Neural Networks Project Report

Yurii Toma
Oliver-Matis Lill

1. Background

This project basically involves participating in the Cdiscount image classification Kaggle competition. In the competition we are given products from the Cdiscount database and our job is to categorize them based on their images.
One of the main complications is that the data is huge, the problem specification is:

- Images have resolution 180x180
- Training set is a 58 GB compressed file with 7 million products represented by more than 12 million images in total (~350Gb when decompressed)
- There are more than 5000 classes
- Test set contains 1.7 million products with 1-4 images for each product

The sheer size of the datasets means that doing the training in our own computers is infeasible, therefore we need to rely on HPC clusters. Since the 58GB training set is compressed (it would be more than 350GB uncompressed) we need to put quite a lot of effort into preprocessing and designing a system to feed the data into our networks.

On top of all that designing the networks itself is obviously quite a lot of work. Our solutions to these complications will be described below.

2. Introduction

Because the main idea of the project is to be able to classify product category by looking only at the image, we think that it make sense to use Deep Convolutional Neural Networks.

To begin with, we are planning to use transfer learning and fine-tuning on ResNet50, VGG-19, and some other pre-trained on ImageNet data networks that are accessible in Keras. For this step we are planning to use only small subset of the whole training data it should also represent only small subset of the whole classes that we need to classify data into (50-100 classes out of total 5000 classes).

We are planning to rely on transfer learning, because it seems to be very time consuming to train the Deep Convolutional Neural Network on such a large dataset from

scratch, we will need tens of training epochs, each of them should run through at least some big subset of the initial train set (e.g. 15%) and then update the weights of the network. Each of these epochs will take few days even on GPU, then to train from scratch we need tens of days on GPU which we don't have. Thus we decided to stop on transfer learning which have been reported as a very powerful way to deal with image classification problems. Because on the top level most object on images are built from the same lines, and corners we then just need to specify combinations of these lines and corners and their connection with the classes in the last layers of the pretrained networks.

Next steps would involve training on a cluster because of the reasons stated in the background section. For this task some Keras functions would be very useful to us and would save some time from reimplementing functions for dealing with big training sets of data (functions like flow_from_directory and fit_generator), because in this project we should be concerned about RAM usage.

On the cluster we would do transfer learning and fit tuning for ~50 models each one on subset of the whole training data, and then we are planning to combine these models into 1 ensemble which will do the whole test images classification. If we would have time it would be also interesting to train the whole model from scratch based on the whole training set.

As a final step we are planning to use the hierarchical structure of the target classification labels. Because we have 3 level tree of categories with roughly 50 nodes at the first layer, 50-100 children per node at 2nd and 3rd levels we can use it in a way that we first classify the image to one (or top 5) of the upper level categories, then based on this we can do 2nd level classification and 3rd (level of interest) classification and the idea is that because we are fixing some class on the upper level the lower level should have smaller number of choices and be able to better learn from smaller amounts of training data.

Of course, we also planning to tune each of our models by applying different hyperparameters values and probably adding/removing some layers. And validate them on validation subset of data, but we should be able to fine tune hyperparameters quickly, so we would either train on subsets of train data or reduce number of train epochs to be able to see what are the behaviour of some hyperparameters at least in the beginning.

3. Methods

In order to efficiently prototype various models, we need a smaller subset of the data that can be used on our own computers. To generate that subset, we used a python preprocessing script, that for each encountered class includes it with 1% chance and then picks all images that are in the included classes. This means that the subset classes have the same amount of images to train on as in the full data and as such should help the subset simulate the full dataset better.

Due to RAM constraints, we need to feed the images from the disk. Conveniently keras has functionality that allows us to do that, but that functionality requires our images to be unpacked from the bson files and categorized into folders. This is handled with another python preprocessing script.

The neural network models were built in Keras. We tried out one simple convolutional model. A lot more effort went into training the powerful pretrained networks provided by Keras, in particular we used: Xception, VGG16, InceptionV3 and ResNet50. They were trained by the so called transfer learning method.

A lot of the training was done on the subset because the networks took very long to train even on the cluster. The pretrained models were trained on full data as well and the results we managed to get will be presented. In the end we were heavily constrained by time and the availability of the cluster.

Lots of products had multiple images 1-4 so we were averaging the predictions for them and getting result for 1 product based on multiple images.

4. Results

A simple convolutional network trained on the 1% of classes subsets achieves a 99% validation accuracy on a single epoch, so training on that subset was not very representative of actual performance and this forced us to take a more blind approach.

We have trained 4 models with the use of transfer learning.

The first one was ResNet50 with the ImageNet weights. We have trained it for 8 separate epochs, the training data was split into train / validation set with the help of keras functions. 0.1 was the validation split. Each epoch was done on subset of the original training set in order to increase training speed. The result of ResNet50 on validation was 0.71.

Here and next we will report only results of validation, because, unfortunately we were not able to produce prediction on the whole test set due to various problems with the code used for prediction. At first we were experiencing memory errors due to loading more RAM than we had, then after doing changes and unpacking of the test data

memory errors have gone. Though the speed was now the problem, we tried predict for each image separately, for batches generated by generator function, but the speed for each image was few seconds, and for batches the problem was that we haven't utilised the GPU, because data generation was yet too slow (or we missed some point here). Interesting thing was the with first tries of transfer learning we were unfreezing all of the layers very soon and the model started to overfit on train, but not on validation. Then we discovered that we need to unfreeze layers with small groups at each epoch and then the results were better.

Xception model was trained in the similar to ResNet50 conditions. Obtained validation result was a bit better comparing ro ResNet50 it was 0.73.

Inception model was trained during the smaller amount of epochs. And the resulting validation accuracy was 0.55.

Top competitors at kaggle reported that their attempts with ResNet, Inception, Xception and VGG16 networks were in range 0.72-0.74, which is quite nice, though because we have used small number of training epochs and haven't used any preprocessing for the images we can expect that once we will receive fast prediction code we will receive ~0.68 accuracy with these models.

5. Discussions

Probably the biggest problem our project faced was the sheer size of the data. It severely limited what we could actually produce and massively slowed down training and testing. Given more time it might have been possible to mitigate this issue by distributing the data over several hard disks across multiple compute nodes, in fact we had many ideas that would have effectively taken advantage of this approach.

One of our ideas was to use a two level classification method where we divide the classes into ~50 subsets and then create one classifier that classifies into subsets and then train classifier for each of the subsets. This is well parallelizable over a larger scale, however it's also a quite complicated approach and it's accuracy will be limited by (level 1 accuracy) * (level 2 accuracy)

Another idea was to simply divide the images into 50 subsets (this times each subset can contain all the classes), train models on those subsets, and then build an ensemble on those models by having the models vote on a class. This approach would have been interesting since here we have the fact that each model has less training data balanced out by the fact that the models form an ensemble. It would be very

interesting to see how it compares to just one model trained on the full data. This idea would have been slightly easier to implement than the two level idea, but still very complicated. Ensembles are generally a powerful tool for classification, so I suspect it would get better results, but it's impossible to tell without trying.

Also we were thinking in a direction of hierarchical models. To use the fact that in the top level categories there are less than 100 classes and train one classifier for the top level and then for each for this classes train one classifier to work inside this top level class.

It's interesting how much easier the problem became when reducing the number of classes while keeping the number of examples per class roughly the same. This implies that in order to properly simulate the difficulty of the problem on a smaller subset, we need a more elaborate way to divide the examples into subsets. One possible way we could do this could go like this: suppose you get a 70% validation accuracy on the full test set with your model. Now try taking 1% of the training examples in such a way that they are chosen from *n* classes. Increase *n* until the model gets a 70% validation accuracy on the subset as well. This should work since increasing the number of classes for the same no. of examples makes classification much more difficult and once n encompasses all classes, it's definitely less to learn for the same goal compared to the full training data. In fact the decrease of accuracy with respect to *n* is almost certainly roughly monotonous, so we can probably massively speed up the process with binary search.

Distribution of classes was very unequal in the dataset, some classes had 10 products in train set and some of them had thousands, which is not very good, so for better training on late stages we were thinking to use some data augmentation.

Overall this was very interesting challenge with largest problem being amounts of data and time needed to make something with the data. We have learned how we can efficiently train a model for such big amounts of data, how to do it bit faster than the simplest way. We have discovered the world of kaggle competitions and will try to participate in future competitions and learn from its discussion forums.