ResLT Paper Implementation:

Implementation of the ResLT model and its training methodology from the paper "ResLT: Residual Learning for Long-tailed Recognition" on the long-tailed CIFAR-10 dataset.

Initial Setup & Core Implementation

1. The Dataset:

- A data loading script to generate long-tailed versions of the CIFAR-10 dataset with varying imbalance ratios (100x, 50x, 10x), matching the paper's setup. The same, that was used in the implementation of ELF was used here again.
- Data Pre-processing: RandomCrop (with padding) and RandomHorizontalFlip. Pixel values were normalized only to the [0, 1] range by ToTensor(), omitting any further normalization.

2. The Model Architecture (ResLT_ResNet32):

- Implemented a ResNet-32 backbone suitable for CIFAR-10.
- The core of the implementation was the ResLT head. This was created using a single nn.Conv2d layer with groups=3, which efficiently acts as the three specialized branches described in the paper.
- Encountered a ValueError because the standard ResNet-32's final feature dimension (64) is not divisible by 3. Resolved this by modifying the final block of the backbone to output 48 channels, a number that is both divisible by 3 and architecturally consistent with ResNet design principles.

3. The Custom Loss Function:

- Implemented the two-part loss function in the training loop.
  - Fusion Loss (L_fusion): A standard cross-entropy loss on the summed output of the three branches.
  - Branch-Independent Loss (L_branch): The specialization component. Implemented this by calculating three separate cross-entropy losses on the *individual* branch outputs, using filtered data subsets (all data for branch 1, medium+tail for branch 2, and tail-only for branch 3).

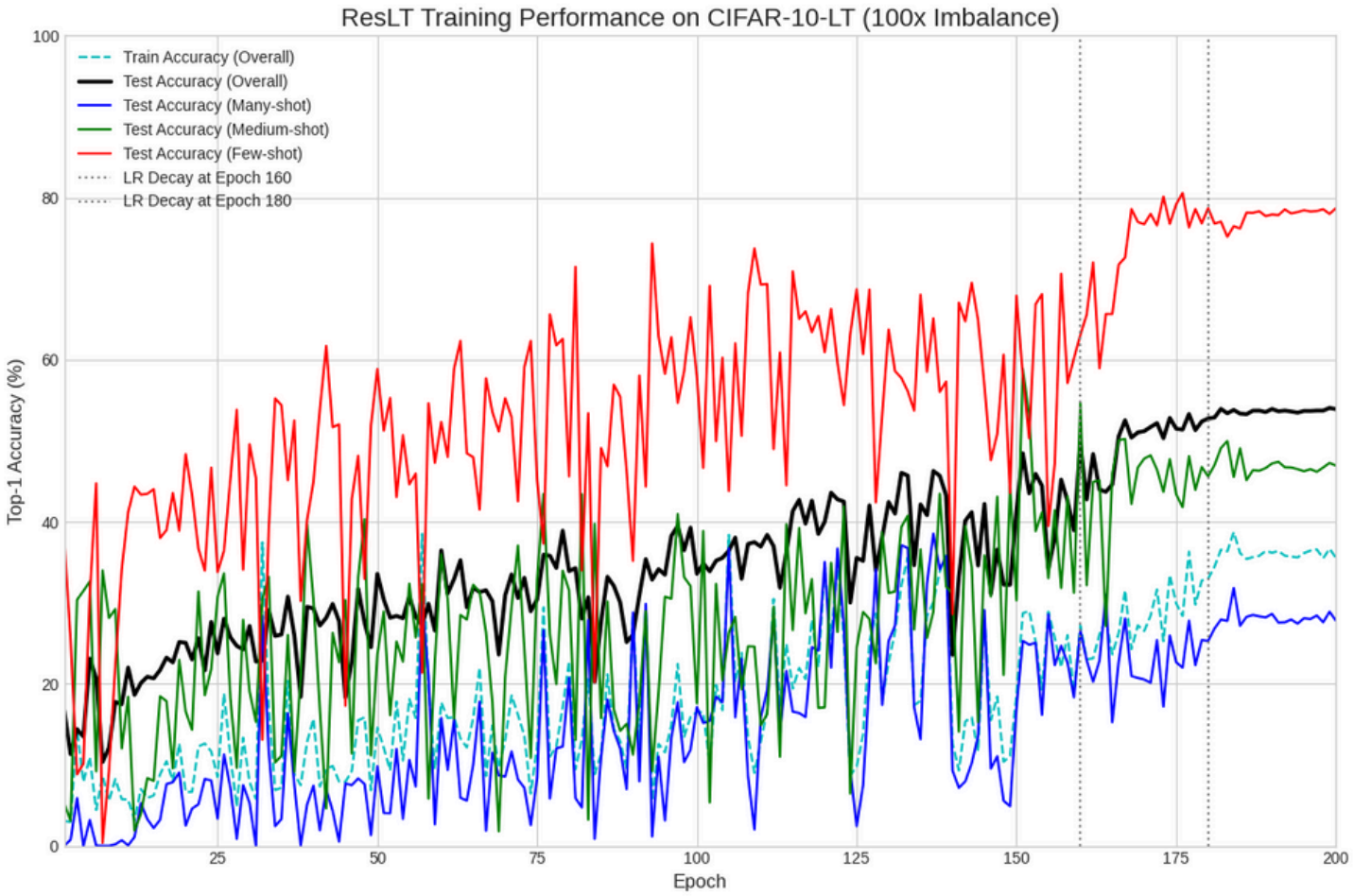
Hyperparameter Tuning & Experiments

With the core components in place, I began a series of experiments on the 100x imbalanced dataset to tune the critical alpha hyperparameter and replicate the paper's results.

Paper's Accuracy	100x	50x	10x
	80.44	83.46	89.06

Experiment 1: The Paper's Value (alpha = 0.995)

- Result: Low overall accuracy (~54%).
- Analysis: The model became a "tail-class specialist." It achieved very high accuracy on the rare, few-shot classes (~78%) but performed terribly on the common, many-shot classes (~28%). The high alpha value caused the model to over-specialize on the tail of the distribution.



The clearest difference can be seen around **epoch 175**. At this point, the **red line (Few-shot accuracy)** is peaking at nearly **80%**, while the **blue line (Many-shot accuracy)** is struggling down around **25%**. It is noted that the model's performance is **wildly different** across the class frequency splits.

Experiment 2: The Opposite Extreme (alpha = 0.9)

- Result: A much better overall accuracy of ~72%.
- Analysis: This created the opposite effect, a "head-class specialist." The model performed very well on the many-shot classes (~83%) but sacrificed performance on the few-shot classes (~68%). This experiment successfully bracketed the optimal alpha range.

Experiment 3 & 4: Zeroing In (alpha = 0.99 and 0.95)

- The run with alpha = 0.99 produced another tail-class specialist, showing the model is highly sensitive in this range.
- The ***run with alpha = 0.95 was the most successful***. It achieved a final accuracy of ~71% and, most importantly, produced ***a balanced model that performed well across all three class splits*** (Many: ~75%, Medium: ~65%, Few: ~73%).

100x Models	Overall	Many	Medium	Few
Alpha= 0.995	53.9	27.8	46.93	78.7
Alpha= 0.9	72	83.37	66.27	67.95
Alpha= 0.99	61.77	43.9	52.5	82.12
Alpha= 0.95	70.79	74.73	64.73	72.45

Pushing for State-of-the-Art Performance

Having found the best alpha value, the final phase focused on closing the remaining ~9% gap to the paper's reported ~80% accuracy.

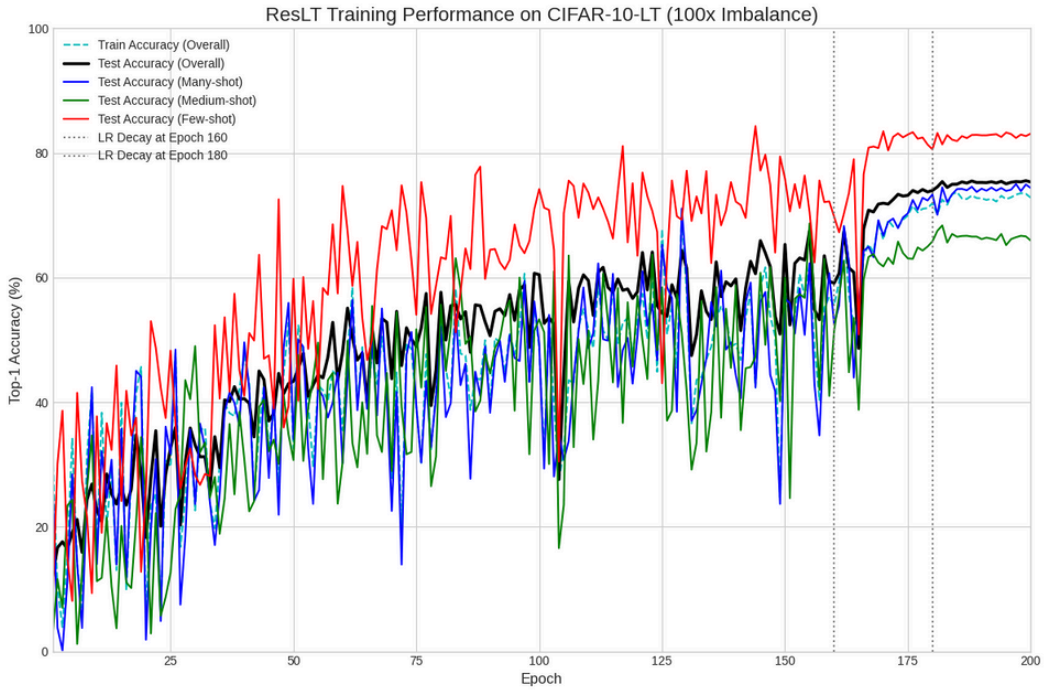
Experiment 5: Adding Stronger Augmentation

- Setup: Keeping alpha = 0.95 and added AutoAugment to the data loader.
- Result: The final accuracy was still around ~71%.
- Analysis: This indicated that while our alpha tuning was solid, we had likely hit the performance ceiling with the current configuration.

Experiment 6 : Adding Label Smoothing

- An additional regularization technique, label_smoothing=0.1 was applied to the loss function.
- This yielded the best accuracy of all the experiments yielding around 75.33%

100x Models	Overall	Many	Medium	Few
Alpha= 0.95	70.79	74.73	64.73	72.45
Alpha= 0.95 with AutoAugmentation	70.22	67.87	62.3	77.92
Alpha= 0.95 with AutoAugmentation and Label Smoothing	**75.33**	**74.37**	**65.93**	**83.1**



Alpha= 0.95 with AutoAugmentation and Label Smoothing	**Paper's Accuracy**	Overall	Many	Medium	Few	**Difference**
100x	80.44	**75.33**	74.37	65.93	83.1	**~5.11**
50x	83.46	**80.14**	80.5	70.1	87.4	**~3.32**
10x	89.06	**86.78**	84.63	79.23	94.05	**~2.28**

Note: Main Changes that were made that were not in the paper:

To make the model work, I had to modify the ResNet-32 architecture by changing the final layer's out channels as 48 instead of 64, as grouped convolution required the in_channels as a multiple of 3.

Further, I performed extensive tuning of the alpha hyperparameter, finding that a different value (0.95) provided a much better accuracy balance than the one reported.

Finally, to push performance closer to the state-of-the-art, I incorporated more techniques like AutoAugment and Label Smoothing which were not part of the original methodology.