Project Proposal 8420

Will Sherrer
Clemson University
wsherre@g.clemson.edu

Ashton Sobeck
Clemson University
asobeck@g.clemson.edu

1 Abstract

Our group explores the effectiveness of Convolutional Neural Networks (CNN) when images that have been reduced by Principal Component Analysis (PCA) have been input as training. If CNN models can still be trained on dimensionally-reduced dataset, it would allow for dataset expansion with not needing to store as much information per image. We use the Canadian Institute for Advanced Research (CIFAR) dataset our training and testing images. We then used PCA to reduced the dataset at different intervals and train 2 CNN architectures we the dimensionally-reduced images. We used compression ratios at 81.25%, 87.5%, 93.75%, and 96.88% to see whether the models could effectively be trained to classify the images. What we found is that there is a correlation between the amount of compression and the models' ability to learn how to classify the images. At many points 81.2% of compression, or the first 6 of 32 principal components of the image, was on par or just below the ability of the standard dataset. We conclude that it is possible to train models with some form of image compression. However further studies should be conducted for fine-tuning the technique.

2 Motivation

Throughout our class of CPSC 8420, we have learned many things. One of the most crucial concepts of the class is certainly Singluar Value Decomposition (SVD). Using SVD, one can utilize PCA to find the important features of data. CNNs are quite effective in identifying patterns within images through the use of filters and non-linearity within their structure. However, when input images get larger and larger, the network can grow in size and in return, require much more computational power to train in a timely manner.

3 Method

The machine learning techniques we plan to implement are PCA and SVD. These will be used to reduce the dimension of an images which the CNN models will be trained and tested on. A technique we could be improving on is seeing if CNN models can maintain performance whilst minimizing input data. We believe this could be a good preliminary study on performance vs. training sample data.

4 Experiment Overview

The experiment that we conducted is using 2 CNN models with different architectures. Having two models will just test if one architecture is better than another with testing. We then take the CIFAR10 dataset images and train the models with the original image as a control. We record the training loss and accuracy as well as testing loss and accuracy. Then, we use PCA to reduce the data further to 3.125%, 6.125%, 12.5%, and 18.75% of the original and train both models with the same hyper-parameters for consistency. We record the training and testing loss and accuracy and plot the data and compare to see if the models are able to learn a reduced image.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

5 Related Research

We found an interesting paper, "Image retrieval method based on CNN and dimension reduction" (https://arxiv.org/pdf/1901.03924.pdf) that also conducts a study on this topic. They found that while CNN's can still have satisfactory performance, further practical and theoretical studies should be conducted to prove the viability and best parameters.

6 Experiments

6.1 Image Compression Comparison

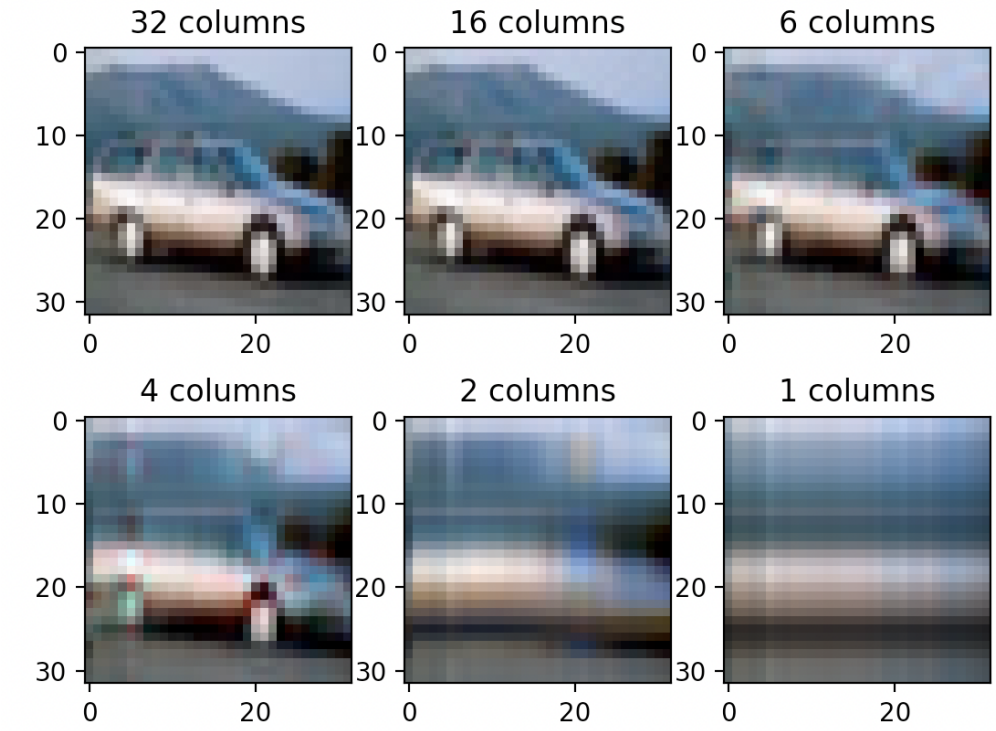

Figure 1: Comparison of Feature Reduced CIFAR10 Images. 32 Columns is the Full Size Image

Above is a comparison of a CIFAR10 image at different compressions from PCA. We noticed that the difference between 32 and 16 features was very similar, so we chose to compare the performance between 32 (100% of the full image), 6(18.75% of the full image), 4(12.5% of the full image), 2(6.125% of the full image) and 1(3.125% of the full image) feature reduction due to the similarity between 32 and 16 principal components.

6.2 Model Architectures

We chose to create our CNNs using Python and the keras library from tensorflow. Keras allows for intuitive model creation, as well as historical data collection for later plotting. As mentioned above, we created two models. One model was smaller with the other being larger. The larger model also used dropout layers, which randomly sets nodes to 0 when outputting so that the model is less prone to overfitting. Figure 2 and 3 are summaries of our model architectures. The larger model utilizes 8 convolutional layers while the smaller model only utilizes two. We expect for the larger model to perform much better than the smaller model.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None, None, 3)]   0

 conv2d (Conv2D)             (None, None, None, 32)    416

 max_pooling2d (MaxPooling2D  (None, None, None, 32)   0
 )

 conv2d_1 (Conv2D)           (None, None, None, 64)    8256

 max_pooling2d_1 (MaxPooling  (None, None, None, 64)   0
 2D)

 conv2d_2 (Conv2D)           (None, None, None, 128)   32896

 max_pooling2d_2 (MaxPooling  (None, None, None, 128)  0
 2D)

 global_average_pooling2d (G  (None, 128)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               33024

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 77,162
Trainable params: 77,162
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Smaller Model Summary

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_4 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_3 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_1 (Dropout)         (None, 16, 16, 64)        0

 conv2d_6 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_7 (Conv2D)           (None, 8, 8, 128)         73856

 dropout_2 (Dropout)         (None, 8, 8, 128)         0

 conv2d_8 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten (Flatten)           (None, 2048)              0

 dropout_3 (Dropout)         (None, 2048)              0

 dense_2 (Dense)             (None, 256)               524544

 dropout_4 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 10)                2570

=================================================================
Total params: 814,122
Trainable params: 814,122
Non-trainable params: 0
```

Figure 3: Larger Model Summary

6.3 Dataset Creation

To create the 6 (one for each compression that we are doing and the control dataset) different datasets that we need, we made use of python's numpy library that has an SVD function. Similar to the image compression problem in homework set 2, we separated each image by its red, green, and blue matrices and used SVD on each matrix. We then took the n features that we wanted from the U matrix that was generated by SVD and put the image back together. After the image was put back together, we add it to a list of images that will then be split up between training and testing. The labels are not changed in any way.

6.4 Model Training

To test the performance of our model architectures, we trained them using Clemson's Palmetto HPC. In total, we trained 12 models, a small and large model for each dataset that we created. For our hyperparameters, we trained for 200 epochs, a batch size of 32, and a learning rate of .001. Additionally, we chose to use the Adam optimizer, as it is very common and a gold standard for many model types in deep learning. We used keras' sparse cross-entropy loss for the multi-classification problem that we are faced with.

7 Results

Based on our results, we confirm our hypothesis that the larger model performs better than the smaller model. As seen in the test loss for the small model, regardless of the amount of features utilized, there is a global minimum of loss and then the loss starts to diverge from the x-axis. The small model begins to overfit the data after about 15 epochs, and we learn that the small model might need to be trained for less time. The large model does not overfit the data, and we believe this to be the case because of the dropout layers in the architecture as well as the larger model most likely learning slower than the small model architecture. We also think that the larger model may need to be trained for longer due to a lack of convergence in the loss and accuracy. The testing loss in the larger model does not converge and smooth out at a certain epoch, so we believe that training longer might help with the convergence of the model. Another approach to help the convergence would be to lower the learning rate to a value such as .0001, so that the larger model learns even slower.

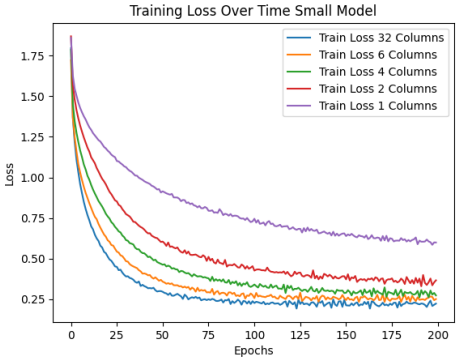

Figure 4: Smaller Model Training Loss

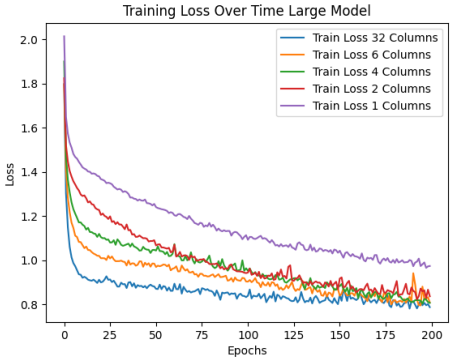

Figure 5: Larger Model Training Loss

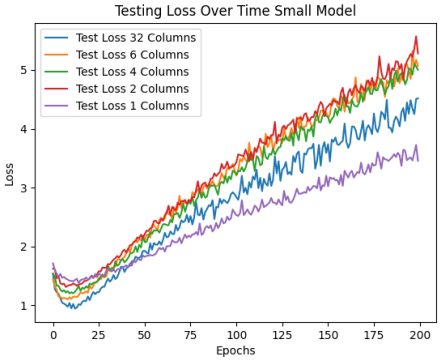

Figure 6: Smaller Model Validation Loss

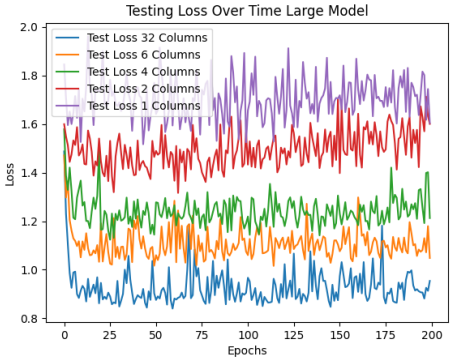

Figure 7: Larger Model Validation Loss

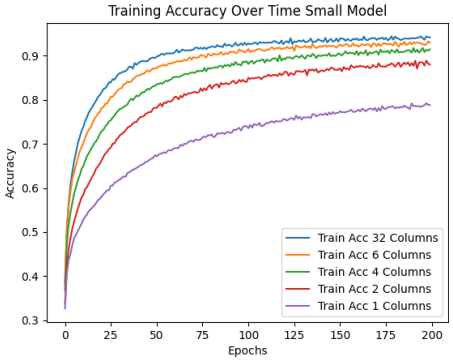

Figure 8: Smaller Model Training Accuracy

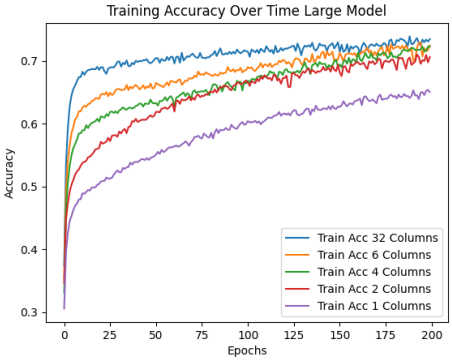

Figure 9: Larger Model Training Accuracy

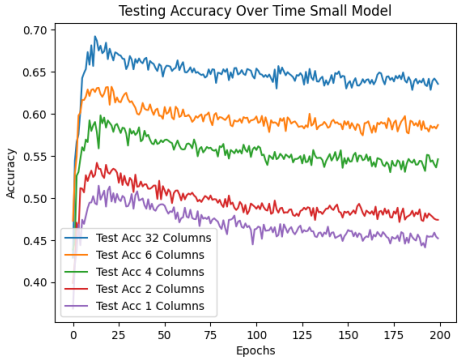

Figure 10: Smaller Model Validation Accuracy

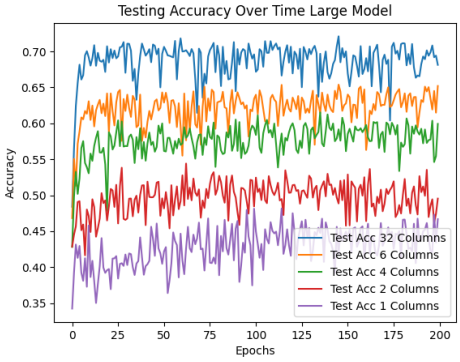

Figure 11: Larger Model Validation Accuracy