GNR638 Mini Project 1 Report

Kavya Gupta (22B1053), Atharva Bendale (22B0901), Soham Dahane (22B0941)

March, 2024

1 Introduction

This report presents the implementation and evaluation of a fine-grained classification model using a Convolutional Neural Network (CNN) on the CUB dataset. The objective is to train a CNN model with an upper limit of 10M parameters, focusing on parameter efficiency, training time efficiency, and accuracy.

2 Dataset Description

The CUB dataset consists of images of birds, categorized into 200 classes. The dataset was downloaded from `https://data.caltech.edu/records/65de6-vp158/files/CUB_200_2011.tgz?download=1` and follows the default train-test split for the task.

3 Methodology

The implemented model adopts a transfer learning approach, leveraging the **MobileNetV2 architecture pretrained on the ImageNet dataset**. MobileNetV2 is chosen due to its lightweight design, making it suitable for the task of fine-grained classification with an emphasis on **parameter efficiency**.

To tailor the model to the specific characteristics of the CUB dataset, additional layers are appended to the MobileNetV2 architecture. The custom layers consist of a fully connected layer with **512 units and ReLU activation**, followed by another fully connected layer with the number of output classes (**200 in this case**). This modification allows the model to adapt and specialize in recognizing fine-grained details of bird images.

During training, two key regularization techniques are employed. First, **early stopping** is implemented to monitor the validation loss. If the loss does not decrease for three consecutive epochs, training is halted to prevent overfitting. Second, **L2 regularization** is applied to the model's weights to prevent excessive complexity.

The loss function used for optimization is the **CrossEntropyLoss**, suitable for multi-class classification tasks. **Stochastic Gradient Descent (SGD)** with positive momentum is chosen as the optimizer to update the model's parameters. This combination aims to strike a balance between accurate classification and model generalization.

4 Architecture Details

The **MobileNetV2 model** serves as the backbone for feature extraction. It has **3.4M parameters**. This architecture comprises **depthwise separable convolutions**, making it computationally efficient while preserving representational power.

```
# Define your custom layers
self.custom_layers = nn.Sequential(
    nn.Linear(num_features, 512),
    nn.ReLU(inplace=True),
    nn.Linear(512, self.num_classes),
)

# Concatenate the MobileNetV2 model with custom layers
self.model.classifier = nn.Sequential(*list(self.model.classifier.children()) +
list(self.custom_layers.children()))
self.model = self.model.to(self.device)
```

The custom layers, added at the end of MobileNetV2, act as a classifier for fine-grained classification. The final layer outputs probabilities for each of the **200 bird classes**. The model is trained end-to-end, updating both the pretrained MobileNetV2 weights and the appended custom layers.

The training process involves **data augmentation and normalization**. **Random resized cropping and horizontal flipping** are applied during training to augment the dataset and improve model robustness. Normalization is performed with **standard ImageNet mean and standard deviation values**.

5 Experimental Setup

The model was trained with the following hyperparameters:

- Learning Rate: 0.001
- Batch Size: 32
- Epochs: 20
- Momentum: 0.9
- Weight Decay: 0.0001
- Image Size: 224

The training was conducted on GPU with ID 0.
The model's training involves data augmentation and normalization. The transformations applied during training and testing are defined as follows:

```
self.train_transform = transforms.Compose([
    transforms.RandomResizedCrop(self.image_size),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

self.test_transform = transforms.Compose([
    transforms.Resize(int(self.image_size/0.875)),
    transforms.CenterCrop(self.image_size),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

These transformations include random resized cropping, random horizontal flipping during training, resizing, and center cropping during testing. Additionally, the images are converted to tensors, and normalization is performed using specified mean and standard deviation values.

6 Results

Figure 1 illustrates the training and validation loss curves, while Figure 2 shows the corresponding accuracy curves. The training was monitored for early stopping, and the final model achieved an accuracy of **73.07%** on the test set.

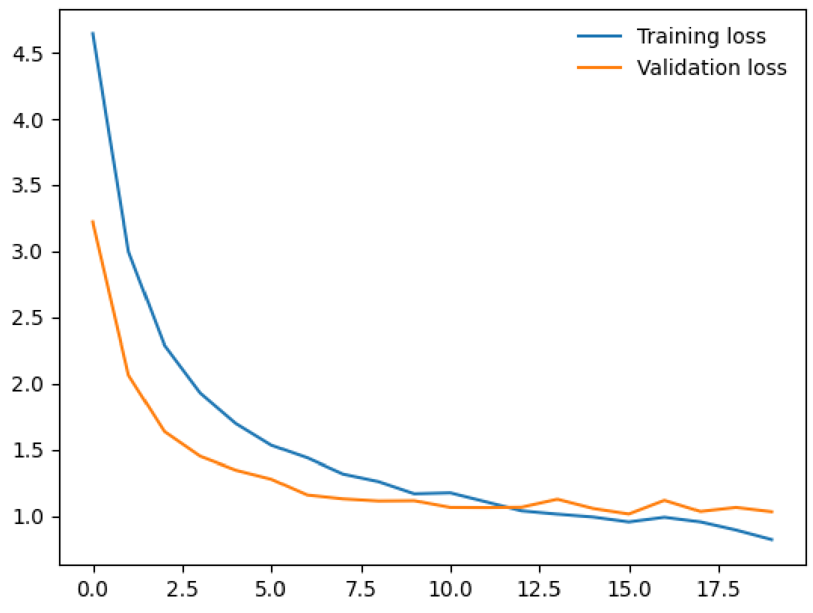

Figure 1: Training and Validation Loss Curves

7 Model Evaluation

The model was evaluated based on accuracy, model complexity (number of parameters), and training time efficiency. The trade-off between accuracy and model complexity was

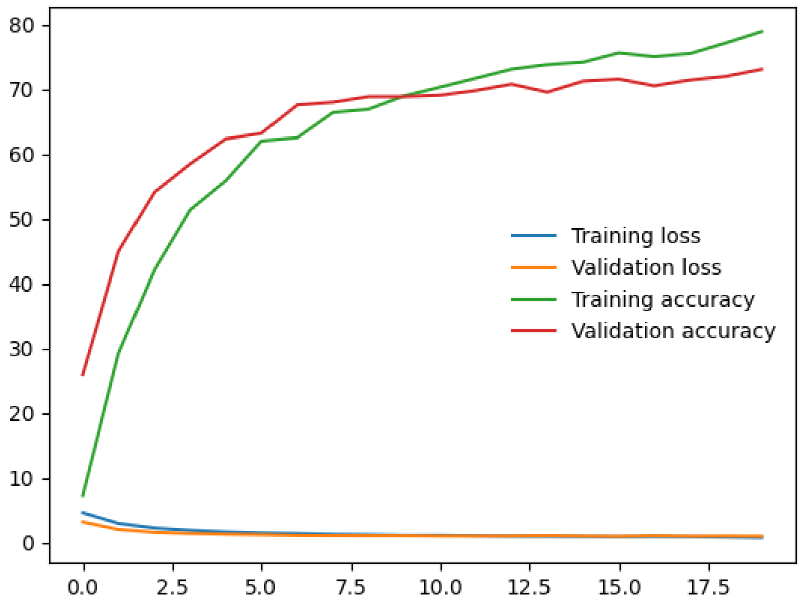

Figure 2: Training and Validation Accuracy Curves

observed, and the final accuracy was achieved with **3.4M parameters**, within the specified 10M parameter limit.

8 Code and Implementation Details

The Python file `cubmodel.py` includes the implementation details of the model. To train the model, run the command:

```
python3 cubmodel.py --tarfile_address /path/to/dataset.tgz --lr 0.001 --batch_size
32 --epochs 20 --momentum 0.9 --weight_decay 0.0001 --image_size 224 --gpu_id 0
```

For testing the trained model, import the `CUBModel` class from `cubmodel.py`, load the model, and use the `predict_from_image_path` or `predict_from_image_array` functions.

9 Conclusion

In conclusion, the implemented fine-grained classification model demonstrated effective training with a balance between accuracy and model complexity. Further optimizations and experiments can be explored for potential improvements.

10 Appendix

The log file (`training.log`), code (`cubmodel.py`) are submitted on Moodle and the final model checkpoint (`final_model_checkpoint.pth`) is available in the provided drive link: `https://drive.google.com/file/d/1W-IQJNeQ3-aaFnb8JOSr8g5ip-a4PP5X/view?usp=sharing`.