GNR638

Machine Learning for Remote Sensing II

MINI-PROJECT-I

Team Members-

Abhinav Vaishnav

Lakavath Sidhartha

TA Name-

Sanarbh Agarwal

Table of content-

1. Introduction

The primary objective of this project is to develop an efficient CNN model capable of accurately categorizing bird images while adhering to strict constraints on model complexity. To achieve this goal, we adopt a data-driven approach, leveraging a default train-test split for training and evaluation purposes.

As part of the project guidelines, efficientNet architecture is used as external model.

The report will delve into the architecture and training details of the CNN model developed for bird image classification. Key aspects such as model architecture, training methodology, training loss, accuracy curves, and final results will be thoroughly discussed. Additionally, the report will provide insights into the challenges encountered during the development process and strategies employed to overcome them.

2. Methodology

1.1. Data collection

Data was made available to us which can be found on this link-
https://data.caltech.edu/records/65de6-vp158/files/CUB_200_2011.tgz?download=1

1.2. Data Preprocessing

To prepare the dataset for training and evaluation, a series of preprocessing steps were applied to the raw data. The preprocessing tasks primarily involved reading and structuring the data obtained from the provided files. The following steps were executed to preprocess the data:

1. Reading Data Files:

The data is stored in three separate text files: `images.txt`, `train_test_split.txt`, and `image_class_labels.txt`. These files contain information about the images, their corresponding labels, and their assignment to training or testing sets.

2. Reading Text Files:

Using the Python `open()` function, the contents of each text file were read into memory. The paths to the files were specified as `D:/GNR638_assign/CUB_200_2011/images.txt`, `D:/GNR638_assign/CUB_200_2011/train_test_split.txt`, and `D:/GNR638_assign/CUB_200_2011/image_class_labels.txt`.

3. Storing Data:

The contents of the text files were stored in variables for further processing. Specifically, the image paths were stored in the `images_text` variable, the train-test split information was stored in the `train_test` variable, and the image labels were stored in the `labels_text` variable.

4. Converting Text to DataFrame:

Using the `pd.read_csv()` function from the Pandas library, the text data was converted into Pandas DataFrames. Each text file was parsed into a DataFrame with two columns: 'Number' and 'Data'. The 'Number' column represents a unique identifier, while the 'Data' column contains the relevant information (image paths, labels, or train-test split).

5. Data Merging:

The DataFrames obtained from parsing the text files were merged to create a single DataFrame (`df`). This DataFrame consists of three columns: 'images', 'labels', and 'is_train'. The 'images' column contains the paths to the images, the 'labels' column contains the corresponding image labels (indexed from 0), and the 'is_train' column specifies whether the image belongs to the training set (1) or the testing set (0).

6. Partitioning Data:

Finally, the dataset was partitioned into training and testing subsets based on the 'is_train' column. Images designated as part of the training set were stored in the `df_train` DataFrame, while those assigned to the testing set were stored in the `df_test`

DataFrame.

1.3. Data analysis

1. Data Shuffling:

The `df` dataframe is shuffled using the `sample` method to ensure that the data is randomized before splitting into training and testing sets.

2. Dataset and DataLoader Creation:

Custom datasets (`CustomImageDataset`) are created using the training and testing data lists generated from the dataframe. These datasets are then used to create PyTorch `DataLoader` objects for efficient batch-wise data loading during training and testing.

3. Data Augmentation and Normalization:

Although commented out in the provided code, typically data augmentation and normalization techniques are applied to the images using PyTorch's `transforms.Compose` function. This step helps in improving model generalization and performance.

4. Model Architecture Definition:

The model architecture is defined using either a custom-defined neural network architecture or a pre-trained model (in this case- EfficientNet). The model is then modified to suit the specific task (e.g., adjusting the output layer to match the number of classes).

5. Model Training:

The model is trained using the training dataset. During training, batches of data are fed into the model, and the loss is computed based on the model's predictions and the ground truth labels. The gradients are then backpropagated through the network, and the optimizer updates the model parameters to minimize the loss.

6. Monitoring Training Progress:

Throughout the training process, various metrics such as training loss and accuracy are monitored. These metrics are typically logged and visualized to track the model's performance over epochs.

7. Model Evaluation:

After training, the trained model is evaluated on the testing dataset to assess its generalization performance. Accuracy metrics are calculated to determine how well the model performs on unseen data.

8. Visualization:

Loss and accuracy curves are plotted using Matplotlib to visualize the training progress and the model's performance over epochs.

1.4. Model Development

Here we used efficientNet B1 to train the model.

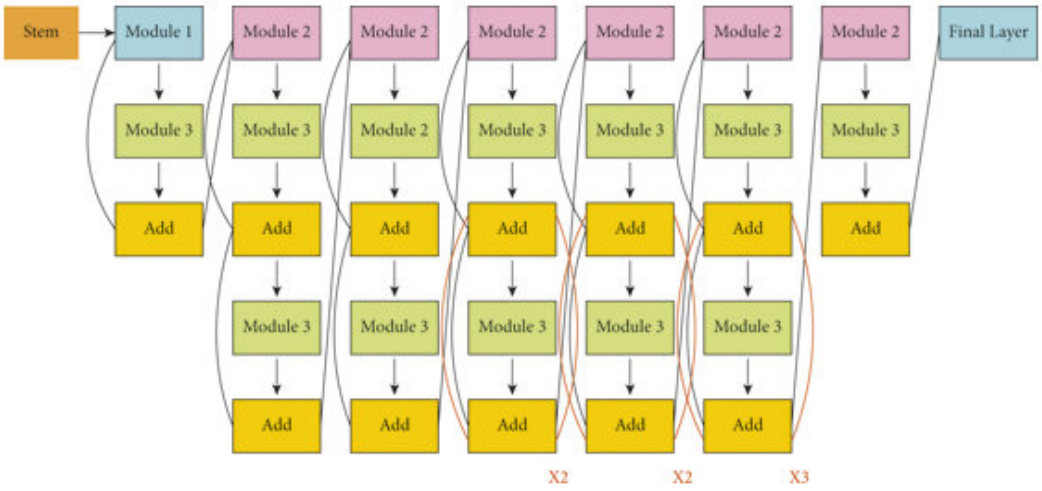
Why EfficientNet B1:

1. Pretrained on ImageNet: EfficientNet B1 was pretrained on the ImageNet dataset, which consists of over a million labeled images across thousands of classes. Pretraining on ImageNet allows the model to learn rich and generalizable features from a diverse range of visual data.

2.It offers competitive performance on various computer vision tasks, including image classification, while being computationally efficient.

3.By leveraging transfer learning, the pre-trained weights of EfficientNet B1 on the ImageNet dataset can provide a good initialization for the model, enabling faster convergence during training.

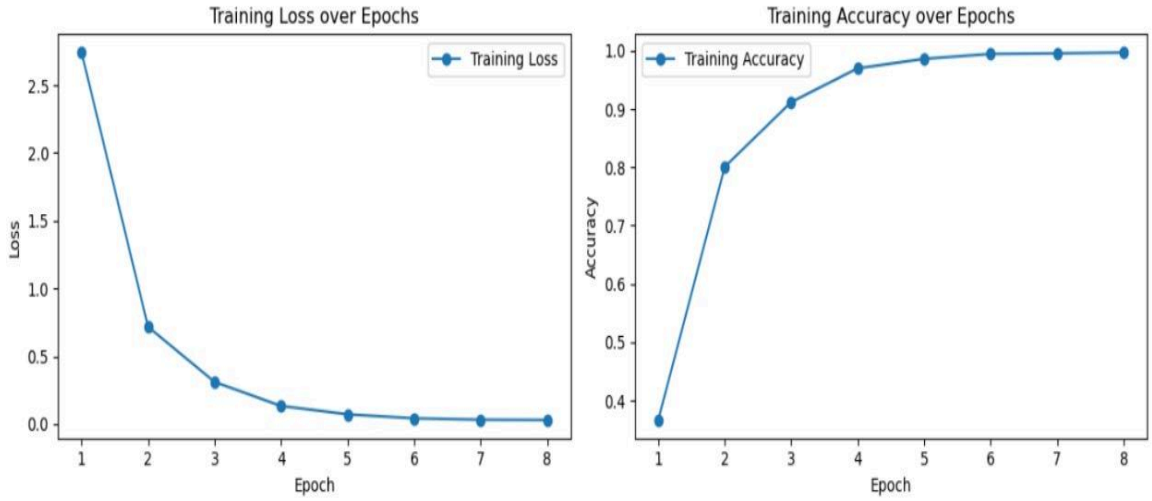
4.Its scalability allows for easy adaptation to different dataset sizes and computational budgets, making it a versatile choice for a wide range of applications.



1.5. Model Evaluation-

Epoch	Loss
0	2.74201
1	0.71807

2	0.31189
3	0.13419
4	0.07291
5	0.04351
6	0.03342
7	0.03113



3. Results-

Training set- Got 5994 / 5994 with accuracy 100.00

Test set- Got 4581 / 5794 with accuracy 79.06

Model Checkpoint-https://drive.google.com/file/d/1Nze_Sv_LgV6u6Uw-FXcVmSFvJqH8HjBn/view?usp=sharing

4. Conclusion-

Generalization: Due to its pre trained nature on ImageNet, the EfficientNet B1 model exhibits strong generalization capabilities. It has learned to recognize a wide variety of objects and patterns, allowing it to perform well on unseen data from the CUB-200-2011 dataset.

5. References-

- https://arxiv.org/pdf/1905.11946
- https://www.researchgate.net/figure/The-model-architecture-of-EfficientNet-B1_fig7_360213068