Selected Topics in Visual Recognition using Deep Learning HW1 Report

- **Name:** 馬楷翔
- **Student ID:** 110550074
- **Github Repo:** https://github.com/seanmamasde/Selected-Topics-in-Visual-Recognition-using-Deep-Learning

Introduction

This report describes the approach that I used to fine-tuning deep convolutional neural networks on a dataset composed of animals and plants. The primary goal here is to improve prediction performance by leveraging transfer learning and ensemble methods. I started with a baseline using RegNet (pre-trained on ImageNet) and then focus on using SEResNeXt101_64x4d (pre-trained on iNaturalist) since its domain is more similar to our target dataset. In addition, we investigate the benefits of bagging with majority voting as an ensemble technique. The experiments, conducted on an A100 40GB GPU, reveal that while a single RegNet model achieved 91% accuracy, the bagging approach with SEResNeXt reached 94% accuracy.

Method

Data Preprocessing

The dataset is organized as follows:

- **train:** 100 folders (classes labeled from 0 to 99) containing training images.
- **val:** 100 folders with validation images.
- **test:** A single folder with test images.

Image pre-processing includes:

- **Training transforms:** Resize to 256, random crop to 224, random horizontal flip, conversion to tensor, and normalization using mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].
- **Validation/Test transforms:** Resize to 256, center crop to 224, conversion to tensor, and identical normalization.

Model Architecture and Hyperparameters

1. **RegNet Baseline:**
    - **Architecture:** `regnety_160` from the timm library (pre-trained on ImageNet).
    - **Hyperparameters:** Batch size = 32, learning rate = 1e-4, 10 epochs.

- **Performance:** Approximately 91% accuracy on the test set.

2. **SEResNeXt with Bagging:**
   - **Architecture:** `seresnext101_64x4d` from timm (pre-trained on iNaturalist).
   - **Bagging Details:**
     - **Ensemble Size:** 10 models.
     - **Training:** Each model is trained on a bootstrapped sample of the training data.
     - **Hyperparameters:** Batch size = 80, learning rate = 1e-4, 20 epochs per model.
     - **Inference:** Majority voting is applied across ensemble predictions.
   - **Performance:** Achieved a test accuracy of 94%.

Model Training and Saving

To prevent loss of progress in case of hardware interruptions, I implemented a mechanism to save models:

- **Validation Check:** After every epoch, the model is evaluated on the validation set.
- **Saving Criteria:** If the current epoch's validation accuracy is higher than the best recorded, the model state is saved as `./saved_models/model{model_index}_{epoch}.pth`.

A representative code snippet is shown below:

```
best_acc = 0.0
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    for images, labels in bagged_loader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    avg_loss = running_loss / len(bagged_loader)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}")

    # Validation evaluation
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = model(images)
            _, preds = torch.max(outputs, 1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
    val_acc = correct / total
    print(f"Validation Accuracy: {val_acc:.4f}")

    # Save model if improved
```

```
    if val_acc > best_acc:
        best_acc = val_acc
        model_save_path = f"./saved_models/model{i+1}_{epoch+1}.pth"
        torch.save(model.state_dict(), model_save_path)
        print(f"Model improved. Saved model to {model_save_path}")
```

Inference and Prediction Output

For inference, a custom test dataset is defined that also returns image filenames. Predictions are written to a CSV file (`prediction.csv`) in the format `image_name,pred_label` , which is then compressed into `pred.zip` for submission afterwards. Existing prediction files are removed before saving new outputs.

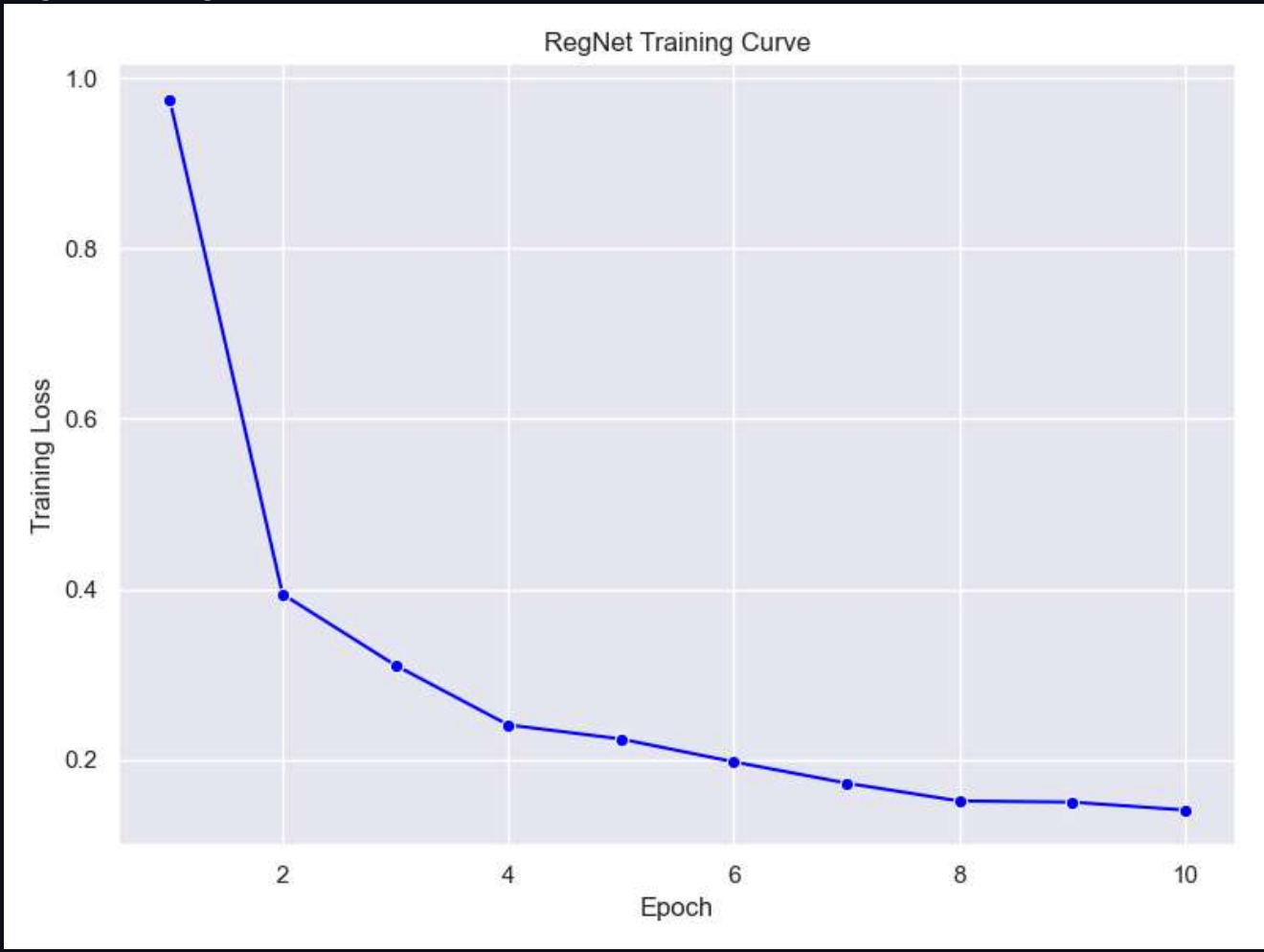
Results

Performance Summary

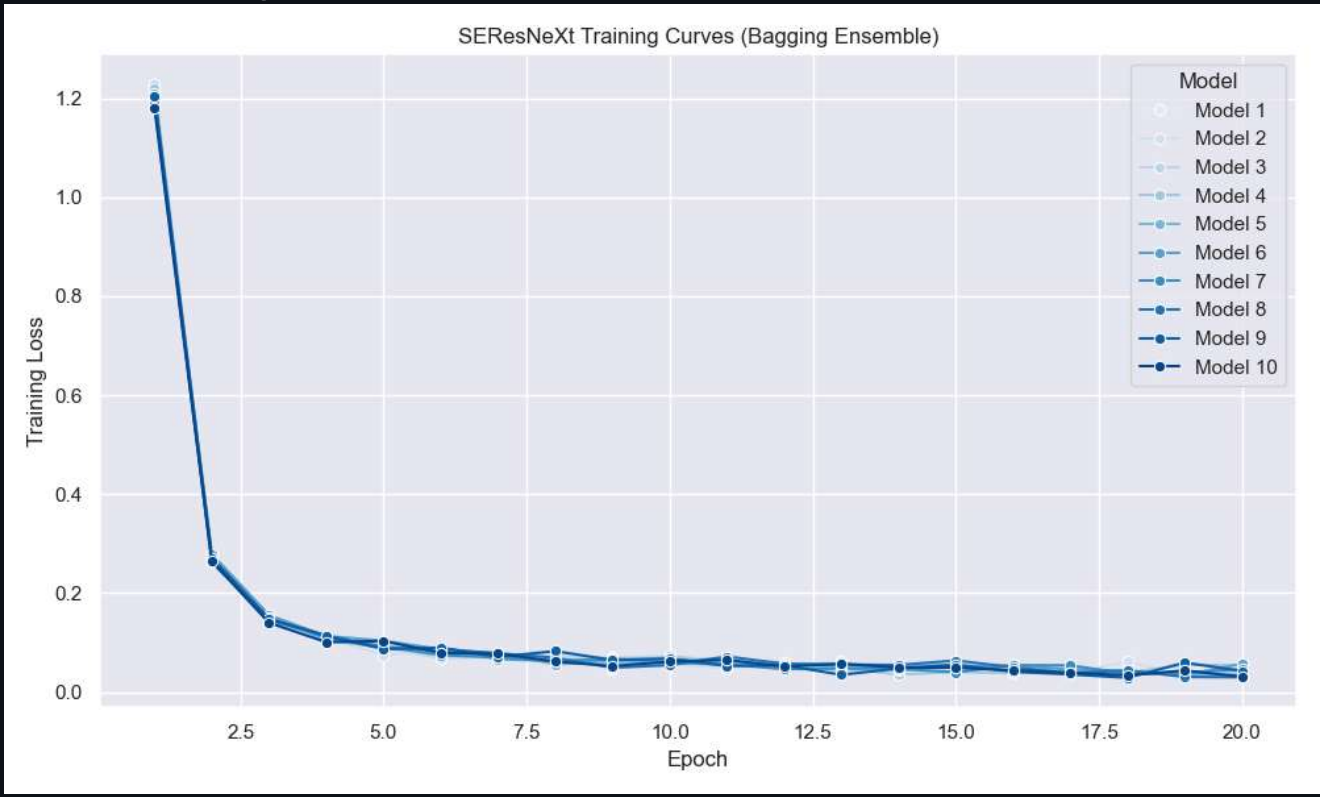
- **RegNet (Baseline):**
  - **Test Accuracy:** 91%
  - **Training Time:** Approximately 40 minutes on an A100 40GB GPU.
- **SEResNeXt with Bagging:**
  - **Test Accuracy:** 94%
  - **Training Time:** Approximately 6 hours.
  - **Observation:** Despite similar parameter sizes (~80M each), the domain alignment (iNaturalist vs. ImageNet) and the ensemble effect from bagging provided significant accuracy gains.

Training Curves

- RegNet training curve



- SEResNeXt training curve



- I feel like the seresnext one got converged faster than the regnet one (comparing the first 10 epochs trained).

Additional Experiment: Bagging and Majority Voting

Hypothesis

We hypothesized that bagging—training multiple models on bootstrapped samples—could reduce model variance and improve overall performance via majority voting. Since SEResNeXt is pre-trained on a domain more similar to our dataset, the expectation was that an ensemble of these models would be more robust, yielding better generalization on the test set.

Methodology

- **Ensemble Creation:**
  Ten SEResNeXt models were independently trained on different bootstrapped subsets of the training data.
- **Model Aggregation:**
  During inference, each model produced a prediction for a given image. The final label was determined using majority voting across all ensemble predictions.
- **Validation and Saving:**
  Each model was monitored for validation performance, and improved checkpoints were saved to mitigate potential training interruptions.

Results and Implications

- **Accuracy Improvement:**
  The bagging approach improved the test accuracy from around 92% (when using RegNet with bagging) to 94% with SEResNeXt, validating our hypothesis.
- **Computational Trade-off:**
  While bagging provided a notable performance boost, it also increased the training time significantly (from 40 minutes without bagging to 6 hours with bagging when training RegNet, and a whopping 8 hours when training SEResNeXt).
- **Implication for Future Work:** Ensemble methods like bagging can be particularly beneficial when the pre-training domain of the model aligns closely with the target dataset. Further experiments could explore optimizing the number of ensemble members or integrating more diverse architectures.

References

- **timm Library:**
  rwightman/pytorch-image-models
- **RegNet Architecture:**
  RegNet: Design Space for High-Performance Image Classification
- **SEResNeXt Architecture:**
  Squeeze-and-Excitation Networks
- **Bagging in Machine Learning:**
  Bootstrap Aggregating (Bagging)

- **iNaturalist Dataset:**
  iNaturalist 2018