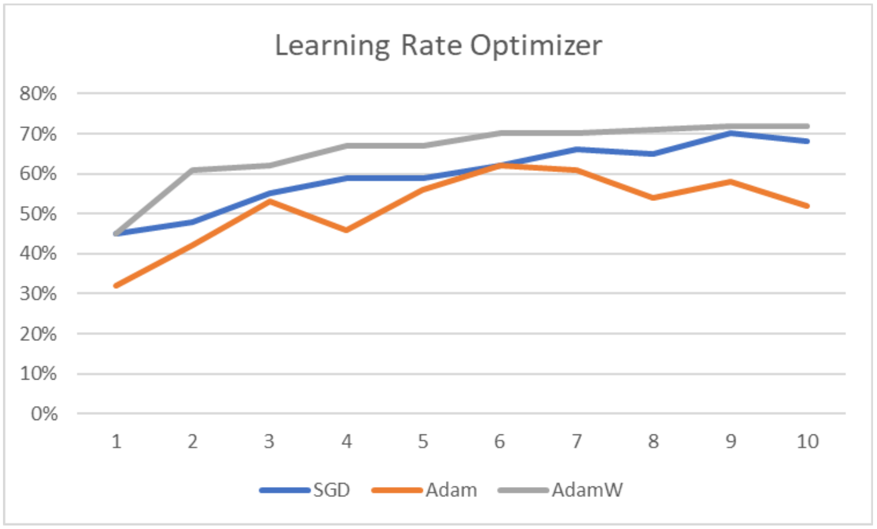
Improve FinalExamNet CNN Accuracy

Introduction
In the final exam, we have been asked to create a CNN model and train the model to achieve an accuracy of at least 62%. This CNN is trained on the CIFAR-10 data, which contains 60,000 32x32 color images in 10 classes with 6000 images per class. During the training process for FinalExamNet, accuracy reaches 63% after 10 epochs, and batch size = 128. For this final project, I want to propose a new CNN that can reach higher accuracy with 10 convolutional and dense layers. Before creating a new architecture, I kept the original CNN and made changes to other aspects, including data augmentation, weight initialization, and optimization.

Data Augmentation
The goal of data augmentation is to generate more data from the original train set to expand the training set and make sure the model learns enough data to make predictions. In the original net, we applied random crop and random horizontal to the data. I see there was room to improve. There was a variety of augmentation techniques but it does not mean trying everything. I came across the AutoAugment Policy for the CIFAR-10 dataset and decided to use this to replace the past transformation. CIFAR-10 AutoAugment Policy contains a variety of image operations and each operation specifies the probability of applying the operation and the magnitude of the operation. When using the AutoAugment function with CIFAR-10 policy, the function would use the transformations that yield the highest accuracy, which is proven based on scientific research. By utilizing AutoAugment for CIFAR-10, I could ensure that I apply the most effective transformations to maximize the accuracy of my CNN.

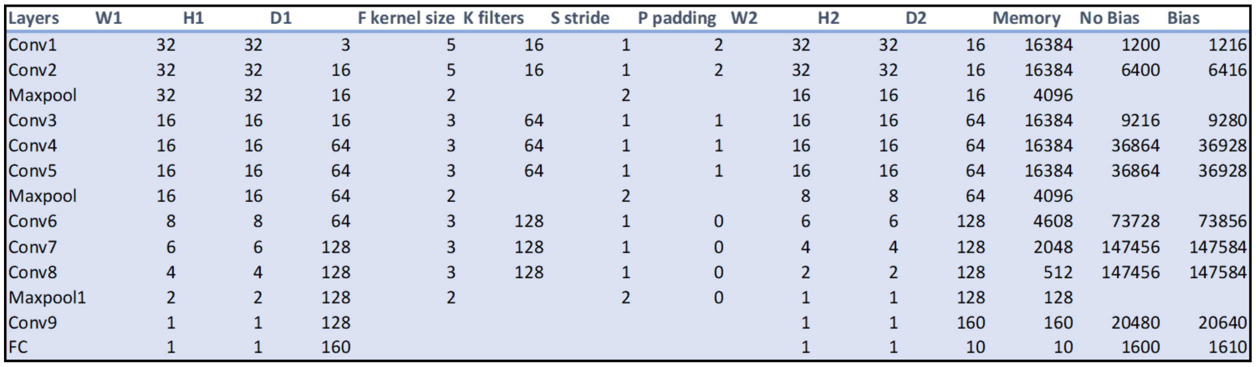
Learning Rate Optimization
The past CNN was trained using a fixed learning rate with Stochastic Gradient Descent. Since the learning rate is fixed, the model can take a long time to reach the optimal point because all parameters are updated after each example. As a result, the model can have high variance in some batches and accuracy cannot pass after a few batch. Also, the fixed learning rate can make convergence happen very slowly.
I consider adaptive learning rate through AdaGrad and RMSProp optimization, but there is a better hybrid of the two called Adam Optimization. First, Adagrad updates the learning rate for each parameter after each time stamp by accounting for the gradient change of the parameter. However, as the accumulated sum of gradient change increases during training, the learning rate can become very small. Thus we have RMSprop, which has a decay rate to have a decaying average of the sum of squared gradients.
Adam (Adaptive Moment Estimation) differs from these methods as it combines RMSprop and Momentum gradient descent. It considers both the first moment in RMSProp and the 2nd moment of the gradients. In this new CNN, I use AdamW, an improved Adam Optimization as it fixed the issue of weight decay and generalizes better. AdamW optimizes weight decay and learning rate separately so the weight decay can still have an optimal value. Keeping old CNN architecture fixed, I tried SGD, Adam, and AdamW and tracked the accuracy rate:



Adam optimization actually did worse than SGD if it did not control for weight decay. SGD might not work as well as AdamW at first, but the gap narrows after more epochs were trained.

CNN Architecture

The task of coming up with a whole new CNN architecture is not easy. Thus, I took some inspiration from state-of-the-art CNN architecture and did something similar. Specifically, I looked at VGC-16 architecture and noticed that it grouped 3 convolutional layers of same size together before the max-pooling layer and repeated the same patterns until the fully connected layer. The kernel size is small, 3x3, and the number of filters increases after each convolutional layer group. My proposed CNN architecture has the following structure:



Layers	W1	H1	D1	F kernel size	K filters	S stride	P padding	W2	H2	D2	Memory	No Bias	Bias
Conv1	32	32	3	5	16	1	2	32	32	16	16384	1200	1216
Conv2	32	32	16	5	16	1	2	32	32	16	16384	6400	6416
Maxpool	32	32	16	2		2		16	16	16	4096		
Conv3	16	16	16	3	64	1	1	16	16	64	16384	9216	9280
Conv4	16	16	64	3	64	1	1	16	16	64	16384	36864	36928
Conv5	16	16	64	3	64	1	1	16	16	64	16384	36864	36928
Maxpool	16	16	64	2		2		8	8	64	4096		
Conv6	8	8	64	3	128	1	0	6	6	128	4608	73728	73856
Conv7	6	6	128	3	128	1	0	4	4	128	2048	147456	147584
Conv8	4	4	128	3	128	1	0	2	2	128	512	147456	147584
Maxpool1	2	2	128	2		2	0	1	1	128	128		
Conv9	1	1	128					1	1	160	160	20480	20640
FC	1	1	160					1	1	10	10	1600	1610

Adjustments in the old CNN are:

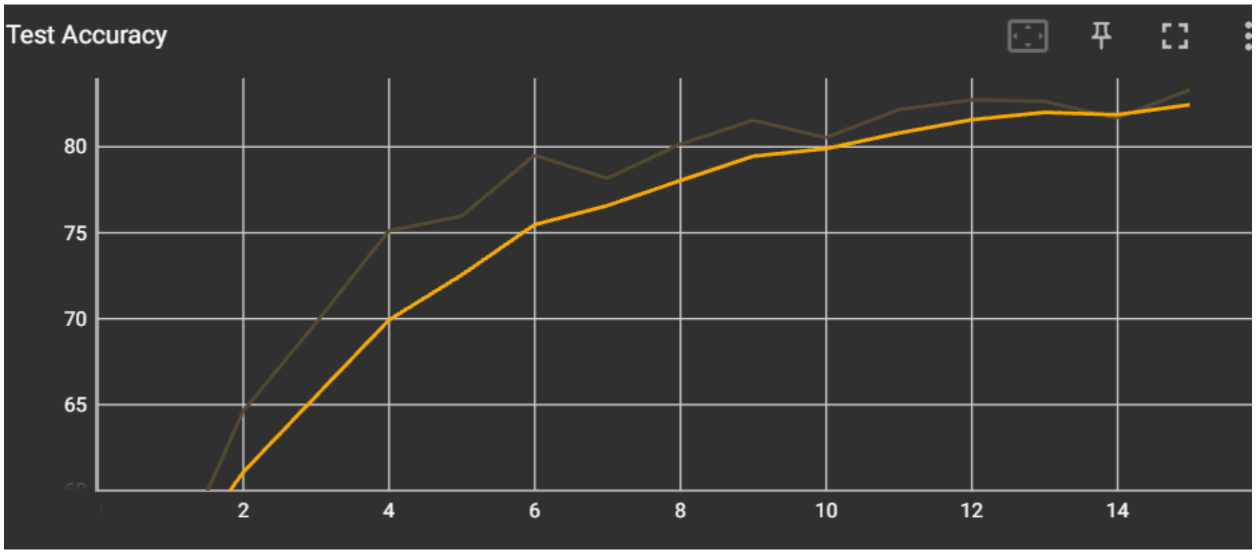
- Increase the number of filters: 16 - 64 - 128
- Decrease kernel size: 5x5 for the first two convolutional layers and 3x3 for 6 next convolutional layers
- Increase the number of convolutional layers: a wider and deeper CNN would detect more patterns and train longer, thus achieving better accuracy. This new CNN adds 3 more convolutional layers to the neural network.
- Reduce fully connected layers: Since the limit is 10 layers, I decide to reduce the fully connected layers to 1 even though I would prefer to have 2 or 3 dense layers. But I prioritize the number of convolutional layers more.

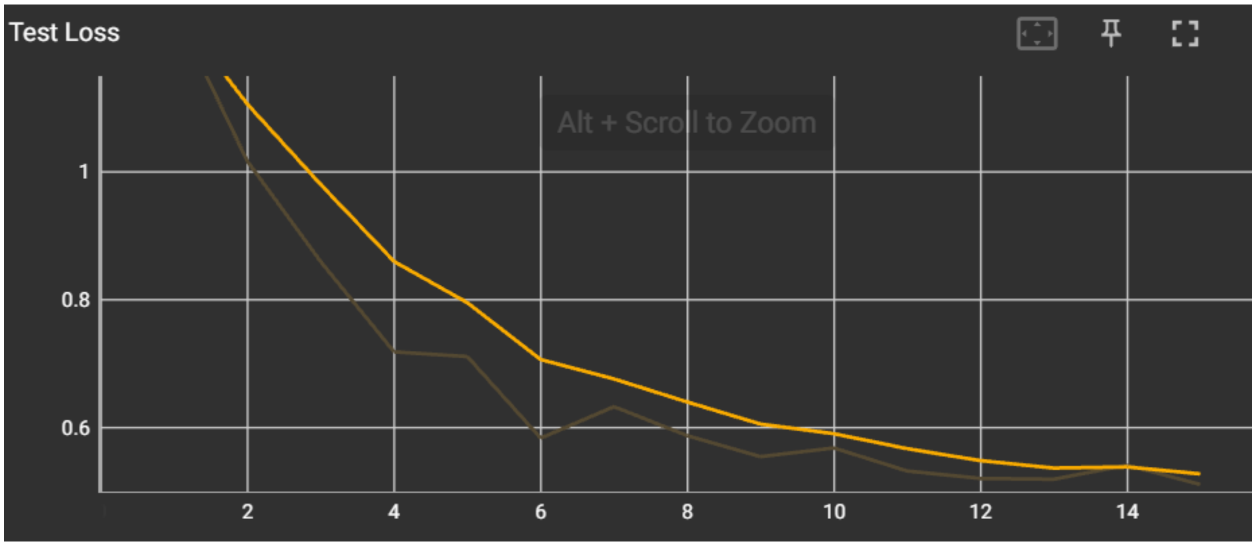
Batch Normalization
Before each Relu Activation layer, I added a batch normalization layer to stabilize the values. By standardizing the inputs, we ensure that the activation layer would work correctly and the neural network would produce an accurate result.

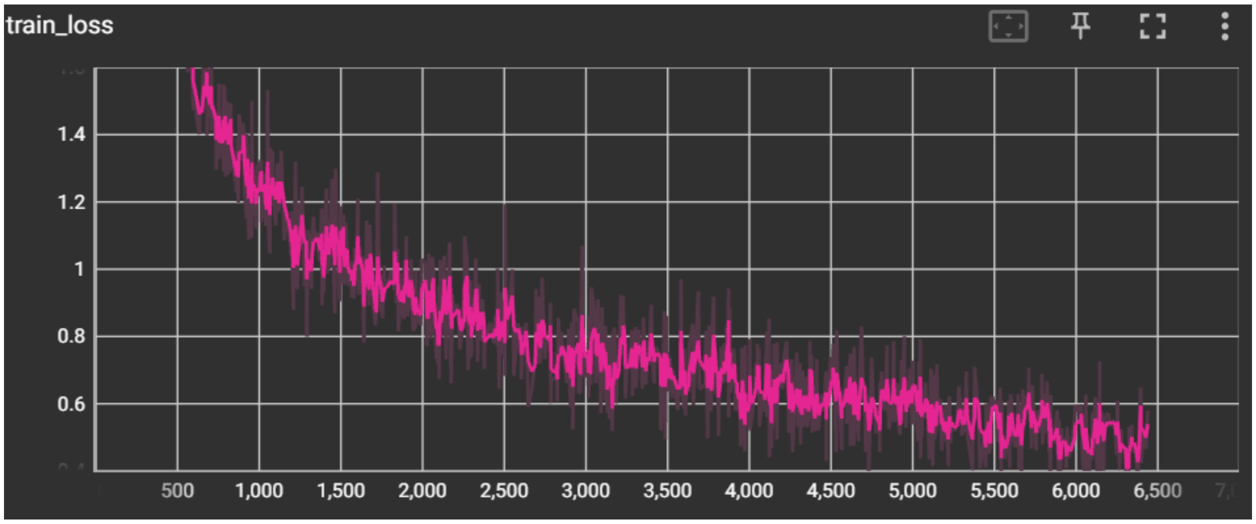
Drop Out
In the only fully connected layer, I replace a batch-norm layer with a drop-out layer. After some trials of both dropout and batch normalization, I found that accuracy did not really improve. In a paper “Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift” by Li, Chen, Hu, and Yang, it seems that only one should be used at a time. Thus, I used batch normalization in convolutional layers and dropout in a dense layer

Activation Function
I did adjust the activation function to leakyRelu in later layers as I want to account for situations where there are negative values. It fixes the “dying ReLU” problem, as it doesn't have zero-slope parts.

Training Process
I tried various modifications of this architecture and recorded the latest training of this CNN in 20 epochs. The model achieves an accuracy of 84%.







Conclusion
Based on the graph, the CNN achieves accuracy at a slower rate after epoch 10. It is possible to achieve higher accuracy if training for more, but it would be computationally expensive as the model seems to reach a plateau around 82-84%.