CS 231N Final Project: Landmark Recognition

Alejandro Dobles
Stanford University
adobles@stanford.edu

Luis Fernando Varela Eleta
Stanford University
lfvarela@stanford.edu

1. Abstract

In this paper we tackle a complex classification problem in computer vision using ensembling over homogeneous models trained with a transfer learning approach over deep residual networks. In the introduction we outline the classification problem. In the Data section we discuss specifics of the dataset, and outline specific challenges. In the Methods section we explain a thorough preprocessing paradigm which addresses those specific challenges and outline the architecture of our model. The Training and Tuning section discusses these processes. We report satisfactory results in the Results section. Finally, a short conclusion and discussion of future work follow.

2. Introduction

With the advent of ultra-deep residual networks in 2015 [1], image classification enjoyed a substantial improvement in performance. This caused the field of image classification to explore increasingly difficult classification problems, embodied by massive datasets with a large number of classes and high imbalances in class frequencies. Such is the case of the Google Landmark dataset, first tackled by by Noh *et al.* [5] in *Large-Scale Image Retrieval with Attentive Deep Local Features*, and which labels photographs of landmarks according to the specific landmark they contain. In this paper, we outline our approach for said classification problem, and discuss the results of said approach.

3. Data

The Google Landmark dataset is available to the public on Kaggle.com, as part of the Google Landmark Recognition Challenge. [1] The training set consists of roughly 1.2 million images, each containing exactly one of almost 15,000 landmarks in the label set. The test set has roughly 117k images, each of which contains zero or more landmarks (some even more than one).

There are certain idiosyncrasies of the data which necessitate an extended preprocessing process. To begin, all 1.2M images in the train set weigh approximately 336GB, which is too large to reasonably use for training. This is due to some images being of very high quality. The images are also of varying dimensions. To standardize the images in the training set, and to reduce its size, we first center-crop the image, cropping the longer dimension in order to obtain a square image. We then downsize each of those square images to 250×250 pixels. This is the same initial preprocessing as done by Noh *et al.* [5].

The data set is also highly imbalanced. Table 1 shows the percentiles of label counts in the data set, where label count represents how many images are in a given label. We can also visualize the label counts in Figure 1.

As we see in Table 1, half of all labels have 14 examples or fewer in the training set. In fact, around 1,000 labels have only 1 example in the training set. Not only do we have many labels with few images, we also have some labels with a very high number of images, as we can also see in Figure 1. For example, the labels that correspond to the two tall bars in Figure 1 have upwards of 50k examples. For context, the tall bar on the left corresponds to the Vatican, while the tall bar on the right corresponds to the Coliseum.

[1] See: https://www.kaggle.com/c/landmark-recognition-challenge/data

Percentile	Label count
10	4
20	5
30	7
40	9
50	14
60	22
70	34
80	62
90	142

Table 1. Percentiles of label counts

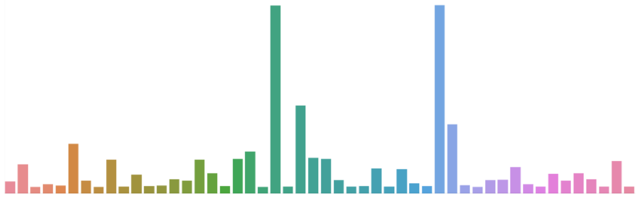

Figure 1. Label Counts

4. Methods

4.1. Preprocessing

Imbalanced data sets have to be handled with care in machine learning contexts as they can easily lead the model to converge to non generalizable optima (for instance by always predicting modal classes). Because of this, we decided to extend the preprocessing stage to incorporate both downsampling of modal classes and upsampling of rare classes. As we will write throughout the paper, we had an initial idea of how to approach the problem, and we made several modifications that lead to a final working version. We will talk about the different approaches as our ”first approach” and our ”final approach”.

4.2. Downsampling

For downsampling, we used a generalized version of ensemble random undersampling (ERUS), which is outlined for the binary classification case by Liu *et al*. [4] In it they generate multiple training sets by random undersampling (RUS), which consists in randomly sampling the number of examples in the smaller class from the modal class. For example, in a binary data set with 100 examples in its 0-class and 1,000 in its 1-class, we would sample 100 examples from the 1-class and use these to generate a balanced training set. ERUS goes one step further and generates multiple of these training sets from which an ensemble of models is trained. The idea is that this allows for each model to train on a balanced data set while taking full advantage of the images in the training set. In fact, Kuncheva *et al*. [3] find that ERUS maximizes geometric mean accuracy of models out of several instance selection methods.

However, the ERUS method as presented in the preceding paragraph is not suitable for our purposes, as we have close to 15K classes instead of two, and if we were to generalize it in the obvious way we would end up with just one image per class on each training set, which is not desirable. Thus, we decided to establish a cutoff number for downsampling.

For our first approach, we decided to make the cutoff 60 images, which, as can be seen in Table 1, means that we undersample approximately 20% of labels. Each of these is then center-cropped to further downsize them to 224×224 (the reason for not downsizing them to this size from the beginning will become apparent later). After getting results of our first approach, we realized that our downsampling was a bit too aggressive, and that even though the idea of having no bias in our data seamed appealing, we were losing the pros of having more data available that we were not using. Therefore, in our final approach we decided to double our cutoff and set it at 120 images. Figure 2 shows an example of downsampling processed on the Coliseum.

4.3. Upsampling

With only downsampling, we still had a highly imbalanced training sets. Thus, to complete the correction we had to turn to upsampling of rare classes. For this, we used standard techniques to bring up the number of examples of rare classes up to 60. We came up with different standard image transformations to upsample images. Transformations include: random crops with zoom, grayscale, mirror images, and increasing/decreasing the brightness. The goal for our first approach was to upsample all images up to 60, to have exactly 60 images for each label. We had several challenges that we had to overcome in this area:

1. Having a uniform distribution for upsampled images: For example, for a class with 10 images, we

Figure 2. Downsample

would need 6 different random transformations for each image.

2. Being efficient throughout the process: Due to the size of the dataset and the long time it takes to load and save images, we wanted to minimize the number of operations as much as possible.

We came up with the following that solves the two challenges described above:

Algorithm 1 Upsampling a rare class

```
1:  $transformations \leftarrow [greyscale, crop, ...]$
2:  $lq \leftarrow lc - 60\%lc$
3:  $uq \leftarrow lc - lq$
4:  $ss \leftarrow$superset($transformations$)
5:  $id2trans \leftarrow \{im\_id : s$ for $s$ in $ss\}$
6:
7:  $ls \leftarrow$sample($id2trans, lq$)
8:  $us \leftarrow \{id2trans - ls\}$
9:  for $im\_id$ in $ls$ do
10:     $ts \leftarrow id2trans[im\_id]$
11:     transform($im\_id$,sample($ts, 60/lc$))
12: end for
13: for $im\_id$ in $us$ do
14:     $ts \leftarrow id2trans[im\_id]$
15:     transform($im\_id$,sample($ts, 60/lc + 1$))
16: end for
```

We apply Algorithm 1 to any rare class, which has a label count of less than 60. With a careful selection of $transformations$, which depends on the number of label counts, the algorithm above will ensure that we always perform enough transformations such we get exactly 60 images, and the number of transformations done to any image will differ by at most one, which is represented on lines 11 and 15. Thus, our new training dataset for any label of this kind will be evenly distributed in terms of original images. The implementation of this algorithm can be seen in our $generate\_datasets$ scripts.

After our first approach was over, we realized that the upsampling was too aggressive and ambitious. First, it was very slow to do due to the multiple transformations that we had to do. On top of that, we would be transforming very rare images up to 60 times. This means that for many images, the original image would show up 60 times in the dataset. Thus, we were not being successful at having an unbiased dataset. We now had a bias towards very rare images, which might have been even worst than the original bias, since those images would not really help us at validation and testing.

Therefore, we decided to tackle the problem a slightly different way on our final approach. We decided to cut the tail of the distribution and completely disregard labels that have less than 10 images, which we concluded that were affecting our performance. Our implementation consisted of getting rid of labels with less than 10 images, which would get rid of around 40 percent of the labels. We still upsampled classes to 60 images. Figure 3 shows an example of upsampling with flipping, grayscale, and increased brightness.

4.4. Preprocessing Notes

After the preprocessing was completed, we would end up with multiple training sets, each of which would be much more balanced than the original training set, including on our final approach. The reason we had multiple training datasets was so we could train multiple models with different datasets, and get the final predictions through majority voting model ensemble, which is described later. All datasets would include the same upsampled images, but they would include different random samples from the downsamples.

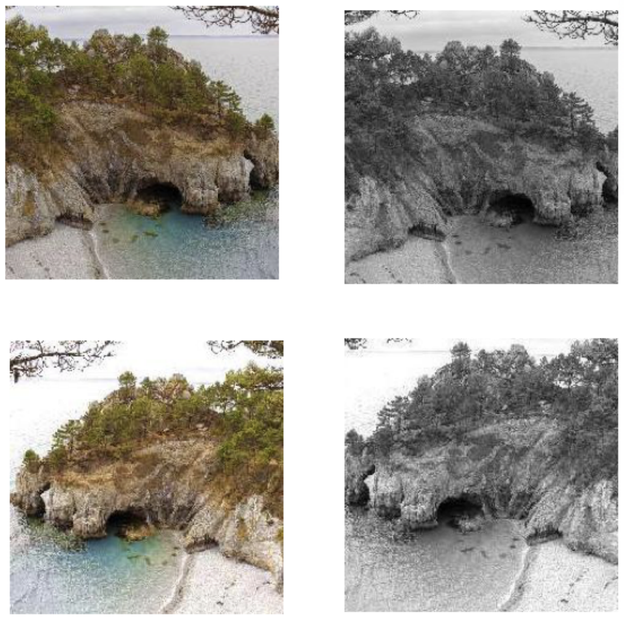
Figure 3. Upsample

The datasets would now contain around 900k images for the first approach, and around 900k images for the final approach. For the first approach, we ended up with exactly 60 images for all classes, while for the final approach, each class had between 60 and 120. We also created a validation set from the original images, taking 2 images from each class, so none of the images in the validation set would be in the training set.

As a note, the final approach led to a substantial increase in training speed, which would help us evaluate metrics more often during training, play around more with hyperparameters, and build a better final ensemble, which is described later.

4.5. Architecture

To tackle this challenging classification problem we decided to transfer learn on a ResNet50 [1] network pre-trained on ImageNet. The latter technique was also employed, *mutatis mutandis*, by Noh *et al*. [5]. We removed all fully connected layers from the end of resnet, and appended one new fully connected layer with 8763 output nodes, corresponding to the 8763 classes of the dataset used in our final approach. During training we ”froze” all but the last n convolutional layers of ResNet50 and the fully connected output layer. We treated n as a hyper-parameter. Initially, we only used $l2$ regularization, which proved not to be

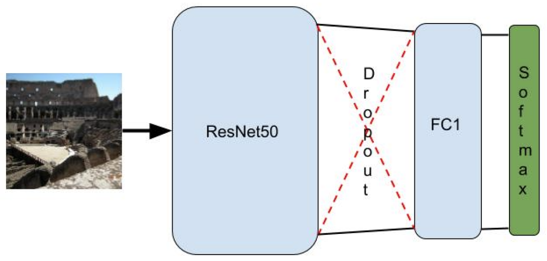

Figure 4. Architecture

sufficient to restrain overfitting, at least without significantly sacrificing performance. This lead us to incorporate an additional dropout layer [2] between the last layer of ResNet50 and the fully connected output layer. This lead us to settle down on the architecture depicted in Figure 4.

Moreover, we used an ensemble of models with the aforementioned architecture in order to make our predictions more robust and boost performance. The ensemble was trained over 3 different datasets, each of which was generated from the original dataset as described in the preprocessing section. The three models used different hyper-parameters but their core architecture was the same.

5. Training and Tuning

Training was carried out on a Google Cloud compute instance running 2 K80 GPUs, and 160GB of hard disk space in order to store all the data. We developed using Keras with a Ternsorflow backend.

Because of the size of the dataset and the limited (in comparison) compute resources available, hyper-parameter tuning had to be less rigorous than we would have liked. The sheer size of the training set, meant that even training for only one epoch lasted several hours, making it impossible to even do a coarse-grained random search over hyper-parameter space. For this reason, we wrote a custom *DataGenerator* class which allowed us to scale down epochs by a chosen factor. This essentially redefined an epoch to be a $\frac{1}{\textit{scale down factor}}$ pass through the training set. With these smaller epochs we were able conduct some very coarse grained tuning of regularization weight and learning rate.

The finer adjustments occurred across model, again due to the limitedness of compute resources vis-a-vis

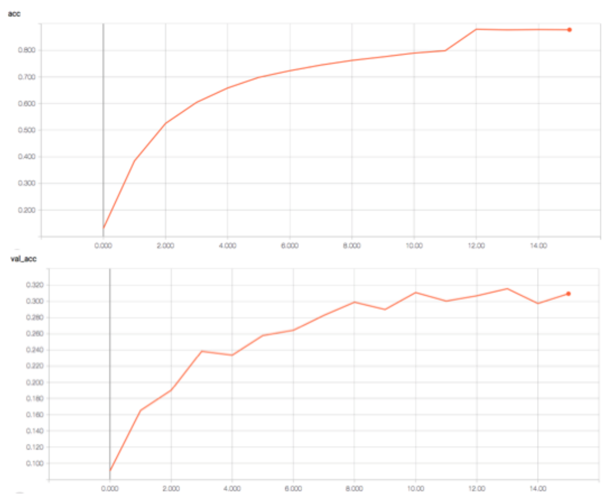

Figure 5. Train accuracy (top) and val accuracy (bottom) of model 1

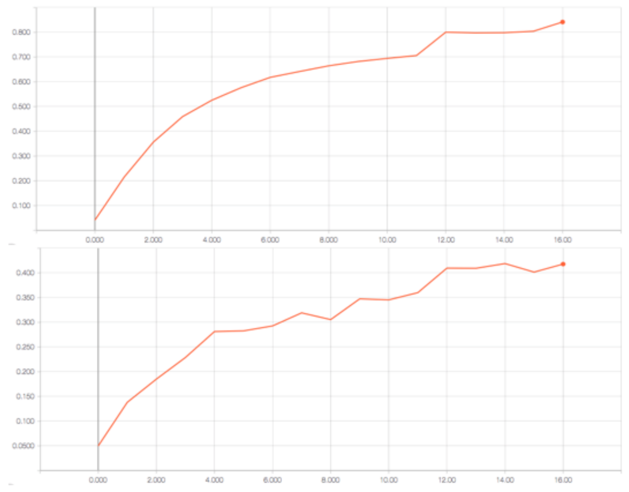

Figure 6. Train accuracy (top) and val accuracy (bottom) of model 2

the size of the dataset. After each model, we evaluated it's performance and made any relevant improvements on the hyper-parameters of subsequent models. For example, after training the first model of our ensemble, we noticed a significant amount of overfitting (as can be seen in figure 5), which we reduced by increasing both regularization weight and dropout rate. This resulted in less overfitting for subsequent models as seen in figure 6.

Furthermore, we used learning rate annealing during training, scaling down the learning rate by a factor of 0.2 every time accuracy began to plateau during training. This can be seen in the spikes in both training and validation accuracy at the 11th epoch on figure 6. In addition, to giving us some additional improvements in performance, the annealing allowed for more freedom in the choice of the initial learning rate, which was useful in mitigating some of the negative effects of insufficient hyper-parameter tuning.

6. Results

While not strictly a valid result metric, we must base the accuracy of our model on our validation data, which we used for hyperparameter tuning, since the test data from Kaggle did not specify the true labels for each image. For our final, we were able to achieve 40 percent accuracy on our validation set, which consisted of two images from each class in the training set. We think that this results are fairly good, given the very high number of labels, the compute power and time limitations that we had.

In terms of evaluation, the Kaggle competition proposes to evaluate the performance of models using the Global Average Precision metric. Since this is what the curators of the data sets propose, and since we have access to the leaderboard scores for the competition, it makes sense to use this as one of our evaluation metrics. That way well be able to get a sense for how good or bad our model is in comparison to contestants from the competition. In addition, well evaluate our model using more standard metrics like accuracy, confusion matrix, etc.

As of the Kaggle competition, we were able to get a final GAP (global average precision) of 0.011, placing us at the 198th spot on the final leaderboard. More than 450 teams participated on the competition. The best model trained on the first approached only reached a GAP of 0.001, so it was clearly a good idea to pivot from our original approach.

7. Conclusion and Future Work

To conclude, we were able to achieve a satisfactory performance given the constrained time and limited computing resources that we had. Our best model ended up being one trained on the data from our final approach. Unfortunately, our ensemble did not turn out to be our best model, but we think that with more time and effort, we could come up with more complex and complete ensembles, which we think have the potential to outperform, ceteris paribus, our current best performance. With more time and compute, we would also do a more involved hyperparameter tuning, exploring changes to the number of trained layers, and more combinations of dropout, regularization and learning rates. We would also like to explore ensembles where each model has a different architecture, as opposed to a homogeneous ensemble.

8. Contributions & Acknowledgements

Both Luis Varela and Alejandro Dobles contributed equally in this project.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[2] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[3] L. I. Kuncheva, . Arnaiz-Gonzlez, J.-F. Dez-Pastor, and I. A. D. Gunn. Instance selection improves geometric mean accuracy: A study on imbalanced data classification. 2018.

[4] X.-Y. Liu, J. Wu, and Z.-H. Zhou. Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39, 2008.

[5] H. Noh, A. Araujo, J. Sim, and B. Han. Image retrieval with deep local features and attention-based keypoints. *CoRR*, abs/1612.06321, 2016.