Investigating Convolutional Neural Networks using CIFAR-10 Image Set

Ricardo H. Sedano

UC San Diego

COGS 181: Neural Networks and Deep Learning

Zhuowen Tu

Mar 22, 2024

Abstract

As seen throughout the homework and course structure, we've seen Convolutional Neural Networks (CNNs) revolutionize image recognition tasks, demonstrating remarkable performance in various domains. In this study, I present an extensive experimental investigation into the impact of architectural variations and optimization strategies on the performance of CNN models for image recognition tasks. I will systematically vary the number of layers, types of activation functions, pooling methods, and optimization algorithms to explore their influence on model accuracy and computational efficiency. Through extensive experimentation and performance evaluation, I'll ultimately analyze the trade-offs between model complexity, computational resources, and recognition accuracy. Finally, the findings provide valuable insights into the design and optimization of CNN architectures for image recognition tasks. The results can also inform the development of more efficient and accurate CNN models for real-world applications in computer vision, object detection, and pattern recognition.

Investigating Convolutional Neural Networks using CIFAR-10 Image Set

Introduction

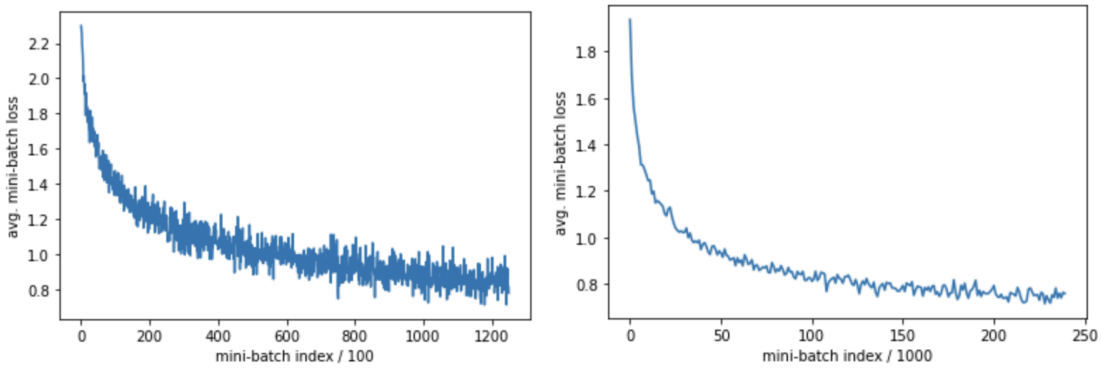
In recent years, image recognition has progressed in significant advancements, particularly, with the widespread adoption of CNN models. These models have demonstrated remarkable capabilities in various applications such as object detection, facial recognition, and medical image analysis. However, there remains a need for further exploration and optimization of CNN architectures to enhance their performance across different datasets and tasks. This research aims to address this gap by conducting an extensive experimentation process that explores the impact of varying CNN architectures on image recognition tasks. Specifically, I will investigate the effects of altering the number of layers, types of activation and pooling functions, as well as different optimization techniques. By analyzing and altering these factors, I'll seek to identify the configurations that yield the most robust and accurate image recognition models. This paper presents the methodology and experimental setup for our study, followed by a detailed analysis of the results and implications for future research and application development.

The basis of the research is derived from the comparison and optimization of the CIFAR-10 image set from Assignment #4, Questions 5 and 6. The task had been to train a CNN with the provided skeleton code, followed by improving the performance of the initially trained CNN. For the experimental portion of this project, we will also report on the same image set for ensured continuity. This allows for us to have a "control sample" of sorts throughout various training transformations, as we can discover combinations of layer numbers, activation and pooling functions, and optimization techniques that either improve or diverge from the original performance.

Method

Prior Knowledge & Initial Process

Based of the performance of the models from Assignment #4, we know that adding an additional convolutional layer as well as another average pooling layer has helped the CNN performance. Particularly with stability of the loss function in the higher iterations of the network (seen below; Question 5 on left, Question 6 on right), it seems apparent that adding more layers benefited the performance of the network. Additionally, complexity in the form of image transformations, like rotation and flipping, assisted in the recognition of the images in the latter network. Last of the differences, the updated network also replaced Stochastic Gradient Descent with Adam. Based on these initial notions, I hypothesize that adding additional convolutional layers will correlate positively with the network optimization and performance. It's also hypothesized if switching of activation and pooling functions have a similar effect on complexity like the image transformations have, then the network may also learn and optimize, thus increasing performance score. It's important to note that for the latter hypothesis, an effect of altering activation and pooling function is compared to an effect of linear image transformation since a change from an activation, pooling, or optimization function to another can also be represented linearly.



Data

The CIFAR-10 dataset consists of 60,000 color images, each sized 32x32 pixels, distributed among 10 distinct categories. Within each category, there are 6,000 individual images. To facilitate training and testing, the dataset is partitioned into five training sets and one test set, each containing 10,000 images. Notably, the test set includes precisely 1,000 randomly chosen images from each category. While the images within the training sets are arranged randomly, it's worth mentioning that certain training sets may exhibit imbalances, with some containing more images from specific categories than others. Despite these variations, collectively, the training sets encompass precisely 5,000 images from each category.

Setup & Model Implementation

For the entirety of the project and previous coding, it will be conducted in UCSD DataHub using a Jupyter Notebook running the CNN in Python using numpy and PyTorch packages. In preparation for our model, the skeleton code provided processes our CIFAR-10 data by converting images to PyTorch tensors and normalizing their pixel values. Additionally, the classes are labeled according to the 10 categories the CIFAR-10 dataset consists of, as mentioned in the Data section. For optimized running within Datahub, and due to slight hardware restrictions, the device was set to ‘cpu’ for the device running the CNN coding.

The actual definition of the network will vary depending on the alteration we commit to the code. Of course, our two functions defined in our network class, `def __init__` and `def forward` mainly are altered from their activation function and pooling function type, as well as varying the number of layers of activation, pooling, and convolution layers are present. On a more complex level, it is imperative that the shape of the layers are correct and updated as more

layers added may induce quicker errors. Last, we will see different optimization functions; the other hyper-parameter to note is the optimizer, yet altered outside of the network class. Having this function outside the network class allows for less room for error interpreting code within the class, so errors can be easier identified if present throughout a particular code cell.

Training

For the sake of the project, the sheer amount of experiments, and the time it takes for my MacBook to compute the code executed throughout the Experiment portion, the epochs in training for each of our combinations will be 15 epochs each, yet the takeaway is the print frequency for each epoch will be 1500. Please note this decision is not intended to sacrifice the integrity or complexity of the Experiment portion of this report. For each of our configurations, we train and report on our loss.

Evaluation Metrics

There will be two direct evaluation metrics as percentages: overall test accuracy and test accuracy for each class. In both instances, the percentage is calculated by dividing the number of correctly predicted images by the number of total images. Additionally, we may factor in a visual evaluation of the loss function and a special sample evaluation of 4 image predictions. It is easier when the model predicts incorrectly as far as visual evaluation, yet if the model is predicting accurately it can be difficult to evaluate the efficacy. For example, evaluating efficacy visually for approaching 100 percent, per se.

Experiment

In order to gauge how well each of the six models is comparatively at predicting images, we must start off with a "control sample" of sorts to base off of whether or not improvement has been made or not. I started on a model with 3 convolutional layers, using the average pooling and ReLU activation function and batch normalization. For the optimization function, this model will use stochastic gradient descent. First, I wanted to test the number of layers, so I decreased the model to 2 convolutional layers, keeping the rest of the configuration. Between the two, we saw a decrease from an overall test accuracy of 72% to 69%. Categorically, the lowest accuracy for the 3-layer model was 58% for *cat,* while the 2-layer model was lower at 43% for *bird*.

Test Accuracy (%) per Model per Overall/Category

Model	*Overall*	*plane*	*car*	*bird*	*cat*	*deer*	*dog*	*frog*	*horse*	*ship*	*truck*
3CNN, AvgPool, ReLU, Batch Normalization, SG Descent	72	68	86	64	58	61	59	75	77	89	81
2CNN, AvgPool, ReLU, Batch Normalization, SG Descent	69	77	77	43	48	68	62	81	77	74	84
3CNN, MaxPool, ReLU, Batch Normalization, Adam	70	74	87	65	53	60	66	77	69	77	77
3CNN, AvgPool, Sigmoid, Batch Normalization,	10	0	0	0	0	0	0	100	0	0	0

SG Descent											
3CNN, AvgPool, ReLU, No Batch Normalization, Adam	73	78	83	68	50	69	67	78	74	86	80
3CNN, AvgPool, ReLU, Batch Normalization, Adam	72	73	75	54	57	76	62	78	78	82	89

Moving on to comparing pooling and activation functions, I changed from ReLU to the Sigmoid activation function, and the result was disappointing for the model. It's assumed that the sigmoid function "forced" the model to predict frog based on only a very small number of predictions and got stuck. So, the overall test accuracy is only 10 percent. When changed from the AvgPool to the MaxPool, the performance went down overall, with only one category reaching accuracy above 80 percent. The overall test accuracy was 70 percent, which is about as accurate as the rest of the models, yet nonetheless is one the lower side for accuracy across the board.

Our highest performing combination happened when we took away the batch normalization I added from Homework 4. After removing the batch normalizations from the three layers, our model improved slightly overall, yet seemingly in a few ways categorically. The overall testing accuracy is 73%, yet we see a 10 percent difference in *plane.* Perhaps though it is balanced from decreasing in *ship* and *truck.*

The final combination substitutes the optimization function of Stochastic Gradient Descent to Adam. Luckily this combination was already performed on Homework 4, Question 6,

thus the data is from that model. We see no change in the overall test accuracy as it says at 72%, yet with the model using Adam, we see less categorical test accuracy consisting of less than 60 percent. I would say this is actually an improvement over the model with SGD since there is more consistency within the categories, and the overall test accuracy is then more representative of the test accuracy throughout the project.

Conclusion

In conclusion, this study embarked on an extensive exploration of Convolutional Neural Networks (CNNs) for image recognition tasks, with a particular focus on architectural variations and optimization strategies. Through systematic experimentation, we investigated the impact of altering the number of layers, types of activation and pooling functions, and optimization algorithms on model performance. Leveraging the CIFAR-10 image dataset, our analysis revealed intriguing insights into the interplay between model complexity, computational efficiency, and recognition accuracy.

Our findings underscored the critical role of architectural configurations in determining CNN performance. Notably, we observed that increasing the number of convolutional layers generally correlated positively with network optimization and performance, albeit with some nuances across different activation and pooling functions. Furthermore, the removal of batch normalization layers resulted in modest performance improvements, suggesting the potential for streamlined architectures in certain contexts.

Interestingly, our experimentation with optimization algorithms demonstrated nuanced effects on model performance. While transitioning from Stochastic Gradient Descent (SGD) to

Adam optimization yielded no significant change in overall test accuracy, it led to improved consistency within categorical test accuracies.

Overall, this study contributes valuable insights into the design and optimization of CNN architectures for image recognition tasks. By elucidating the trade-offs between architectural complexity, optimization techniques, and recognition accuracy, our findings provide practical guidance for the development of more efficient and accurate CNN models in real-world applications such as computer vision, object detection, and pattern recognition. Moving forward, future research endeavors may delve deeper into fine-tuning architectural configurations and optimization strategies to further enhance CNN performance across diverse datasets and applications.

References

Brownlee, J. (2020, August 28). How to Develop a CNN From Scratch for CIFAR-10 Photo Classification.https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

Final Project Requirements (question @221). (2024). https://piazza.com/class/lr6no75x201390/post/221

Homework Assignment #4. COGS 181:Neural Networks and Deep Learning.

Krizhevsky, A. (n.d.). CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html

reusing skeleton code from homework (question @248). (2024). https://piazza.com/class/lr6no75x201390/post/248