A Comparative Analysis of Multiple Convolutional Neural Network Architectures

COGS 181

Ahsan Bari, A13865703

Abstract

In this paper I will determine the efficacy of four different premade convolutional neural networks (CNN). The four architectures are VGG, GoogLeNet, ResNet, and AlexNet. In order to determine the effectiveness of each architecture, they will all be tasked with sorting the Tiny ImageNet database consisting of multiple different types of photos each containing a different image within multiple types of subsets.

Introduction

In order to consistently compare the four architectures, they will all be using the same hyperparameters and image sizes. The percent metric will be observed by comparing the number of correct classifications compared to the number of total classifications. Furthermore, in order to make sure there are enough trials within the classifier training, there will be 11 epochs for each sample each measured at iter = 100n. The dataset being used has 200 different types of photos with unique objects within the training set. Of the 200 different objects, there are 500 different iterations of each object i.e., one of the 200 categories is mountains and there are 500 mountain pictures in that section. There is another validation set which contains 10000 different images each containing the same sections as the training set. Lastly in the test category there are an additional 10000 pictures that are unique from both the training set and the validation set.

The different classifiers being used were provided through the PyTorch Vision GitHub repository, the repository includes other multiple CNN not being used. They vary in their hyperparameters and their core engine.

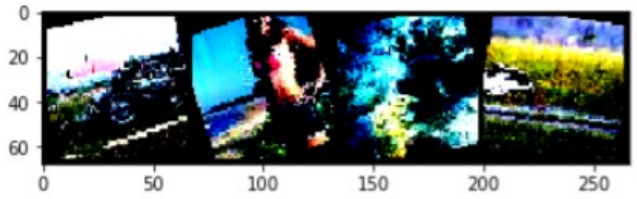
The first of the CNN, GoogLeNet, is a deep convolutional neural network that is based on . An architecture known as “Inception”. According to the PyTorch documentation, a native Inception module relies on 3 hidden convolution layers and one max pooling layer. Through continuous layers of the Inception module, the GoogLeNet network does a forward feed propagation. The next CNN used was VGG. VGG uses an incremental increase of hidden convolutional layers combined with a ReLu layer which eventually ends in a max pooling layer to create a dense output (SoftMax). The next CNN used was AlexNet. AlexNet is one of the more recent CNNs and it utilizes a ReLu unit rather than a tanh function. Additionally the AlexNet architecture uses an overlap in its max pooling layers in order to reduce error in overfitting. As a result, theoretically AlexNet is supposed to have to highest classification accuracy as opposed to its equally trained counterpart architectures. Lastly, a ResNet architecture will be used to classify the Tiny ImageNet database. ResNet is the architecture that preceded AlexNet. The two both rely upon a multi-hidden layer network that involves a ReLu layer before pooling. ResNet is unique in that as forward and backward propagation occur, some layers are skipped to avoid an issue of “vanishing gradients”. As a result a layer’s weight can be carried into multiple different hidden layers before approaching a max pooling layer; therefore, a more accurate estimate can be made. Consequently, the processing time for ResNet can take much longer than other CNN architectures because it skips over layers and goes over them again.

Methods

The hyperparameters for each of the architectures was determined by the given sample code provided in the rubric. The normalization factor was [0.4802, 0.4481, 0.3975] , [0.2302, 0.2265, 0.2262], the images were also sorted in a random way and shifted in a random angle to increase training sensitivity. The images were then labeled with their respective label and added to a tuple object. After the neural network was declared, the forward and backward propagation processes were initiated. The parameters were then optimized using the PyTorch optimization function. The loss was then recorded and added to a list that tracked the losses per 100 sample epochs.

Results

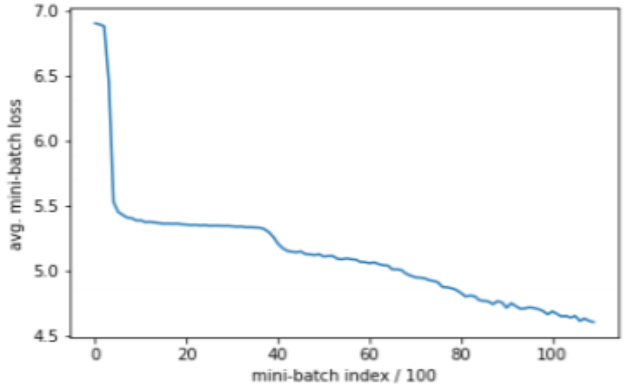
Before the images are classified through the CNN classifiers, the images are presented in random, distorted way.



Each neural network ran for 11 epochs iterating 10 times each. And their results are listed as follows.

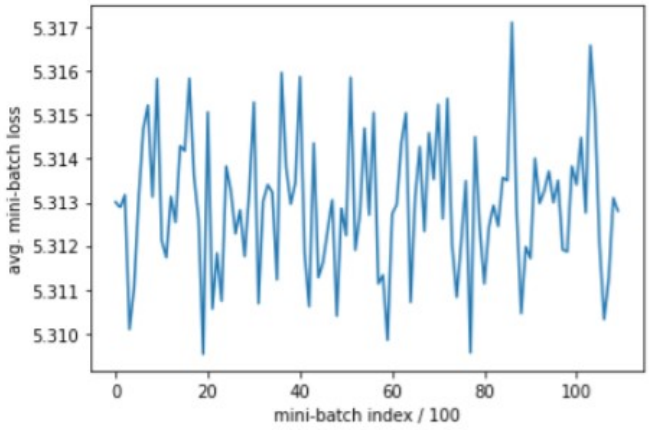
AlexNet:

The loss function for the AlexNet displayed a consistent decrease over the iterations. The total network classification accuracy came out to 15% over 10000 images.



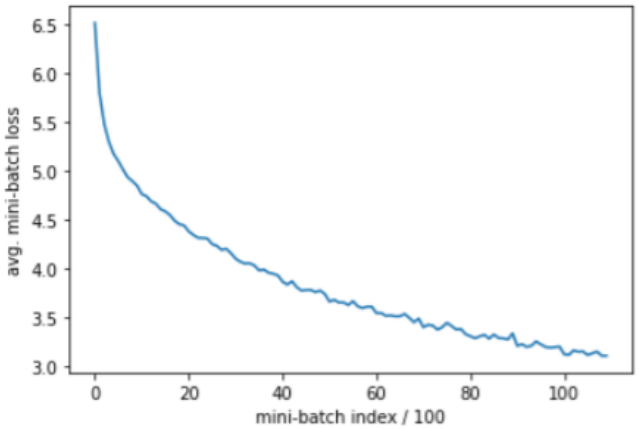
GoogLeNet:

The loss function for the GoogLeNet appeared to oscillate around a single point over the iterations. The total network classification accuracy came out to 17% over 10000 images.



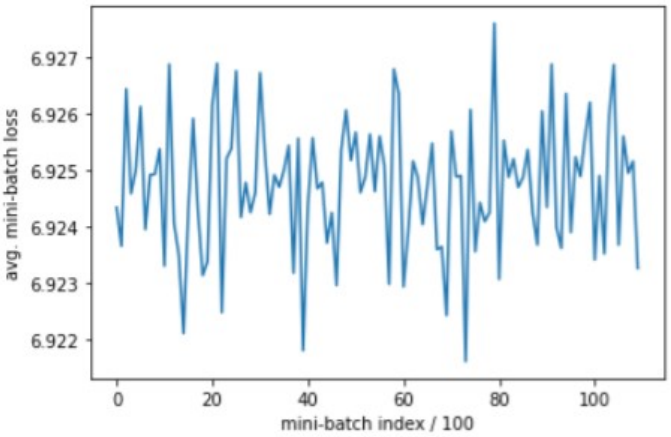
ResNet:

The loss function for the ResNet displayed a consistent decrease over the iterations. The total network classification accuracy came out to 20% over 10000 images.



VGG:

The loss function for the VGG appeared to oscillate around a single point over the iterations. The total network classification accuracy came out to 23% over 10000 images.



Discussion

The VGG was the most accurate classifier for the Tiny ImageNet database. That being said, it was also one of the most computationally demanding. The VGG script ran for an hour to complete, the next highest, ResNet at 20% accuracy completed in less than half the time of the VGG. ResNet completed within 15 minutes and classified the dataset to 20% test accuracy. The ResNet script is an architecture that stems from AlexNet.

Being a newer architecture it is possible that it is more suited to classify a relatively large image database. The skipping of layers may have also contributed to the efficiency because the images were randomly skewed and distorted. Skipping may have broadened the scope of the classifier. Some difficulty with this project stemmed from the size of the database. The runtimes for some of the script went into hours to compile and execute the code. A possible solution for this could be to run this on a more efficient system with a more stable connection to the Jupyter server.

References

Homework 5 and 6

https://github.com/pytorch/vision

https://stackoverflow.com/questions/46972225/how-to-open-local-file-on-jupyter

https://pytorch.org/hub/pytorch_vision_googlenet/

https://neurohive.io/en/popular-networks/vgg16/

https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951

https://en.wikipedia.org/wiki/Residual_neural_network

https://github.com/tjmoon0104/pytorch-tiny-imagenet