Clayton Cohn
clayton@claytoncohn.com
27 Sep 2020

DePaul University
CSC594: Advanced Deep Learning
Prof. Noriko Tomuro

Assignment 1

Overview

The purpose of this assignment was to model the TensorFlow dataset ”Rock, Paper, Scissors” [1] with a Convolutional Neural Network (CNN) via transfer learning. We were instructed to experiment with a variety of architectures and then submit our results to a Kaggle competition [2]. I also experimented with fine-tuning, data augmentation, and training a model from scratch (no transfer learning, i.e. no weights). Additionally, we were instructed to visualize a CNN (trained from scratch) via its filters and feature maps. My best model was the VGG19 [3] CNN with ImageNet weights, fine-tuning, and data augmentation. It achieved an error of 0.038 and an accuracy of 98% which placed third in the Kaggle competition.

Methods

The first model I tried was VGG19. I implemented it via the Keras documentation [4]. Initially, I set the model’s pretrained weights to ”None” so that they would be randomly initialized. As expected, performance was underwhelming: validation accuracy struggled to eclipse 40% after five epochs. Next, I implemented VGG19 with ImageNet weights. The validation accuracy achieved 100% within the first two epochs—obviously a drastic sign of overfitting. Considering the training dataset was small (3000 instances), overfitting was an inevitability. This is not uncommon in transfer learning, as one of the primary advantages of transfer learning is the ability to utilize deep learning to model small datasets. To combat this, I implemented data augmentation [5].

I added a sequential data augmentation layer to the VGG19 model that randomly flipped and rotated the training set instances during preprocessing. This improved accuracy on the test set by a few percentage points. At this point, I started experimenting with other models on the augmented dataset. The specific models will be further discussed in the next section.

After experimenting with the various models, VGG19 was still the best-performing, so this was the model that I chose for fine-tuning implementation. For fine-tuning, I unfroze the top four convolution layers so that they could be trained. Fine-tuning augmented my accuracy considerably, significantly reducing my error. I also trained VGG19 from scratch (weights=”None”) to see how the performance compared to the transfer learning approach. The various errors and accuracies are located in the ”Results” section.

Models

I experimented with four different models: VGG19, ResNet152V2 [6], Xception [7], and InceptionResNetV2 [8]. These models were chosen due to their high reported accuracies in the Keras documentation [4]. For each model after VGG19, I only implemented transfer learning with ImageNet weights and augmented data. The reason for this was that randomly initialized weights already proved to be ineffective at modeling the dataset, and ImageNet weights without data augmentation had a propensity for overfitting the dataset. Therefore, I chose to compare each model via ImageNet weights and augmented data, as I surmised that this combination would yield the best performance (i.e. lowest test set error).

Results

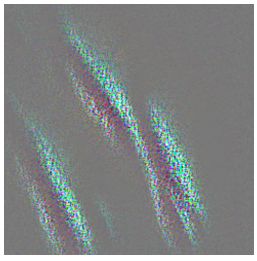
As stated earlier, my best results were with the VGG19 model and ImageNet weights. Data augmentation helped improve my accuracy by reducing overfitting, and fine-tuning the top 4 layers of convolution added an additional performance boost. The errors and accuracies for the various VGG19 implementations are listed below in Table 1, with the best-performing model highlighted in yellow. Errors and accuracies for the other models mentioned in the previous section were not specified due to their underperformance relative to VGG19. I did not experiment extensively with hyperparameter tuning. I implemented both RMSProp and Adam optimizers and noticed a slight performance increase with RMSProp. The learning rate for both optimizers was kept at 0.0001 except for the fine-tuning process where it was reduced by a power of 10 to 0.00001.

VGG19 Model	**Error**	**Acc.**
Transfer Learning (TL), no weights	1.971	0.79
TL, ImageNet weights	0.425	0.88
TL & Data Augmentation (DA)	0.113	0.96
DA & Fine-Tuning	0.038	0.98

Table 1: VGG19 models with errors and accuracies.

Visualization

To visualize the model, both feature maps and filters were produced. I utilized the code provided from François Chollet's book [9] for the feature maps, and I used the Keras documentation [10] (also courtesy of Chollet) for the filters. Instead of using my best performing model for the visualization process, I opted to train (and overfit) VGG19 from scratch to see exactly what the model had learned from our training data (as opposed to ImageNet). I trained the model for 50 epochs with random weight initializations on the original (unaugmented) training set. Figure 1 illustrates a filer and several feature maps.



(a) A filter

(b) Some feature maps

Figure 1: A filter and some feature maps

Difficulties

Overall, I did not encounter many difficulties with the assignment or its implementation. It was clear from the start that overfitting would be an issue, as the dataset was small in size; however, I knew that this could be remedied with data augmentation. There were some minor TensorFlow versioning issues between the various references, but this is to be expected—much of the documentation for machine learning becomes obsolete within several months of publication. Lastly, I did run into some GPU memory issues with Colab. I remedied this by reducing the batch size from 128 to 16.

Reflections

I thought this was a great assignment. It was my first foray into computer vision, and I thought it was a great opportunity to implement models with various CNN architectures. Furthermore, this was also a nice opportunity to see the power of the ImageNet weights. I had heard about ImageNet for some time, but this was my first experience seeing its power first-hand. I also enjoyed comparing and contrasting the various implementation methods, i.e. transfer learning and fine-tuning. Lastly, I thought visualizing the filters and feature maps was an incredibly effective way of ascertaining a visual understanding of the model used to make its predictions. Altogether, I thoroughly enjoyed this assignment.

References

[1] Laurence Moroney. *Rock, Paper, Scissors Dataset*. Feb. 2019. URL: http://laurencemoroney.com/rock-paper-scissors-dataset.

[2] Noriko Tomuro. *CSC 594 HW#1: Part 1 (Fall 2020) Classification of Rock-Paper-Scissors images*. Sept. 2020. URL: https://www.kaggle.com/c/csc-594-hw1-pt1-fall-2020.

[3] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. eprint: arXiv:1409.1556.

[4] Chollet et al. *Keras Applications*. Mar. 2015. URL: https://keras.io/api/applications/.

[5] TensorFlow. *Data augmentation*. Sept. 2020. URL: https://www.tensorflow.org/tutorials/images/data_augmentation.

[6] Kaiming He et al. *Identity Mappings in Deep Residual Networks*. 2016. eprint: arXiv:1603.05027.

[7] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2016. eprint: arXiv:1610.02357.

[8] Christian Szegedy et al. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. eprint: arXiv:1602.07261.

[9] Francois Chollet. *Deep Learning with Python*. 1st. USA: Manning Publications Co., 2017. ISBN: 1617294438. URL: https://condor.depaul.edu/ntomuro/courses/594ADL-2020fall/Chollet/Chollet-v1-convvis.pdf.

[10] Chollet et al. *Visualizing what convnets learn*. May 2020. URL: https://keras.io/examples/vision/visualizing_what_convnets_learn/.