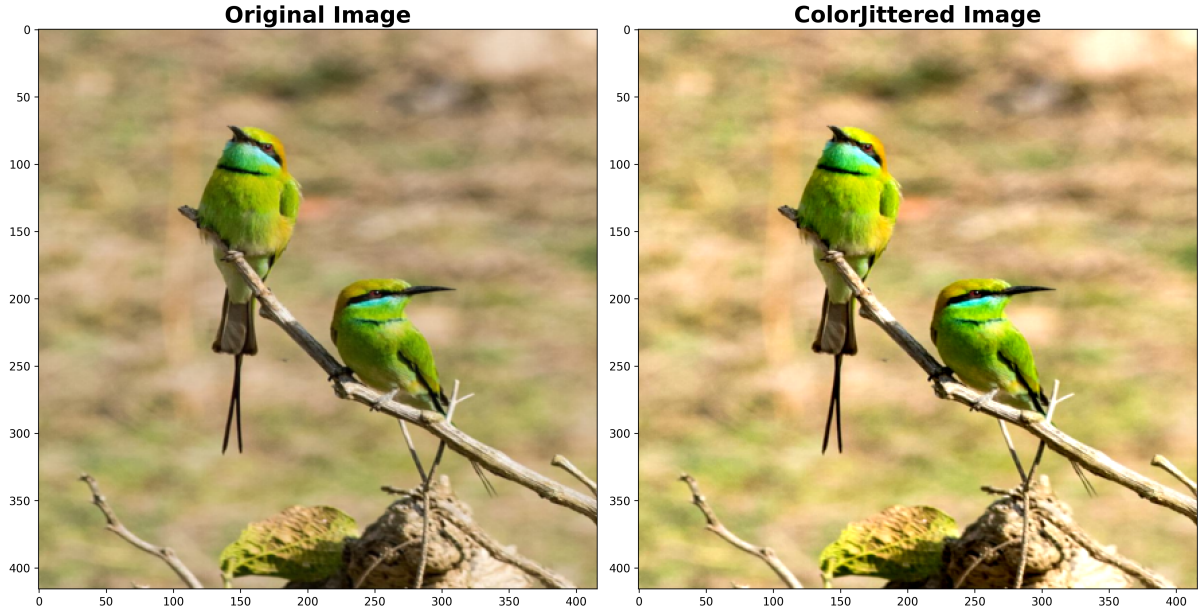
1. *RandomHorizontalFlip*



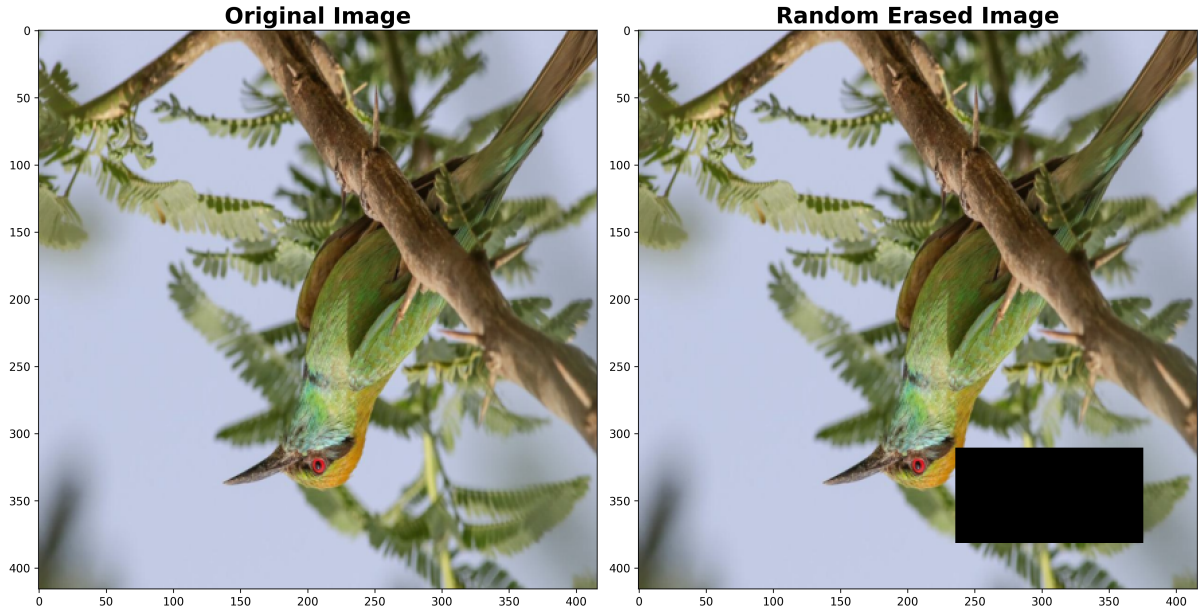
2. *RandomRotation*



3. *ColorJitter*



4. *RandomErasing*



3.2 Data Preparation

Data preprocessing steps include:

1. Providing paths for train and validation sets
2. Calculating mean and standard deviation for normalization
3. Applying image transformations (resize, normalize)

4 Model Building

The model is built using PyTorch and includes:

- A base class for image classification
- Convolutional and Depthwise Separable Convolutional blocks
- Fully connected layers with dropout

The `ResNet9` class defines the complete model architecture.

4.1 Model Architecture

The ResNet9 model consists of the following components:

1. Initial Convolutional Block
2. Depthwise Separable Convolutional Layers
3. Residual Blocks
4. Global Average Pooling
5. Classifier

4.1.1 Layer Details

- **conv1**: Initial Convolutional Block
  - Conv2d: 3 input channels, 64 output channels, 3x3 kernel, stride 2, padding 1
  - BatchNorm2d: Normalizes the 64 channels
  - ReLU6: Activation function
- **conv2**: Depthwise Separable Convolution
  - Depthwise Conv2d: 64 channels, 3x3 kernel, stride 2, padding 1, groups=64
  - Pointwise Conv2d: 64 input channels, 128 output channels, 1x1 kernel
  - BatchNorm2d: Normalizes the 128 channels
  - ReLU6: Activation function
- **conv3**: Depthwise Separable Convolution
  - Depthwise Conv2d: 128 channels, 3x3 kernel, stride 2, padding 1, groups=128
  - Pointwise Conv2d: 128 input channels, 256 output channels, 1x1 kernel
  - BatchNorm2d: Normalizes the 256 channels
  - ReLU6: Activation function
- **conv4**: Depthwise Separable Convolution
  - Depthwise Conv2d: 256 channels, 3x3 kernel, stride 2, padding 1, groups=256

  - Pointwise Conv2d: 256 input channels, 512 output channels, 1x1 kernel
  - BatchNorm2d: Normalizes the 512 channels
  - ReLU6: Activation function
- **conv5**: Depthwise Separable Convolution
  - Depthwise Conv2d: 512 channels, 3x3 kernel, stride 2, padding 1, groups=512
  - Pointwise Conv2d: 512 input channels, 1024 output channels, 1x1 kernel
  - BatchNorm2d: Normalizes the 1024 channels
  - ReLU6: Activation function
- **avgpool**: Adaptive Average Pooling
  - AdaptiveAvgPool2d: Reduces spatial dimensions to 1x1
- **classifier**: Fully Connected Layer
  - Dropout: With probability p=0.2
  - Linear: 1024 input features, 25 output features (classes)

Key Points:

- All convolutional layers (except the first) use depthwise separable convolutions for efficiency.
- Stride 2 is used in `conv1`, `conv2`, `conv3`, `conv4`, and `conv5` for downsampling.
- ReLU6 is used as the activation function throughout the network.
- Two residual blocks (`res1` and `res2`) are used to facilitate gradient flow.
- The final classifier uses dropout for regularization before the linear layer.

#	Layer	Output Shape	Param #
1	**conv1: Initial Convolutional Block**		
1.1	Conv2d (3 in, 64 out, 3x3, stride 2, pad 1)	[-1, 64, 208, 208]	1,728
1.2	BatchNorm2d (64 channels)	[-1, 64, 208, 208]	128
1.3	ReLU6	[-1, 64, 208, 208]	0
2	**conv2: Depthwise Separable Convolution**		
2.1	Depthwise Conv2d (64, 3x3, stride 2, pad 1, groups=64)	[-1, 64, 104, 104]	576
2.2	Pointwise Conv2d (64 in, 128 out, 1x1)	[-1, 128, 104, 104]	8,192
2.3	BatchNorm2d (128 channels)	[-1, 128, 104, 104]	256
2.4	ReLU6	[-1, 128, 104, 104]	0
3	**conv3: Depthwise Separable Convolution**		
3.1	Depthwise Conv2d (128, 3x3, stride 2, pad 1, groups=128)	[-1, 128, 52, 52]	1,152
3.2	Pointwise Conv2d (128 in, 256 out, 1x1)	[-1, 256, 52, 52]	32,768
3.3	BatchNorm2d (256 channels)	[-1, 256, 52, 52]	512
3.4	ReLU6	[-1, 256, 52, 52]	0
4	**conv4: Depthwise Separable Convolution**		
4.1	Depthwise Conv2d (256, 3x3, stride 2, pad 1, groups=256)	[-1, 256, 26, 26]	2,304
4.2	Pointwise Conv2d (256 in, 512 out, 1x1)	[-1, 512, 26, 26]	131,072
4.3	BatchNorm2d (512 channels)	[-1, 512, 26, 26]	1,024
4.4	ReLU6	[-1, 512, 26, 26]	0
5	**conv5: Depthwise Separable Convolution**		
5.1	Depthwise Conv2d (512, 3x3, stride 2, pad 1, groups=512)	[-1, 512, 13, 13]	4,608
5.2	Pointwise Conv2d (512 in, 1024 out, 1x1)	[-1, 1024, 13, 13]	524,288
5.3	BatchNorm2d (1024 channels)	[-1, 1024, 13, 13]	2,048
5.4	ReLU6	[-1, 1024, 13, 13]	0
6	**avgpool: Adaptive Average Pooling**		
6.1	AdaptiveAvgPool2d (1x1)	[-1, 1024, 1, 1]	0
7	**classifier: Fully Connected Layer**		
7.1	Dropout (p=0.2)	[-1, 1024]	0
7.2	Linear (1024 in, 25 out)	[-1, 25]	25,625
Total			**1,307,417**

Table 1: Model Parameters

4.2 Training Pipeline

The training process includes:

1. Evaluation function for validation
2. Training loop with:
    - Gradient clipping
    - Learning rate scheduler

4.3 Saving the Model

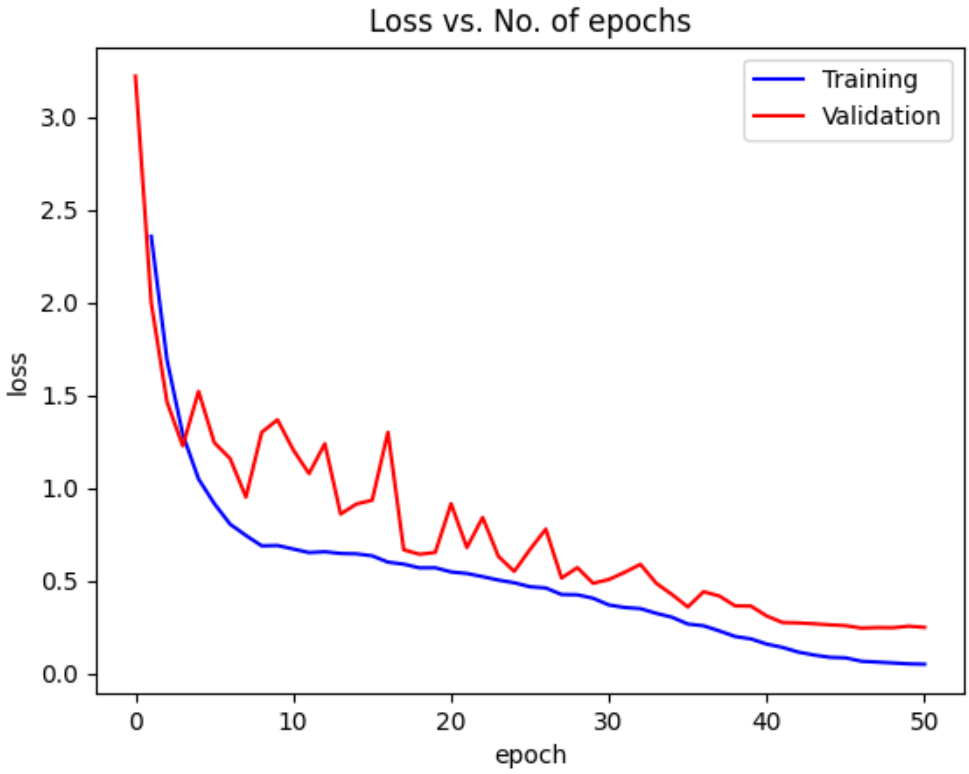
The model is saved in two formats:

1. Model state dictionary
2. Scripted version using `torch.jit.script`

5 Plotting Metrics

- Accuracy vs. Epochs

- Loss vs. Epochs



- Learning Rate