Plankton Image Classification

Sagar Chordia
Stanford University
sagarc14@stanford.edu

Romil Verma
Stanford University
vermar@stanford.edu

Abstract

This paper is in response to the National Data Science Bowl's automated Plankton image classification problem. It evaluates the performance of many different experiments run on medium-architecture Convolutional Neural Networks.

Experiments evaluated include different architectures, feature extraction from multiple architectures followed by linear and non-linear classification, data augmentation and model ensembles. The experiments together yield a classification accuracy of 70% and multi-label log loss of 1.15 on validation set.

The learned approaches can be ported to deep ConvNets as future work for significant improvements.

1. Introduction

Given a supervised sample dataset of types of planktons, we are applying Convolutional Neural Networks for image classification. Recent advances in neural networks have showed that CNN with various techniques perform extremely well for image classification as they learn complex detectors. Convolutional filters operate on localized region of images and multiple layers in CNN can combine several localized regions and can understand the complete image in entirety.

Figure 1. Sample Images from three plankton classes

2. Background

2.1. Dataset

Hatfield Marine Science Center at Oregon State University provided 30,458 manually labelled images of planktons as training set and another 130,401 plankton images as the test evaluation set. Each of the images are classified into one of the 121 classes of planktons, including different unknown class of images grouped together based on shape. The distribution graph shows that distribution of training images is skewed and hence model learns less about the infrequent classes.

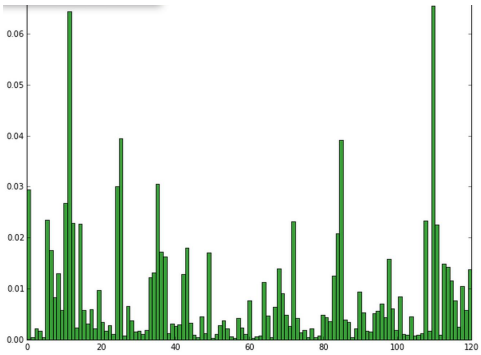

Figure 2. Distribution of number of images per class

2.2. Evaluation Metrics

We used a 90%/10% split of the 30,000 images training dataset into train and validation sets. Given very small available dataset, to improve model robustness a 90% train split was necessary. K-fold cross validation was unfeasible given training is very expensive in terms of computation and also time. The validation split was used for our own evaluations in approaches below. The models were evaluated on two metrics - the validation accuracy and multi-class validation loss. The loss is defined as -

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$$

where N is the number of test images, M is number of classes, y_{ij} indicates if observation i is in class j and p_{ij} is learned probability that observation i is in class j.

$$accuracy = \frac{\sum_{i=1}^{N} 1[y_i = \bar{y_i}]}{N}$$

where N is the number of test images, y_i indicates class of observation i and $\bar{y}_i$ indicates true class of observation i

3. Approach

- We experiment with different layered self-trained Convolutional Neural Networks, varying them from 2 layers to 6 and using them as base models for many of our experiements.
- Given the base models and pretrained models from AlexNet [1] and BVLC GoogleNet [2], we experiment with transfer learning and training a softmax classifier as the final layer.
- We try to extract penultimate features from the baseline and pretrained models, combine them and train a single linear classifier on top. The premise of this approach was not to take the better predictions of the different Softmax layers but to let the Classifier make a single prediction given more features.
- We tweak the hyper-parmeters such as RMS momentum, dropout, learning rate, regularization penalty and filter size for our experiments.
- We also try model ensembles using the best learned models so far.

4. Experiment

4.1. Uniform probability

As a baseline we performed this experiment where images can belong to any of 121 classes with uniform probability. Thus each image is classified randomly into one of the classes with uniform probability which gave log loss of 4.79 comparable to 10 percentile on the Kaggle competition leaderboard.

4.2. Data Augmentation

We performed some experiments where we tried to augment data with methods like random rotation, random crop and random contrast. Since all images are in grey-scale we didn't try random tint. Almost all images have single central object and hence cropping didn't tend to help. Between rotations and contrast, we observed that random rotations was the most effective data augmentation technique.

4.2.1 Rotation

Randomly rotating images by 90, 180 or 270 degress helped, given images are orientation agnostic, hence it is very possible that any rotation of the planktons should classify to correct class. We augmented the data by sampling 50% of images, randomly rotating them and then sampling same number of images in as in the original training set for training the model.

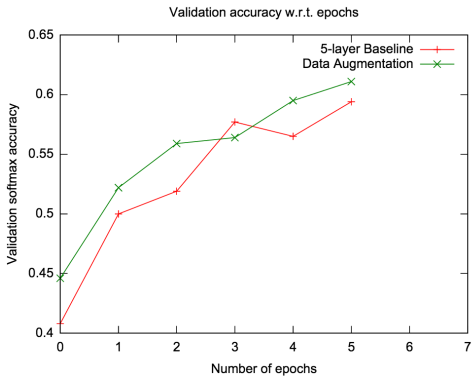

Figure 3. Validation Accuracy for Baseline and Augmented Train Set

Data augmentation on average gives 2.2% improvement on validation accuracy per epoch and getting as high as 4% improvement in one.

4.3. Convolutional neural network

We performed various experiments on CNN, some major of them are noted here:

4.3.1 Number of layers in CNN

We experimented with 2,3,5,6 layers in CNN network. There was clear increase in log loss and validation accuracy as number of layers were increased from 2,3 to 5. 6-layered network improved training accuracy but didn't improve validation accuracy implying overfitting. But on addition of augmented data 6-layers did show improvement over 5-layer.

Since number of training examples are 30k huge CNN architecture is expected to overfit. This can be avoided by increasing the number of training samples by data augmentation.

We started training more complicated architectures ($>$ 10 layers) on GPU instances in terminal.com. But because of frequent downtime of GPU instances, we decided to abandon terminal.com and instead use local machine to train networks. Because of CPU mode on local machine, we stopped at 6-layer ConvNet and decided to show improvements using this as baseline.

The shallower ConvNets converge in terms of accuracy and loss much faster than the deeper ones. Further, the deeper ConvNets 5-layer gave better results for both metrics. ConvNet running for 6 epochs achieves an accuracy of 61% and validation loss of 1.35

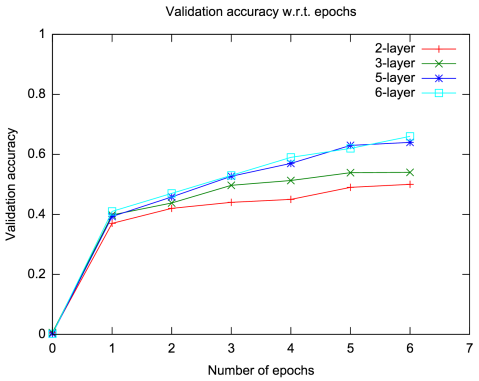

Figure 4. Validation Accuracy of 2,3,5,6 layer convnet models

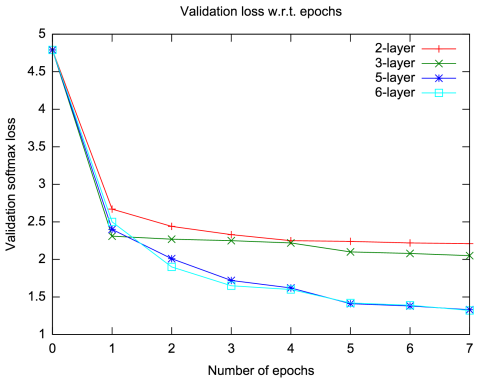

Figure 5. Validation Loss of 2,3,5,6 layer convnet models

4.3.2 RMS momentum, dropout parameter, filter size

Parameter sweep on 5 and 6 layer network indicated 0.9 as best value for RMS prop and 0.8 value for dropout [3] Literature suggests that smaller values for filter tend to perform better for deeper network. We observed that for our architecture (max 6 layers) however filter size of 5 performed best.

4.3.3 Learning rate

Log Loss was plotted as function of number of iterations and the shape of resulting curve was used to tune learning rate. When lr was too small, loss decreased very slowly and when lr was too large loss function had very steep decrease. Learning rates for each architecture was chosen so that loss function obeys approximately exponential decay.

4.3.4 Regularization

We plotted training accuracy and validation accuracy against number of iterations to tune regularization(reg) values. When the gap between training accuracy and validation accuracy was high (training $>$ validation) it indicated large overfitting on dataset and hence indicated low regularization value. Value was chosen to minimize the gap between two curves but also increase validation accuracy.

4.4. Linear classifier on top of CNN

Typically we observed that increasing number of epochs, training accuracy for model improved but at the cost of increased training time. Hence the motivation for these experiments was to train last layer of ConvNet for many more iterations as compared to CNN. So we conducted series of experiments by using the activations of CNN as features to linear models.

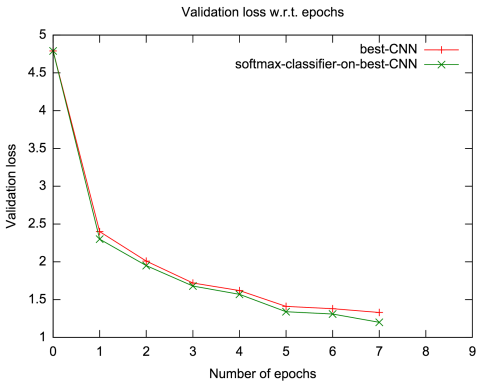

Figure 6. Improvement in loss with linear classifier on top of CNN

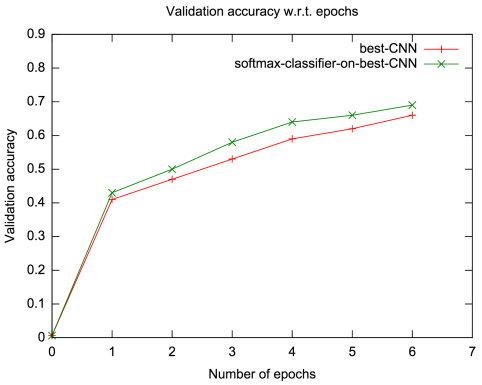

Figure 7. Improvement in accuracy with linear classifier on top of CNN

4.4.1 Features from CNN trained earlier

We fetched 128 dimensional feature vector from a convnet we trained before (2/3/5/6 layer CNN) and trained softmax classifier on these features. Typically we observed improvement of about 5-6% on validation accuracy by having separate linear layer on top of each architecture.

4.4.2 Features from standard CNN (BVLC reference model)

High level idea was to extract features from pre-trained standard architectures. In this experiment we ran our dataset on BVLC reference model in Caffe and used second last layer activations as features. Since BVLC expects size of 256 $\times$ 256 we resized our images to that size. Softmax trained only on these extracted features didn't show any improvement, the max performance achieved was 45%. Possible reasons could be -

Loss of precision. Original images typically of 50 $\times$ 50 were resized to 256 $\times$256 and thus had significant loss of precision.

Sparse features. The features learned were of 4096 dimension and were very sparse. Typically one needs lot of data for training of sparse features but limited size of 30k images posed a challenge to us.

4.4.3 Features from multiple CNNS

When we combined features from 2,3,5,6 layer convnet, it improved the performance by another 2-3%. But careful anaylsis revealed that using features only from 5 and 6 layer convnet gave better performance as compared to using features from all convnets. This happens because of feature importance indicating that features coming from 2 and 3 layer convnets added noise.

4.5. Model ensembles

In this set of experiments we tried averaging predictions from various combination of models. The best performing ensemble composed of 5 linear classifiers on top of features coming from 5 and 6 layer self-trained convnet. This ensemble gave 70% validation accuracy and 1.15 as validation loss. This performance gives us 1.22 as test log loss and ranks us in top 30% teams in the kaggle contest.

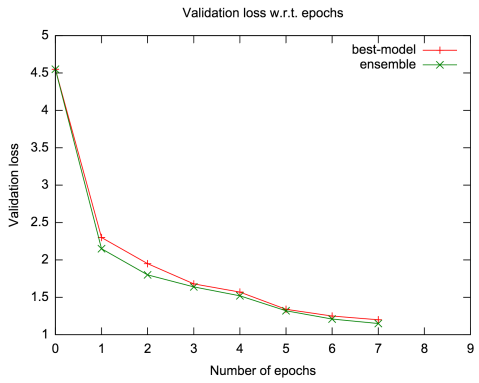

Figure 8. Improvement in loss because of ensembling various model predictions

5. Challenges

5.1. Correlation Between Classes

We created a confusion matrix plotting the predicted labels of validation set images and their true labels to see bottlenecks on accuracy.

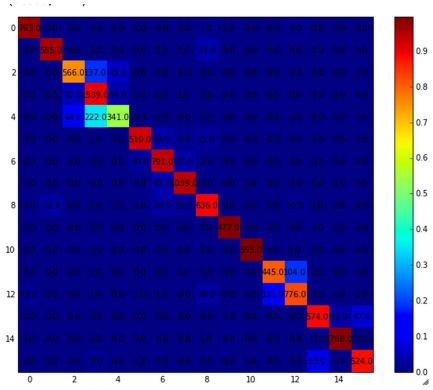

Figure 9. Correlation between True and Predicted labels of 16 most frequent classes

Upon evaluating correlated classes, we found that some classes are very highly correlated such as here classes 2, 3 and 4 refer to chaetognath non sagitta, chaetognath other and chaetognath sagitta. Similarly, classes 11 and 12 refer to protist noctiluca and protist other. These sets of classes are physically same so its difficult to accurately predict between them.

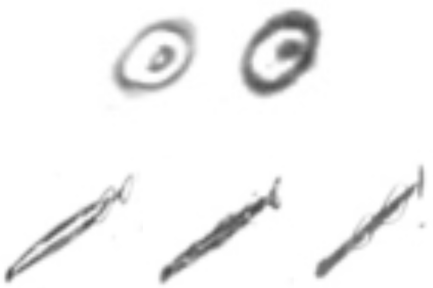

Figure 10. [above] images of protist classes; [below] images of chaetognath classes

5.2. Shallow ConvNets

Most of our experiments have been run on Shallow ConvNets as our approach was to try various models and optimize under that setting assuming that the best working techniques can easily be ported to a deeper network giving better accuracy and loss values. [4]

6. Future Work

- Use deeper networks trained on better hardware (GPU)
- Incorporate hierarchy of classes in the loss function. Currently all misclassification are treated equally. But since entire taxonomy of classes is available we can potentially penalize the misclassification based on distance between two classes.
- Since ensemble of linear classifier improved performance, natural extension is to use random forest or any non-linear classifier on top of features generated from CNN

7. Conclusion

Convnets tend to perform very well for image classification. Data augmentation improves the performance and reduces the over-fitting on training data. Shallow convnets with ensemble of linear classifiers on top of it are very good proxy for deeper convnets.

8. Acknowledgements

Thanks to Kaggle.com for hosting the competition and to Hatfield Marine Science Center at Oregon State University for providing the supervised dataset.

9. Supplementary Materials

The source code for the project can be found at `https://github.com/sagarc/plankton_classification`
The teamname on kaggle contest is under name of Sagar Chordia.

References

[1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

[2] Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).

[3] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

[4] Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman, ICLR 2015.*