Homework 06 – Group 15: Changes to the architecture

Our initial model showed some overfitting, since our testing accuracy plateaued noticeably. Thus, we intend to optimize our model by implementing the following changes:

Architecture:

The first change we introduced was a bottleneck in the convolutional blocks with a kernel size of 1. Our intention was to boost the performance of the level (and thus the network), while at the same time saving computational resources, as specified in the lecture.

Furthermore, we introduced a fully connected (dense) layer at the end of the network. This aids in providing a network in more expressive power with the cost of being more computationally intense.

The next changes to the architecture were the introduction of Dropout Layers and Batch Normalization Layers. Dropping random units makes it harder for the network to learn specific datapoints and thus helps to prevent overfitting on unseen data.

Batch Normalization normalizes the activation of a layer so we get an approximately standard normal distribution

Futher optimizations:

- Dataset normalization
    - We normalized the data using the built in tf per image normalizer. Data normalization is prudent to make reaching local minimums in the loss landscape less likely and to reach a faster convergence.
- Early stopping with patience parameter
    - Early stopping is almost never a bad thing to as you don't want to overfit your model on the training data (i.e., specialize too much on the seen data.
- Checkpointing (saves best model and loads it when stopped)
    - This is generally also a good idea as it makes sure that your best model is always saved during training.
- Different kernel weight initialization
    - For some reason our model seemed to plateau in the first few epochs before finally starting to train. We suspected that this might be the case due to an unfavorable weight initialization. Upon changing the way the layers were initialized said problem went away.
- Optimizer change
    - Based on this github repo (https://github.com/adhishthite/cifar10-optimizers) Adadelta might perform favorably and was therefore chosen as the optimizer.
- Weight plotting

- We now also plot the weights associated with each layer in order to get more insight on the inner workings of our network