CSCE 479/879 Homework 2: Convolutional Architectures and CIFAR-100

Derek DeBlieck, Sabrina Fowler, Grace Hecke, Abby Veiman

October 31, 2025

Abstract

Image classification is a classic problem in computer vision. We trained two distinct convolutional architectures, each with two distinct sets of hyperparameters, on the CIFAR-100 dataset. All four of these classifiers perform well, with the best performing model attaining a top-1 accuracy score of 56%. Differences in model performance are discussed, and future research questions raised.

1 Introduction

For this assignment we developed convolutional neural networks to classify images from the CIFAR-100 dataset. This dataset consists of 100 classes, many of which are similar to each other, e.g. 'beetle' and 'cockroach'. [1] Therefore we needed to train models more complex than those we have worked with so far.

We tested out two different CNN architectures, and for each we varied the optimizer, learning rate, L_2-regularization, and dropout rate. The two architectures varied by number and depth of convolutional blocks.

Overall, our four models performed relatively well. Both of our shallower CNNs performed virtually identically on top-1 and top-5 accuracy. Our deeper CNNs had more variance in their performance, with one performing worse than the shallower ones and the other performing better.

2 Problem Description

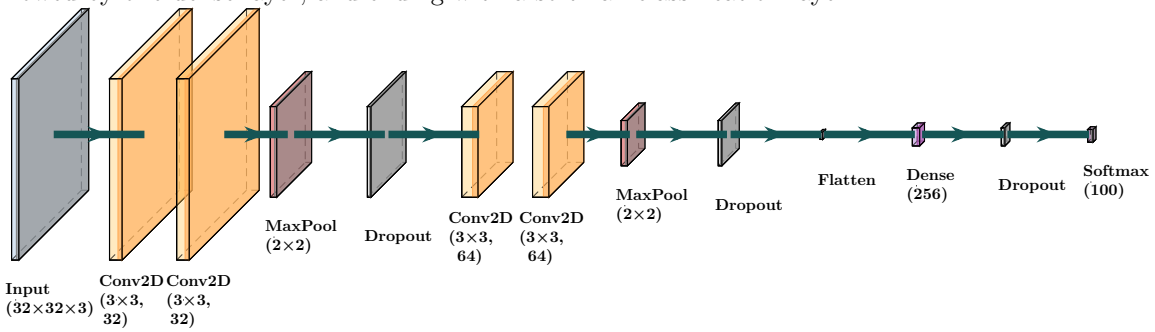
The problem we worked on was to use convolutional neural networks to classify images from the CIFAR-100 dataset. The CIFAR-100 dataset contains 60,000 32×32 color images, evenly distributed across 100 classes. Each class consists of 500 training images and 100 testing images. The image data is formatted as a `numpy` array of `uint8`s where each row represents a single image. The label data is given as a list of numbers in the range 0-99. [1]

The motivation for this problem is to develop a convolutional architecture that can handle significantly more classes than the 10 classes present in the

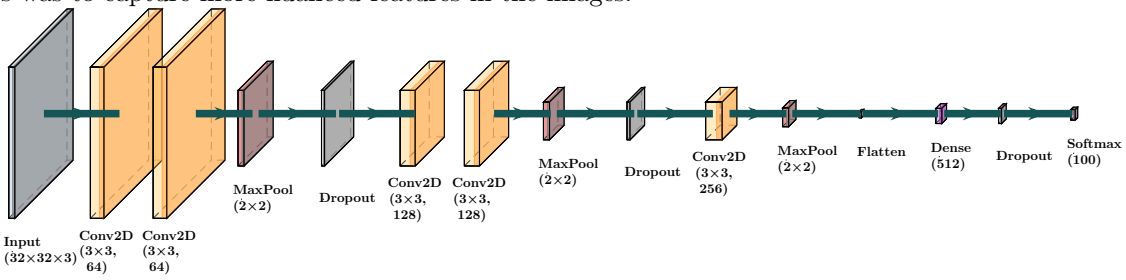
CIFAR-10 or MNIST datasets. This will be interesting because the 100 classes also fit into 20 superclasses [1], so many are similar and could be reasonably confused for each other. We will have to train our models to have attention to detail in order to be able to differentiate between similar classes. The goal is for the architectures we develop to be able to classify test images accurately into one of the 100 classes.

3 Approaches

For our first architecture we wanted to start small so that we didn't have something unnecessarily complex for the task at hand. This architecture consisted of two blocks with two convolutional layers with max pooling and dropout, followed by one dense layer, and ending with a softmax classification layer.



For our second architecture we built upon the first one by adding more filters to our convolutional layers. The purpose of introducing more convolutional layers was to capture more nuanced features in the images.



4 Experimental Setup

The dataset was loaded using `tensorflow.keras.datasets.cifar100`. For our experiments, the training set was split into 50,000 images for training and 5,000 images for validation, while the test set contained 10,000 images. All images were normalized to the $[0, 1]$ range and the labels were converted to

one-hot encoding. The training data was shuffled before splitting to ensure randomness. Additionally, data augmentation was applied during training using `ImageDataGenerator`, including rotations up to 10°, width and height shifts of up to 8%, horizontal flips, and zoom of up to 8%.

Two convolutional neural network (CNN) architectures were tested. Model A is a smaller network with two convolutional blocks, each consisting of two convolutional layers followed by max pooling and dropout. Then there is a single fully connected layer, and the softmax output. Model B is deeper, with three convolutional blocks, each increasing in the number of filters. To prevent overfitting, this network uses higher dropout rates and stronger L_2-regularization. Model B also has a larger fully connected layer before the softmax output.

We conducted experiments with different combinations of optimizers, learning rates, L_2-regularization, and dropout. The batch size was fixed at 256, maximum epochs were set to 100, and early stopping with a patience of 5 epochs was used. Table 1 summarizes the hyperparameter configurations for each model and run. We used callbacks including early stopping (monitoring validation loss and restoring best weights), reducing learning rate on plateau, and saving the best model with `ModelCheckpoint`.

Model	Run	Optimizer	Learning Rate	L2 Reg	Dropout
A	1	Adam	0.001	1×10^{-4}	0.2
A	2	SGD (momentum=0.9)	0.01	5×10^{-4}	0.3
B	1	Adam	0.001	5×10^{-4}	0.4
B	2	SGD (momentum=0.9)	0.01	1×10^{-3}	0.5

Table 1: Hyperparameters for CIFAR-100 experiments.

All models were compiled with categorical crossentropy loss. The metrics monitored during training included Top-1 accuracy and Top-5 accuracy using `TopKCategoricalAccuracy(k=5)`. When enabled, data augmentation was applied to the training data. Each model was trained for up to 100 epochs or until early stopping criteria were met, after which the best model weights were restored.

The primary evaluation metrics were Top-1 accuracy, defined as the fraction of samples correctly classified, and Top-5 accuracy, defined as the fraction of samples for which the true label is among the top five predicted probabilities. Ninety-five percent confidence intervals for Top-1 accuracy were computed using the standard normal approximation:

$$\mathrm{CI}_{95\%} = \hat{p} \pm 1.96\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

where $\hat{p}$ is the observed accuracy and n is the number of test samples. Additionally, confusion matrices were generated for visual inspection, and macro-averaged precision, recall, and F1 scores were reported.

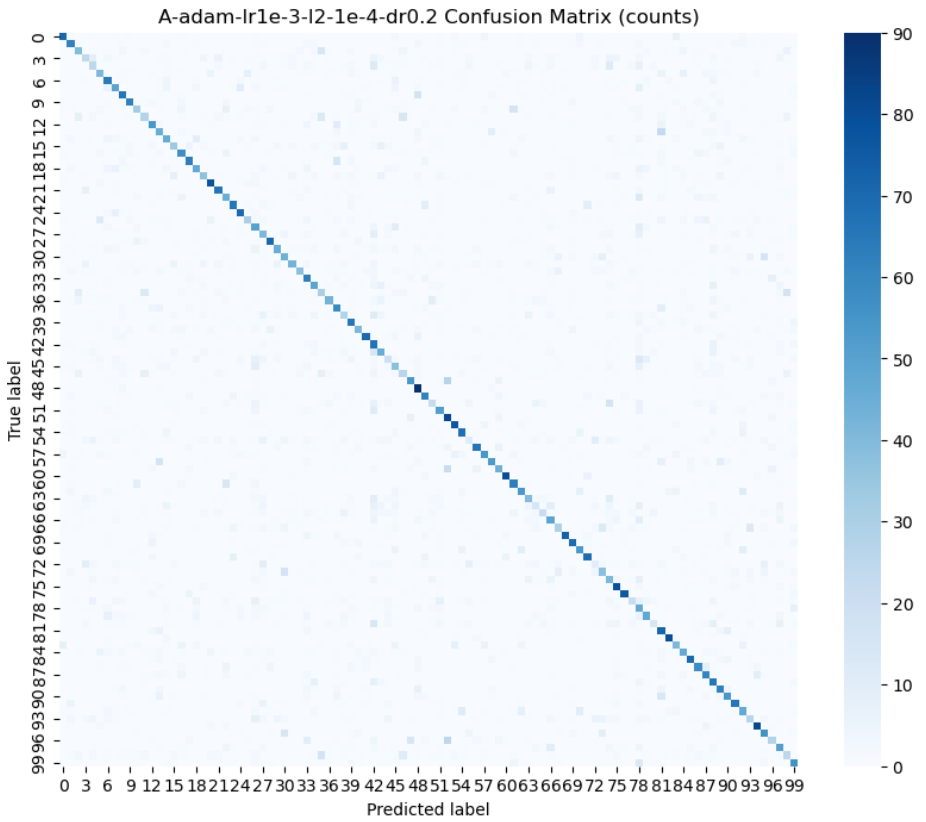


Figure 1: Model 1 Run 1 Confusion Matrix - Note that the chart's title lists the hyperparameters used

5 Experimental Results

We tested two different model architectures with two sets of hyperparameters each, for a total of four models.

Architecture 1, described in section 3 above, performed similarly across both sets of hyperparameters. In the case of the first run (i.e., the first set of hyperparameters used), the model was trained for 55 epochs before early stopping halted the training process. As we used a patience interval of 5, we reverted back to the weights trained through 50 epochs to evaluate on the test set. This model attained a 50.53% top-1 accuracy on the test set with a 79.45% top-5 accuracy. Additionally, this model has a 95%-confidence interval of 49.55-51.51% for top-1 accuracy. Finally, this model achieved (macro) precision, recall, and F1 scores of 51.48%, 50.53%, and 0.4999 respectively. Figure 1 is this model's confusion matrix.

For the second run of architechture 1, this time with different hyperparam-

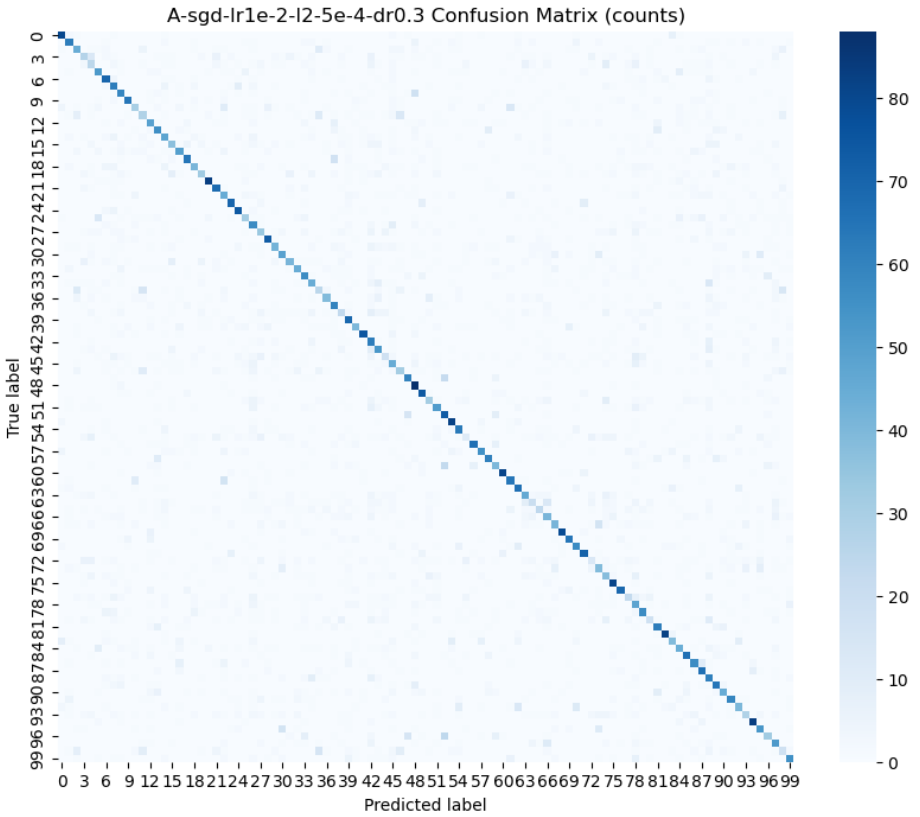


Figure 2: Model 1 Run 2 Confusion Matrix

eters, we again trained for a maximum of 100 epochs. This training run was stopped early after 77 epochs. It attained a top-1 accuracy of 51.37% and a top-5 accuarcy of 79.04%, with a 95%-confidence interval of 50.38%-52.34% for top-1 accuracy. This model achieved (macro) precision, recall, and F1 scores of 51.50%, 51.36%, and 0.5067, respectively. Figure 2 is architecture 1, run 2's confusion matrix

Our second architecture was also trained using two different sets of hyperparameters. Again, descriptions of these hyperparameters can be found in section 3. The model trained using the first of these two sets of hyperparameters was stopped early after 70 epochs. It attained a top-1 accuracy of 45.81% and a top-5 accuracy of 75.79%. Additionally, it has a top-1 accuracy 95%-confidence interval of 44.83-46.79%. Finally, it achieved precision, recall, and F1 scores of 46.28%, 45.81%, and 0.4491, respectively.

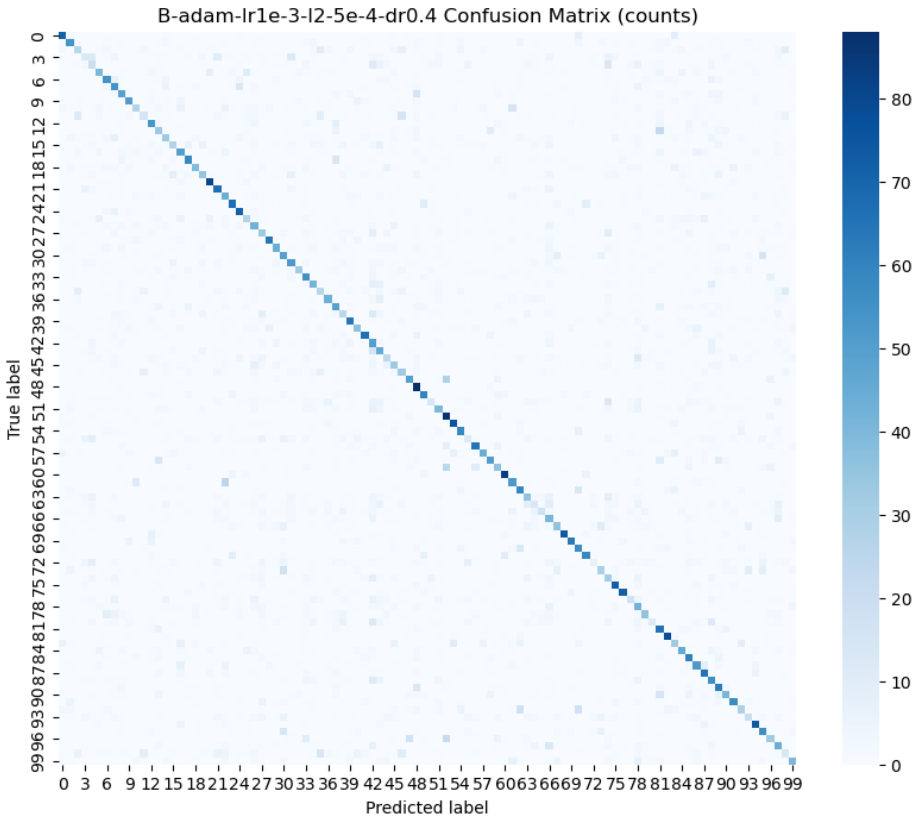


Figure 3: Model 2 Run 1 Confusion Matrix

Finally, we ran trained a fourth model using our second architecture with a new set of hyperparameters. This model trained for 80 epochs before it was stopped early. It achieved a top-1 accuracy of 56.02%, a top-5 accuracy of 82.28%, with a top-1 accuracy 95% confident interval of 55.05-56.99%. It also attained precision, recall, and F1 scores of 55.93%, 56.02%, and 0.5560, respectively.

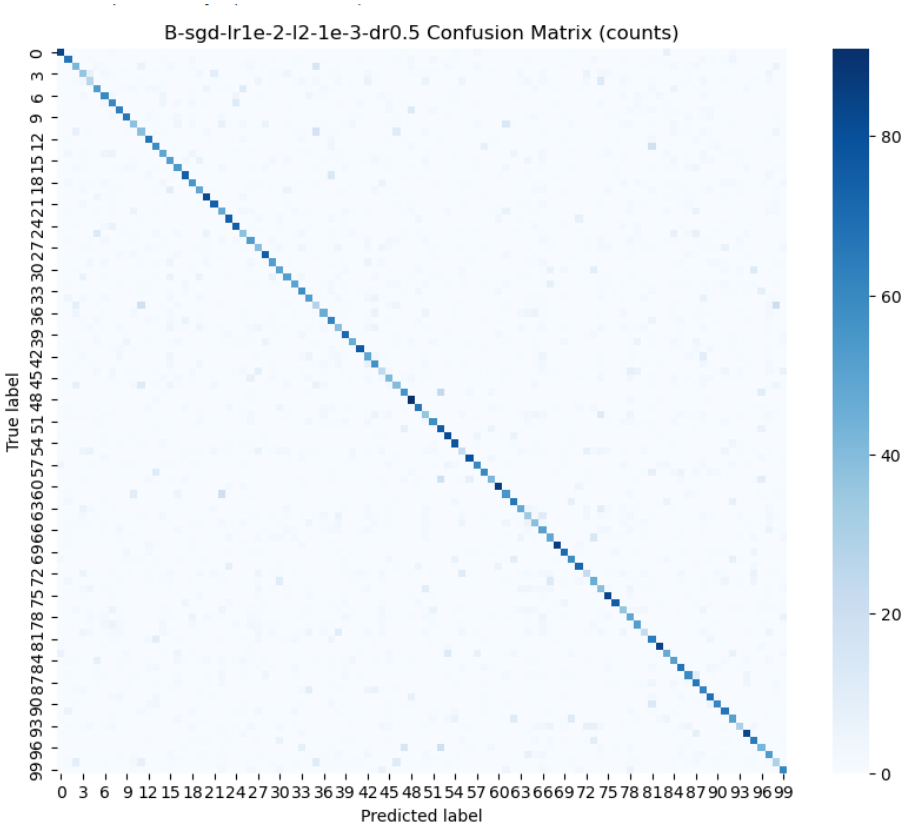


Figure 4: Model 2 Run 2 Confusion Matrix

6 Discussion

The first two models performed quite similarly. The changes in optimizer, learning rate, L_2-regularization, and dropout rate did not have a strong impact on model performance. The difference in performance would be much more noticable if the model performed better, but as the top-1 accuracy was only around 50% in both models, a change of $< 1\%$ in accuracy does not feel like a substantial difference. However, there was a difference in the number of epochs needed to attain relatively similar results. The model using the Adam optimizer stabilized much more quickly than the model using SGD.

The first model trained with the second architecture was the worst performing of the four models. This was originally a disappointment, as we had expected (at least slightly) improved performance as we increased the number of layers in the architecture. However, after testing this new architecture with a first set of hyperparameters, the results were disappointing. It performed worse than a simpler model and took more epochs to train. For this second architecture,

we increased our dropout rate, as we had more nodes in this architecture than in the first and were more concerned about overfitting. We had also increased the amount of L_2-regularization as a second level of protection against overfitting. Perhaps we were too cautious, as this model third model performed quite poorly. When reviewing this model's performance after each epoch, this model's performance looks quite distinct from those of the other three models. In the other three models, the loss on the validation set is actually lower than the loss on the test set in the early epochs. We believe this is due to the dropout happening in the test data. However, in all three of the other models, early stopping does not occur until the loss on the test data is either equal to or below the loss on the validation set. But in the case of our third model, the loss on the validation set is still much lower than the loss on the test set when early stopping is implemented (see the two figures below). Perhaps we needed an increased patience interval for this model, but is rather unclear to us why this would be so different than architecture 1 run 1, which also used the Adam optimizer.

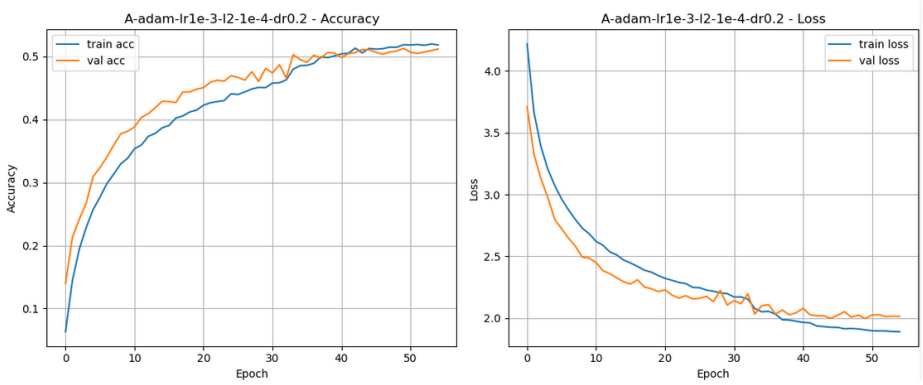


Figure 5: Accuracy and Loss by Epoch - Model 1 Run 1

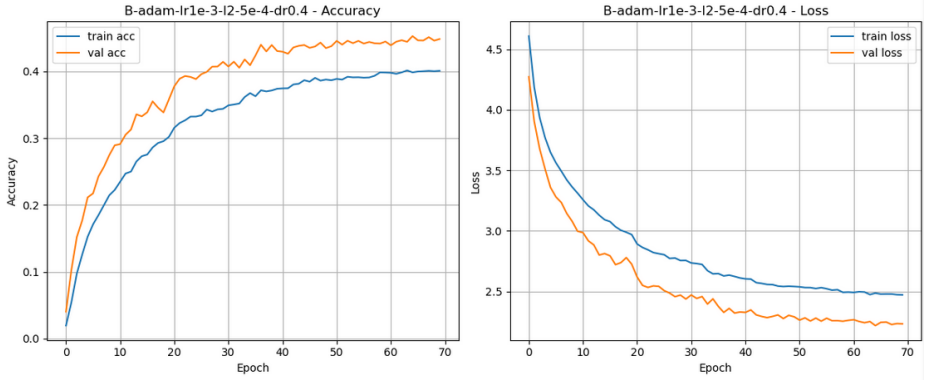


Figure 6: Accuracy and Loss by Epoch - Model 2 Run 1

While the results of our third model were slightly disappointing, the last model was by far the best performing. This was slightly surprising to us, as the third model (model B, Run 1) performed so poorly. Truthfully, it is slightly mysterious to us why this model saw such an improvement in performance. Our hypothesis is that the second architecture required more epochs to converge, and the higher learning rate in the fourth model allowed it to converge quickly enough before being stopped by early stopping. This conclusion is perhaps supported by the figure below (note how the train/validation loss begin to flatten out twice before continuing downward), but additional testing would be necessary to confirm or deny our suspicions.

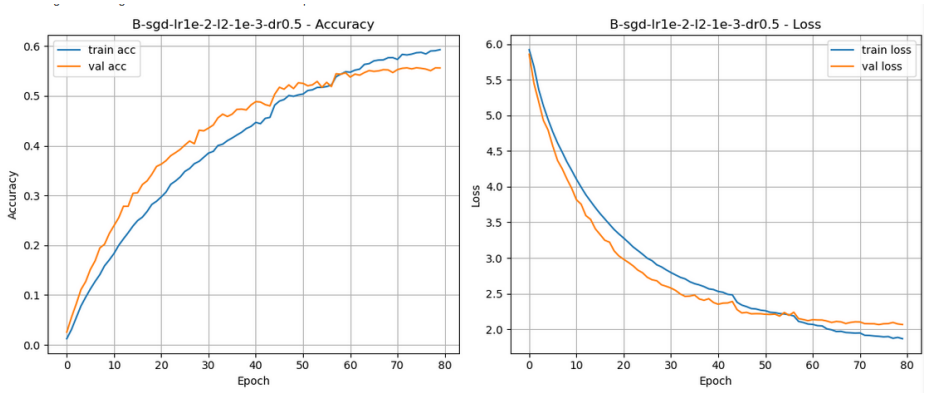


Figure 7: Accuracy and Loss by Epoch - Model 2 Run 4

7 Conclusions

Convolutional architectures are a reliable, high-performing method for image classification. Even our worst performing model was able to correctly classify

images nearly half of the time, an impressive feat when dealing with 100 labels. However, architecture design and hyperparameter selection can significantly impact model performance. In many cases, it isn't entirely clear why small changes seem to have a relatively large impact on performance. This highlights the need for hyperparameter tuning as well as the testing of multiple architectures. In future work, we would like to explore in more detail how the learning rate and L_2-regularization can be used to explain the difference in performance between our third and fourth models. Additionally, we would like to test whether our third model can be improved if given a much longer patience interval.

Table 2: Contributions by team member for this assignment.

Team Member	**Contribution**
Abby Veiman	Code first draft
Grace Hecke	Code finishing and organizing
Sabrina Fowler	Writeup sections 1-4
Derek DeBlieck	Writeup sections 5-7, Abstract

References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL: `https://api.semanticscholar.org/CorpusID:18268744`.