Reproducing AlexNet

David Schrier 260827890

Theo Janson 260868223

Jacob McConnell 260706620

Reproducibility Summary

The goal of this paper was to reproduce AlexNet's results and claims and perform an ablation study. In particular, we were able to reproduce a Top-5 error rate of 18.2%, a Top-1 error rate of 40.7%, We also confirmed that Inferior performance for less than 5 convolution layers by 2% was deemed to be true. By reproducing the results, this shows that despite small deviations and tests that the model itself is dynamic. We also tested the use of other activation functions, average pooling and non-overlapping pooling.

Scope of Reproducibility

This paper aims to reproduce the AlexNet model developed in 2010 and trained on the ImageNet ILSVRC-2012 data. AlexNet is a Deep Convolutional Neural Network that won the ImageNet ILSVRC-2012 classification challenges. The model achieved a top-5 error rate of 18.2%, and top-1 error rate of 40.7% in 2012. In addition to reproducing the results from the ImageNet LSVRC-2012 dataset, we will be analyzing the other conclusions that the authors of AlexNet drew when developing their models.

Methodology

We implemented the AlexNet model using the open source PyTorch implementation. We copied the source code directly, so that the model could easily be changed for the purpose of hyperparameter tuning and ablation studies. We used the pretrained weights from Pytorch due to the cost of training the model.To evaluate the model and the changes we made to it we used subsets of the ImageNet 2012 validation set because it was a manageable size and we had access to the ground truth for the images. In terms of resources we used mostly the GPU's available on google Colab. One team member had a subscription to Colab Pro which was used to fine tune the model for ablation studies.

Results

We were able to reproduce similar top-1 and top-5 error rates of 42.83%.and 21.00% respectively for the pre-trained AlexModel by testing it on portions of the 2012 ILVRC validation dataset. This is within 3% of the reported values for the AlexNet Model in the paper. This supports the original claim of the paper.

What was easy

It is relatively easy to use the Pytorch implementation of AlexNet in order to classify images . Modifying AlexNet and even evaluating its accuracy posed challenges however.

What was difficult

The main difficulties we faced were due to the fact that AlexNet was trained on a dataset of 1.2 million RGB images, which took the authors five to six days on two GTX 580 3GB GPUs. Our laptops cannot easily handle that much that, and training a single model would take weeks, as the process updates 60 million parameters. Though we used PyTorch's pretrained weights for our model, these weights would need to be retrained for changes made to hyperparameters or model architecture. Given that we were training with 35k images, this dataset proved to be small.

Communication with original authors

We did not communicate with the original authors. However, we did send an email to the author of the tutorial we followed for finetuning Torch Vision models. We wanted to clarify some assumptions they made regarding the format of the image folder, but got no reply.

1 Introduction

In this paper we looked to reproduce the classifier from the the paper "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krixhevvsky, Ilya Sutskever and Geoffrey E. Hinton. This classifier is commonly referred to as AlexNet. This classifier was originally trained using the ImageNet ILSVRC-2010 dataset and was submitted to that years associated competition. The classifier classifies images among 1000 categories. It was submitted again to the ILSVRC-2012 competition, trained on the ImageNet LSVRC-2012 dataset. Due to the unavailability of the 2010 dataset, we only reproduced the model trained on the 2012 dataset. This reproduction study includes reproducing the main results obtained by the authors, performing ablation studies and finetuning the model's hyperparameters.

2 Scope of Reproducibility

We reproduced the results and the claims, or conclusions drawn by the authors. The main result was the error rate the model achieved on the c dataset in the corresponding competition. We do not have access to the labels of the test set from which this result was obtained, so we tested the model using the validation set, which contains labels. The paper also highlighted many design choices that were chosen by performance. These choices include the use of ReLU activation to accelerate training, Dropout layers to reduce overfitting and decrease the error rate on the validation set, the use of overlapping max-pooling, and the use of five convolutional layers. We also wrote code to perform a grid-search over the momentum, batch-size and learning rate hyperparameters to determine if our model's performance is optimized using the same hyperparameters. Unfortunately, we did not have the time resources to perform a grid search over momentum and learning rate. The paper's tested claims are listed below:

- Top-5 error rate of 18.2% (training on 1.2M images, testing on 150k images)
- Top-1 error rate of 40.7% (training on 1.2M images, testing on 150k images) Faster training using non-saturating ReLU activation, than saturating Sigmoid and TanH activations.
- Inferior performance for less than 5 convolution layers by 2%: removing any convolutional layer results in inferior performance. Removing middle layers resulted in a loss of 2% top-1 accuracy
- Better performance using overlapping pooling than non-overlapping pooling. Overlapped max-pooling layers with a kernel size of 3 and a stride of 2 generally produced less overfitting than max-pooling layers with both a kernel size and a stride of 2.
- Optimal stochastic gradient descent batch size of 128.

In addition to demonstrating these claims, we will also produce an additional result by substituting max-pooling layers with average pooling layers.

Link to code: https://colab.research.google.com/drive/18TIsrOO1BbQbKhC1IZ8FMY_wK5uiofYyscrollTo=fm8s1tdOP_iQuniqifier=1

3 Methodology

The authors used two GTX 580 3GB GPUs to train the model on 1.2 million images. We used the PyTorch implementation of the model from the Torch Vision module. This model has the same architecture as the primary one described in the paper and is also trained on the ILSVRC-2012 dataset. To tune the model, we used Google Colab pro resources: a NVIDIA TESLA P100 GPU. We could not implement the exact same methodology due to a lack of hardware resources. To finetune the model, we followed a Pytorch tutorial for finetuning TorchVision models, and edited the code to suit the purposes of this study. Our methodology for the ImageNet dataset is discussed in section 3.2.

3.1 Model descriptions

AlexNet is a Deep Convolutional Neural Network. The model has 60 million parameters, 650k neurons. It consists of five convolutional layers, followed by ReLU nonlinear activations. Some of these are then followed by max pooling layers. A model architecture can be found in figure 1 of the appendix. The first convolutional layer takes in a 224x224 3 channel RGB image as a tensor object and filters it using 64 kernels of size 11x11x3 with strides of 4 pixels. This is followed by ReLU activation and max pooling with a 3x3 kernel and a stride of 2. The second convolutional layer filters using 192 kernels of size 5x5x48. This is also followed by ReLU activation and max pooling with 3x3 kernels. The third convolutional layer filters using 384 kernels of size 3x3x192. This is also followed by ReLU activation. The fourth convolutional layer filters using 256 kernels of size 3x3x384, following by ReLU. Finally, the fifth layer filters also using 256 kernels of size 3x3x256, followed by ReLU. Before the output produced by the last layer is classified, the output is passed through an adaptive average pooling layer with a 6x6 kernel size. The output is classified by being passed through a dropout layer, a 4096 neuron fully connected layer, a ReLU layer, another dropout layer, another fully connected layer, another ReLU, a third fully connected layer, and finally a Softmax layer to produce category probabilities. ReLU layers were used to introduce non-linearities and allows the model to converge faster than using Sigmoid or Tanh. The dropout layer sets the output of each neuron with a probability of 0.5 to 0, so they do not contribute to back propagation or predicting. The model learned using stochastic gradient descent, with with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. The model optimized for cross entropy loss. To initialize the model, pretrained weights were used. These was trained on the ImageNet ILSVRC-2012 dataset.

3.2 Datasets

The training set was available for download, but was impractically large (130 GB) for the purposes of this study. The test set lacks labels, so cannot be used to obtain performance metrics. We elected to only download the validation set and split it up into a test set a training set and a validation set. The training set was composed of the first 35000 images in the validation set of the 2012 ImageNet Dataset. The validation set we used were images 35001 - 40000 in the original validation set. We downloaded it from ImageNet from the following link: http://www.image-net.org/challenges/LSVRC/index. The ImageNet dataset was created and is run by Stanford and Princeton university. Unfortunately the labels used by ImageNet did not match those used by the PyTorch AlexNet pretrained model. In order to resolve this we found a document on GitHub that contained the category IDs used by ImageNet with references to the names of the categories[3]. We then used those names to link the Category ID's from ImageNet to the category IDs from Pytorch using a list of Pytoch's category names where the entry of the name is that categories ID.

3.3 Hyperparameters

The stochastic gradient descent hyperparameters were a batch size of 128, a learning rate of 0.0005, and a momentum of 0.9. These hyperparameters were used for our reproducability and ablation studies. We also developed code to perform a grid search for these hyperparameters. However, due to a lack of time resources, we only tested a batch size of 8, 128, and 500. The batch size of 128 proved to be a good balance between accuracy and training time. While it did perform slightly worse the a batch size of 8 (top-5 accuracy of 80% vs 79%). It is worth noting that as we were finetuning the hyperparameter using only 35k examples, the model could only perform marginally better.

3.4 Experimental setup and code

In order to use AlexNet we had to preprocess the images before the model could make predictions on them. This preprocessing consisted of downsampling our images to sizes of 256 by 256. We then centered and cropped the image to be size 224 by 224. Next we normalized the mean and the standard deviation of the RGB channels of the pixels using the following values: mean = 0.485, 0.456, 0.406 ; std = 0.229, 0.224, 0.225. This is slightly different from what was done in the paper but was necessary to be compatible with the Pytorch implementation of AlexNet. In the AlexNet paper their preprocessing consisted of down sampling and cropping the images to a size of 256 by 256 and then centering the raw RGB values of the pixels over the entire training set. Because we are using the pre-trained model we had to use the processing suggested by PyTorch. For each of the models implemented, the stochastic gradient descent hyperparameters were set to the values discussed in section 3.3.

The top-5 and top-1 error rates were obtained by initializing a PyTorch AlexNet model using pretrained weights, and evaluating the accuracy using 5000 examples from our test set. The error is the obtained by subtracting the accuracy from 1. The ablation studies proved to be a considerable challenge. We initialized the model by passing it a list of the default feature and classification layers.To remove the last convolutional layer, we simply removed the layer from the initialization, and its corresponding key value pair from the dictionary of model weights.

We also replaced the PyTorch ReLU activation layers with Sigmoid and Tanh Pytorch activations. Finally, we replaced the PyToch max pooling layers with PyTorch average pooling layers.

For each of these changes, we then finetuned the original model weights by training on the new architectures and optimizing the weights using stochastic gradient descent with a cross entropy loss criteria.

To fine tune the model we used training code from Pytorch's website[4].

PyTorch vision model learning works best using a PyTorch dataloading method for better readability and modularity. This process was used in the tutorial we followed, which relied on the there being a directory to two folders named "train" and "val" which are each formatted in the same manner as the original training set from ImageNet for 2012. In order to create a training and validation set we used modified shell scripts derived from a script we found on github [5]. These scripts were modified to only move the images with the numbers 1-35000 for the "train" folder into the category folders. Then after moving the filled category folders into the train folder we ran the second modified script which created new category folders and moved images with number 35001-40000 for the "val" folder. We then moved those filled category folders into the "val" folder. We could then upload the 'train' folder and the 'val' folder to the google drive directory that our collab had access to allowing the training code to fine tune our model.

Link to code:

https://colab.research.google.com/drive/18TIsrOO1BbQbKhC1IZ8FMY_wK5uiofYyscrollTo=VafL0Q6NsZw6uniqifier=1

3.5 Computational requirements

We upgraded our Google Colab plan to have access to more computational resources. We ran this study on a NVIDIA TESLA P100 GPU, and consumed about 14.5 hours of GPU runtime. The average time to predict the labels of 5000 images is 40 seconds, and the average time to finetune the model on 35k images with a batch size of 128, over 10 epochs is 80 minutes.

4 Results

4.1 Results reproducing original paper

Experiment 1: We aimed to support the claim that AlexNet produces an 18.2% top-5 error rate and 40.7% top-1 error rate (training on 1.2M images, testing on 150k images). We obtained a top-5 error rate of 21% or an accuracy of 79%, which is within 3% of the claimed result. Moreover, we achieved a top-1 error rate of 42.83%, which is within 2.12% of the claimed result.

Experiment 2: The paper claimed that removing a single convolutional layer degrades performance. In particular, they claimed that removing a middle layer decreased accuracy by 2%. We tested the removal of the 5th convolutional layer. We initialized this model and fine-tuned it on 35k images. This produces a top-5 error rate of 30.5% and a top-1 error rate of 56.0%. This result somewhat supports the claim and is discussed in section 5. Figure 3-4 in the appendix plots accuracy against epochs.

Experiment 3: The paper claimed that overlapped pooling with stride 2 and kernel of 3 performed slightly better than non overlapped pooling with a stride of 2 and a kernel of 3. We tested this out, obtaining top-5 error rate of 21% and a top-1 error rate of 42.83% for overlapped pooling and top-5 error rate of 10.1% or an accuracy of 2.5%, which does not support the claim.

Experiment 4: The paper claimed that the model produced better performance with ReLU activation than with Sigmoid and Tanh. We failed to reproduce this.

4.2 Results beyond original paper

Experiment 5: We removed both the second to last and the last convolutional layers and their ReLU activations, and then finetuned the model on 35k images. This resulted in a top-5 accuracy of 9.3% and a top-1 accuracy of 2.15%. Figure 1-2 in the appendix plots accuracy against epochs.

Experiment 6: We replaced the max pooling layers with average pooling layers with the same kernel. This resulted in a top-5 accuracy of 2.6% and a top-1 accuracy of 1.1%, though we did not fine tune the model to these layers.

5 Discussion

The results form experiment 1 are rather expected as the result was obtained using the PyTorch pretrained weights.

Unlike the authors of the original paper we were unable to remove a convolutional layer networks in the middle of the AlexNet model, since we could not train from scratch. We attempted to remove a middle layer by replacing a convolutional layer with an identity layer composed of weights which would forward its input to the next layer and would not be updated by back propagation. This did not work because layers after the removed one depend directly on the preceding layer. When removing a given layer, all the weights of those after it are meaningless and the model needed to be essentially trained from scratch to produce reliable results. We then tried various methods for removing a middle layer from the network. This included replacing the convolutional layer with the PyTorch identity layer, and tried replacing that layer's weights with a tensor composed of (kernel size x kernel size) matrices of 0s with a 1 at their centers, so that the a convolutional layer would produce no change from input to output. We then 'froze' that layer's weights such that they would not be updated by SGD, and performed finetuning. For both methods, after 10 epochs of training, the model still produced random accuracy. This is most likely due to the lack of data used for finetuning the model. We used 35k images. When a middle layer is removed, the weight's of the layers after it are essentially random as they depended directly on this last layer and not the ones before it. As such, more data and computational power would be required to fully retrain these layers. We attempted to retrain, but got no significant increase in performance.

Removing the last convolutional layer did produce an accuracy only 2% worse than that of the full five layer model. Lower convolutional layers learn primitive patterns (ie. lines, shapes) which can be fed to higher layers that can take the the relationships between more primitive patterns in order to recognize objects composed of them. As such, removing the last layer should result in the model making predictions on less defined patterns, resulting in slightly lower performance, which was confirmed by our results. The weights of the remaining layers were finetuned to these patterns. We also removed both the last layer and the second layers and finetuned our model. As the shapes predicted on the third layer are presumably quite primitive compared to those used by the fifth layer to make predictions, the performance ought to be pretty bad, which was confirmed in our results.

Experiment 4 and 6 substituted various components in the model: average pooling for max pooling, and Sigmoid and TanH activations for ReLU. As the pretrained model weights were learned using ReLU, these weights were essentially meaningless when tested using Sigmoid and Tanh. Retraining the weights for these activation functions would require training from scratch, which is out of the scope of this paper. Curiously, using average pooling for forward propagation did not result in a random accuracy of 0.001, but of 1%. Naturally, the model would require lots of training to achieve the same accuracy as the pretrained model.

5.1 What was easy

It is relatively easy to use the Pytorch implementation of AlexNet in order to classify images. Modifying AlexNet and even evaluating its accuracy posed challenges however.

5.2 What was difficult

Various aspects of a thorough reproduction study were difficult to implement. As an image classifier, AlexNet was trained on a dataset comprised of 1.2 million RGB images. This process took the authors five to six days on two GTX 58 3GB GPUs. Our laptops cannot easily handle that much that, and training a single model would take weeks, as the process updates 60 million parameters. Fortunately, PyTorch provides pretrained weights. Many of our tests consisted in making changes to the model and assessing the impact on performance. The main disadvantage of pretrained weights is that we cannot conclude that a given change will produce better results if the model were trained from scratch using that change. In effect, we are simply tuning the model using the valedation data. Moreover, the ImageNet LSVRC-2010 dataset is not available for download. As such, the study aims to reproduce only the results from the 2012 competition. Finally, as the ImageNet LSVRC-2012 is an 80GB folder, we elected to only download the labelled validation set, and to use this dataset for model fine tuning and testing.

5.3 Communication with original authors

We did not communicate with the original authors. However, we did send an email to the author of the tutorial we followed for finetuning Torch Vision models. We wanted to clarify some assumptions they made regarding the format of the image folder, but got no reply.

6 Summary

To summarize, we achieved roughly the same accuracy as the authors of the paper. We also demonstrated through ablation studies that removing a layer reduces the performance, and that removing two layers provides a very poor classifier. We also attempted to substitute various design choices such as activation and pooling techniques, but these required to much additional training to produce the same accuracy as the original model. We also performed a grid search over the batch size and determined that the authors chose a good middle ground.

We did attempt to remove the dropout layers as the authors claimed that they are key to reducing overfitting in the model. A further reproduction study could explore this idea, which proved to require too much additional training on our side to reproduce reliable results.

7 Statement of Contribution

Jacob and Theo wrote the code and the tests and contributed to the report. David also helped out on the report.

References

[1] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015

[2] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012): 1097-1105.

[3] https://gist.github.com/aaronpolhamus/964a4411c0906315deb9f4a3723aac57

[4] https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html

[5] https://raw.githubusercontent.com/soumith/imagenetloader.torch/master/valprep.sh