Maximizing Feature Extraction in DNNs for Transfer Learning

Ayon Borthakur AB2535@CORNELL.EDU
Matthew Einhorn ME263@CORNELL.EDU
Cornell University, Ithaca, NY United States

Abstract

Transfer learning enables applying the learning from one dataset on another to improve performance. We replicate previous studies on the relationship between layer depth and feature dataset specificity in AlexNet to DenseNet. We also explored implementing a novel data driven layer that could help extract more features from the first dataset, but we didn't find a significant improvement.

1. Introduction

Deep learning relies on large datasets for its success in image classification. One technique for handling smaller datasets is transfer learning, where a network is pre-trained on a large dataset and then adapted to the target dataset (Yosinski et al., 2014). Transfer learning could also be useful for less curated online datasets, such as eBird, where many of the training classes doesn't have sufficient training examples (Horn & Perona, 2017).

The idea behind transfer learning is that earlier layers learn general features that are combined and specialized in the later layers specifically to the current dataset. To adapt to a different dataset, the latter layers are replaced and retrained. As opposed to only learning the minimal features for the task we would like to force the original network, by increasing the data complexity, to learn many different independent features in the earlier layers. By learning diverse features, it should improve target dataset performance.

We here attempted to improve upon this technique by introducing a novel binary classifier layer in parallel to the traditional classifier layer. By randomly mapping existing class labels into two new classes we hypothesize the network can be made to learn many general features of objects so that it can map unrelated classes to the same label. Recently, (Arpit et al., 2017) found that a network learned the data patterns even when randomizing the individual example labels, but the network will learn data features first. We suspect that DenseNet's lack of redundancy is well suited for learning more independent features.

2. Related Work

(Yosinski et al., 2014) classically studied at the transferability of a layer's features as a function of layer depth in a network trained on ImageNet and found that generality decreased with depth. (Kruithof) later used the CIFAR-10 dataset to highlight its effectiveness on smaller data-sets.

3. Methods

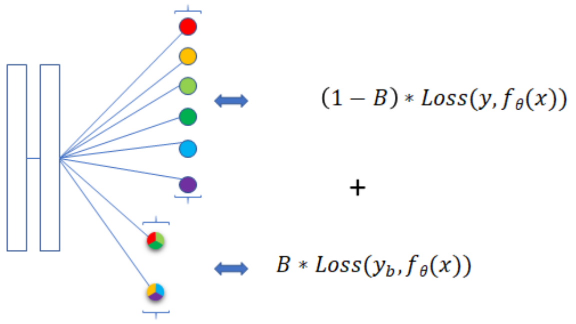

Figure 1. **Schematic of binary classifier** — A binary layer, labeling the classes as 0 or 1 each is added to the network in addition to the traditional classification layer. Combined loss is $L = (1-B)*Loss(y, output) + B*Loss(y_b, output)$, where y_b is the new binary layer loss.

3.1. Transfer learning

It is believed that the first layer of deep convolutional neural network learns some basic features such as edges and bars of different orientations, color blobs etc. These structures are observed across multiple datasets. But the later layers become more specific to the trained examples.

For two datasets, A and B, the standard transfer learning procedure is to first train a network(N_1) with dataset A. Some layers of N_1 are frozen and a new network N_2 is created by adding some new layers to the frozen layers. This method is effective in improving the test accuracy on dataset B compared to a network N_2 with no pre-training with dataset A. This strategy has been successfully applied in improving test accuracy on sensor and medical image datasets by pre-training with ImageNet dataset. (Yosinski et al., 2014) showed that transferability of features is inversely dependent upon the layer depth of a network. While (Yosinski et al., 2014) used AlexNet for his study, we here first analyze the effectiveness of transfer learning in a DenseNet (Huang et al., 2016) which achieves state of the art using less parameters by concatenating features learned in all layers. In addition to freezing the transfered layers we also look at reducing the learning rate for these layers instead.

3.1.1. Binary classifier

The aim is to extract more features from dataset A to increase performance on dataset B. Accordingly we tried to make the problem of classification harder to a network by introducing a binary classifier layer in parallel to the fully connected last layer (Figure 1). We generate a binary dataset from A by randomly assigning 50% classes to class 0 and the rest to class 1. For example, given a dataset $A : (x, y)$ with class labels $y = \{1, 2, 3, 4, 5, 6\}$, we randomly partition it into two equal size subsets $y_0 = \{2, 4, 5\}$ and $y_1 = \{1, 3, 6\}$ to generate y_b. We assign label 0 to y_0 and label 1 to y_1. The new dataset $A^{'}$ is (x, y, y_b).

We compute the loss as the weighted sum of the binary and normal classification layers; $L = (1 - B) * Loss(y, output) + B * Loss(y_b, output)$, where B is the weight value and $B \epsilon \{0, 1\}$.

3.1.2. Training & testing procedures

4. Experimental Evaluation

4.1. Transfer Learning Results

We trained DenseNet on the CIFAR100 dataset to investigate the effect of layer depth on transfer learning similar to (Yosinski et al., 2014). We used $DenseNet - BC(L = 100, k = 12)$, with 3 dense blocks based on (Amos, 2017) implemented in PyTorch.

We split the 100 classes into 2 random groups A and B with 50 classes each. We train the network on A (see 4.2) to get pre-trained weights. To train on B, we initialize with the pre-trained weights and reset the final classification (FC) layer. The training schedule is 40 epochs with learning rate (lr) drops at 26 and 36 using lr values of (0.01, 0.01, 0.001), except for the FC layer that starts at 0.1.

For the 1st experiment, to test layer freezing, we pick a layer N and freeze the weights of all the layers upto and including that layer e.g. block 2, layer 9 denoted 2_9 in Figure 2. The Base plot in figure 2 shows the mean test error of two replications at epochs 1, 21, and 40 for B at various N. We find that freezing more layers increases error, with dramatic increases for the final layers of block 3. On the other hand, the error rate on B vs. A is lower even after the first epoch. See Supp. 7 for full training curves.

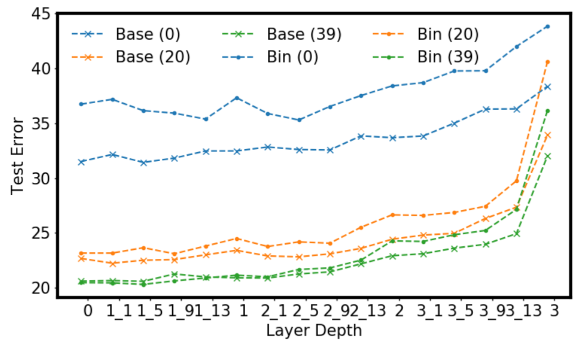

Figure 2. **Effect of layer depth freezing on transfer learning** — The test error on dataset B for the baseline and binary condition (loss weight is 0.4) for epochs 1, 21, 40 when all the layers upto and including layer X_Y (X-axis) is transfered from A and frozen. Remaining layers are fine-tuned for 40 epochs with reduced learning rate. Mean of 2 tests. X_Y means block X, layer Y within block X. There were 3 blocks, 0 means no blocks were frozen, 3 means all blocks were frozen

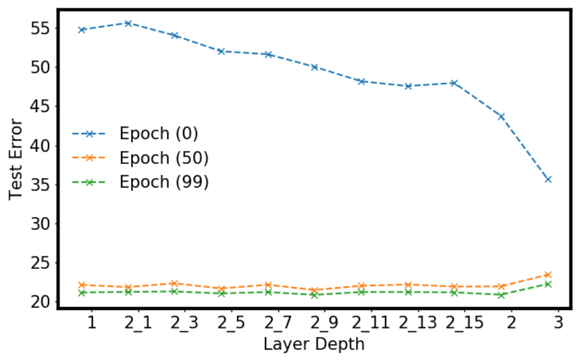

Figure 3. **Effect of layer depth on transfer learning** — The test error on dataset B for epochs 1, 51, 100 when all the layers upto and including layer X_Y (X-axis) is transfered from A and fine-tuned for 100 epochs with reduced learning rate. Remaining layers are reset with normal learning rate. X_Y means block X, layer Y within block X. There were 3 blocks, 1 means block one was retained, 3 means all blocks were retained

For the second Experiment, to test layer fine tuning, after picking N, we reset all the layers after N and set its initial lr to 0.1 and keep the weights for layers ¡= N and reduce its initial lr to 0.01. We fine tuned for 100 epochs (see 4.2) with a lr change at 51 and 76 similar to the 1st experiment. Figure 3 shows that although initially retaining more layers reduces the error, at 100 epochs retaining more layers makes little difference. See Supp. 8 for full training curves.

4.2. Binary CIFAR Results

4.2.1. DenseNet Results

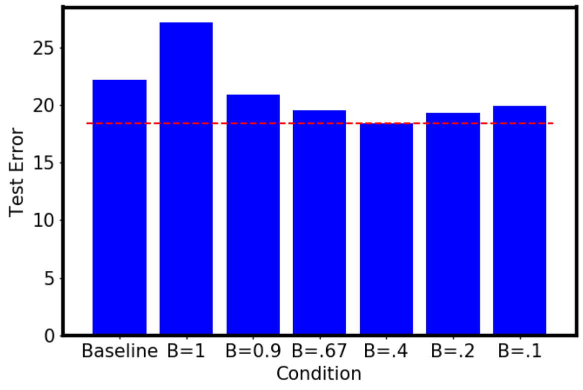

Figure 4. **Test error on dataset B for various loss weights** — Comparing the test error on dataset B when A was trained with various loss weights for the binary classifier. Also see 5.

In this section we investigate the effect of adding a binary classifier layer to DenseNet with CIFAR100. We split CIFAR100 into two random groups of 50 classes each - dataset **A** and **B**. With a random 32 crop and horizontal flip and using SGD with momentum (0.9) we trained **A** for 175 epochs with a lr of 0.1, 0.01, and 0.001 at epochs 1, 126, and 151, respectively. Although typical training is 300 epochs we used early stopping for time efficiency. Using the pre-trained weights from the best epoch, we transfered its weights and reset the last fully connected (FC) classification layer, then fine tuned 100 epochs on **B** with learning rate (lr) drops at 51 and 76 using lr values of (0.01, 0.01, 0.001), except for the FC layer that starts at 0.1. This is for the baseline case, for the binary case to create the binary classifier we split dataset A into two groups and assign them a 0 or 1 and add the binary layer's loss to the FC loss

As mentioned we have a choice of how much to weigh the binary loss; we chose a value of B=0.4 based on figure 4, which also showed that adding a binary classifier improved performance. When trying multiple binary layers simultaneously each with a different random class shattering we observed decreased performance (Sup 10) so we only used one binary layer. The above used a different random A/B class split for each test. We did not use a validation set because although we picked the best binary loss weight of 0.4 using figure 4, we were only looking for any potential difference between binary and baseline rather than claiming a best result.

To remove that uncertainty, we ran many paired binary vs baseline experiments where A/B classes were identically split for each pair of binary and baseline tests and we found

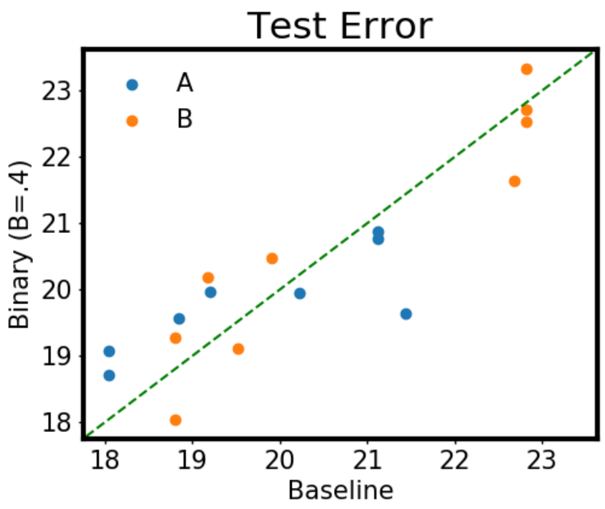

Figure 5. **Baseline and binary test error for dataset A and B** — For each random A/B dataset split shows the test error of A and B for the binary (loss weight is 0.4) vs. baseline condition. Also see 5. One can see a clustering of dataset B where some A/B split did worse than others.

no improvement with the binary classifier (Figure 5). We observed that there seems to be some clustering for dataset **B** test error and that some class splits were better at generalizing from A to B. However, overall we observe no improvement adding a binary classifier. See Sup 13, 11, and 12 for example full train and test error and loss curves. Figure 2 (and associated Sup 9) shows the effect on **B** of freezing upto layer N when **A** is trained with a binary classifier - it seems to do worse than baseline, but the overall result from 4.1 holds. However, it also shows if all the layers are fine-tuned (leftmost points), at the last epoch there seems to be little difference between binary and baseline.

4.2.2. Long tail

We set out to test whether a binary classifier could improve feature extraction, thereby requiring less data for dataset B. To test, for CIFAR100 we split the dataset A/B and then reduced the number of examples per class in B from 500 to 60. We then fine tuned for 40 epochs retaining all the weights from A except for the final layer, but reducing the lr similar to above. We compared test error on dataset B using weights from dataset A trained with and without a binary classifier with weight loss 0.4 for the same class split. The final epoch results from two runs, each with 7 replications is in table 1.

We observe that the two baselines perform differently indicating that different dataset splits generalize differently from A to B. In all cases the binary condition does worse indicating that adding a binary classifier to A reduces performance when there's not enough data in B. Strangely, when

Baseline 1	Binary 1	Baseline 2	Binary 2
31.81	38.86	35.57	37.12

Table 1. **Test error for dataset B when B is reduced to 60 images per class** — Compares the test error on B when B is reduced and shows two different binary vs. baseline tests with different A/B splits each. Each is the mean of 7 tests.

the baseline did better, the binary did worse. See Sup 14 for the full testing curve.

4.2.3. WRN Results

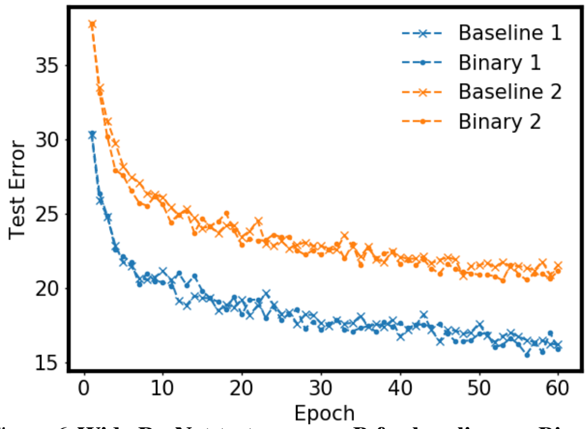

Figure 6. **Wide ResNet test error on B for baseline vs. Binary** — Two runs of the binary and baseline condition with different random A/B splits for each run, although the same between binary and baseline. Shows that some datasets A are better than others at generalizing to B.

To test whether adding a binary loss would improve performance when using a different network architecture we test CIFAR100 in a Wide ResNet - WRN-28-10 (drop 0.3) (Zagoruyko & Komodakis, 2016) using the PyTorch implementation from (Yang, 2017). We compared the baseline network with one that adds a binary layer as described, with loss weight 0.4. We tested twice, each with a different A/B split, but maintaining the same between baseline and binary. Figure 6 shows that there's little difference between binary and baseline for each test, however, the two A/B splits result in wildly different outcomes, evoking similarity with DenseNet that showed two test error clusters on B in figure 5.

4.3. ImageNet Results

We also tested once on ImageNet, extracting 100 classes and testing similarly to CIFAR100 but on $DenseNet-121(k = 32)$ using binary loss weight 0.4. The binary condition showed some improvement (sup 15), however, the baseline and binary conditions used different A/B class split, so given the previous results it seems likely the binary condition had a class split that better generalized from A to B, rather than that binary performed better.

5. Discussion

Our results showing that layer depth correlates with feature specificity in dataset A concur with previous work, including (Yosinski et al., 2014). This is especially evident when freezing the latter layers (figure 2). Although we believed initially that DenseNet would not build features in layer hierarchies, our results suggest that it does, but compared to previous networks it's not as strong for earlier layers.

When fine tuning for 100 epochs, we observed that the test error when resetting layers after N didn't depend much on N, suggesting 100 epochs is sufficient to largely retrain the network from scratch and using less epochs, like 40, is more appropriate in transfer learning. This seems to suggest that SGD is robust to initialization.

In Figure 5 Comparing test error when adding a binary layer vs. the baseline both used the same A/B dataset split and although we observed that each performed differently, binary did not improve. However, initially when looking at the effect of changing binary layer weighing on the loss (Figure 4), we used different random A/B splits when comparing baseline to binary. Similarly for the initial CIFAR100, wide ResNet and ImageNet single runs. In all cases binary did better than baseline. However, when comparing binary to baseline on the same A/B split, there was no improvement. It's unclear why initially the binary tests seems to have correlated with class splits where A transfers better to B; we suspect there may not have been sufficient randomness in the initial runs as we started from the same GPU seeding for the random number generator. Although that should not have affected across architectures and datasets.

The most convincing results that the binary condition is not working is from 4.2.2. That showed that different A/B splits generalizes differently, yet the binary still always did worse.

6. Conclusion

We replicated in DenseNet previous work that showed layer depth correlates with dataset specificity. Although compared to other architectures, it became most pronounced in DenseNet in the final layers.

We also investigated whether adding a binary loss layer would encourage the network to extract more features for transfer learning. Although we had some initial encouraging results, we were unable on average to show any improvement when adding this layer, rather the positive results seemed to have been confounded with the A/B dataset splits, where some are more amenable to transfer learning.

Acknowledgments

The authors greatly appreciate Professor Kilian Q. Weinberger as well as the course teaching assistants, Geoff and Jake, for paper readings which inspired this project as well as insightful discussions throughout the semester.

References

Amos, Brandon. densenet.pytorch: A PyTorch implementation of DenseNet, November 2017. URL https://github.com/bamos/densenet.pytorch. original-date: 2017-02-09T15:33:23Z.

Arpit, Devansh, Jastrzbski, Stanisaw, Ballas, Nicolas, Krueger, David, Bengio, Emmanuel, Kanwal, Maxinder S., Maharaj, Tegan, Fischer, Asja, Courville, Aaron, Bengio, Yoshua, and Lacoste-Julien, Simon. A Closer Look at Memorization in Deep Networks. *arXiv:1706.05394 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1706.05394. arXiv: 1706.05394.

Horn, Grant Van and Perona, Pietro. The Devil is in the Tails: Fine-grained Classification in the Wild. *SciRate*, September 2017. URL https://scirate.com/arxiv/1709.01450.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q., and van der Maaten, Laurens. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, August 2016. URL http://arxiv.org/abs/1608.06993. arXiv: 1608.06993.

Kruithof, Klamer. Object recognition using deep convolutional neural networks with complete transfer and partial frozen layers. volume 9995. International Society for Optics and Photonics, October . doi: 10.1117/12.2241177. URL https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9995/99950K/Object-recognition-using-deep-con e-transfer/10.1117/12.2241177.short.

Yang, Wei. pytorch-classification: Classification with PyTorch, November 2017. URL https://github.com/bearpaw/pytorch-classification. original-date: 2017-05-10T05:33:36Z.

Yosinski, Jason, Clune, Jeff, Bengio, Yoshua, and Lipson, Hod. How transferable are features in deep neural networks? *arXiv:1411.1792 [cs]*, November 2014. URL http://arxiv.org/abs/1411.1792. arXiv: 1411.1792.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

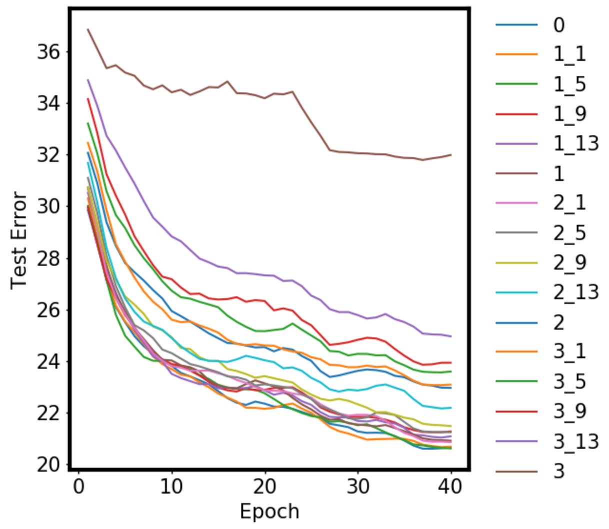

Figure 7. **Effect of layer depth freezing on transfer learning** — The test error on dataset B for all epochs when all the layers upto and including layer X_Y (X-axis) is transfered from A and frozen. Remaining layers are fine-tuned for 40 epochs with reduced learning rate. Mean of 2 tests. X_Y means block X, layer Y within block X. There were 3 blocks, 0 means no blocks were frozen, 3 means all blocks were frozen. CIFAR100 on DenseNet.

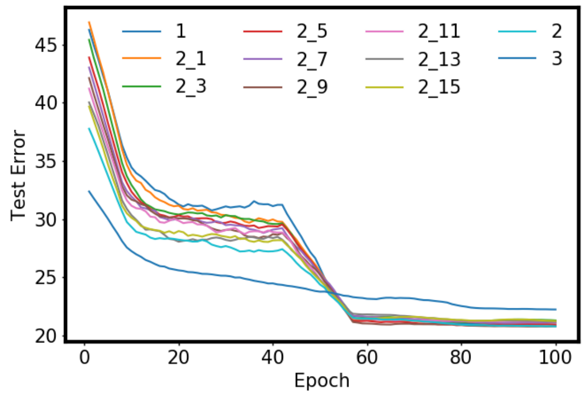

Figure 8. **Effect of layer depth on transfer learning** — The test error on dataset B for all epochs when all the layers upto and including layer X_Y (X-axis) is transfered from A and fine-tuned for 100 epochs with reduced learning rate. Remaining layers are reset with normal learning rate. X_Y means block X, layer Y within block X. There were 3 blocks, 1 means block one was retained, 3 means all blocks were retained. CIFAR100 on DenseNet.

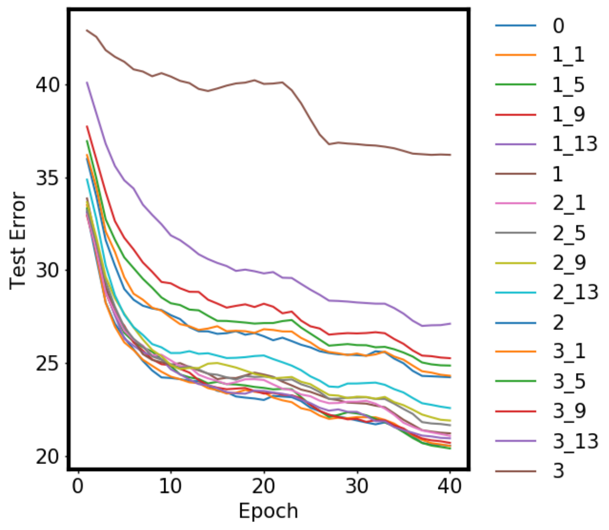

Figure 9. **Effect of layer depth freezing on transfer learning with binary condition** — The test error on dataset B for the binary condition (loss weight is 0.4) for all epochs when all the layers upto and including layer X_Y (X-axis) is transfered from A and frozen. Remaining layers are fine-tuned for 40 epochs with reduced learning rate. Mean of 2 tests. X_Y means block X, layer Y within block X. There were 3 blocks, 0 means no blocks were frozen, 3 means all blocks were frozen. CIFAR100 on DenseNet.

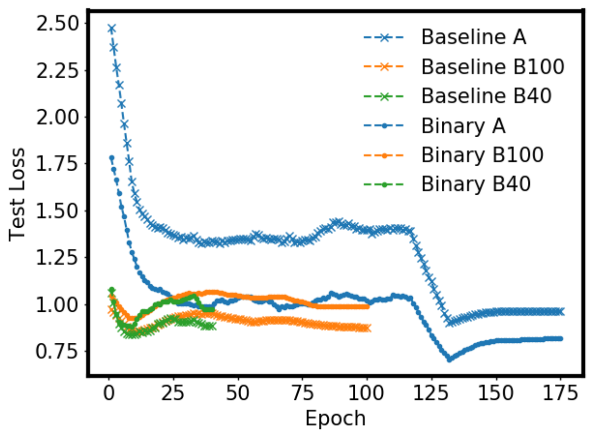

Figure 11. **Example test loss of A and B for baseline vs binary** — Compares the test loss of dataset A and B for the baseline and binary (loss weight is 0.4) conditions for CIFAR100 on DenseNet. B100, B40 mean it was fine-tuned on B for 100, and 40 epochs, respectively.

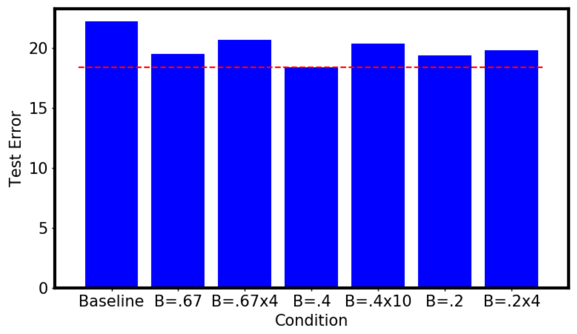

Figure 10. **Test error on dataset B for various loss weights** — Comparing the test error on dataset B when A was trained with various loss weights and number of binary layers for the binary classifier. B=bxn means the binary loss weight b was distributed equally across n binary layers each with a different binary split. B=b means there was only a single binary layer. CIFAR100 on DenseNet.

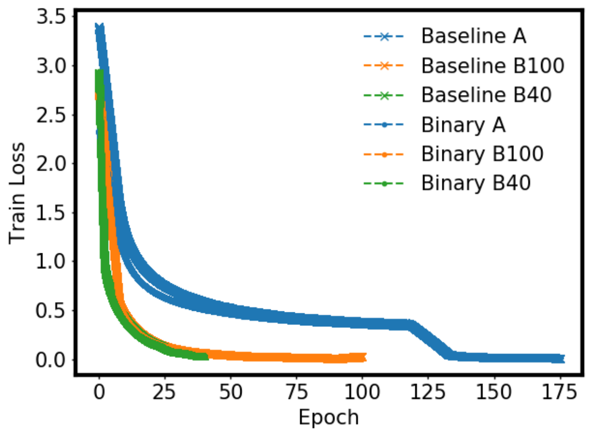

Figure 12. **Example train loss of A and B for baseline vs binary** — Compares the train loss of dataset A and B for the baseline and binary (loss weight is 0.4) conditions for CIFAR100 on DenseNet. B100, B40 mean it was fine-tuned on B for 100, and 40 epochs, respectively.

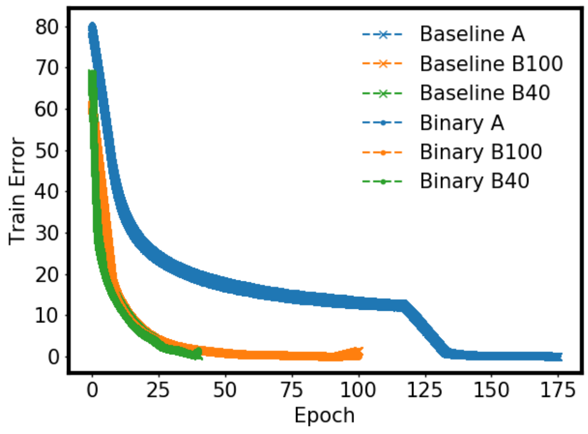

Figure 13. **Example train error of A and B for baseline vs binary** — Compares the train error of dataset A and B for the baseline and binary (loss weight is 0.4) conditions for CIFAR100 on DenseNet. B100, B40 mean it was fine-tuned on B for 100, and 40 epochs, respectively.

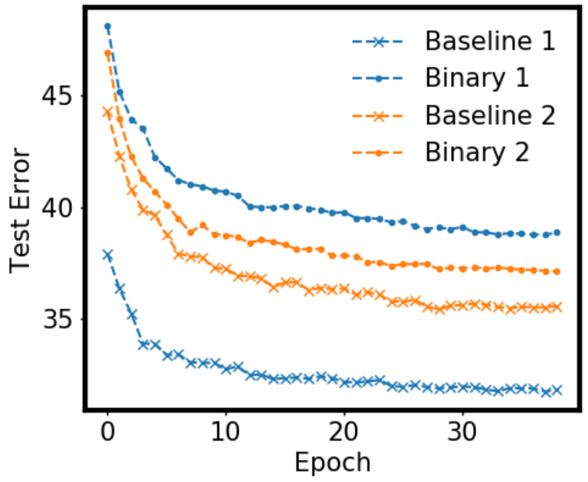

Figure 14. **Test error for dataset B when B is reduced to 60 images per class** — Compares the test error on B when B is reduced and shows two different binary vs. baseline tests with different A/B splits each. Each is the mean of 7 tests. CIFAR100 on DenseNet.

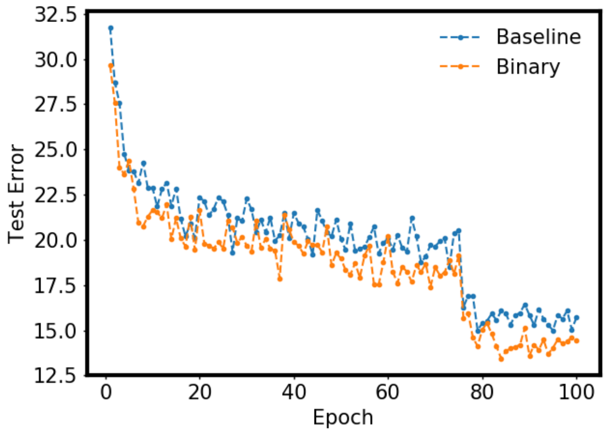

Figure 15. **ImageNet-100 test error baseline vs binary condition** — Compares the test error on baseline vs binary (loss weight is 0.4) using 100 classes extracted from ImageNet on DenseNet. Split is 50/50 A and B. However, the baseline and binary used a different A/B class split.