CS230

Xceptional Landmark Recognition

Tyler Yep
Stanford University
tyep@stanford.edu

Heidi Chen
Stanford University
hchen7@stanford.edu

Abstract

The 2019 Google Landmark Recognition Challenge [9] involves classifying popular landmarks from a massive dataset of images, with few training examples for any one landmark. In our model, we experiment with Xception networks, an alternate form of the Google Inception network proposed by Chollet [2], as well as compact bilinear pooling and various forms of image attention, inspired by the Sirius paper winner [1] from the 2018 challenge. Our submission achieved 53rd place in the official Kaggle competition on the private test set with a GAP of 0.06406.

1 Introduction

Landmark Recognition is a difficult visual classification task. Given the massive amounts of image data on online platforms such as Instagram and Facebook, we require smarter algorithms and models in order to classify images at scale. In this project, we aim to identify notable landmarks in images, which can help with image captioning or geotagging. From a broader perspective, advancements in landmark recognition can improve surveillance for national security or improve the performance of mapping systems. With this end goal in mind, we create a competitive model in the Kaggle challenge associated with this difficult task.

2 Related work

Xception is an architecture proposed by Chollet [2] that outperforms ResNet and Inception v3. The model proposed by Chollet [2], heavily leverages depthwise separable convolutions and residual connections for all layers, drastically cutting down on the computation time and the number of parameters in the model. Since our baseline model uses a ResNet50 as the core architecture, we decided to explore an Xception network as our starting point.

Xception was also used by last year's 19th place model [3] in the Kaggle challenge. This baseline froze the first 85 layers of Xception and then trained on the 70 remaining layers, and then added a generalized mean pool plus an activation layer as well as a customized loss function. We considered some of these embellishments for our model, but since the 2019 landmark dataset was nearly twice as large, we did not think we would find as promising results with any of these minor finetuning improvements.

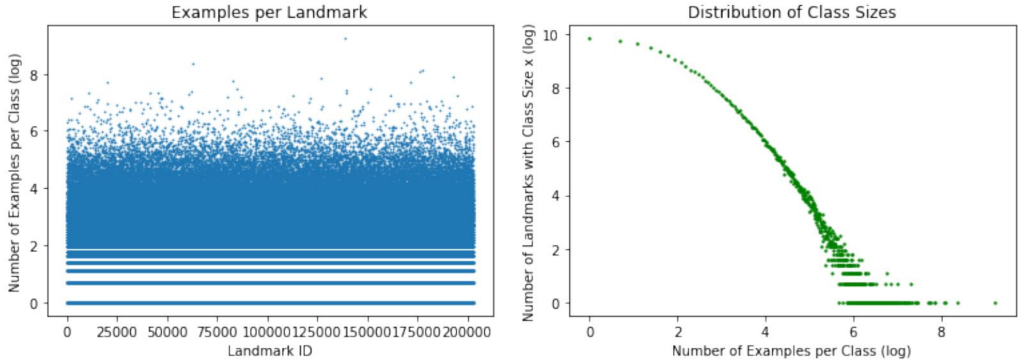
Sirius, the second place entry for the 2018 Landmark Recognition Challenge, implements Attentive Bilinear Convolutional Neural Networks with Re-Ranking [1]. We are interested in this model due to its novel attention architecture and use of DELF to re-rank confidence scores. The model first propagates input through a basic CNN (ensembled VGG16, ResNet) trained on ImageNets. The output activations are then fed through a bilinear pooling layer, which uses second order statistics to better model feature correlation. The output is convolved through a spatial attention network on guide images marked in the Google Dataset [9]; the final classification layer outputs a landmark

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

label and confidence value. To optimize confidences for GAP scoring, the entire test output is then re-ranked by DELF feature matching between test images and train images of the predicted class. Candidate pruning of indoor images, global features extracted from the bilinear pooling layer, and model ensembling were also used in the re-ranking. We believe that integrating Xception into this model and modifying the attention layers will lead to improvements on the 2019 challenge.

3 Dataset and Features

The Google Landmark Dataset [9] contains images with human-annotated labels of their associated landmarks. Images depict the landmark from various angles and locations within and outside the landmark. The dataset contains over 5 million images and over 100,000 landmark classes. Each class may be represented anywhere from one to thousands of times in the training set. Consequently, a major difficulty of this classification problem is handling a vast number of categories with few data points within these categories. The distribution of the entire dataset is depicted below.



We trained our models on the compressed 52 GB version of the dataset with 256x256 images, and only included landmark classes that contained at least 100 examples. The final training data subset consisted of 6512 classes and 1.2 million images. Our development data subset was randomly sampled from the remaining images, and our test set was the provided stage 2 test submission on the official Kaggle competition page.

In analyzing the provided train dataset, we found that there were some mislabeled and incorrectly rotated images. Based on our data exploration, we decided it would not significantly hurt our model to include these images, given that these mislabeled examples only made up a very small fraction of the dataset. Additionally, to help our network learn to classify landmarks with very few training examples, input images were center cropped, horizontally flipped, and width/height shifted randomly. We ensured that our data augmentation techniques preserved the landmark for the majority of the data.

4 Methods

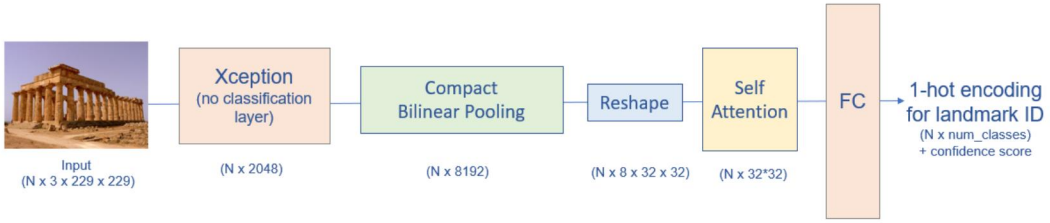


Figure 1: Final model architecture.

Our model loosely follows the structure of Sirius, which utilizes in the following order: a pretrained CNN, several pooling mechanisms, and various attention layers. In our model, we pass each batch of images through an Xception model pretrained on ImageNet, with 85 layers frozen and the remainder

tuned. Then, we use a compact bilinear pooling layer and try several different types of attention layers in order to train our model to focus on the relevant places in the image likely to contain landmarks. After our attention layers, we add a small number of full-connected layers in order to shrink the representation size down to the softmax layer output.

In compact bilinear pooling, we take the outer product of each Xception output vector with itself to functionally encode second-order descriptors of the landmark, and use Count Sketch dimension reduction to minimize computational costs, from 4194304 parameters (2048^2) to 8192. Based on methods proposed in [4], we adapt an existing compact bilinear implementation [6] in Tensorflow to take in 1D vectors instead of 2D feature maps.

For our attention layer, we experimented with and implemented Parameter-Free Spatial Attention proposed in [12], and adapted the attention module with spectral normalization implemented in Self-Attention GANs (SAGAN) [5] [11]. For our inputs to both models, we reshaped our 1D input vector int a series of 2D features maps of size 16 x 16 (Spatial) and 32 x 32 (SAGAN). Our final model ultimately used Self-Attention.

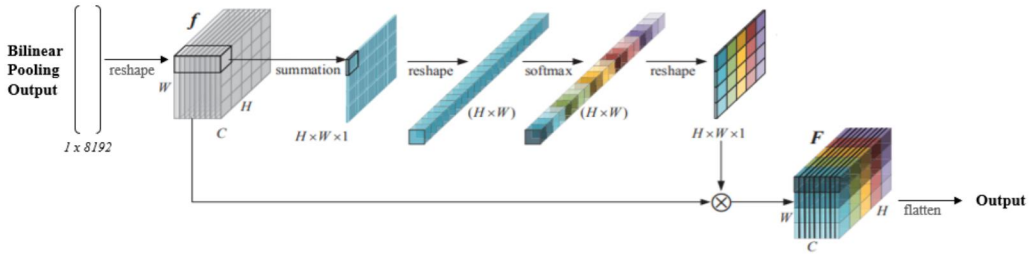


Figure 2: Parameter Free Spatial Attention (figure adapted from [12]).

In Spatial Attention, we sum over our 2D feature maps channel-wise to capture global localities. We then reshape the maps back to a 1D vector, calculate its softmax, and output the final reshaped 2D map element-wise multiplied by the input. We were interested in Spatial Attention due to its elimination of extra parameters and potential to capture high level features that may be perturbed locally by changes in camera viewpoint.

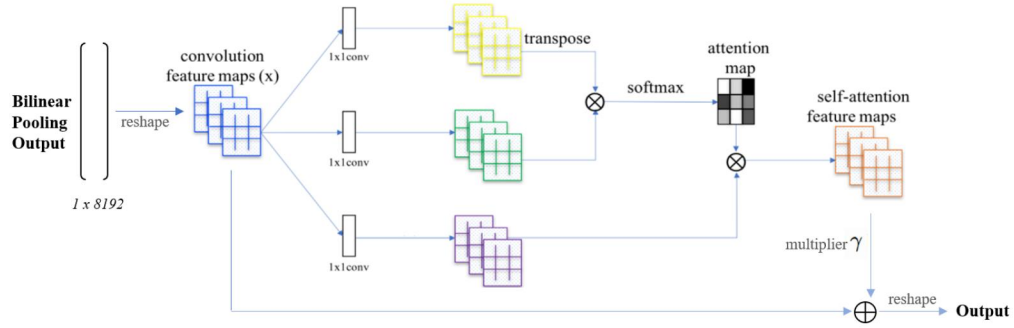


Figure 3: Soft Self Attention Module (figure adapted from SAGAN [5]).

In Self-Attention GANs, our feature maps are copied into three different branches which perform 1D convolutions. The first two branches are then transpose-element multiplied, then element-multiplied with the third branch to form the self-attention feature map. This map is then scaled by a hyperparameter γ and added to the original input. We chose to include Self-Attention GANs because their 1D convolutions have the potential to encode spatial information that may be lost in smaller kernel convolutions or zero padding.

5 Experiments/Results/Discussion

5.1 Evaluation

The Kaggle evaluation metric for landmark recognition is Global Average Precision (GAP). Given a list of N predicted landmark labels and corresponding confidence scores, the evaluation sorts the list in descending order of confidence and computes as follows:

$$GAP = \frac{1}{M} \sum_{i=1}^{N} P(i) rel(i)$$

where M is the total number of queries with at least one landmark from the training set visible in it (some test images may not contain landmarks), $P(i)$ is the precision at rank i, and $rel(i)$ is an indicator of prediction i's correctness.

5.2 Results & Discussion

After running our baseline ResNet50 model, we adapted and tuned an Xception network in Pytorch pretrained on ImageNet. We decided to allow the gradients to update the last 85 layers of the model (like the 19th place Xception network submission).

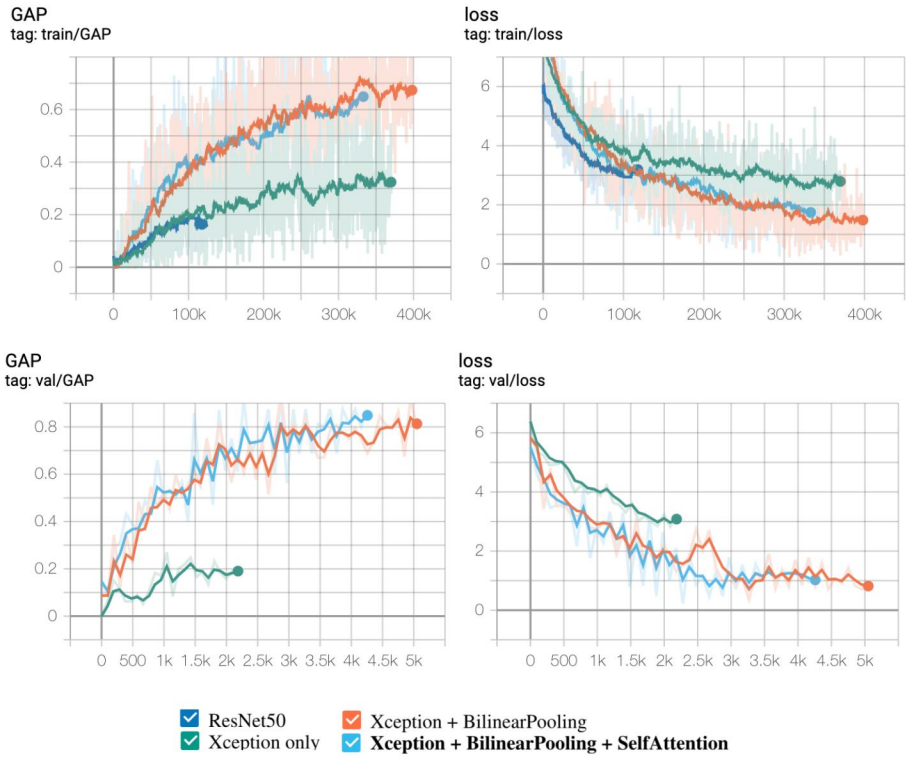


Figure 4: Relative train and dev set loss and batch GAP curves.

We used the Adam optimizer with a learning rate of 3e-4, evaluating on cross-entropy loss. Following Karpathy's [8] recommendation on using Adam with a learning rate of 3e-4, we decided to use a consistent optimizer and learning rate for all models, since we were focusing on the effects of major architectural changes to our model. We decided to run each model for roughly 300k iterations (20 epochs, where our epochs were defined as 15000 iterations, since a full pass through the dataset before plotting would be too computationally expensive.

In Figure 4, we see a smoothed version of our noisy train loss and GAP curves. While the graph's noise may be due to setting our learning rate too low to start with, we also suspect that the noise may

simply result from plotting our batch gap and loss too frequently. Since we were only able to use a batch size of 64, the enormous size of our dataset makes the noise is unavoidable. Despite the noisy curves, we are still able to compare them and see that our final model, Xception + BilinearPooling + SelfAttention performs significantly better than Xception alone and our baseline, ResNet50. However, we can also see that Xception + BilinearPooling and Xception + BilinearPooling + SelfAttention achieve very similar performances, indicating that adding self-attention does not significantly help our model focus on important parts of the image. From our table of results in Figure 5, we can see that spatial attention did not help the GAP or loss at all.

Model Metrics on Dev Set	**GAP**	**Loss**
ResNet50	0.241	3.183
Xception only	0.674	1.301
Xception + SpatialAttention	0.119	3.079
Xception + BilinearPooling	0.812	0.908
Xception + BilinearPooling + SelfAttention	**0.841**	**0.932**

Figure 5: Dev Set GAP & Final Loss for each model. The model with the best GAP is highlighted.

To investigate the reason behind this, we decided to create saliency maps for our base Xception model [10]. In Figure 6, we can see the model clearly upweighting the more characteristic parts of a landmark, such as the roof of the building or repeating structure.

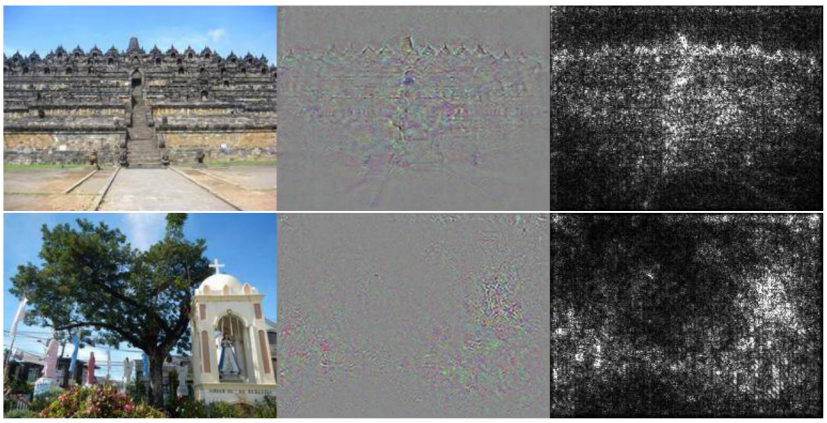


Figure 6: Saliency maps using vanilla backprop saliency.

From the visualizations, we can see that the Xception network on its own is already able to attend to key parts of the landmarks and update its weights to favor these types of structures. We can theorize that this may be because the Xception network is already able to maintain and prioritize disjoint pieces of spatial information from its more sophisticated depthwise separable convolutions and residual connections.

6 Conclusion & Future Work

From our results, we can see that Xception networks provide a powerful boost in performance on the Google Landmark Recognition Challenge over a baseline ResNet50. Furthermore, we see that Bilinear Pooling as well as various types of attention work well with the Xception network in achieving a higher overall GAP, however to a lesser extent as advertised by Sirius [1]. We suspect that due to its lower number of parameters, the Xception network may respond more favorably to increasing model complexity, which our attention modules did not necessarily add.

If we had more time on this project, we would likely explore various re-ranking techniques on our model. Though we attempted to ensemble our models and tried to reweight confidence scores with DeLF, we did not find much success . However, we may find more success using based on other similarity search algorithms such as Facebook's Faiss [7] or efficient implementations of kNN.

7 Contributions

Both Tyler and Heidi contributed equally to the project overall. Tyler spent a significant amount of time implementing the initial Xception model in both PyTorch and TensorFlow and loading the datasets. Heidi spent a significant amount of time designing the bilinear pooling and spatial/self-attention layers and creating the dev set and dev loss. Both Tyler and Heidi worked together to write the model architectures and create saliency maps and visualizations. Tyler and Heidi both worked together to write the report and create figures, and both appreciate all of the hard work the other has put into this quarter-long project.

References

[1] Bor-Chun Chen and Larry Davis. Deep representation learning for metadata verification. *IEEE Winter Applications of Computer Vision Workshops (WACVW)*, 2019.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[3] Jan Daldrop. Finetuning the xception cnn with a generalized mean pool (and custom loss function), 2018.

[4] Yang Gao, Oscar Beijbom, Ning Zhang, and Trevor Darrell. Compact bilinear pooling. *CoRR*, abs/1511.06062, 2015.

[5] Dimitris Metaxas Augustus Odena Han Zhang, Ian Goodfellow. Self-attention generative adversarial networks. 2019.

[6] Ronghang Hu. Compact bilinear pooling in tensorflow, 2018.

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[8] Andrej Karpathy. A recipe for training neural networks. 2019.

[9] Hyeonwoo Noh, Andre Araujo, Jack Sim, and Bohyung Han. Image retrieval with deep local features and attention-based keypoints. *CoRR*, abs/1612.06321, 2016.

[10] Utku Ozbulak. Pytorch implementation of convolutional neural network visualization techniques, 2019.

[11] David Keetae Park. Pytorch implementation of self-attention generative adversarial networks (sagan), 2019.

[12] Haoran Wang, Yue Fan, Zexin Wang, Licheng Jiao, and Bernt Schiele. Parameter-free spatial attention network for person re-identification. *CoRR*, abs/1811.12150, 2018.

https://github.com/TylerYep/landmark