Exploring the Role of Optimizers and Architectures in Active Learning
Progress Report

Khadiza Sarwar Moury, Md Hasibul Hasan Shovo, Sayed Abdullah Ali, Youssef Midra
Concordia University, Canada
Email: khadizasarwar.moury, mdhasibulhasan.shovo, syedabdullah.ali, youssef.midra@mail.concordia.ca
Roll no: 40343620, 40329539, 40333397, 40341729

I. Introduction

Deep learning models have achieved remarkable success in image classification tasks, but this success often comes at the cost of requiring large volumes of labeled data. In real-world scenarios, labeling such data can be expensive, time-consuming, and in some domains—such as medical imaging or satellite imagery—requires expert annotators.

Active Learning (AL) offers a compelling solution to this problem by selecting the most informative or uncertain samples from a large pool of unlabeled data for annotation. Instead of randomly choosing data points, AL prioritizes instances that are expected to improve model performance the most when labeled. A key AL strategy is *uncertainty sampling* [1], where the model queries data points it is least confident about, such as those with ambiguous class probabilities or high entropy. This approach helps the model refine its decision boundaries, especially during the early stages of training.

In this research, we aim to explore how different choices of optimizers (SGD, Adam, AdamW, RMSProp) and model architectures (ResNet18, ResNet34, ResNet50) [5] affect the performance and efficiency of an active learning pipeline using uncertainty sampling. Using the CIFAR-10 dataset, we simulate a real-world budget-constrained labeling scenario by incrementally increasing the labeled dataset over 10 active learning cycles. Through systematic experimentation, we intend to uncover insights into how these fundamental design decisions influence not just model accuracy but also sample efficiency and computational cost—key considerations for deploying image classification systems in practical, resource-constrained environments.

II. Research Objectives

- To assess the effectiveness of various optimizers (SGD, Adam, AdamW, RMSProp) in the context of active learning using uncertainty sampling.
- To evaluate how different ResNet architectures (ResNet18, ResNet34, ResNet50) impact the performance of active learning on the CIFAR-10 dataset.
- To compare the overall performance (accuracy, sample efficiency, and computational cost) of these models and optimizers under a fixed labeling budget.

III. Methodology

This research adopts an iterative active learning framework based on uncertainty sampling, designed to evaluate how different model architectures and optimizers impact learning efficiency and performance. The study uses the benchmark image classification dataset CIFAR-10 [6].

For model architectures, three variants of ResNet—ResNet18, ResNet34, and ResNet50—are trained using four optimizers: SGD [3], Adam [2], AdamW [4], and RMSProp [3].

In each experiment, the active learning process begins with no initially labeled data. In the first cycle, 1,000 samples are selected and labeled based on uncertainty sampling. In each subsequent cycle, the model selects an additional 1,000 most uncertain samples from the unlabeled pool and adds them to the labeled training

set. This continues over 10 cycles, resulting in a total labeling budget of 10,000 samples per run.

After each cycle, the model is trained using the updated labeled set to ensure consistent evaluation. Model performance is evaluated after every cycle using several key metrics: accuracy on a held-out test set, sample efficiency, computational efficiency, and the F1-score. This methodology is applied systematically across all model–optimizer combinations, allowing a detailed comparison of how these factors influence the effectiveness of active learning.

IV. Experiment Setup

- **Computing Resource:** Experiments are conducted using Concordia's High-Performance Computing (HPC) Facility, Speed.
- **Model Architectures:** ResNet18, ResNet34, and ResNet50 represent different network depths, balancing complexity and computational cost.
- **Optimizers:**
  - **SGD:** Standard optimization technique, often used with momentum.
  - **Adam:** Adaptive learning rate optimizer using first and second moment estimates.
  - **AdamW:** A variant of Adam that decouples weight decay from the optimization step.
  - **RMSProp:** Adjusts learning rate per parameter based on recent gradient magnitudes.

V. Task Distribution

- Md Hasibul Hasan Shovo worked on the implementation of active learning.
- Khadiza Sarwar Moury is working on adding optimizers and generating results for different model-optimizer pairs.
- Youssef Midra prepared the progress report.
- Sayed Abdullah Ali working on preparing the final report.

VI. Performance Evaluation

The performance of each model–optimizer combination is assessed after every cycle using these metrics:

- **Accuracy:** Tracks model's ability to classify correctly across cycles.
- **Sample Efficiency:** Measures performance relative to the number of labeled samples.
- **Computational Efficiency:** Evaluates training time and memory consumption.
- **F1-Score:** Captures balance between precision and recall.
- **Label Efficiency:** Accuracy gain per percentage of labels used.
- **Accuracy vs Budget:** Accuracy improvement over random selection at a given labeling budget.

VII. Results and Progress

At the current stage, the following milestones have been achieved:

- CIFAR-10 dataset preprocessing and augmentation (random cropping, horizontal flipping) completed.

```
def get_dataset(args):
    if args.dataset in ('cifar10', 'cifar100'):
        dataset = cifar.CIFARDataset(args)
        dataset = dataset.dataset
        args.nTrain, args.nClass = len(dataset['train']), 10
    return dataset, args
```

- Models initialized and trained for the first cycle using 1,000 uncertainty-sampled labels.

```
def train_epoch(models, criterion, optimizers, schedulers, dataloaders,
                epoch, epoch_loss):
    models['backbone'].train()
    models['module'].train()
    correct, total = 0, 0
    for inputs, labels in
    tqdm(dataloaders['train'],leave=False,total=len(dataloaders['train'])):
        inputs, labels = inputs.to(args.device), labels.to(args.device)
        optimizers['backbone'].zero_grad()
        optimizers['module'].zero_grad()
        scores, features = models['backbone'](inputs)
        target_loss = criterion(scores, labels)
        _, preds = torch.max(scores.data, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
        if epoch > epoch_loss:
            features[0] = features[0].detach()
            features[1] = features[1].detach()
            features[2] = features[2].detach()
            features[3] = features[3].detach()
        pred_loss = models['module'](features)
        pred_loss = pred_loss.view(pred_loss.size(0))
        m_backbone_loss = torch.sum(target_loss) / target_loss.size(0)
        m_module_loss   =
        lossnet.LossPredLoss(pred_loss, target_loss, margin=args.margin)
        loss = m_backbone_loss + args.weight * m_module_loss
        loss.backward()
        optimizers['backbone'].step()
        optimizers['module'].step()
    schedulers['backbone'].step()
    schedulers['module'].step()
    return 100 * correct / total
```

- Experiments completed for ResNet18 and ResNet34 using the SGD optimizer.

```
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
    super(ResNet, self).__init__()
    self.in_planes = 64
    self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=False)
    self.bn1 = nn.BatchNorm2d(64)
    self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
    self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
    self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
    self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
    self.linear = nn.Linear(512*block.expansion, num_classes)
    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out1 = self.layer1(out)
        out2 = self.layer2(out1)
        out3 = self.layer3(out2)
        out4 = self.layer4(out3)
        out = F.avg_pool2d(out4, 4)
```

```
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out, [out1, out2, out3, out4]
def ResNet18(num_classes = 10):
    return ResNet(BasicBlock, [2,2,2,2], num_classes)
def ResNet34(num_classes = 10):
    return ResNet(BasicBlock, [3,4,6,3], num_classes)
def ResNet50(num_classes = 10):
    return ResNet(Bottleneck, [3,4,6,3], num_classes)
def ResNet101(num_classes = 10):
    return ResNet(Bottleneck, [3,4,23,3], num_classes)
def ResNet152(num_classes = 10):
    return ResNet(Bottleneck, [3,8,36,3], num_classes)
```

- Accuracy across all 10 classes recorded for these configurations.

```
def get_uncertainty(models, unlabeled_loader):
    models['backbone'].eval()
    models['module'].eval()
    uncertainty = torch.tensor([]).to(args.device)
    with torch.no_grad():
        for (inputs, labels) in unlabeled_loader:
            inputs = inputs.to(args.device)
            scores, features = models['backbone'](inputs)
            pred_loss = models['module'](features)
            pred_loss = pred_loss.view(pred_loss.size(0))
            uncertainty = torch.cat((uncertainty, pred_loss), 0)
    return uncertainty.cpu()
```

- **Results**: Fig. 1 shows the accuracy of active learning with ResNet-18 and Resnet-34 with SGD optimizer. The accuracy of both ResNet-18 and ResNet-34 both increases when the number of labeled data increases in training. ResNet-34 performed slightly better than ResNet-18 as ResNet-34 has a deeper architecture. Fig. 2 shows the training snippet of ResNet-50 model with SGD on the Concordia's High-Performance Computing Facility, Speed.

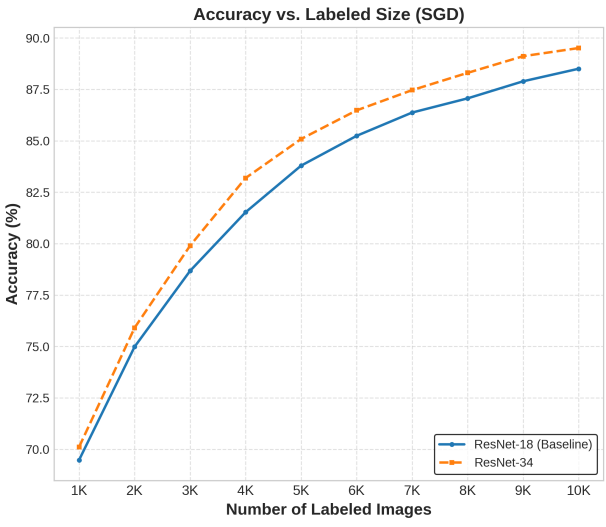

Fig. 1. Difference of accuracy between ResNet-18 and ResNet-34 with SGD optimizer.

VIII. Next Steps

The next phase will focus on completing all active learning cycles and model–optimizer combinations:

```
Results for (Optimizer: sgd, Model: resnet50) saved to results/cifar10/results.json

--- Running ResNet50 (Optimizer: sgd, Model: resnet50) Approach (Trial 4/5) ---
Training Set 1000
>> Train a Model.
>> Finished.
Trial 4/5 || Cycle 1/10 || Labelled Data 1000: Test Accuracy 75.50%
Training Set 2000
>> Train a Model.
>> Finished.
Trial 4/5 || Cycle 2/10 || Labelled Data 2000: Test Accuracy 80.22%
Training Set 3000
>> Train a Model.
>> Finished.
Trial 4/5 || Cycle 3/10 || Labelled Data 3000: Test Accuracy 82.57%
Training Set 4000
>> Train a Model.
>> Finished.
Trial 4/5 || Cycle 4/10 || Labelled Data 4000: Test Accuracy 83.98%
Training Set 5000
>> Train a Model.
>> Finished.
Trial 4/5 || Cycle 5/10 || Labelled Data 5000: Test Accuracy 85.09%
```

Fig. 2. Training snippet of ResNet-50 with SGD optimizer.

- Integrate and test the remaining optimizers: Adam, AdamW, and RMSProp.
- Run all 10 active learning cycles for each model–optimizer pair.
- Conduct comparative analysis of trade-offs among accuracy, sample efficiency, computational efficiency, and F1-score.
- Compile a comprehensive final report summarizing findings and recommendations.

IX. Conclusion

This research is progressing as planned, with initial results offering valuable insights into the role of model architectures in active learning. The findings will contribute to a deeper understanding of how to design more efficient active learning systems that achieve higher performance with fewer labeled samples.

References

[1] Y. Yang and M. Loog, "Active learning using uncertainty information," *2016 23rd International Conference on Pattern Recognition (ICPR)*, IEEE, 2016, pp. 2646–2651. doi: 10.1109/ICPR.2016.7900034.

[2] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2017.

[4] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.

[6] A. Krizhevsky, "Learning multiple layers of features from tiny images," Technical Report, University of Toronto, 2009.