DD2424: Deep Learning in Data Science

GROUP PROJECT

Group 12
Lucy McCarren | mccarren@kth.se
Maira Khan | mairak@kth.se
Brynja Þorsteinsdóttir | brynjat@kth.se

May 24, 2022

Abstract

Convolutional Neural Networks (CNN) are commonly used for Image classification. In this paper, we propose numerous methods to build and train a modern ConvNet architecture from scratch. Since the deeper neural networks are more difficult to train, for this, we extend the VGG networks to look more like Residual Network (ResNet) and show that these networks have low complexity and are easier to optimize.

The approaches used in our study includes Weight Decay, Dropout, Data Augmentation, Standardization, Batch Normalization, Cosine Annealing, making VGGs look like ResNet, extensive data augmentation and exploring various Normalization techniques. In this paper, we demonstrate results at 89.16 % by empirically examining CIFAR-10 data. Moreover, the experiments used in the paper are implemented using Tensorflow deep learning packages.

1. Introduction

The default project 1.2 was chosen to build and train a modern ConvNet architecture. This project was selected to explore the concept of training a convolutional network from scratch for CIFAR-10 photo classification, as CNNs are commonly used to classify images. It has a major advantage over its predecessors as it automatically detects important features without human intervention and is also computationally efficient.

The goal was to choose the model giving the highest accuracy, for which various approaches were used including different types of regularization (Dropout, Weight Decay and Data Augmentation), different types of Normalization (Batch, Layer and Instance), Standardization, Adam Optimizer, Cosine Annealing and making VGG to look like ResNet with 3 VGGs as the baseline model.

The full code used to implement our project is available here: Github Project

2. Related work

To build a modern ConvNet [5] architecture from scratch choosing VGG models is a good starting point since they are easy to understand and achieve good performance. VGG was introduced in the 2015 paper "Very Deep Convolutional Networks for Large-Scale Image Recognition", where the 16 layer deep VGG outperformed other networks, including a 19 layer deep VGG network [6].

Shortly after VGG, ResNet was introduced in the paper "Deep Residual Learning for Image Recognition". ResNet is also a convolutional neural network. The concept of "skip connections," is the strength of this network. Skip connections enables ResNet to go deeper then any network had done before, the paper even has an example of 1202 layer deep network on the CIFAR-10 dataset [3].

3. Dataset

An open-source library, Tensorflow package is used to import images from the CIFAR-10 dataset. The dataset contained 60000 32*32 color images in 10 classes including airplane (0), automobile (1), bird (2), cat (3), deer (4), dog (5), frog (6), horse (7), ship (8) and truck (9), where the labels are a list of 10000 numbers ranging between 0 to 9. The dataset is further divided into five training batches and one test batch, each containing 10000 images. Moreover, these images are colored images in RGB format, so every image had a shape (32,32,3), where 3 represents having 3 different channels: red, blue, and green. The dataset was initially normalized by dividing each image by 255 as pixel values range from 0 to 256, to convert it to the range from 0 to 1. This helps convergence faster while training the network.

4. Methods

In order to build this ConvNet architecture from scratch we used Tensorflow Keras and closely followed the steps of the tutorial How to Develop a CNN From Scratch for CIFAR-10 Photo Classification to construct our initial baseline network architecture consisting of 3 VGGs with 73.79 % accuracy. We played around with different combinations of the regularization strategies shown in the tutorial including dropout, weight decay and data augmentation.

- Dropout regularly throughout the network - **accuracy 82.8%**
- Weight decay (L2 regularization) - **accuracy 72.95%**
- Data augmentation - **accuracy 84.04%**

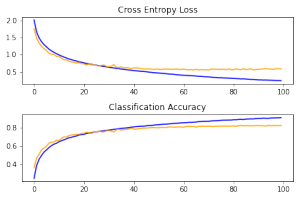


Figure 1: Dropout

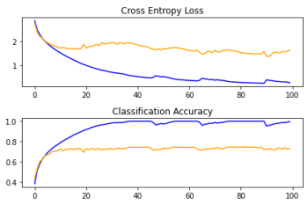


Figure 2: Weight decay

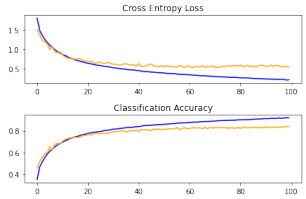


Figure 3: Data augmentation

4.1. Choosing our baseline model

After exploring different combinations of regularization strategies and batch normalization, we obtained the best performance by training the baseline network with Increasing Dropout and Batch Normalization. This gave an accuracy of 84.23%. Hence, we used this as our training model for the parts of the project described hereafter.

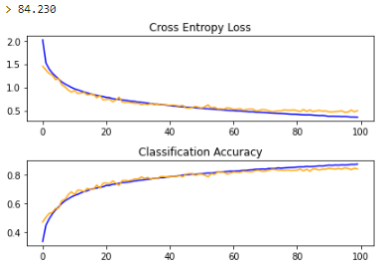


Figure 4: Initial Baseline with Increasing Dropout and Batch Normalization

Note: Although data augmentation increased the accuracy even further, we did not include it in our final model as the computation time for this was too high, so in order to increase efficiency we decided to exclude it for the next steps.

In section 5, we describe some basic experiments that we implemented alongside our baseline model in order to get our final set-up. We experimented with using standardization, and different kinds

of optimizers including SGD and Adam. We also tried out a different learning rate scheduler, and changing the order of the regularization strategies.

In section 6, we explored some further extensions to improve performance. These extensions included making the VGG network look more like a ResNet. We also tried more extensive data-augmentation methods. We implemented different kinds of normalization methods to see if we could maintain performance levels with a simpler normalization practice.

5. Experiments - Basic project

Several approaches were considered for the final setup in order to determine if they improve or degrade the performance. To explore further, the data was standardised, Adam optimizer was used, a different learning rate scheduler was tried and the order of batch normalization/dropout was adjusted to see what effect their changes had on the model performance. The results are presented in the following table and the graphs for each method can be found in the appendix.

Method	Accuracy
Standardizing the data	84.49%
Adam optimizer	87.72%
Cosine Annealing Warm Restarts	62.04%
Changing the order of dropout and batch norm	83.52 %

Table 1: Experiments - Basic Project

Standardizing the data. The data for the training model was standardized by making the standard deviation one and the mean zero. The standardized data boosted the performance to 84.49% and had a faster computation time..

Adam optimizer. Stochastic Gradient Descent (SGD) and momentum optimization were replaced with the Adam optimizer, which is a combination of two gradient descent methods. In addition to making the model more accurate by achieving 87.72 % accuracy, it was also able to train the network in less time, compared to that required to train the initial model.

Different learning rate scheduler. A different learning rate scheduler was implemented with `CosineDecayRestarts` from Tensorflow. The function is based on the paper "SGDR: Stochastic Gradient Descent with Warm Restarts" [4]. The learning rate scheduler was added to the final training model, and the result obtained accuracy up to only 62.04%.

Changing the order of dropout and batch norm. Using batch normalization first followed by dropout resulted in an accuracy of 84.23%. In contrast, applying dropout first followed by batch normalization resulted in a slightly reduced accuracy of 83.52%. Changing the order does not seem to have much effect on the accuracy.

5.1. The Final Set-up

Although, highest accuracy was achieved with the combination of Baseline with increasing Dropout, Batch Normalization, and Adam Optimizer around 87.72%, but we decided to add Standardization as well in the final setup as it speeds up the computational time. Therefore, A total of 87.10 % accuracy was achieved by combining the VGGs Baseline with increasing Dropout, Batch Normalization, Standardization and Adam Optimizer since all of these methods led to an improvement in performance and we used this final setup model for section 6 where we applied further extensions to the project.

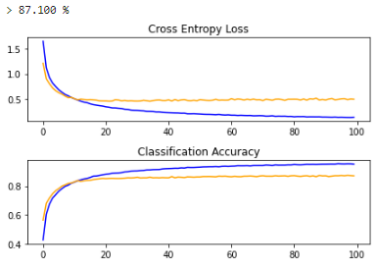


Figure 5: Final Network

6. Experiments - Project extensions

6.1. Make the VGG network look more like a ResNet

Originally, ResNet was developed to address the degradation problem in deep networks, which occurs when increasing the network's depth degrades the performance both on test data and on training data. The degradation problems can be addressed by using Residual Network (ResNet) with skip connections, by making deeper network a subset of a shallower network. In the skip connection, training is omitted from a few layers and the output is connected directly. Hence, the approach is to make our initial VGG network architecture to look more like a ResNet, in order to allow the network to fit the residual mapping instead of layers learning the underlying mapping. [1]

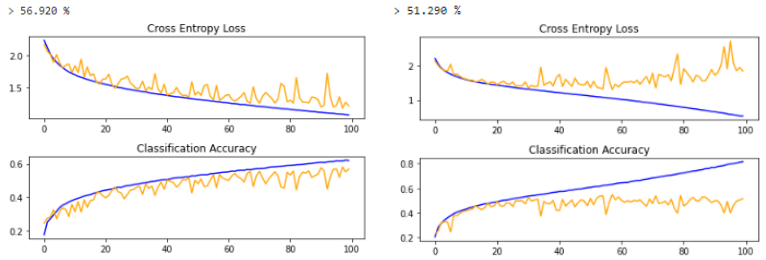


Figure 6: Left: VGG like ResNet, Right: Deeper VGG like ResNet

Although making VGG look like ResNet doesn't increase the network's overall performance as seen in the figure 8, but it does boost up computation speed. Furthermore, the network was made deeper by adding a fourth block, but again its performance deteriorated further. However, the accuracy can be improved by making the network deeper and including more epochs.

6.2. More extensive data-augmentations

More extensive data-augmentations can give a performance boost since it adds more training data into the model and reduces data over-fitting. However, not all data augmentation techniques make sense for the CIFAR-10 dataset. The dataset is low-resolution and thus distorting the images might increase irrelevant data since important features get lost. Adding photo-metric augmentations did not improve the accuracy nor did rotation, scaling and other types of geometric data augmentations except shifting the images in width and height, and flipping them horizontally did improve the accuracy up to 89.16%.

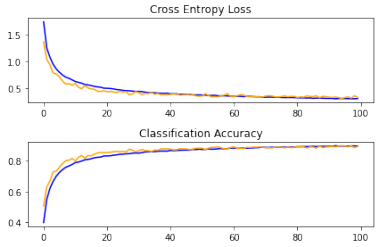
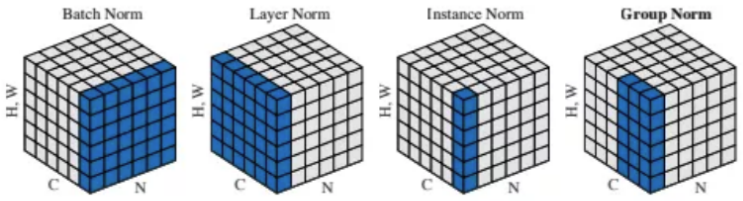


Figure 7: Extensive Data Augmentation

6.3. Different kinds of normalization

Layer normalization normalizes input across features instead of normalizing input features across the batch dimension as is done in batch normalization. Layer normalization and instance normalization are very similar to each other but the difference between them is that instance normalization normalizes across each channel in each training sample instead of normalizing across input features in a training sample. As the name suggests, Group Normalization normalizes over group of channels for each training sample. We can say that, Group Norm is in between Instance Norm and Layer

Norm. [2]



A visual comparison of various normalization methods

We explored these alternatives to batch normalization to examine the effect on the model's accuracy and obtained the following results:

- Batch Normalization: Accuracy 87.2%
- Layer Normalization: Accuracy 86.72%
- Group Normalization: Accuracy 86.3%
- Instance Normalization: Accuracy 84%

We could still maintain high levels of accuracy and more efficient computation time using Group and Layer Normalization. Instance normalization was slightly less accurate but most computationally efficient.

7. Conclusion

The highest accuracy achieved was the 89.16% in section 6.2, where data augmentation was added to the final set-up from section 5.1. ResNet performed worse than was expected, perhaps the network should be made even deeper and have different extensions to improve the accuracy in future applications. However, ResNet outperformed all other models in terms of computational time and hence is easier to optimize. The learning rate did also perform worse than expected, different choice of parameters might increase the performance a bit or using it with another optimizer, for example Adam instead of SGD or by standardizing the data. It takes many failures and trails to improve a network, and the options and different combinations to improve or worsen the network are many. It would be interesting to see our model used on a different image dataset and see how well it performs for future work.

Another lesson learned from the project was that computational efficiency is a key element of implementing neural networks. Due to some difficulties with getting a quota increasing on Google Cloud Platform we were not able to use the GPU resources there. Although GPU resources on Google Colab were sufficient for this project, obtaining additional resources would be vital for further explorations.

References

[1] S. Bhattacharyya. *Understand and Implement ResNet-50 with TensorFlow 2.0*. 2020. URL: https://towardsdatascience.com/understand-and-implement-resnet-50-with-tensorflow-2-0-1190b9b52691.

[2] A. Bindal. *Normalization Techniques in Deep Neural Networks*. 2019. URL: https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8.

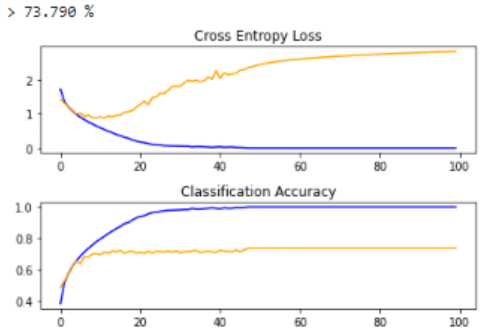
[3] et al He Kaiming. *Deep Residual Learning for Image Recognition*. 2015. URL: http://arxiv.org/abs/1512.03385.

[4] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. URL: http://arxiv.org/abs/1608.03983.

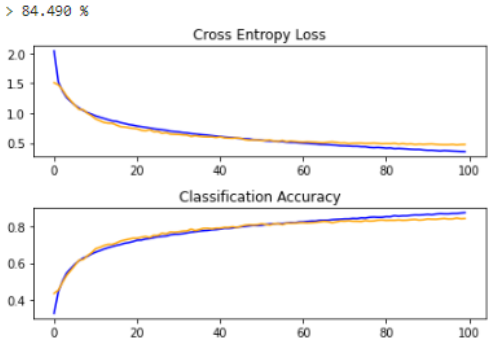
[5] T. A. Mohammed S. Albawi and S. Al-Zawi. *Understanding of a convolutional neural network*. 2017. URL: doi:%2010.1109/ICEngTechnol.2017.8308186.

[6] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. URL: http://arxiv.org/abs/1409.1556.

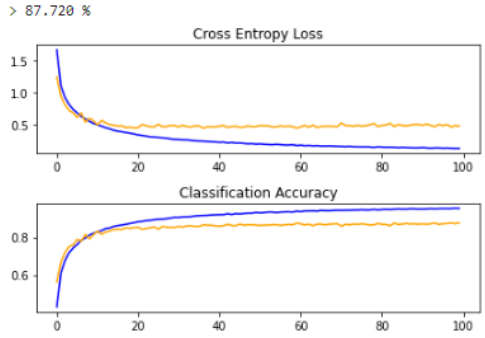
Appendix



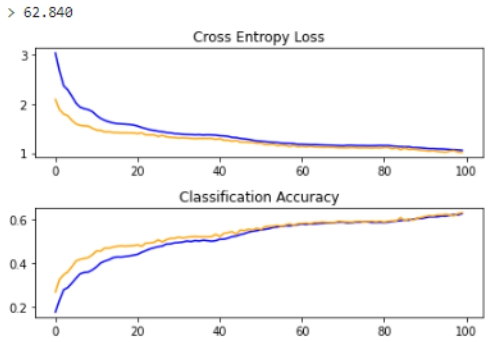
VGGs Baseline



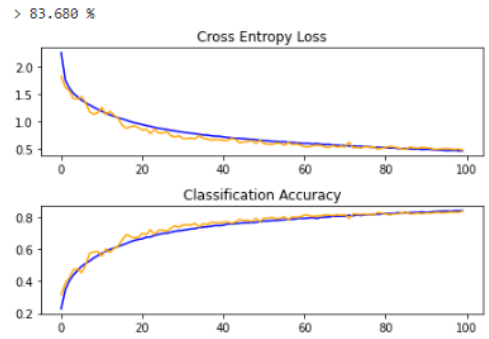
Final Training Model with Cosine Annealing Warm Restarts



Final Training Model with Adam Optimizer



Final Training Model with Cosine Annealing Warm Restarts



Final Training Model with Changed order of dropout and batch norm