CS231N: World Landmark Recognition

Omar Alhadlaq
Stanford University
hadlaq@cs.stanford.edu

Saud Alsaif
Stanford University
salsaif@stanford.edu

Abstract

In this paper, we tackle the problem of recognizing landmarks from the Google Landmarks Dataset. We experiment with basic convolutional neural networks as well as advanced architectures such as VGGNet and Xception to find out which model performs best. We explore different extensions and parameters for the models, pick the best model and apply performance boosting methods such as model ensembles and data augmentation. Our best model evaluated on the test set achieves an accuracy of $98.66\,\%$ *and a global average precision score of* $97.31\,\%$*. We also present error analysis and explore how the model visualizes the landmarks.*

1. Introduction

Recent advances in computer vision enabled technologies such as face recognition, which is now part of social networks and smart phones to help people organize their photos by faces. Now, as many people travel to famous landmarks, the next step is to enable identifying landmarks in photos and help people organize their landmarks photo collection and know more about these landmarks.

As part of the Google Landmark Recognition Challenge [4], Google released over a million images of landmarks all around the world. This dataset enables us to build systems that can identify landmarks from pixels. The problem can be treated as a complex classification problem with 15K labels.

The inputs to our models are images from the landmarks dataset. We use convolutional neural networks (CNN) architectures to classify each image to a landmark in the dataset. We expect the models to perform well on images with easy angles in the top landmarks, like a front shot of the Colosseum. At first, the models will have a hard time classifying images with obstructions. We also expect the models to have difficulties classifying landmarks that are in close proximity of each others and share some common geographical features.

The structure of this paper is as follows: Section 2 presents recent work in classification problems, methods to boost performance, and software architectures. Section 3 extensively details the dataset, shows some examples, and presents the subset we work on as well as train-dev-test divison and data preprocessing. Section 4 presents the models we use to tackle the problem, the performance boosting methods, the hypotheses we want to investigate, and the evaluation metrics. Sections 5 and 6 present the quanitative and qualitative results in addition to analysis and discussion. Finally, Section 7 concludes the paper.

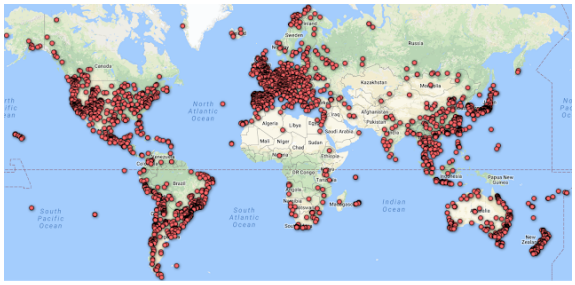

Figure 1: Geographic distribution of landmarks in the dataset [5].

2. Related Work

The problem is a classification problem on a very specific object: landmarks. The inputs are raw images from the dataset that has been preprocessed to have the same size. We use CNNs to extract the features from these images and infer the spatial properties before passing these features to the classifier. CNNs have shown to be the most powerful tool in extracting image features [8]. In recent years, different CNN architectures have been proposed to tackle the classification problem, such as VGGNet, ResNet, and Inception [12, 6, 13]. These architectures vary in depth, size of filters, and the way layers are connected. More recent architectures aimed at getting the best of both worlds and combined the ResNet and Inception components, such as Xception, which used depthwise separable convolutions and residual connections, presented a more efficient use of

parameters, and outperformed previous hybrid architectures like Inception-V3 [2, 14].

Training CNNs from scratch can be resource intensive, many opt for using transfer learning to import filters and weights from classifiers that has been trained on large datasets, like Imagenet [10]. Using pretrained weights helps tremendously especially in the first layers where the filters are extracting low level features that are common between all images. Transfer learning is an efficient way to initialize the weights of the network [11], then the choice to freeze them or train them is a hyperparamet thas is subject to investigation.

Training one model on the dataset is not sufficient for achieving high performance. A common method to boost performance is to ensemble multiple classifiers or multiple snapshots of the same classifier at different points in the training [9, 7]. Another boosting technique often used in computer vision problems to help models generalize better to unseen examples is data augmentation. Data augmentation is an indirect regularization method where the images are distorted in various ways, such as mirroring, stretching, blurring, and applying saturation. This method is commonly used in training almost all CNN architectures [8, 12, 6].

In this work, we use software frameworks that simplify importing models and weights and building upon them. Mainly, we use Tensorflow and Keras to build our training and evaluation pipeline and optimize the computations [1, 3].

3. Dataset

The dataset we use is from the Google Landmark Recognition Challenge [4]. The dataset contains 1,225,029 training images and 117,703 test images. The images span 14,951 landmarks that are spread all over the world as can be seen in Figure 1.

There are many challenges associated with the dataset. First, the imbalance in labels as illustrated in Figure 2. We are aiming to classify the top 100 labels, which include 400,000 images. The least common label among the top 100 has 1094 samples, and the most common label has 50,337 samples. The reason we picked the top 100 labels is to fit to the available computing resources, as well as to have enough samples for each label to be properly classified. Second, the images for one landmark vary a lot as shown in Figure 3. For one landmark we can have images from different angles, different distances, and different orientations. The images can also be taken from inside the landmark, which can have different features than the outside. The images might only show a small portion of the landmark, and it can be obstructed by other objects like trees and people. All of these variations will increase the difficulty of identifying a landmark, since the sets of features that define a landmark can vary depending on the angle and where the shot was taken from.

We wrote a script that downloads the images of the 100 most popular landmarks, and saves a downsized copy of each of them at 224×224 pixels. These images make up a dataset of 400k data points. We then split the dataset into: train, development, and test sets, where the train set has a representational 95% portion of the data (380k), the dev set has 1% (4k), and finally the test set has 4% (16k). Even though there is a high variance between the number of samples for each landmark, overall the cumulative distribution shows that we need to classify all landmarks correctly to achieve high accuracy. In other words, the model cannot overfit to the few landmarks with large numbers of samples and get an impressive accuracy. This is illustrated for the training and development sets in Figure 4.

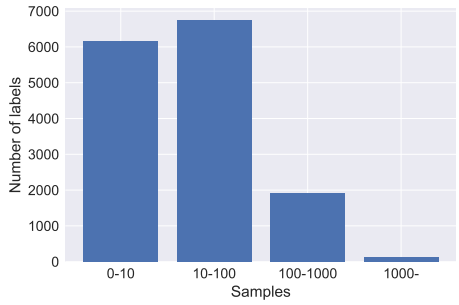

Figure 2: Distribution of samples between the labels. 12,915 labels have less than 100 samples. There are 6 labels with over 10k samples, 2 of them have over 50k samples.

4. Methods

We examine multiple model structures for solving the landmark recognition problem. In all of the model structures we use, we apply the softmax function on the scores at the output layer. The softmax function takes a vector of real-valued scores in $\mathbb{R}^d$, and squashes it to a vector of values between zero and one that sum to one, that is also in $\mathbb{R}^d$. The ith element of the output vector can be interpreted as the probability of the input being classified as the ith label, and is defined as:

$$f_i(z) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Also, for all of our models, we optimize for the multi-class cross entropy loss function. The loss of one example is:

$$L_i = -\log(f_{y_i}(z))$$

where y_i is the ground truth label for that example.

Figure 3: Example of images from one landmark, Alhambra Palace. The samples can contain shots of the landmark from different angles and distances, from the inside and from the outside. The images can also contain other objects, like people, trees, or other buildings.

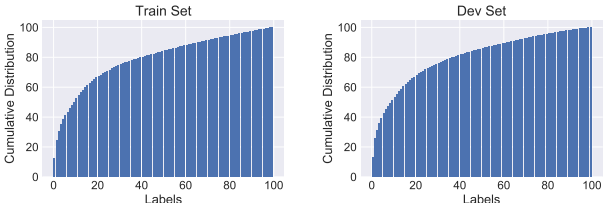

Figure 4: Cumulative distribution of labels of the training and dev sets. The x-axis labels are in descending order in terms of number of samples. This distribution shows that 80% of the labels cover almost 90% of the data.

4.1. Model Architectures

4.1.1 Basic ConvNet

We implement a basic ConvNet as our baseline. This model has two Conv-ReLu-MaxPool layers with 32 (3×3) filters in each, followed by another Conv-ReLu-MaxPool layer with 64 (3×3) filter. The output is then flattened and passed to 3 fully-connected layers with 4096, 4096, and 100 units. The first two fully-connected layers are each followed by a dropout layer. Finally the output is passed to a softmax layer.

4.1.2 VGGNet

We also explore the realm of advanced ConvNet architectures. VGGNet has a uniform architecture that consists of 16 convolution layers, all of which has a 3×3 filters [12]. Every two or three convolution layers are followed by a 2×2 MaxPool layer. The number of filters for the convolution layers starts with 64 filters and doubles after each MaxPool layer reaching 512. The output is then flattened and passed to passed to 3 fully-connected layers with 4096, 4096, and 100 units. The first two fully-connected layers are each followed by a dropout layer. Finally the output is passed to a softmax layer. See Figure 5 for an illustration of the architecture.

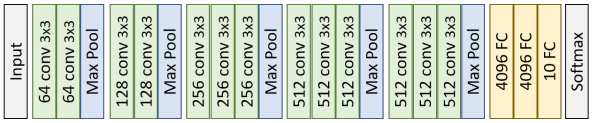

Figure 5: The architecture for VGGNet with 16 convlolutional layers.

4.1.3 Xception

The architecture of the Xception ConvNet is a linear stack of depthwise separable convolution layers with residual connections. A depthwise separable convolution consists of a depthwise convolution followed by a pointwise convolution. The depthwise convolution is a spatial convolution performed over each channel independently. The pointwise convolution projects the channels output by the depthwise convolution onto a new channel space, an example of this is a 1x1 convolution. The full architecture has 36 convolutional layers, used as a base of the network for feature extraction. These 36 layers are structured into 14 modules that have linear residual connections around each one of them, except for the first and last modules. An illustration of a single module is given in Figure 6. The network was designed for image classification tasks and therefore this convolutional base is followed by a fully-connected softmax layer.

4.2. Transfer Learning

For the VGGNet and Xception architectures, we use pre-trained weights for these models on the Imagenet dataset. We then freeze all layers and only train the last fully-connected layers to make predictions for our landmarks recognition task. We hypothesize that pre-trained image networks act as a generalized feature extractor which can be very helpful for extracting higher level features with less data and less train time. In addition to training models with freezed convolutional base, we also examine training unfreezed full networks over the existing Imagenet weights–

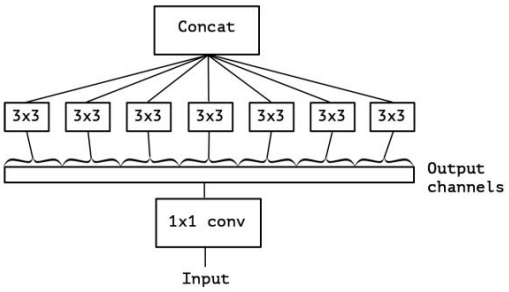

Figure 6: An Xception module with one spatial convolution per output channel of the 1x1 convolution [2]

except for the fully-connected layers; where we always start from randomly initialized weights.

4.3. Model Extensions

The standard Xception model has a one fully-connected layer at the end. In this paper, we try different extensions to the model, namely we examine the effect of adding three fully-connected layers with dropout and how it compares to including just a single layer.

4.4. Data Augmentation

To boost the performance of our models, we try to reduce overfitting by implementing multiple data augmentation techniques. Specifically, for every training image, independently of other images and other epochs, we flip the image with a probability of 0.5. Additionally, we change the brightness by at most 12%, where the amount is randomly and uniformly decided. Similarly, we change the saturation by a saturation factor randomly and uniformly picked from the rage $[0.5 - 1.5]$. We train our model without data augmentation, and then we later introduce it to our models to see how effective it is for our task.

4.5. Ensemble Learning

We hypothesize that ensembling our best performing models would increase the performance by a few percentages. Thus, we take the prediction scores of our best performing models and average them weighted by the accuracy of the corresponding model on the dev set. The final combined scores are then used to make the prediction for the ensembled model.

4.6. Evaluation Metrics

We use two metrics for evaluating the models. First, for evaluation on the validation set and checking for model progress we use accuracy. Second, for overall performance on the test set we follow the competition's evaluation metric, the global precision average (GAP). For each test image, we predict at most one landmark label and a corresponding confidence score. Each prediction is treated as an individual data point in an ordered list of predictions (label and confidence pairs), and we compute the average precision of that list.

If the model has given N predictions, then the GAP can be computed as:

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

where:

- **N** is the number of predictions returned by the model.
- **M** is the number of images in the test set with at least one landmark (some images may include no landmarks).
- **P(i)** is the precision at rank i of the list. It is the cumulative sum of correct predictions over i.
- **rel(i)** is the relevance of prediction i; its 1 if the i-th prediction is correct, and 0 otherwise.

5. Results and Discussion

In this section we present the results of all experiments we perform. For each model we first tune hyperparameters like regularization, learning rate, dropout rate, optimizer, and batch size on a small subset of the training set that has 30k images. We then pick the best hyperparameters for each model. The best hyperparameters of all models we experiment with are presented in Table 1. With the exception of the basic model, we train the variations of VGGNet and Xception on the small subset first, then train the best model among them on the full training set. This is done to save time since these models require an extensive amount of training hours. The first result we obtain is aimed at setting the baseline. The basic model achieves a dev accuracy of 79.5%. The history of accuracy is shown in Figure 7.

After setting the baseline we want to pick the best advanced model. We have multiple variations of VGGNet and Xception. All of them are trained on the small training subset to accomodate time constraints. The variations are as follows: First we want to know if it is better to freeze the pretrained Imagenet weights or to update them as we train the model. This variation was applied on VGGNet and Xception. Second, we want to know if it is better to add only one dropout and dense layer on top of Xception (Xception1), or to add three dense layers where the first two layers are followed by dropout (Xception3). This variation was only applied on Xception since VGGNet has three dense layers at the end by design.

Model	Regularization	Learning Rate	Dropout	Optimizer	Batch size
Baseline	5×10^{-5}	1×10^{-3}	0.5	Adam	64
VGGNet	5×10^{-5}	1×10^{-4}	0.5	Adam	64
Xception1	5×10^{-5}	1×10^{-4}	0.5	Adam	32
Xception3	5×10^{-3}	1×10^{-4}	0.5	Adam	32

Table 1: Best hyperparameters for each model variation.

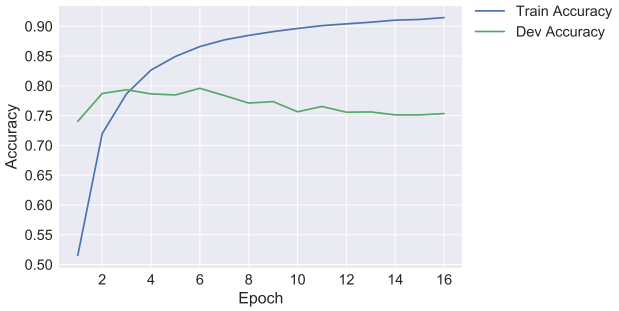

Figure 7: Accuracy of basic model. In the first few epochs the dev accuracy is better than the training accuracy because the training accuracy is a moving average of the accuracy of each training batch.

	CE Loss		Accuracy	
Model	train	dev	train	dev
VGGNet freezed	0.86	1.11	84.08 %	79.02 %
VGGNet unfreezed	0.57	**0.82**	92.24 %	**86.95** %
Xception1 freezed	0.12	2.4	96.79 %	59.28 %
Xception3 freezed	3.99	4.49	68.95 %	58.43 %
Xception1 unfreezed	0.04	2.85	98.83 %	66.08 %
Xception3 unfreezed	0.5	5.06	98.39 %	66.12 %

Table 2: Best loss and accuracy evaluated on the small training set and the dev set for each model variation.

We run the six variations on the small training subset after tuning the hyperparameters of each variation. The accuracy evaluated on the dev set is shown in Figure 8. The best variation among them is VGGNet with unfreezed weights. The full results with the training accuracy and loss is presented in Table 2. Something interesting we found is that the Xception models are hard to train. We tried many hyperparameters and many variations, yet they tend to overfit and perform poorly on the dev set. Xception1 seems to be learning and generalizing better than Xception3. In general, unfreezing the Imagenet weights tends to give better results.

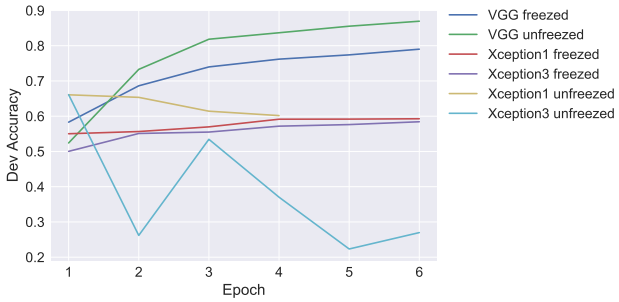

Figure 8: Accuracy of multiple variations of VGGNet and Xception models trained on a small subset of the data and evaluated on the dev set.

Now that we have an answer for which variation is best. We can proceed to train the best variation on the full training data (380k images). Another question that we want to answer from the best model is the effectiveness of data augmentation. We train VGGNet with unfreezed weights on normal training data and on the augmented training data (VGGNet-DA). The dev accuracy results are shown in Figure 9. We find that applying data augmentation gives a very small boost to dev accuracy.

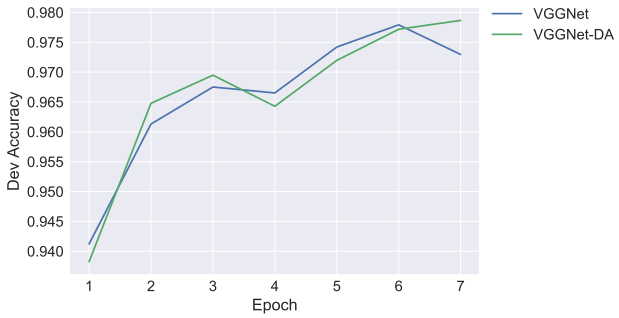

Figure 9: Accuracy of VGGNet and VGGNet with data augmentation. Data augmentation slightly improves the performance of VGGNet.

Model	Dev Accuracy	Dev GAP
Baseline	79.59 %	63.31 %
VGGNet	97.79 %	95.61 %
VGGNet-DA	97.86 %	95.68 %
Ensemble1	98.76 %	97.31 %
Ensemble2	**98.80%**	**97.56%**

Table 3: Best dev accuracy for models after performance boosting. Ensemble1 combines all three models, ensemble2 only combines the VGGNet models.

Finally, the last performance boosting we try is model ensembles. We try two ensembles, one combines the baseline, VGGNet, and VGGNet-DA (Ensemble1). The other only combines VGGNet and VGGNet-DA (Ensemble2). The dev accuracy and dev GAP results are shown in Table 3. Even though GAP is supposed to be applied on the test set only, we apply it on the dev set to show its correlation with the accuracy and get a sense of what values are expected.

We are confident that our best model is the ensemble that only contains VGGNet and VGGNet-DA. This is the model that we will try on the test set and find the final test accuracy and test GAP. The best model achieves a test accuracy of $98.66\,\%$ and a GAP score of $97.31\,\%$. For comparison, the winner of the Kaggle competition achieved a GAP score $30.4\,\%$. The reason there is a huge difference is the number of classes. In the original competition there are 15k classes, while we are classifying 100 classes. The GAP score depends on the order of prediction and is affected by the class imbalance, which is hugely present in the full dataset as shown in Figure 2. Overall, our best model succeeds in classifying landmarks with high confidence. The errors are mainly stemming from landmarks that are adjacent to each other or landmarks that are very similar despite being in different parts of the world. These errors and more analysis of the best model are explored in the next section.

6. Analysis

In this section of the paper, we show qualitative results and more in-depth analysis. The results shown in this section are obtained from our best performing model.

6.1. Confusion Matrix

Figure 10 (left) visualizes the results of our model on the dev set as a confusion matrix, where in cell i, j of the matrix, it shows how much of class i is classified as class j. We can see that, for all classes, the diagonal, by far, has the highest scores. This indicates that the model is performing well on all classes. In other words, the high accuracy results are not due to performing well in classes with higher number of examples.

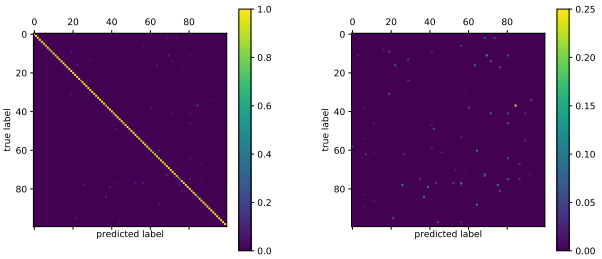

Figure 10: The normalized confusion matrix evaluated on the dev set (left), and the same matrix with misclassifications emphasized.

Moreover, Figure 10 (right) depicts the same confusion matrix, while ignoring the correctly classified samples and emphasizing the misclassified ones. The highest misclassification rate between two classes is the one between class 37 and class 84, where it misclassifies the image about $\%25$ of the time. The first class is the Krakw Cloth Hall and the other is the Town Hall Tower of Krakw. Figure 11 shows a screen shot from Google Maps that clearly shows that the two landmarks can very frequently both appear on the same picture, which is a sensible explanation for why the model is confusing the two classes. Most images of each of the two classes include the other one, which shows that there is not much room for improvement. Even for humans it is difficult to tell from an image which landmark of the two it is, given that the pictures are taken from different angles and does not always show the landmark completely.

Another two classes that are commonly confused are 56 and 77. The first class is the skyline of Shanghai and the other class is the skyline of Chicago (shown in Figure 12). The two classes are for city skylines, but a human can easily distinguish between the two. Therefore, although the misclassification rate here is much lower than the one in the previous paragraph, this one has a room for improvement.

6.2. Visualizing the Network

6.2.1 Saliency Maps

Here, we visualize the saliency maps of three randomly picked examples, shown in Figure 13. As we can see, in all three examples the model is clearly highly activated only by the pixels of the actual object in each case. In the Colosseum example the model ignores everything including people and surroundings, and focuses on the small Colosseum structure in the background. This demonstrates that the model has learned well to identify some of the landmarks in our dataset and ignore obstructions such as people.

Figure 11: A screen shot from Google Maps showing two landmarks in our dataset; Cloth Hall and the Town Hall Tower of Krakw.

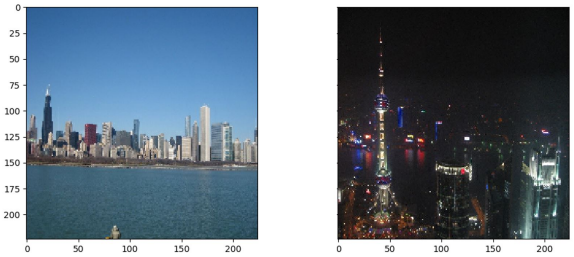

Figure 12: Two pictures from our dataset showing the skyline of Chicago (left) and the skyline of Shanghai (right).

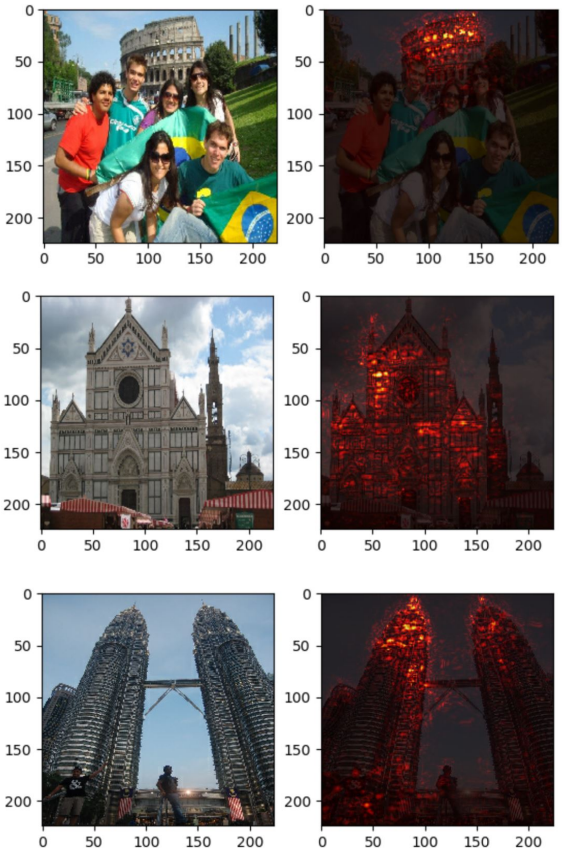

Figure 13: The saliency maps for the Colosseum, the Basilica di Santa Croce, and the Petronas Towers, respectively.

6.2.2 Class Visualization

Here, we visualize some of the classes learned by the model, shown in Figure 14. First, the model attempts to visualize the ancient city of Selinunte. The visualization shows different parts of the landmark where one can easily see the pillars that identify Selinunte. Second, The visualization of Hagia Sophia clearly shows the identifying features of a mosque, namely a minaret and a conical dome. Finally, one can see different parts of a castle in the visualization of the Vianden Castle.

We can see that the model is able to draw and visualize some of the most identifying features of the landmarks. This visualization demonstrates that the model has learned these identifying features for the landmarks in the dataset.

7. Conclusions

In this paper, we explored several variations of CNN architectures such as freezing or unfreezing the pretrained weights, extending the Xception model with one or three dense layers with dropout. We found that Xception tends to perform poorly on the dev set, while VGGNet with unfreezed weights performed best among all variations. We applied ensemble methods and data augmentation on VGGNet and achieved almost $1\,\%$ improvement in dev accuracy and $2\,\%$ in dev GAP. We showed that the errors stemmed from landmarks that are adjacent in location, or landmarks that look fairly similar, like city landscapes. The best model was able to capture the essential features of the landmarks as shown by the saliency maps and the class visualizations.

Areas of future work include investigating more into why Xception has poor generalization performance. Adding more classes from the landmark dataset and stress testing the models when the number of classes is huge.

Code Repository

Our code is available at:
https://github.com/hadlaq/Google-Landmarks-Recognition

Contributions

Omar worked on dataset preprocessing, data pipelining, and model visualization. Saud worked on dataset explo-

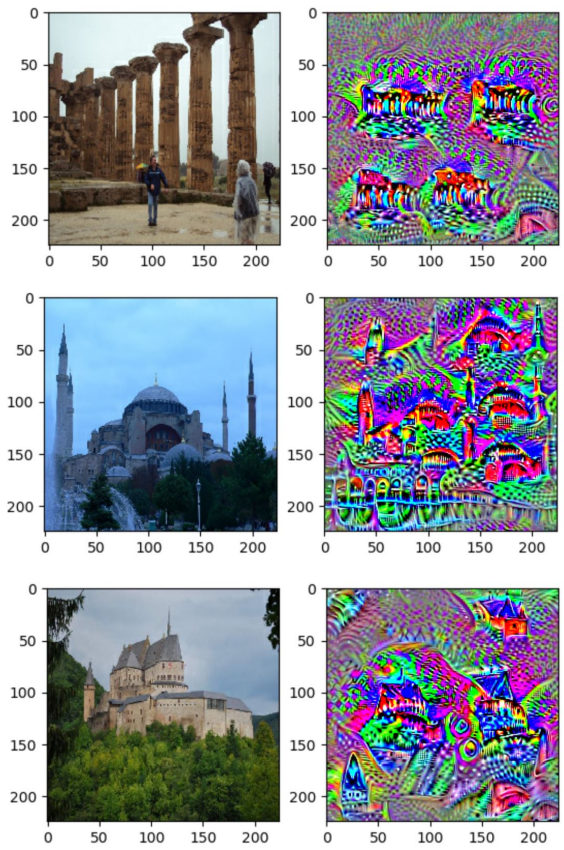

Figure 14: The visualization of the classes of Selinunte, Hagia Sophia, and the Vianden Castle, respectively.

ration, hyper-parameter tuning and experimentation. All other parts of the project were equally divided efforts by all members.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions.

[3] F. Chollet et al. Keras. `https://keras.io`, 2015.

[4] Google. Google Landmark Recognition Challenge. `https://www.kaggle.com/c/landmark-recognition-challenge`, 2018.

[5] Google AI Blog. Google AI Blog: Google-Landmarks: A New Dataset and Challenge for Landmark Recognition. `https://ai.googleblog.com/2018/03/google-landmarks-new-dataset-and.html`, 2018.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[11] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.