ACM MLab: Bringing Down BARCO Using ResNets

Luke Bousfield, Athena Hernandez, and Tony Wang

December 1, 2023

1 Architecture

Initially, we experimented with many different model architectures including simple linear networks, convolutional neural networks (CNNs), and ultimately residual networks (ResNets). Our initial ResNet model, characterized by its residual blocks, performed with the highest accuracy. Further refinement and structural modifications were made in order to tailor the ResNet for specific tasks and improve overall accuracy.

2 Training

The training process of our model involved utilizing backpropagation, a technique in which the weights of the neural network are updated iteratively based on the computed gradients of the loss function with respect to the model parameters.

Additionally, we employed the cross-entropy loss function alongside an Adam optimizer, which uses both momentum and adaptive learning rate methods helping convergence during training.

```
model = ResNet(ResidualBlock, [3, 4, 6, 3])
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-5)

model.to(device=device)
```

After using geometric hyperparameter testing, we found an optimal configuration. We systematically varied batch size, learning rate scheduling, weight decay, CNN stride, and CNN padding.

We found that a learning rate of 0.01, subjected to a 33% decay at the end of each epoch was the most effective choice. When deciding on a learning rate and its decay, we tried to find a balance between swift convergence while avoiding divergence and oscillations and without compromising accuracy. Through experimentation, we also found that a batch size of 64 and a weight decay of 0.00001 converged the fastest to a high-performing accuracy.

```
batch_size = 64
learning_rate = 0.01
```

```
num_epochs = 40
```

3 Results

To mitigate overfitting, data augmentation techniques such as random horizontal flips and image crops were employed. These measures were successful and effectively reduced the gap between training and testing accuracy by approximately 10%.

```
from torchvision.transforms import v2
aug = v2.Compose([
    v2.RandomHorizontalFlip(),
    v2.RandomResizedCrop(size=(224, 224), antialias=True)
])
```

We also experimented with various dropout layers, of about 20%, but this decreased the overall accuracy of the model too much while also not significantly reducing overfitting. Thus, we decided not to utilize dropout layers for the final model.

```
self.dropout = nn.Dropout(0.2)
```

Additionally, we randomly assigned 2500 of the images of the training set and 613 to the test set to ensure that we train on as much possible data while still having a sizeable sample to test. This was about a 80% to 20% split between training and testing, respectively.

```
from torch.utils.data import random_split, DataLoader
train_len = 2500
val_len = 613
train_set, val_set = random_split(all_data, [train_len, val_len])
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=True)
```

We were stuck at around 80% accuracy, so we took a look at what images our model was failing to recognize. It appeared that less colorful birds are more prone to false detection because our model heavily relies on bright colors to make its determinations.

Overall, our work demonstrates the significance of tailored architectural modifications, training strategies, and data preprocessing in enhancing ResNet performance for image classification tasks. The model's reliance on color intensity for classification presents an avenue for future improvements to generalize its recognition capabilities.