Computer Vision PSET 3 Write up

Bayard Walsh

February 2024

1 configs

Training was done through Google Collab on a T4 GPU on torch 2.1.0. Note for the final iteration of the final model I trained on an A1000 GPU, as my baseline was performing well enough that it indicated increasing the size of the model and power of the GPU would lead to the required performance bound.

2 imageclassification.ipynb

2.1 MNIST

I began with the MNIST data set classification because of the images were gray scale, and with a more simple data set I believe I could reach the 70% classification threshold quickly. Data loading was straightforward, I decided to use a batch size of 16, and an 80/20 split for training/testing. Because both MNIST and CIFAR10 were classification tasks, the training and testing functionality was the same. For training I followed the recommendations of training the model, moving to the GPU, getting loss (I used a cross-entropy loss calculation), zeroing the gradients, computing a backward pass and then stepping forward. While my iteration loop is standard, I decided to record time and maintain a list of scores and time for each epoch while I was training to generate graphs of model performance. I also decided to print out the results every 25 iterations in training, to better illuminate the model performance as it trains and to fit the larger batch size of CIFAR10. During this process my training function returns a list of tuples, however I removed this functionality for the final version of the code for the test script, and returned nothing instead.

For testing I moved the x,y values in the loader to the GPU, predicted the output, and then counted the number of correctly classified samples, which I then divided by overall samples to get the accuracy.

For the MINST data set I built a small model with 4 convolution layers of the following size: $64, 128, 256, 512$ with standard 3x3 sized filters of increasing

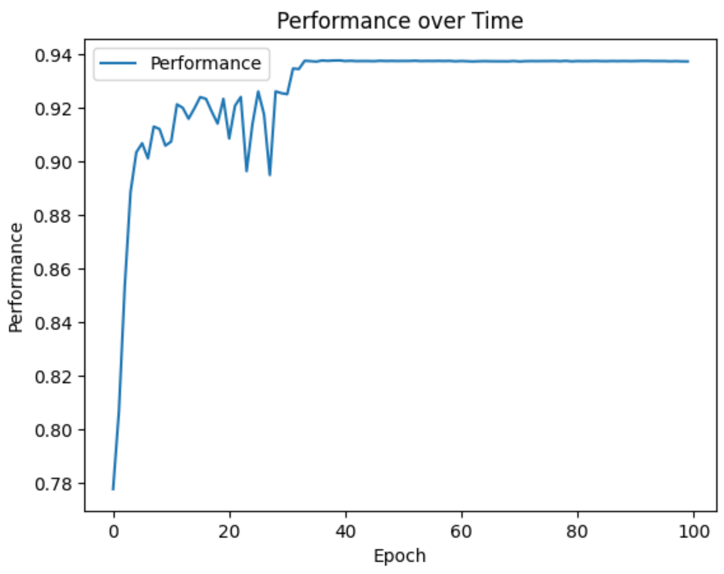

Figure 1: MINST, RUNTIME FOR 100 EPOCHS: 32 MIN. BEST SCORE 93.7625 reached at epoch 41. Final score at 93.725

size. Each convolution layer is followed by a ReLU activation function to introduce non-linearity. Next I added max pooling layers (pool) with a 2x2 kernel between each convolution layer, to help reduce computational complexity and control over fitting. Finally, I fully connected linear layers to reduce the dimensions of the model to 10 for the purpose of classification, which worked for both MNIST and CIFAR10.

Given more time and compute resources, I believe I could achieve similar results (above the 70% threshold) with a slightly larger batch size/ reduced model size to speed up training runtime. This is because the model converged quickly and showed little improvement at further epochs, so some compute resources could have been saved with smaller parameters. Conversely, an overall larger model (such as adding a Resblock) would likely lead to better performance with increased runtime. The model also showed some volatility within the 20-40 epoch range, so reducing the learning rate by a bit would lead to more stable convergence behavior.

2.2 CIFAR10

For this data set, my model differed the most in terms of having to expand out the size of the model to reach the same accuracy threshold and to adopt to the RGB factor for CIFAR10 images. I was able to use the same approach in terms of data loading, training and testing functionality described above, however the

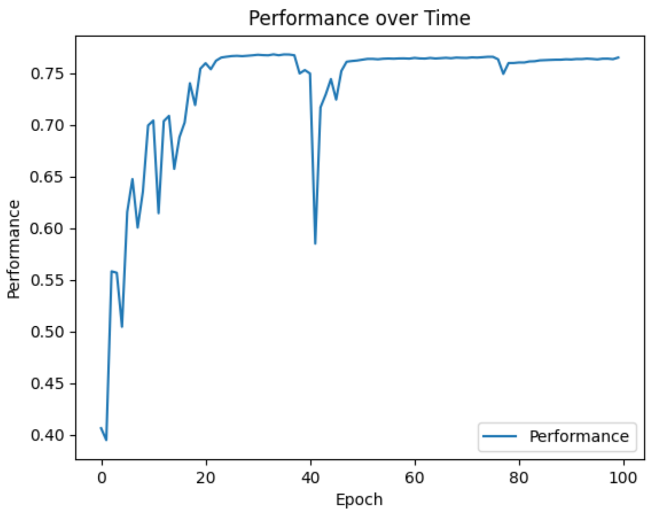

Figure 2: CIFAR10, RUNTIME FOR 100 EPOCHS:20 MIN. BEST SCORE 76.83 reached at epoch 35. Final Score at 76.5225

model architecture was expanded because the data set was harder, and required more computation to reach the bound of 70% accuracy. In terms of loading, I started with a 2d convolution layer of input size 3, rather than 1, because CIFAR10 has RGB colors instead of gray scale for MNIST. Similar to the MNIST model, I used 4 convolution layers, however in this case I added ResBlock layers between each convolution frame as recommended in the write up. Note that I based my model architecture on the one described in the *Deep Residual Learning for Image Recognition* paper, specifically the approach of expanding the model up to size 512. Unlike the model described in the paper I applied the residual networks between every other convolution layer, which worked fine in terms of reaching the training bound, though this approach does mean that the residual layers function more like linear layers - the paper mentions that residual layers had increased performance with 2-3 convolution layers in between. Therefore to increase the model performance in the future, I could add more convolution layers to take better advantage of the ResBlocks.

The ResBlock architecture attempts to reduce the vanishing gradient problem and consists of two convolution layers with batch normalization and ReLU activation functions. The block also includes a downsample path, triggered when the input size or the number of channels changes, achieved through a 1x1 convolution and batch normalization. The final output is obtained by adding the downsampled input (identity) to the processed output, followed by another ReLU activation. This design allows gradient modification during backpropagation while preserving important features in the skip connection. After the

ResBlock layers I followed the same approach as used in the MNIST model in terms of max pooling and using 3 fully connected linear layers to reduce the model to 10 parameters for classification of the 10 options for the photos. I also increased the batch size to 128 to increase speed of training, which explains why the model performs faster than the MNIST model with its expanded size.

3 imagesegmentation.ipynb

For the image segmentation model, the data loading and training aspects were the same from the previous 3 models. Also, I copied the functions from the utils.py file directly into the imagesegmentation file and loaded the data directly into collab, changing the file paths for the collab environment. Otherwise the utils were the same. In terms of testing, the implementation differs from the previous model because of the size of the vectors, as each y vector has dimensions 64x32x32 (for a batch size of 64, the ground truth label for each pixel in the 32x32 image). Because the testing shape wasn't a single digit I adopted the testing file so the model checked each digit in the 32x32 sized image.

For the first iteration of model training, I modified the best performing version of the CIFAR10 model and adopted it to the pixel classification task. These models were slightly different because of the shape of the data and I added some batch normalization blocks between convolution layers. The model eventually reached a final score of 59.11447265625% accuracy after 100 epochs, so I decided to increase the sophistication of the model as the performance was promising but below the 70% threshhold.

In the second model, I added an additional residual block and two convolution layers after the third residual block. The goal of adding size to the model was to increase the model's capacity. This increased size slightly improved performance at the cost of more runtime, however I cut off training early because the performance plateaued and wasn't approaching 70% and I wanted to save compute units. This version of the model eventually reached a final score of 65.160849609375% accuracy after 50 epochs

For the third iteration of the model, I once again added more size to the overall model. In my second iteration, the number of output channels increased from 64 to 512 before the final convolution layer with 20 output channels (for the 20 possible outcomes). However, in the third iteration, I added two additional convolution layers with batch normalization and an extra residual block, resulting in a more complex architecture. This increased complexity led to a slower training process, however as I wanted to hit the baseline 70% accuracy, I continued with the approach of trading off training time for increased size of the model. The model eventually reached a final score of 67.664384765625% accuracy after 34 epochs, and I decided to increase the sophistication of the model once again as the model plateaued slightly below the thresh hold goal.

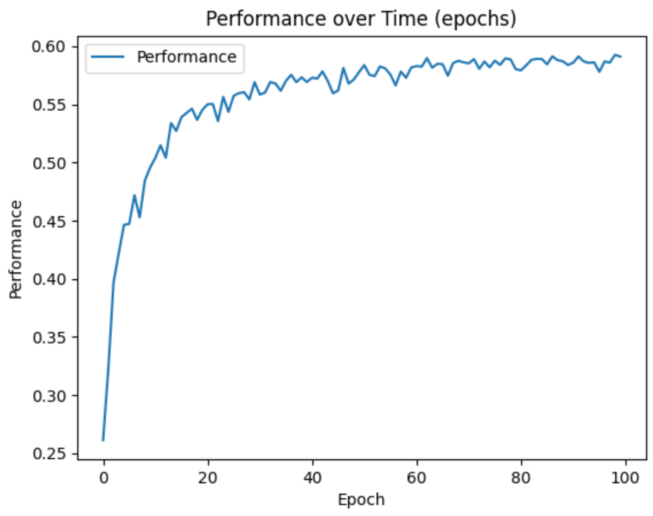

Figure 3: first iteration: segment dataset, 100 epochs: 151.514306 MIN. BEST SCORE 59.267421875 reached at epoch 99. Final score at 59.11447265625

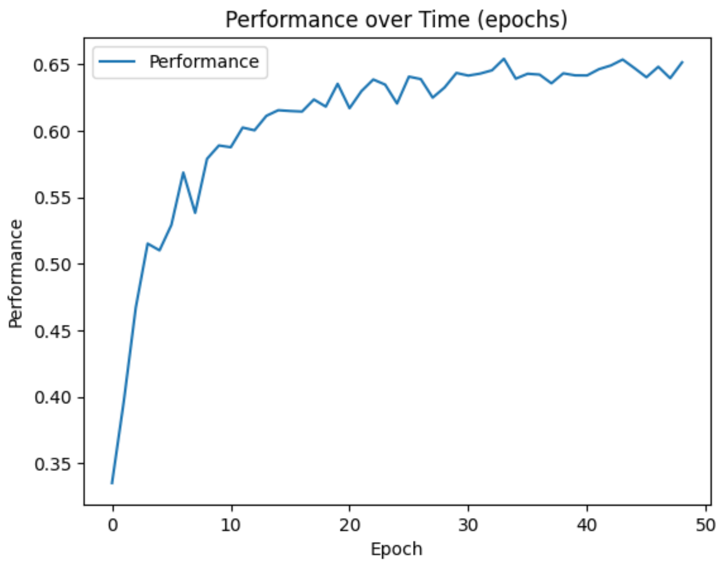

Figure 4: second iteration: segment dataset, 50 epochs: 236.943253203 MIN. BEST SCORE 65.432373046875 reached at epoch 35. Final Score at 65.160849609375

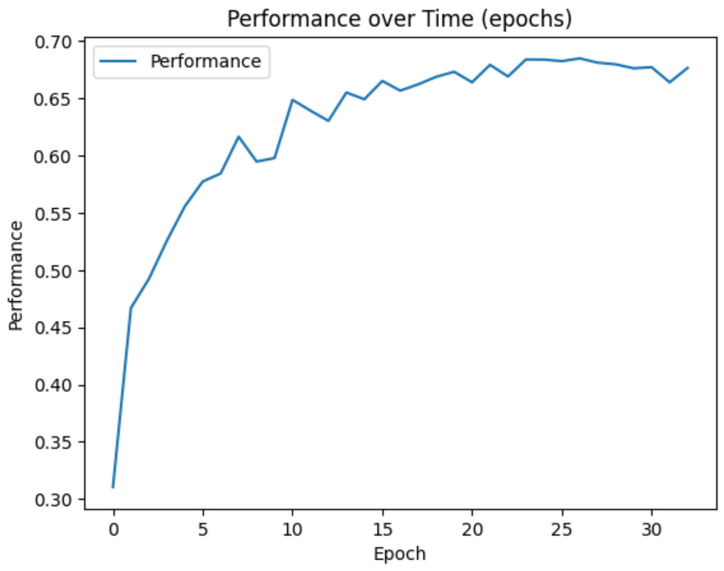

Figure 5: third iteration: segment dataset, 34 epochs: 211.102264 MIN. BEST SCORE 68.498642578125 reached at epoch 28. Final Score at 67.664384765625

For the forth and final iteration, I increased the size of the model architecture from the third version. In the forth iteration, I introduced two more convolutional layers with their corresponding batch normalization layers, followed by a new residual block, aiming to increase the model's complexity. Another considerable change between training session was the GPU type; I switched from a T4 GPU to an A1000 GPU on the final iteration, which significantly decreased the training runtime. Google Collab wasn't timing out for any of my previous models, because I was terminating the training session when the model stopped improving, so while the more powerful GPU decreased training time, the accuracy of the model should be similar regardless of GPU type. The model eventually reached a final score of 75.3687890625% accuracy after 100 epochs.

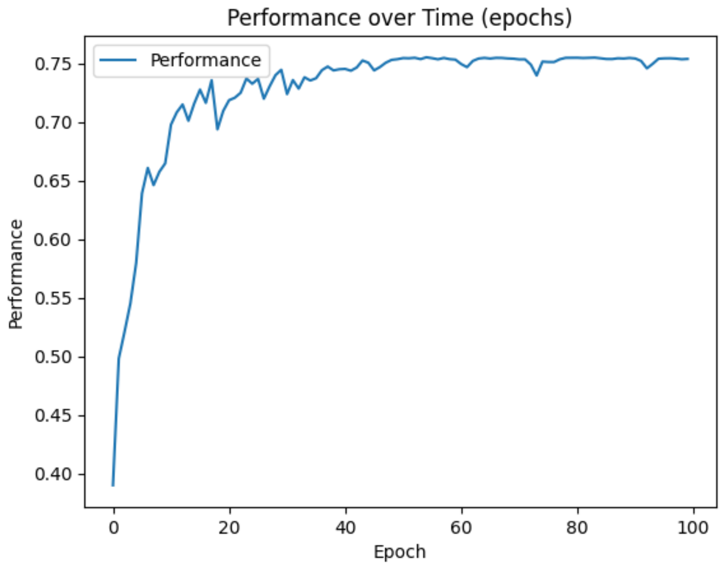

Figure 6: forth iteration: segment dataset, 100 epochs: 143.612789 MIN. BEST SCORE 75.506806640625 reached at epoch 56. Final Score at 75.3687890625