Exploring the Impact of Activation Functions and Normalization Techniques on VGG-16 Architecture

Kevin Hutchins khutchins2@wisc.edu
Ley Kwan Choo lchoo2@wisc.edu
Paul Brochhausen pbrochhausen@wisc.edu
Zongliang Han zhan226@wisc.edu

Abstract

This paper presents a comparative study of the object detection model VGG-16 using various activation functions and normalization techniques. We investigate the impact of these techniques on the performance of the model in terms of accuracy and speed of training. Our findings provide insights for the best combination of activation functions and normalization techniques to improve the performance of object detection models.

1. Introduction

Normalization refers to the process that transforming the data to be on a similar scale, it usually recenter and rescale our data such that is between 0 and 1 or -1 and 1. Scaled features will have zero mean and unit variance. Normalization can help training of neural networks as the different features are on a similar scale, which helps to stabilize the gradient descent step when using larger learning rates. Models can also converge faster for a given learning rate.

However, normalizing the inputs to the network only affects the first hidden layer. To solve this issue, one of the most widely used methods is Batch Normalization, originally proposed in (Ioffe, 2015), which mainly normalizes the input of each layer of the network, making the distribution follow a standard normal distribution with a mean of 0 and a variance of 1. This ensures that the input range falls within the sensitive range of the activation function, avoiding saturation and preventing gradient vanishing, which speeds up the convergence of the model.

Another method is Layer Normalization. Layer Normalization does not depend on batch size, and it normalizes the inputs of a layer by computing the mean and variance across all features for a single example.

In this paper, we applied 3 different activation functions to each normalization techniques. The functions are ReLU, Tanh and Swish. ReLU and Tanh are well-known, and are standard options for traditional machine learning. Swish is a newer function much like ReLU, which is unbounded above to avoid saturation. It has lower bound which indicate that strong regularization for large negative numbers. Its non-monotonicity property ensures the ability to preserve small negative weights, with a much lower possibility of killing a neuron.

2. Proposed Method

To determine the best combination of activation function and normalization technique, several models were created with every possible combination of activation function and normalization technique. Since we have chosen three activation functions and three normalization techniques, a total of 9 models were created. They were run for 10 epochs, at which point their final accuracy on the test set was recorded. Throughout the training, loss and accuracy on the test and training datasets were recorded after each epoch.

3. Experiments

3.1. Model Architecture

The model architecture chosen was the VGG-16 architecture. The VGG-16 model was proposed by Karen Simonyan and Andrew Zisserman in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition." The CNN was used in the authors' submission for ImageNet. The VGG-16 model architecture is used for image classification and was chosen for its power and simplicity of construction. An image of the model architecture is detailed below. As the network progresses, the parameters are fixed and the depth of the network is increased by adding more convolutional layers. The addition of layers is possible because of the small (3x3) filters in all layers. VGG-16 is easy to create as, before becoming a fully connected network, it follows a simple pattern of layers. One limitation of this model is the many parameters that can lead to exploding gradients, which is dealt with through normalization.

3.2. Dataset

The dataset used was the CIFAR-10 dataset. This dataset contains 60,000 color images of 10 mutualy exclusive classes of objects that are 32 X 32. Those classes being: airplane, automobile, bird, cat, deer, dog, frog, horse,

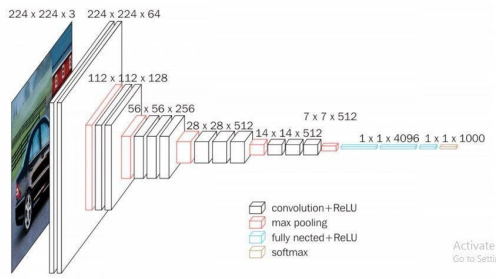

Figure 1. An image of VGG-16 Architecture

ship, and truck. The dataset was split into a set of 50,000 images used for training and 10,000 images used for validation. We chose this dataset for two reasons. First, it is an extremely common dataset used for object recognition and can serve as a recognizable benchmark. Second, the dataset has a smaller image size, which leads to less computation and a smaller training time for the nine models.

3.3. Software

We used Pytorch software to manipulate the dataset and create the model. The model was run on Google Colab GPUs.

3.4. Activation Functions

The activation functions we have chosen are the ReLU function, the Tanh function, and the Swish function. The ReLU and Tanh are very common activation functions and were chosen because of their popularity. The Swish function is a newer activation function, and is a smoothed, continuous version of the ReLU function that creates better gradients. Swish fixes the problem of dead neurons that a function like ReLU has from stochastic gradient descent with the parameters not updating. We chose the Swish function because of its novelty and wanted to compare it to the two standard functions.

ReLU:

$$max(0, x)$$

tanh:

$$\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Swish:

$$x * sigmoid(x)$$

3.5. Normalization Techniques

The normalization techniques we chose were Batch Normalization (BN), Layer Normalization (LN), and having no normalization. BN is the most popular normalization technique and is traditionally used in the VGG-16 model architecture.

However, there is more randomness introduced when BN is being used in a model. The statistics of Mean and Standard Deviation differ for every batch that is sampled randomly which causes the logits to jitter.

Layer Normalization is a newer normalization algorithm that only normalizes across layers which introduces less noise. We wanted to explore if this feature is good enough to replace or beat the performance under Batch Normalization. Also, Layer Normalization was chosen because it has done impressively well in transformers and LLMs. We wanted to see if that performance transferred to CNNs like VGG-16. The third option, no normalization, was done to see the impact of no normalization. As expected, it led to gradient explosion which had the ultimate effect of models without normalization being unable to retain information and learn.

4. Results and Discussion

We performed experiments using the VGG-16 architecture using the 9 different model configurations, with a focus on how stable the training is and how the performance of the model is based on training accuracy, validation accuracy, and training loss. Unless otherwise noted, the default batch size is set to 100 and the learning rate is set to 0.001.

4.1. Normalization

As stated in the introduction above, we experimented with the model with 2 different normalization methods. With no normalization as the control group, the other 2 model was set up with Batch Normalization and Layer Normalization respectively. We set the ReLU as the activation function for all 3 settings. Following are the results that was obtained from our training.

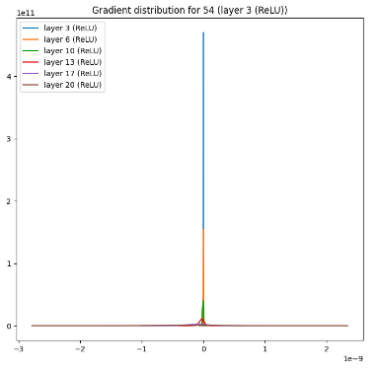

Figure 2. Statistics of Gradient after first 4 layer with No Normalization

The statistics of the gradient diagram tell us how the gradients are controlled after passing through several layers. It gives us a good visualization of how stable the gradient is

across different layers and subsequently indicates how stable the training is.

Figure 4 illustrated how the gradient would vanish across layers without any normalization technique, which is expected.

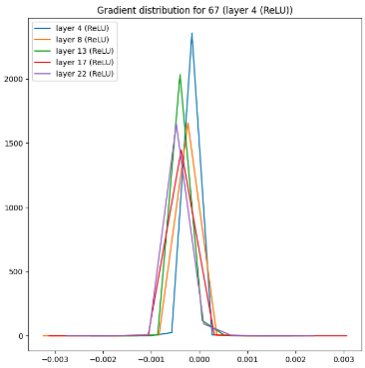

Figure 3. Statistics of Gradient after first 4 layer with Batch Normalization

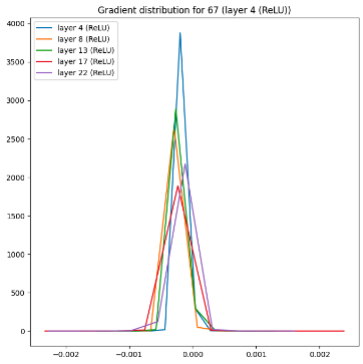

Figure 4. Statistics of Gradient after first 4 layer with Layer Normalization

Based on Figure 5 and Figure 6, it is clear that both Batch Normalization and Layer Normalization methods have very stable gradient statistics across the first 4 layers. Stable gradient statistics indicate that both settings have rather stable training processes, so there is very unlikely chance of us witnessing a vanishing or exploding gradient. This ensures there will be less dead neurons created during training, which makes the training more efficient.

However, Figure 5 and 6 showed that models with Batch Normalization significantly outperformed the other 2 normalization modes. In Figure 5, we could see that the Batch Normalization model converges significantly faster than Layer Normalization model. To explain this scenario, Ioffe & Szegedy (2015) [1] suggested that the Batch Normalization technique can help prevent overfitting during

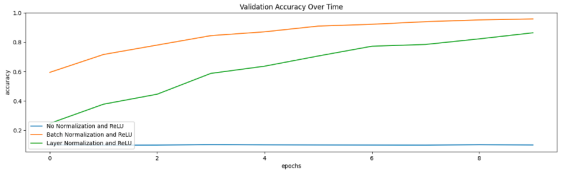

Figure 5. Statistics of Gradient after first 4 layer with Layer Normalization

Figure 6. Statistics of Gradient after first 4 layer with Layer Normalization

training by providing an implicit form of regularization. The randomness that was introduced during the process of batch normalization acted as a data augmentation similar to a weight decay method. Specifically, the inverse of the signal-to-noise ratio can act as the decay factor in this approach (Krogh & Hertz, 1992 [3]; Rifai et al.,2011 [2]). As a result, the Batch-Normalization model has a better generalization compared to the Layer Normalization model.

4.2. Activation Functions

As previously stated, the activation functions we chose to compare were: tanh, ReLU, and swish. In the original VGG-16 model architecture, when the model was introduced, the ReLU activation function was originally used. This may be due to ReLU being common at the time, as other activation functions such as swish not being used until recently.

Figure 7 shows the statistics of the gradient for ReLU, similar to the gradient diagrams used for the normalization results. The gradients across layers appear to be stable for the ReLU activation function.

Figure 8 shows the statistics of the gradient for Tanh. Differing from figure 7 where the gradients were more stable, the lines for layer 8 and layer 17 are much lower than the other layers. So the gradient tends to follow a decreasing/increasing pattern in more layers.

Lastly, figure 9 shows the statistics of the gradient for the swish activation function. Similar to the gradient for ReLU the gradients across layers appear to be stable. This would make sense as ReLU and swish are fairly similar activation functions. With swish being a smoother ReLU.

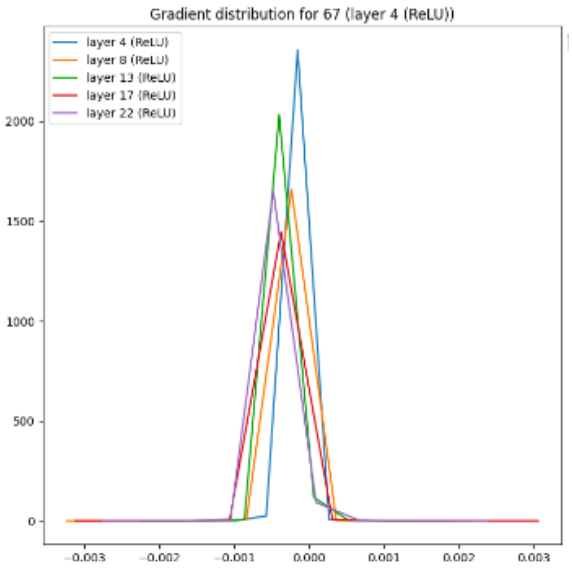

Figure 7. Statistics of Gradient after first 4 layer with ReLU

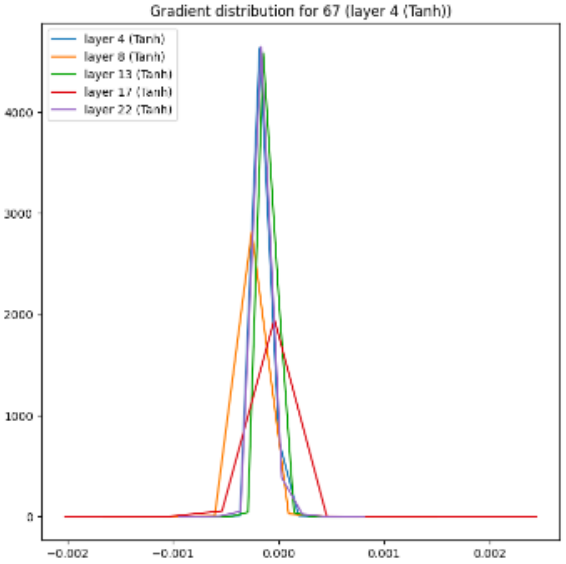

Figure 8. Statistics of Gradient after first 4 layer with tanh

5. Conclusion

In the future, futher experimenting could be done using other models and/or datasets for comparing the activation functions used. As well as for comparing Batch and Layer Normalization. These experiments would be helpful in being able to generalize what activation functions and normalizations are more accurate, or if there are certain scenarios where one is better than another.

Overall, for the VGG-16 architecture normalization is required to deal with exploding gradients, which may be in part due to the large number of parameters required for the model. When we did not use normalization the model failed to learn. This can be seen with the relatively low accuracies when no normalization was used, even when changing the activation functions. Also, when comparing Batch Normalization and Layer Normalization, Batch Normalization was consistently better for this model architecture. When comparing activation functions (ReLU, Tanh, and swish), swish outperformed both ReLU and Tanh, while ReLU was better than Tanh.

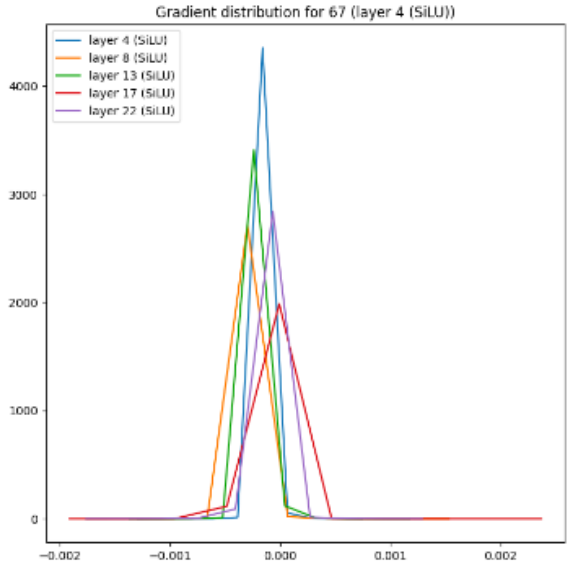

Figure 9. Statistics of Gradient after first 4 layer with swish

6. Contributions

All members split the work and contributed equally.

References

[1] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In ICML*, 2015.

[2] A. Krogh and J. A.Hertz. Generalization in a linear perceptron in the presence of noise. *Journal of Physics A: Mathematical and General*, 25(5):1135, 1992.

[3] Y. B. Salah Rifai, Xavier Glorot and P. Vincent. Adding noise to the input of a model trained with a regularized objective. *arXiv preprint arXiv:1104.3250*, Apr 2011.