Assignment 1: CIFAR-100 Image Classification with Keras

David Gardner
May 30, 2025
Computer Vision Research Experience for Undergraduates
Center for Research in Computer Vision
University of Central Florida
(Dated: May 31, 2025)

I. INTRODUCTION

A. Theory Overview

The purpose of this assignment is to familiarize myself with different Convolutional Neural Network (CNN) architectures, coding in python with keras, and working with the Newton kernel. As an electrical engineering student with little experience in deep learning and computer vision, I took this assignment as an opportunity to gain more context before starting the research project.

II. DATASET

The CIFAR-100 dataset is a collection of labeled 32x32 colored images. There are 100 classes with 600 images for each class. The dataset is split into training and testing bins. There are 500 images and 100 images in the training and testing bins, respectively. The 100 classes are further broken down into 20 superclasses [1].

The images are from work at MIT and NYU where they compiled a dataset of 80 million images from the internet with labels based on the text search that generated the image [1]. Alex Krizhevsky created the CIFAR-100 dataset by paying students to label images from the 80 million images dataset, taking a subset of 100 classes from the 80 million images dataset. Only images that had one instance of the class, were photorealistic, and were images of objects were included. Partially obstructed images were included if the student was able to identify the object. Listed here are the 100 classes and 20 superclasses [1].

Superclass	**Classes**
Aquatic mammals	beaver, dolphin, otter, seal, whale
Fish	aquarium fish, flatfish, ray, shark, trout
Flowers	orchids, poppies, roses, sunflowers, tulips
Food containers	bottles, bowls, cans, cups, plates
Fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
Household electrical devices	clock, computer keyboard, lamp, telephone, television
Household furniture	bed, chair, couch, table, wardrobe
Insects	bee, beetle, butterfly, caterpillar, cockroach
Large carnivores	bear, leopard, lion, tiger, wolf
Large man-made outdoor things	bridge, castle, house, road, skyscraper
Large natural outdoor scenes	cloud, forest, mountain, plain, sea
Large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
Medium-sized mammals	fox, porcupine, possum, raccoon, skunk
Non-insect invertebrates	crab, lobster, snail, spider, worm
People	baby, boy, girl, man, woman
Reptiles	crocodile, dinosaur, lizard, snake, turtle
Small mammals	hamster, mouse, rabbit, shrew, squirrel
Trees	maple, oak, palm, pine, willow
Vehicles 1	bicycle, bus, motorcycle, pickup truck, train
Vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

TABLE I. Superclass and Classes in CIFAR-100

III. METHODS

To explore architectures for the CIFAR-100 dataset, I took existing models and adjusted their parameters. I borrowed architecture ideas from Alex Krizhevsky's 2009 paper "Learning Multiple Layers of Features from Tiny Images", the original paper from which this dataset is from. I also read online resources for pre-defined architectures on Geeks4Geeks.

Krizhevsky's paper includes the concept of whitening images before passing them through the model. I attempted to implement this algorithm. I also tried to match the architecture in Krizhevsky's paper.

The code is adopted from my python script for Assignment 0 using Keras to construct the model architecture. I used ChatGPT, Stack Overflow, and online documentation for assistance. I used both the Google Colab GPU runtime kernel and the Newton cluster GPUs.

Now we will discuss the theoretical ideas behind archi-

tecture elements I included.

Whitening Transformation: Krizhevsky's 2009 paper attempted to prevent the model from creating filters based on unimportant features that all color images of real objects tend to have [1]. For instance, the corners of images typically are the same color and pixels that are close together tend to be similar in color. The idea is that the model may focus too much on these irrelevant similarities between images instead of on key defining characteristics of objects.

These patterns across real color images manifest in the covariance matrix of the random vector input. Let Y be a random vector image input where each element of Y is a flattened 32x32x3 pixel color image with elements from 0 to 255, and the mean of each element of Y is normalized to 0. To remove any irrelevant correlations between pixel colors at different pixel locations, we desire to find a whitening transformation W such that the covariance matrix of $W(X)$ is the identity matrix.

Given n samples of X, let X be a $32 \times 32 \times 3$ by n matrix with the ith column containing the ith sample of X. We can approximate the correlation matrix of Y by

$$C \approx \frac{1}{n-1} XX^T.$$

We also desire that the covariance matrix of a transformed X, called $A := WX$, satisfies

$$\frac{1}{n-1} YY^T = I,$$

meaning the covariance matrix of Y is approximately the identity matrix. There are many such W, so we choose W such that $W = W^T$. Taking these expressions together, we get that

$$W = \left(XX^T \frac{1}{n-1} \right)^{-\frac{1}{2}} = C^{-\frac{1}{2}}.$$

Notice C is symmetric, so it is diagonalizable. That is, there exists an orthonormal change of basis matrix P and a diagonal matrix D where

$$PDP^{-1} = PDP^T = C.$$

This gives us an explicit expression for W, with

$$W = PD^{-\frac{1}{2}}P^T.$$

Here, $D^{-\frac{1}{2}}$ just means D with all of its elements raised to the $-\frac{1}{2}$ power. Also, another benefit of whitening is that whitening speeds up the convergence of the model parameters [2].

Batch Normalization: The concept of batch normalization was first introduced in the 2015 paper "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" [3]. Batch normalization filters can be added between learning layers to reduce "internal covariate shift", as described in the paper. That is, batch normalization reduces the effects of nonlinearities, regularizes outputs from learning layers, and increases the optimal learning rate to make training faster [3].

The internal covariate shift refers to "the change in the distribution of network activations due to the change in network parameters during training", as stated in the 2015 paper [3]. For each mini batch, the sample mean and variance are measured then the batch is normalized. This is a similar idea to whitening but it occurs on the inside of the hidden layer.

IV. EXPERIMENTS

I ran the following experiments:

Model B is the architecture that I created for Assignment 0. There are leaky ReLu activation functions after each layer.

Model C is the model from Geeks4Geeks without any change.

Model D is the model from Geeks4Geeks with the whitening pre-processing step.

We fix the batch size at 32 and the epoch number at 200 because these numbers worked best for optimizing runtime and higher system learning rate.

V. RESULTS

The accuracy rates at the end of each of the 200 epochs for each model id shown in the following figures. The final model performances on the training data (accuracy and loss) and on the testing data (val accuracy and val loss) are also shown below.

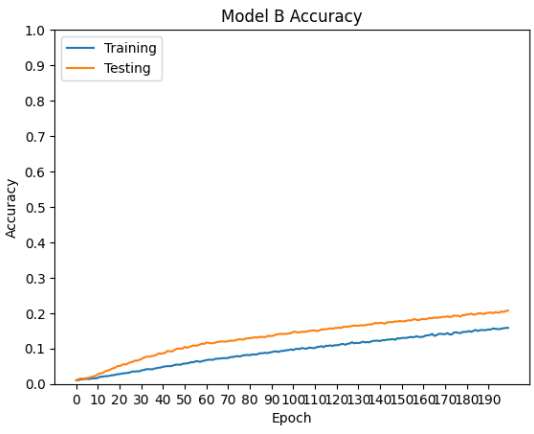

FIG. 1.
The model from assignment 0 struggled to classify the images.

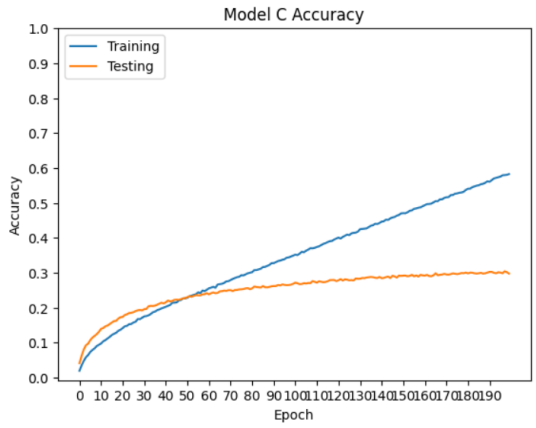

FIG. 2.
The model from Geeks4Geeks outperformed my original model from Assignment 0.

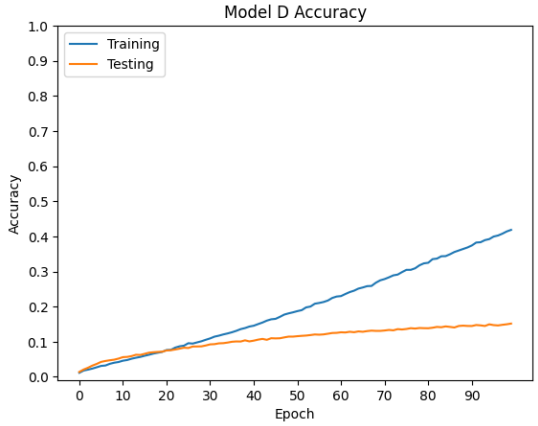

FIG. 3.
Whitening decreased model performance.

VI. DISCUSSION

The whitening algorithm surprisingly hurt the performance of the model, with the Geeks4Geeks model D performing with a 15.2% validation accuracy, worse than the Geeks4Geeks model C without the whitening with a validation accuracy of 30.42%. If I had more time, I would have looked into implementing more conventional architectures like ResNet and VGG-16. I will likely do this on the weekend.

This assignment helped me better understand how to develop the architecture for a deep learning model. A major challenge was choosing where to start. Next time, I would have just jumped right into the literature review instead of struggling to create a model from scratch without much assistance.

[1] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, Technical Report (University of Toronto, Toronto, ON, Canada, 2009) https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf.
[2] S. Wiesler, A. Richard, R. Schlüter, and H. Ney, Mean-normalized stochastic gradient for large-scale deep learning, in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2014) pp. 180–184.
[3] S. Ioffe and C. Szegedy, engBatch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv.org (2015).

TABLE II. Model Performance Comparison

Model	Accuracy	Loss	Val Accuracy	Val Loss
Model B	0.1570	3.7305	0.2071	3.5998
Model C	0.5842	1.8610	0.3042	3.0037
Model D	0.4186	2.6224	0.1517	3.8537

VII.