Deep Learning Mini-Project— Fall 2022

Written by Eduardo Calzadilla

New York University - CS-GY 6953 / ECE-GY 7123

Abstract

Machine Learning techniques have been widely implemented in diverse settings. The implementation of such models has become easier due to the open-sourced nature of their development, which facilitates transfer-learning. There are instances in which available pre-trained networks will yield significantly accurate and generalizable results - needing to change only a few top layer parameters. In other instances certain applications might require programmers to alter premade models and train from scratch. However, given the large range of parameters available for tuning - requiring more then a few million can lead to incredibly long training periods that are not viable for productionized uses.

In this paper we will explore different pre-designed models and alter their parameters to achieve a high level of accuracy ($> 80\%$) for Image classification susing the CIFAR-10 Image Dataset. Basing our models on the Train CIFAR10 with PyTorch Github Tutorial (link to repo), we will analyze different pre-designed models and then change some of the establishedparameters to increase accuracy until the desired level.

Given initial results based on 2 epochs the GoogleNet and MobileNetV2 structures were selected. Out of these we continued with the GoogleNet and summarize performance results of altered structure in Figure 3. We found the highest accuracy with **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"** and the following learning parameters: LR = 0.05; and WD = 5E-5. The Test results being 85.01% and 86.31% respectively, and training session taking about 20-30 min to complete.

Introduction

Starting with the pre-designed models found in the Train CIFAR10 with PyTorch Github Tutorial (link to repo) - and given the accuracy results stated, we chose to do a preliminary test with the following models:

- ResNet18
- GoogleNet
- DenseNet
- MobileNet
- MobileNetV2
- SENet18

We then trained each of these for two epochs and compared the training and testing accuracy scores, as well as number of trainable parameters. Out of these, GoogleNet and MobileNetV2 scored the highest overall training and testing scores as well as having the lowest number of parameters (summarized in Figure 1), which gives us more freedom in pulling levers for increased accuracy without exceeding our 5 million parameter limit.

Given initial results based on 2 epochs the GoogleNet and MobileNetV2 structures were selected. Out of these we continued with the GoogleNet given the higher relative accuracy and it's relative low amount of initial features to work with.

We can see their performance summaries in Figure 2 - given the higher performance and capacity for tinkering with hyperparameter tuning, we will continue with the GoogleNet architecture for this study.

A brief overview of the GoogleNet architecture as designed in [1] Brownee, J. et. al. shows a 22-layer altered CNN. The alteration mostly comes from the introduction of Inception layers, the same which are required to

Copyright

Methodology

Starting with a higher number of parameters than allowed, we first need to reduce the number of parameters, while leaving all of them without any previous training. Given widespread information on Convolutional Neural Networks (CNNs) we know that decreasing a kernel size will decrease the number of parameters. According to [2] Sood, et al., and others we see that a typically optimal number is 3 (3x3). We verified both with 3 and 5 leaving all else the same, and indeed found 3 yields the least amount of trainable params 6,166,250 vs 6,175,466. Also important to note, is the increased accuracy of the larger kernel size, which is to be expected.

While this does not dramatically reduce the number of our parameters for training - we know that the number can, however, be drastically reduced based on the amount of in-

put channels as well as the number of channels manipulated throughout the model. We trained the 3 for 20 epochs, and saw no real difference between the 3 in terms of Accuracy and Loss, and since MobileNetV2 is an "improvement" on MobileNet and has a lower number of features already present in the model. Therefore, we continued with GoogleNet, to easily drop trainable parameters without drastically changing performance.

Minimizing the channels led to decreasing the number of parameters by about 3/5. Meaning, the model us now under our constraint of 5 million features and can be continued to have its other hyperparameters altered. However, the resulting accuracy was far reduced when compared to the original model. In order to keep track of performance - we re-trained and re-tested at every iteration to compare with the original model provided by kuangliu [7].

The initial response was to change kernel sizes to slightly increase complexity in the hopes of increasing image feature detection. After changing the initial kernel sizes, we were still unsatisfied with accuracy scores and looked at changing all intermediate kernel sizes, with the exception of the first 2D Convolution layer in each Inception layer. Since increasing the Kernel size increases complexity if padding is included, we expected an increase in parameters, but not necessarily accuracy. Another strategy was to alter Kernel sizes for the MaxPooling and AvgPooling layers. We decreased the kernel sizes to decrease complexity, but increased padding for greater contextual awareness, hoping for higher accuracy without a ballooning of parameters [4,6]. To further detail the changes we made:

- **GoogleNet_k5:** Intake layer kernel size, changed from 3 to 5
- **GoogleNet_0_5:** All channels throughout the model reduced by a factor of 2
- **GoogleNet_k5_0_5:** All channels throughout the model reduced by a factor of 2, and intake layer kernel size was changed from 3 to 5
- **GoogleNet_k3_k5_0_5_p2:** Changes applied to GoogleNet_k5_0_5 remain, and all Inception layer Kernel sizes were changed from 1 to 3 (with the exception of the first Convolution layer in each Inception layer). Padding was increased from 1 to 2, to increase complexity, and/or maintain proper sizes.
- **GoogleNet_k3_k5_0_5_p2_k5:** Changes applied to GoogleNet_k3_k5_0_5_p2 remain, and the Avg Pool layer kernel size was changed from 8 to 5
- **GoogleNet_0_5_v5:** Changes applied to GoogleNet_k3_k5_0_5_p2 remain, the Avg Pool layer Kernel size was changed from 8 to 7 and it's Padding increased from 1 to 2, and the Max Pool layer Kernel size was changed from 3 to 5

This led to the final two models we decided to continue with - nicknamed **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"** , which then had their learning rate (**LR**) and weight decay (**WD**) further altered.

At the end of 8 Epochs for the altered architectures, we did not see accuracy scores at or above 80%. And, given that it took about 30-45 min for the full training and testing sessions of each of these models using a cuda environment in Amazon SageMaker Studio Lab - the same are not suitable for tasks that require training from scratch. To further iterate on these, we considered changing the LR and WD. Increasing the former was expected to speed up the epochs, while decreasing the same should increase accuracy [5]. In terms of weight decay however, we expect an increase to lead to less overfitting if present [6].

Results

Choosing the base architecture

When choosing the base architecture, as described in the **Methodology** section, we ran all available models for just two epochs, evaluating Losses, Accuracy Scores, number of features and time to train in a cuda environment. The results are summarized in Table 1. We can see that only GoogleNet, MobileNet and MobileNetV2 are worth pursuing given the above 4 factors.

	Train Loss		Train Accuracy		Test Loss		Test Accuracy			
Model Name	Epoch 1	Epoch 2	Epoch 1	Epoch 2	Epoch 1	Epoch 2	Epoch 1	Epoch 2	Trainable Params	Time to Train & Test GPU (min)
ResNet18	1.978	1.414	28.63	47.65	1.555	1.28	41.56	53.47	11,173,962	75
GoogleNet	1.583	1.043	41.8	62.44	1.38	1.115	50.09	60.58	6,166,250	45
DenseNet	1.733	1.259	36.86	54.51	1.573	1.038	43.87	62.91	6,956,298	53
MobileNet	1.782	1.385	34.21	49.47	1.484	1.257	46.23	53.48	3,217,226	35
MobileNetV2	1.763	1.19	35.36	57.02	1.371	1.184	49.23	59.12	2,296,922	35
SENet18	1.571	1.033	41.51	62.93	1.511	1.087	50.49	61.612	11,260,354	80

Table 1: Training and test accuracy scores for base models provided by kuangliu

Reduction of Channels and Kernel Size changes

Here we can indeed see that what was most effective for lowering trainable parameters was the number of input channels, and overall intermediate channel numbers. When decreasing the total number of channels throughout the model by a factor of 2 we obtained 1.8 million parameters when compared to the 6.16 million in the original model - as can bee seen on row 3 in Table 2 (corresponding to version **"GoogleNet_0_5"**)

While the result in terms of number of parameters was unsurprising, our accuracy fell further than expected to **68%**.

In hopes of increasing accuracy, we looked at kernel sizes, and found the scores detailed on Table 2. This shows that after 5 epochs, **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"** had the highest test accuracy scores, with 78.15% and 78.05% respectively. They also maintained a low amount of trainable features.

As expected when we increased Kernel Size we saw an increase in complexity (i.e. parameters) as well as a slight effect in accuracy scores. In order to get even higher accuracy we tried increasing the padding, and while we found a significant increase in parameters we saw no increase in accuracy, leading us to believe our model was experiencing overfitting. To further increase generalizability we increased the padding - however found no change to accuracy. Next, we moved to the pooling layers, which significantly simplify our image features. Increasing the Kernel size here surprisingly dropped the accuracy scores by a significant amount (about a 10% drop). Thus, it could be instead that the padding and kernel sizes are increasing the complexity but

Version	Change	Trainable Params	End test Accurary (8 epochs)
GoogleNet_k3	Original	6,166,250	72.51%
GoogleNet_k5	Initial layer Kernel_size = 5	6,175,466	78.39%
GoogleNet_0_5	Channel sizes 1/2	1,802,730	73.73%
GoogleNet_k5_0_5	Initial layer Kernel_size = 5 ; Channel sizes 1/2	1,802,730	78.48%
GoogleNet_k3_k5_0_5_p2	All layers Kernel_size = 3 changed to Kernel_size = 5; sans first layer in Inception ; Channel sizes 1/2 ; Intermediate layer Padding = 1 to Padding = 2	3,800,340	78.15%
GoogleNet_k3_k5_0_5_p2_k5	All layers Kernel_size = 3 changed to Kernel_size = 5; sans first layer in Inception ; Channel sizes 1/2 ; Intermediate layer Padding = 1 to Padding = 2 ; Avg_Pooling Kernel_size =8 to Kernel_size = 9	3,814,890	68.03%
GoogleNet_0_5_v5	All layers Kernel_size = 3 changed to Kernel_size = 5; sans first layer in Inception ; Channel sizes 1/2 ; Intermediate layer Padding = 1 to Padding = 2 ; Avg_Pooling Kernel_size = 8 to Kernel_size = 7 && Padding = 1 to Padding = 2 ; Max_pool layer Kernel_size = 3 to Kernel_size = 5	3,799,530	78.05%

Table 2: Training and test accuracy scores for models based on the GoogleNet architecture.

essentially blurring out the visual context, rather than providing some. Some researchers have shown that a 3x3 pooling layer is usually the best for increasing training accuracy and maintaining generalizability [7,8]. Then, decreasing the Avg Pool layer Kernel size to 5 we found an increase in accuracy, to further alter the same we then increased the Max Pool layer Kernel size to further increase testing accuracy.

Overall **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"** had the highest test accuracy scores, while having flexibility in number of features (if further structural iterations are necessary).

Training Hyperparameter Tuning

The the performance of the selected models was then verified with different learning parameters. In order to deal with the expected overfitting to our training data - we tinkered with the LR and WD, the results are shown in Table 3.

Version	Change	End train Accuracy (10 epochs)	End test Accuracy (10 epochs)
GoogleNet_k5_0_5	LR* = 0.1 to LR = 0.2	82.32	80.03
	LR = 0.1 to LR = 0.05	88.25	84.04
	WD** = 5E-4 to WD = 5E-3	65.89	43.01
	LR = 0.2; WD = 5E-5	89.01	82.98
	LR = 0.05; WD = 5E-5	89.58	85.01
GoogleNet_0_5_v5	LR* = 0.1 to LR = 0.2	80.72	68.02
	LR = 0.1 to LR = 0.05	87.1	83.03
	WD** = 5E-4 to WD = 5E-3	73.05	59.17
	LR = 0.2; WD = 5E-5	87.01	83.86
	LR = 0.05; WD = 5E-5	89.91	86.31

Table 3: Training and test accuracy scores for models given different learning parameters, based on the altered GoogleNet architectures - **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"**.

As expected an increase in the LR led to overall less accuracy, with **"GoogleNet_0_5_v5"** appearing more susceptible to a change in LR (Figure 1b). Decreasing the LR by half, greatly increases training accuracy and testing accuracy. While we could stop here, we included some additional changes for comparison.

Since there still could be some degree of overfitting, which could balloon in other datasets, we experimented with the WD. We saw that an increase in WD greatly decreases accuracy scores, particularly when related to testing accuracy. It also seems the **"GoogleNet_k5_0_5"** model was much more susceptible to a WD change (Figure 1a).

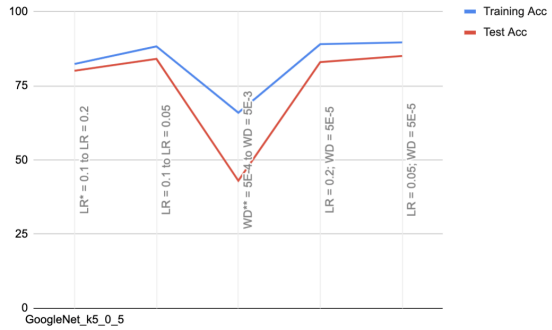

Figure 1a: GoogleNet_k5_0_5 scores

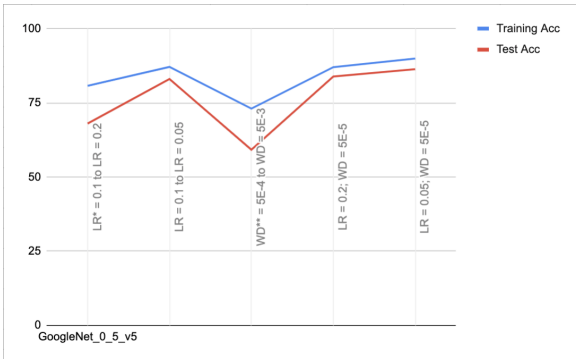

Figure 1b: GoogleNet_0_5_v5 scores

Conclusion

We found the highest accuracy scores for both training and testing with the altered GoogleNet models: **"GoogleNet_k5_0_5"** and **"GoogleNet_0_5_v5"**, set with a LR = 0.05; and WD = 5E-5. The Test results being 85.01% and 86.31% and each took about 25 min to complete in a cuda environment. This should satisfy our needs for ad-hoc training and testing with limited computing resources. However, to further understand one of our models, we trained **"GoogleNet_0_5_v5"** for 25 epochs and found that after the 11th Epoch the accuracy scores start to converge at 93% for training and 88% for testing.

Appendix

Please find the model architecture in the following Git Repo (link to repo)

References

[1] Brownlee, J. (2020, September 11). Understand the impact of learning rate on neural network performance. MachineLearningMastery.com. Retrieved November 27, 2022, from https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/: :text=The

[2] Shivani Sood, Harjeet Singh (2022). Effect of Kernel Size in Deep Learning-Based Convolutional Neural

Networks for Image Classification. ECS Transactions, 107(1), 8877.

[3] Shaikh, J. F. (2020, May 14). Inception network: Implementation of googlenet in Keras. Analytics Vidhya. Retrieved November 27, 2022, from 'https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/'

[4] CIFAR-10 examples¶. CIFAR-10 examples - An open source AutoML toolkit for neural architecture search, model compression and hyper-parameter tuning (NNI v2.0). (n.d.). Retrieved November 27, 2022, from https://nni.readthedocs.io/en/v2.0/TrialExample/Cifar10Examples.html

[5] Brownlee, J. (2020, September 11). Understand the impact of learning rate on neural network performance. MachineLearningMastery.com. Retrieved November 27, 2022, from https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/: :text=The

[6] Zhang, A., Lipton, Z. C., Li, M., amp; Smola, A. J. (2020). Chapter 3: linear Neural Networks for Regression. In Dive into deep learning release 0.14.3. essay. associated website: https://d2l.ai/index.html

[7] Kuangliu. (n.d.). Kuangliu/Pytorch-CIFAR: 95.47