ImageNet image classification challenge comparison between modified LeNet and VGGNet

Rui Ren, Wan Hin Choi

cogs 181 spring 2019 Instructor: ZhuowenTu

University of California, San Diego

ruren@ucsd.edu

***Abstract* - Convolutional Neural Networks have always been the most popular neural network for image classification field. LeNet founded the popularity of CNN made us use LeNet-5 through this project. Shared similarity between LeNet and VGGNet is use of convolutional layers, pooling function, and activation function, we decided to further develop those network and make comparisons by solving same ImageNet micro image classification challenge.**

Index Terms - CNN, VGG, LeNet, ImageNet micro.

I. Introduction

Convolutional Neural Network have been used successfully in image classification. Convolutional Neural Network uses convolution which is one kind of linear function. It takes convolution instead of general matrix multiplication in at least one of the layers. [2] There are three main parts in a convolutional neural network which is convolution layers, pooling layers, and activation function. Convolution takes the split image input and breaks into different features, pooling changes the size of different features and activation function converts different features to an output.

LeNet is a convolutional neural network that was created by LeCun in 1998 which is the foundation of all the convolutional neural networks. It is a seven layer neural network that was first used to recognize hand-written digits. In LeNet, it takes the input. Then, we split that into different parts which is called features and run the first convolution layer. After that, we go through the max-pooling and goes to the second convolution layer and max-pooling. Finally we use ReLU and soft-max to get the output after the second pooling layer.

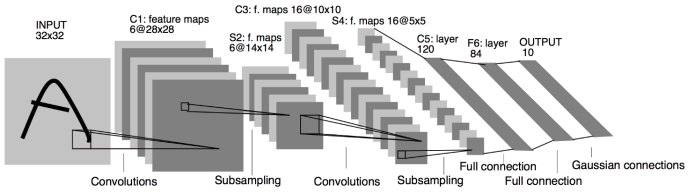

Figure 1: LeNet Architecture Image is From : https://medium.com/@pechyonkin/key-deep-learning-architectures-lenet-5-6fc3c59e6f4

Going into VGGNet, it was created by Simonyan and Zisserman in 2014. One of the features is that It contains 3x3 convolution and a lot of filters with max-pooling layer after convolution layer. And VGGNet becomes the most popular convolutional neural network feature for large image dataset. VGGNet can have both sixteen layers or nineteen layers. [1]

Given that VGG was the runner up in ILSVRC(ImageNet Large Scale Visualization Recognition Competition) 2014, it outperforms a lot other past convolutional neural network architectures. In VGG-16, we separate it into five parts. The first two parts both contain two 3x3 convolution layers and one max pooling layer. The last three parts contain three 3x3 convolution layers and one max pooling layer. And then it goes through activation function with ReLU and softmax activation function. [1]

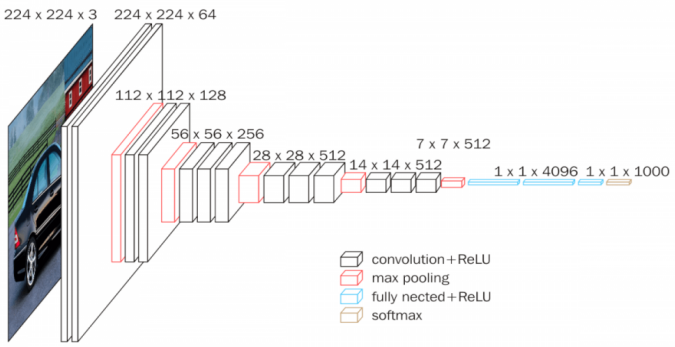

Figure 2: VGGNet Architecture. Image is from: https://neurohive.io/en/popular-networks/vgg16/

The tiny ImageNet dataset contains 100000 64x64 images with 200 different types of color images. With 80000 images in the training data set and 10000 in both testing and validation set.

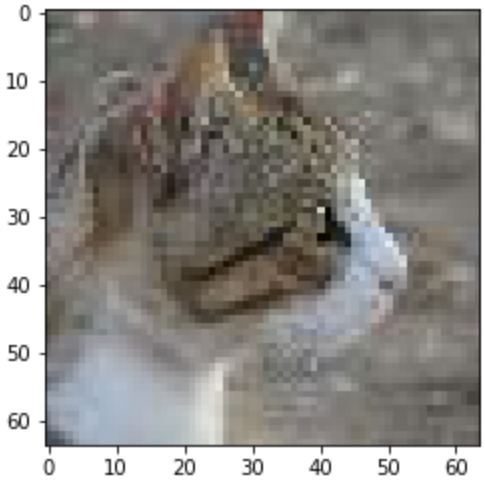

Figure 3: Cat

Figure 4: Race Car

Adam (adaptive moment estimation) Optimizer is an algorithm for first-order gradient-based optimization of stochastic objective functions. The method is used in a lot of neural networks. Since it only requires little memory, it has been used a lot in large data set. Adam Optimizer also solves the dataset problems which are very noisy and/or sparse gradients. [3]

SGD only uses one batch at a time per iteration. However after it goes through every iteration, SGD works well but it is very noisy. The term "stochastic" means that the one example for each batch is chosen at random. And then gradient descent is the most common method that is used to optimize deep learning networks. It was first proposed in the 1950s by Terry Sejnowski. Gradient descent can update each parameter of a model at every iteration, find out the rate of change and how a change would affect the objective function, pick a direction that would lower the error rate, and continue iterating until the objective function converges to the minimum on the function. [4]

In activation, there are linear and nonlinear. We do not use linear function in neural network because it does not help in complex or various parameters. ReLU, tanh and sigmoid are non-linear. In sigmoid, it goes like an S shape from 0 to 1. Tanh goes from -1 to 1 and have similar shape to sigmoid. Y goes negative as x is below zero Zeros stays at zero, and positive x goes with positive y. While ReLU, y is 0 for x that is less than or equal to zero. For above 0, y goes linear.

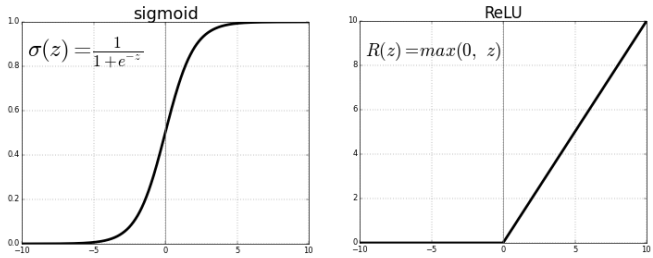

Figure 5: Activation Function Image is from: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

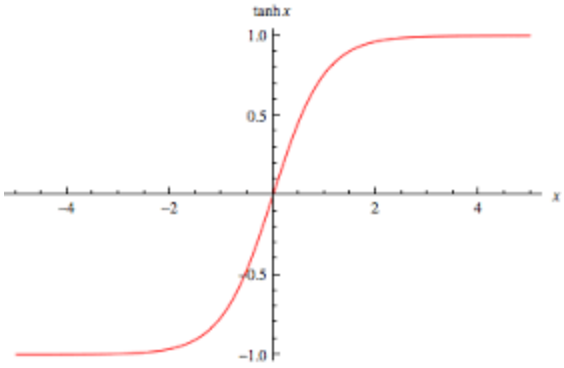

Figure 6: Tanh function. Image is from: http://mathworld.wolfram.com/HyperbolicTangent.html

II. Method and Architecture

We used keras-gpu and tensorflow backend to generate all the networks throughout this project. Making sure we are implementing both LeNet and VGG architectures as authentic as possible to their original authors. Then we tried to depart from what was given from the original paper, constructing networks by various combinations of different aspects. As instructed, changing hyper parameters and structures include: a) varying number of layers. b) adopting different optimization methods, (Adam vs. stochastic gradient descent). c) different pooling function, (average pooling vs. max pooling). and d) different activation functions, (ReLu vs. sigmoid, and tanh).

As mentioned above LeNet-5 at first its proposal was trained and optimized specifically to MNIST, hand witten image dataset contains 28*28 images. and VGG-16 was

designed to adopt 224*224 images assigned by ImageNet competition. Minimum adjustment has been made to both networks adopting our task of training on tiny imageNet dataset which uses only 64*64 images. imageNet micro raining dataset were divided into 200 categories and each category has 500 variation of image. All of our networks are all trained sequentially through 200*500, equally 100000 images at a batch size of 64. Exceptions are of section that implemented PCA and VAE, where there were too many images to process as a whole, so we need to modify the training dataset slightly for training. We used validation dataset as it is with validation annotation text provided, generally there are 10000 images contains all 200 categories distributed randomly that appeared in training dataset.

We defined original LeNet and VGG as our baseline, then compared their performance upon each modification as checkpoints. Each checkpoint we trained 10 and 30 epochs respectively and markdown the training and testing accuracy. The reason behind 10 and 30 epoch on single network separately is because we found that sometimes networks will have sharp change of behavior, for example sudden increase of accuracy or a decrease of loss, after within or around training at 10 epoch. (fig. 5) Training up to 30 epochs was not only because the sake of time, we have so many variations for just one architecture, and they need to train separately, but we shall be confident enough that learning rate are somewhat predictable after 30 epochs.

III. EXPERIMENT

LeNet was implemented 16 years ahead of VGGNet they are very different networks. Howeverm VGGNet is much deeper than LeNet, but has homogeneous layers compositions such as 3 x 3 convolutional layer, adopted relu activation function. And 2 x 2 max pooling layer after convolutional layers. After flatten the output from convolutional layers follows fully connected layers. All hidden layer are equipped with the rectification non-linearity.

Aside from their depth, we established the baseline using uniformed 3 x 3 convolutional layers and with relu activation function, followed 2 x 2 max pooling layer and fully connected layer with relu rectifier, then using Adam as optimizer.

Author in the original paper used stochastic gradient descent instead of Adam, so we compared SGD and adam. For SGD, we tried different learning rate, momentum, and either with or without checking nesterov method. Within SGD we've tried a) learning rate equals 1e-5, without nesterov and momentum. b) a learning rate of 0.1, with 0.9 momentum nad 1e-6 of decay, this is what the author used.

Next section, we evaluate the outcome after replacing each occurrence of max pooling to average pooling.

We also tried different activation function. original VGG used ReLu after each hidden layer. Replacing ReLu to sigmoid, we expected very different outcome because the difference between how they punish wrong predictions. ReLu is more often used my deep neural networks, but sigmoid on a deeper neural network was suspected for vanishing gradient. We also test LeNet with its legacy activation function tanh in this section.

At last section, We tested PCA and VAE as data-preprocessor.

IV. RESULTS

LeNet with adam optimizer training accuracy of 43.74% and test accuracy of 15.06%. Training loss was 1.3457, and test loss was 5.6627. VGG with adam optimizer had training accuracy of 88.03%, and test accuracy of 26.82%. Training loss was 0.3839, test loss value was 5.9903. LeNet with stochastic gradient descent had a training accuracy of 0.65% and test accuracy of 0.67%, training loss value of 5.3046, testing loss value 5.3055. VGG with SGD without momentum, had 0.49% and 0.43% training and testing accuracy respectively. The original VGG had 0.51%, 0.5% of training and testing accuracy respectively, and 5.3 for both training and testing loss values.

Examined variation with different pooling function. Lenet with average pooling had 43.47% and 17.85% training and test accuracy, 2.5 and 4.25 training and testing loss value. VGG with average pooling had 87.34% and 25.4% of training and testing accuracy, 5.29 for both train test loss values.

Tested with different activation functions. LeNet with sigmoid had 8.3% and 7.8% training and testing accuracy, 4.5 for both train and test loss values. Original LeNet which uses tanh function had 54.86% and 23.95% training and testing accuracy, 2.01 and 3.54 training and testing loss values respectively. VGG with sigmoid had 0.49% and 0.5% of training and testing accuracy, 5.29 for both training and testing loss values. The following table has better visualized comparison of those results.

Training Results
COMPARISON

Network.	Network	Train Accuracy	Test Accuracy	Train Loss	Test Loss
1	LeNet - adam baseline	43.74%	15.06%	1.3457	5.6627
	VGG - adam baseline	88.03%	26.82%	0.3839	5.9903
2	LeNet - SGD	0.65%	0.67%	5.3046	5.3055
	VGG w/o momentum	0.49%	0.43%	3.3479	3.3481
	VGG - SGD nesterov original	0.51%	0.50%	5.3029	5.3015
3	LeNet w/ avg pooling	43.47%	17.85%	2.4907	4.2511
	VGG w/ avg pooling	87.34%	25.4%	5.2988	5.2985
4	LeNet sigmoid	8.38%	7.82%	4.5391	4.5561
	LeNet tanh original	54.86%	23.95%	2.0177	3.5404
	VGG sigmoid	0.49%	0.5%	5.2984	5.2983
5	hw5 CNN network	80.97%	14.76%	0.9574	5.2954
6	PCA	6.75%	0.55%	15.0301	16.018
	VAE	19.76%	1.83%	2.7462	12.725

[a]epoch=30, batch size=64

VGG aka “Very Deep Convolutional Networks” was known for its large memory consumption and painful slowness when come to training. Tell form result we’ve obtained, VGG outperforms LeNet at all around variations. Using adam as optimizer yields best test and train accuracy throughout this project, adam runs much faster than SGD additionally. Outcome from network implemented stochastic gradient descent was dismal. We spend most of the time training and optimizing with SGD but never achieved anything better than random guessing. We tried both momentum and without momentum, also adjusting learning rate, even more we adopted learning rate and nesterov parameters directly from the original paper in VGG case. We might have tried normalizing the data or more learning rate settings next time.

Replacing max pooling by average pooling was not helpful at all. We always found decreased accuracy and increase loss when we switched to average pooling.

As expected that replacing relu to sigmoid activation function will deteriorate deep neural network, in our case VGG. Vanishing gradient descent could be a problem here for VGG as backpropagation becomes longer the derivation becomes smaller. Even though LeNet with sigmoid function didn’t yield any better result than its ReLu counterpart. tanh, activation function used in its original place performed quite well which almost reach the best accuracy throughout the testing accuracy. (23.95% for LeNet tanh, and 26.82% overall best VGG with adam) This might be caused by steeper slope of tanh function and its negative feedback capability which sigmoid and ReLu don’t have.

In this project, original LeNet and original VGG didn’t guaranteed for optimal training and testing results. Even though, They were excellent networks training and testing their original problem, but performed less desired at our imageNet micro challenge. We found overfitting to training dataset has been a serious problem throughout our networks. Because of lack of time, we only trained each network up to 30 epoch. 30 epochs were less than enough and could be a major reason kept us from obtaining better results. We observed that accuracy increased consistently towards until the end of training. (e.g. fig. 5 and fig.6)

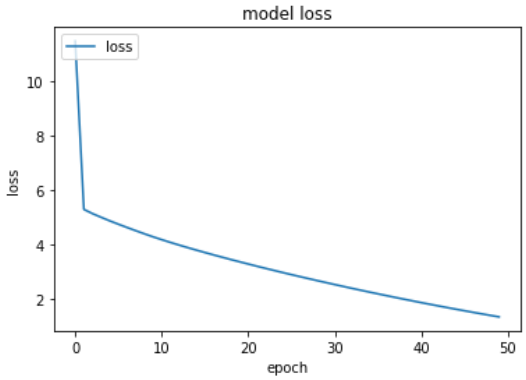

Fig. 5 training loss value curve for LeNet
w/ adam optimizer,max_pooling, epoch=50
jupyter notebook attached to gradescope

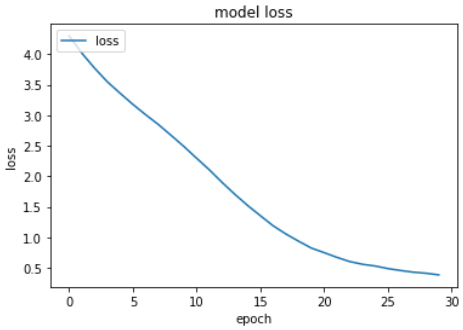

Fig. 6 training loss value curve from VGGNet
w/ adam optimizer, max_pooling, epoch=30
jupyter notebook attached to gradescope

V. Conclusion

VGG and LeNet has their own strength and shortage. VGG had better training and test accuracy than LeNet throughout this project. VGG and LeNet are very different convolutional networks due to their structure. VGG has its iconical 16 layered 3 x 3 filtered convolutional layer and 2 x 2 max pooling function. LeNet has only 5 layers but had tanh as activation function. Given VGG’s depth makes it extremely memory and time consuming to train. Where LeNet is more efficient on memory and training speed. VGG with adam achieved best result from this project. LeNet with tanh was not far from the best, given its efficiency on training and memory size.

We had failed using stochastic gradient descent in any sense from this project. The accuracy from networks using SGD was always near a random guess. We were suspected either a learning rate doesn’t fit our data parameters or we need done more data pre-processing such as normalizing, VAE, and PCA the data.

Overall, we should have trained for more epochs for each network. We spotted that accuracy was constantly decreasing until the end of the training process, which indicates there was room for improvement. If we had more time to work on this project I will train each network to at least 100 epochs.

VAE and PCA data pre-processing was implemented more shallower than other section. From the data we obtain from those methods there are definitely some potential to improve if used properly with other neural networks.

VI. Acknowledgment

Much thanks to Professor Tu for clear explanations to model parameters and hyperparameters, which makes this project possible.

VII. References

[1] Karen Simonyan, Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”
[2] Goodfellow, Ian, et al. *Deep Learning*. MIT Press, 2016.
[3] Diederik P. Kingma, Jimmy Ba, “Adam: A Method for Stochastic Optimization”
[4] Sebastian Ruder, An overview of gradient descent optimization algorithms”