Final Project. Google Landmark Recognition Challenge

Adil Nygaard
Stanford University
CS231N
adiln@stanford.edu

Uzair Navid, Aamnah Khalid
CS230
unavid@stanford.edu
aamnah@stanford.edu

Abstract

Google's landmark recognition challenge is an unprecedented opportunity to build robust classifiers and models to predict images of famous buildings around the world. In this project, we have implemented a variety of models in order to tackle this task, as well as used several visualizations to qualitatively analyze our model. Maximum activation, saliency maps and occlusion sensitivity show that our model has learned to identify meaningful features of the architecture of a class. For non-landmark images classified to a landmark class, we propose using DELF[14] to extract the landmark's features and comparing extracted architectural features to reject non-landmarks.

1. Introduction

In the Spring of 2018, Google released a challenge on Kaggle, calling for a comprehensive classifier on recognizing and categorizing landmark images. These images, are part of a wider initiative to create more robust image classification systems.Described as the largest worldwide dataset for recognition of landmarks, it contains more than 2 million images. [2] These contain 30,000 landmarks from across the globe, approximately thirty times more classes than in most publicly available datasets. The size of the dataset has helped tackle a longstanding problem of a lack of large annotated datasets.

2. Related Work & Literature Review

The quintessential problem of image classification is one that has been studied extensively over the years, and continues to be one of the core areas of focus in Artificial Intelligence. We were thus able to consult a wealth of literature with respect to techniques and approaches, and analyze how these could be mapped to our problem of classifying landmarks.

Li et al's paper on Landmark Classification sheds light on the level of improvement that can be gained by utilizing extra features such as captions and keywords. [13] However, the classification approach based on k-means clustering did not seem as promising as some of the deep learning models we explored, partly because of we hope to generalize our model to various landmarks. [13] Images of landmarks can look fairly different based on image capture conditions, so a deep learning-based approach seemed to better tackle these variations.

We also considered approaches such as object-based image retrieval, as researched by Philbin et al [15]. However, it appeared that, for now, these systems require more research into evaluating the effectiveness of the ranking function "as the corpus size grows." [15] Similarly, Torralba et al's binary code approach seemed promising for general scene recognition tasks, but we did not feel it would be the optimal tool for our problem. [18] Part of the reason for this is that their objective is to cluster semantically similar images, such as all images of flowers, and includes techniques such as a pixel-wise vote between the nearest neighbors. [18] We felt that this could cause us to underfit the data, given that many landmarks can be seen as being semantically similar architectures, and occlusion could affect the pixel-wise voting mechanism.

On the other hand, deep learning techniques such as Deep Residual Networks seemed can able to handle over 1 million images, while maintaining an ensemble "3.57% error on the ImageNet test set."[10] Similarly, Xception models also showed promising results on datasets spanning 350 million images across 17000 classes, highlighting a high propensity for generalizing. [8] These are thus some of the models we decided to implement.

The success of the deep convolutional neural networks in ImageNet classification led us to decide to use ImageNet weights, which are already available in Keras, to initialize the weights in our trainable models. [12] Furthermore, we decided to use dropout as our regularization technique, which drops nodes from layers with a given probability, and can help prevent our network from overfitting by preventing complex co-adaptations. [17][11] Overfitting has been a concern for our project due to the varying amount of dataset images available classes.

We studied visualization techniques next in order to do find the best methods for our qualitative analysis of the model's results and high-level features. One approach suggested by Erhan et al was to employ activation maximization. [9] Matthew D. Zeiler and Rob Fergus note that this method "find(s) the optimal stimulus for each unit by performing gradient descent in image space to maximize the unit's activation". [19] However, they criticize it for requiring careful initialization and not giving any information about the unit's invariance. They instead suggest using occlusion which helps check if the network is truly identifying the location of the object being classified or merely its surrounding context. [19] Zelier et al., also use deconvolutional networks to illuminate trends that activate particular units.However, Zhou et al., in a more recent paper points out that this approach only analyzes convolutional layers, "ignoring the fully connected thereby painting an incomplete picture of the full story." [20] They instead support generating class activation maps by employing global average pooling.

Since this approach seemed more promising and we had limited computation power, we chose to implement it over the former. Furthermore, saliency maps, described by Simonyan et al., also involve deconvolutions so we decided to implement them to compute "the spatial support for a given class in a test image" by mapping pixels in the input to the output. [16]

We also studied the use of an attentive local feature descriptor suitable for large-scale image retrieval, referred to as DELF (DEep Local Feature). [14] The CNN based local-features extractor can be learned with weak supervision using image-level labels only. Noh et al have proven it to be robust against queries that have no correct match in the database. The technique uses key-point selection of densely extracted features to avoid clutter and has been trained on the Google Landmark dataset with higher precision than most local and global feature descriptors. A major challenge with landmark recognition is the lack of a good dataset of non-landmark images. Due to its resistance to false positives, DELF can be used to reject non-landmark images wrongly classified as landmark images.

3. Data

3.1. Kaggle Dataset

The challenge was composed of a large dataset of holiday photos that included famous landmarks, and the objective was to classify an image to the correct landmark. The dataset was rather large, originally containing over 14500 different classes (with each class representing a landmark), and 1.3 million images in the entire training dataset. Image capture conditions vary as the images include as noise, "occlusions, different viewpoints, weather, and illumination."[2]. Some landmarks only had one or two photos in the training set, and some test landmarks were not contained within the training dataset at all. Common occlusions included people and vehicles. The training dataset was already labelled and could be used for Keras models which require correctly labelled images. The dataset was provided to us in a CSV file, which had the name of each image, the website url to download it from and the class that the image belonged to. We wrote a script to download each image with the following labelled format:

[class number]_IMG_[image number].jpg

with the class number and the image number being replaced by their actual values respectively.

3.2. Our datasets

For training, we sorted the CSV file[1] to select 50 different landmarks each of which had over 500 images in the dataset. Comprising of 25,000 images, this dataset was the larger of the two we trained on. We also had a smaller dataset composed of 10 classes, each of which had twenty-five or more images within them, leading to a dataset of approximately one thousand images.

Using a sklearn [21] package, we split our data into training, validation, and test sets. The test set was approximately 15% of the dataset. We split the remaining training set into 85% training and 15% validation. Our script placed each of these into separate folders to allow a flow from directories to data generators available in Keras.

3.3. Image Augmentation

To account for noise from image capture conditions, we utilized zoom, horizontal flips, and rotations for image augmentation.

4. Methods/Experiments

4.1. Variable Testing

We started tuning parameters on the ResNet model available on the CS230 github [3]. We modified this slightly to fit with our smaller 10 class dataset by changing the way it reads in input, and then we used it as the first experiment of our project. The metric we maximized for all our experiments was accuracy. We tried changing image size, batch size and the number of epochs. The results were as follows:

64 x64 images:

batchSize = 32	epochs = 10
Train accuracy: 0.998	loss: 0.044
Dev accuracy: 0.835	loss: 0.493
Test accuracy: 0.849	loss: 0.486
batchSize = 32	epochs = 15
Train accuracy: 1.00	loss: 0.015
Dev accuracy: 0.858	loss: 0.434
Test accuracy: 0.891	loss: 0.410
batchSize = 16	epochs = 10
Train accuracy: 0.998	loss: 0.057
Dev accuracy: 0.827	loss: 0.563
Test accuracy: 0.866	loss: 0.503

128 x128 images:

batchSize = 32	epochs = 10
Train accuracy: 0.988	loss: 0.058
Dev accuracy: 0.866	loss: 0.497
Test accuracy: 0.857	loss: 0.408
batchSize = 32	epochs = 15
Train accuracy: 1.00	loss: 0.013
Dev accuracy: 0.843	loss: 0.520
Test accuracy: 0.857	loss: 0.432

[1] Refer to sort.py

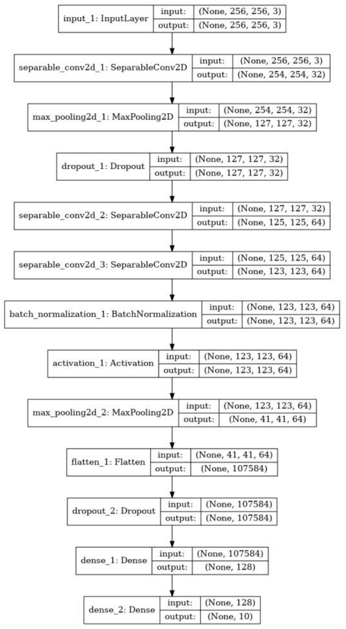

input_1: InputLayer | input: (None, 256, 256, 3) | output: (None, 256, 256, 3)
separable_conv2d_1: SeparableConv2D | input: (None, 256, 256, 3) | output: (None, 254, 254, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 254, 254, 32) | output: (None, 127, 127, 32)
dropout_1: Dropout | input: (None, 127, 127, 32) | output: (None, 127, 127, 32)
separable_conv2d_2: SeparableConv2D | input: (None, 127, 127, 32) | output: (None, 125, 125, 64)
separable_conv2d_3: SeparableConv2D | input: (None, 125, 125, 64) | output: (None, 123, 123, 64)
batch_normalization_1: BatchNormalization | input: (None, 123, 123, 64) | output: (None, 123, 123, 64)
activation_1: Activation | input: (None, 123, 123, 64) | output: (None, 123, 123, 64)
max_pooling2d_2: MaxPooling2D | input: (None, 123, 123, 64) | output: (None, 41, 41, 64)
flatten_1: Flatten | input: (None, 41, 41, 64) | output: (None, 107584)
dropout_2: Dropout | input: (None, 107584) | output: (None, 107584)
dense_1: Dense | input: (None, 107584) | output: (None, 128)
dense_2: Dense | input: (None, 128) | output: (None, 10)

Figure 2: Layers in custom model

We found a tendency to overfit with the larger number of epochs. Increasing image size seemed to have no discernable effect while being computationally expensive. We speculated that this may be due to the way the model was implemented, and it not necessarily was disadvantageous to have a larger number of pixels, since that provides more data to train on.

4.2. Model Testing

Model	Training Accuracy	Test Accuracy
Separable Convolution 2D	0.95	0.79
VGG 16	0.93	0.99
ResNet 50	0.99	0.98
Xception	0.96	0.90

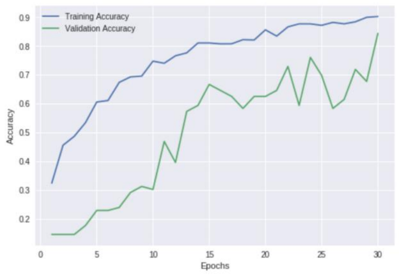
Figure 1: Results on 10 Classes

We started testing models on the 10 landmarks dataset. We created a custom model (see figure 1), implemented a VGG16, an Xception model[8], and a residual neural network.[10] Each model used a Softmax classifier, suitable for a multiclass problem. The activation value for the i^{th} class is calculated using the following equation, where x is the set of input values for the activation layer:

$$f(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

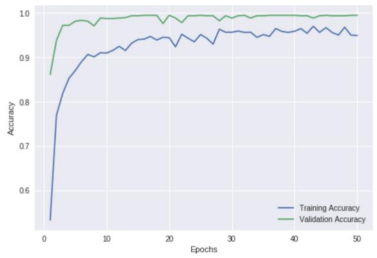
All images were resized according to the input size of the model we were training on which was 256x256 for all models except the ResNet. This size seemed large enough to accurately represent the complexities of images. Dropout probability was kept at 0.4 for all models.
All of our models were constructed or derived from Keras.

A) The first model we created was a custom model that we put together using various Keras layers. We added dropout layers in order to prevent overfitting, and we also created it out of several Separable Convolutional layers. These differ from normal convolutional layers as they convolute over each channel of the input image separately, and then use each of those to update the weights accordingly. This usually leads to more accurate models, since it allows each image to provide even more data per pixel than in a normal ConvNet. We have also added the appropriate pooling and activation layers in order to then get the output to be a softmax layer where we can predict the percentage that the given image would be in one of the ten categories. We got the following accuracies.



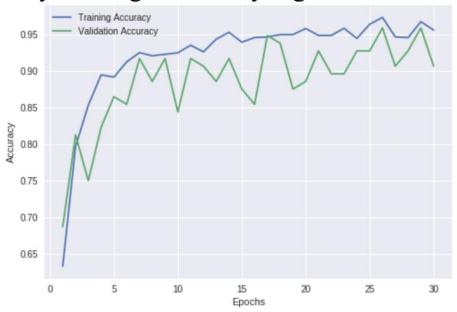
We originally had some issues with our model not converging, and the validation accuracy staying low, and jumping back and forth, but we determined it was a learning rate issue, so we changed the learning rate to .0001 from .001. Reducing the learning rate definitely helped in getting the model to converge. We also originally used ADAM optimization, but that did not seem to actually help in any way, so we switched to RMSprop, although we stuck to ADAM with the rest of the models we were training.

B)Next, we trained on a VGG-16 [22]. It is comprised of 16 layers: it has 3x3 convolutional layers stacked in increasing depth and max-pooling layers combined to make a deep convolutional base out of a small number of layers. We added a dense layer and a dropout layer to prevent. We initialized the weights of our VGG16 on the ImageNet dataset. To save time, we only made the last block trainable, thus exploiting the pre-trained ImageNet weights. The validation and train accuracies were fairly comparable:

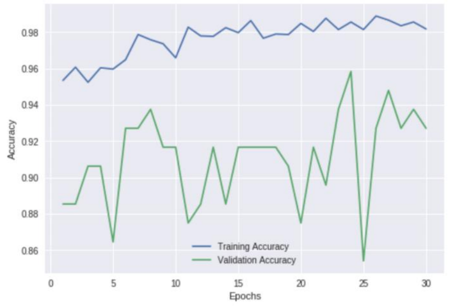


Here we can see that due to the ImageNet already being good weight initialization, accuracies increased quickly. This was the best overall model for accuracy, with an average training accuracy of 99.4% (see table for others).

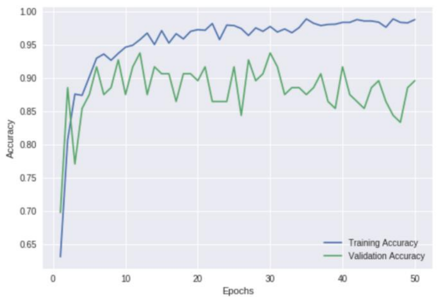
C) We then trained an Xception model. The Xception model is a modification of the Inception model, except using depth-wise separable convolutions. The inception module is to act as a "multi-level feature extractor" by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network. Output from these filters is then stacked along the channel dimension and before being fed into the next layer in the network. [5] Like other models, dropout and Softmax layers were added. Only training the last layer gave this accuracy plot:



Training every layer in Xception caused overfitting:

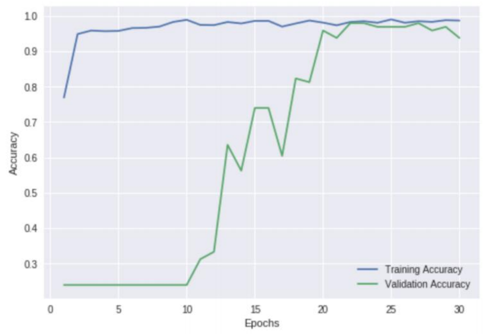


Validation and training accuracies approached similar



trends as we trained on 50 epochs instead of 30:

D) For our Residual Network (ResNet) model, we used the prebuilt model of ResNet50 from Keras, initialized to ImageNet. On its own, ResNet50 is quite a deep network with 50 residual layers, using skip connections to avoid the problem of vanishing gradients. To fit this to our problem, we made the last convolutional layer trainable, added a 128-unit dense layer (ReLU activation), and added an output layer with Softmax activation with our data. This lead to the following graph:



The accuracy graph for the ResNet shows that the validation accuracy takes some time before it starts to catch up with the training accuracy. The deep residual neural network seemed to take quite a long time to find the patterns and weights necessary to make good evaluations on the validation set, but once it learned those weights, it quickly reached equality between the training and validation accuracies.

For all of the above models, we also tested the test accuracies and compared it to the average validation and average training accuracies. All of that is shown within our iPython notebook.

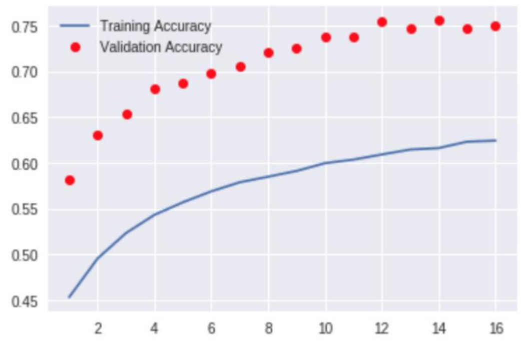
4.3. Large Dataset Model

Based on the earlier testing, we decided to proceed with our larger dataset, consisting of the 50 classes each containing 500+ images in each. We used the same learning rate as we had with the models on the smaller dataset, and we also used the same ADAM optimizer as we had previously. However, we increased the batch size and the number of epochs and steps per epoch. The batch size we increased to 32, from 20, since even though increasing batch size can sometimes be akin to decreasing the ability of our model to generalize, we also thought that since we were drastically increasing the dataset, it would be fine, and it turned out to be completely alright. We also increased the number of epochs from 30 to 50, this is because the VGG16 comes pre-trained with the ImageNet weights, and we realized that if we did not retrain all of the blocks in the VGG16, then it is just configured to the ImageNet dataset, and our maximum activation pictures ends up resembling images from the ImageNet dataset. We increased the number of steps per epoch to 2000, this was so that each epoch had the possibility of including every image within our training set. Since with the batch sizes, and the number of steps per epoch, then 64000 images would be selected each epoch, which should accommodate for the size of the dataset.

Unfortunately, all of the above made it quite computationally expensive, so we stopped training the model after 15 epochs. Due to the tendency of the pre-trained Keras models to over-fit, or to have such a high validation accuracy from the start, that training on the dataset would actually decrease the overall accuracy. In order to combat this tendency to overfit, and to make the

VGG16 more accommodating to our dataset, we added some dense layers, with a final softmax activation layer outputting the prediction compared to our 50 classes.

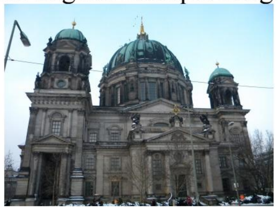
In the graph below we see the training accuracy versus the validation accuracy of the modified VGG16 model trained on the large dataset of 50 landmarks. We were only able to train it for 15 epochs, since it was taking too long to compute, but we were able to achieve a validation accuracy of above 75% and a training accuracy of above 60%.



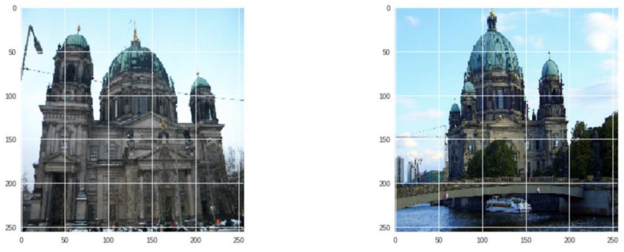
4.4. Visualizations

In order to demonstrate the visualizations and how they would work on our dataset, we will take an example landmark image. For our example, we will take an image of the Berlin Cathedral, which was classified as category 1553 in the original Kaggle dataset, but was category 18 in our large dataset. We used the Keras-vis [4] library in order to implement a variety of visualizations.

Below we see the original example image.



We then reduce the image to the 256x256. Here it is represented side-by-side with another example image within the same class.



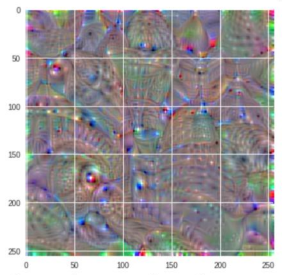
The model correctly predicted the class when given the inputs of the two example images.

We will then use this image and these images in order to do visualizations of our model and how our model interacts with the images. We will first try to represent our filters in our model by using a maximum activation image. The maximum activation image is composed of the image that would maximize the filter output activations of each filter layer in the model. It can mathematically be represented by the vanishing of the following expression [4].

$$\frac{\partial Activation\ Loss}{\partial input}$$

This produces an image that maximize our model's filters to predict the current class. This allows us to understand what sort of input patterns activate a particular filter. So we are maximizing with respect to the input, as opposed to with respect to the output, and we are not in fact visualizing the filters themselves, but more accurately, the images and parts of the images that would increase the activation of those filters to the maximum extent. If we run this on the class of the Berlin Cathedral and on our VGG16 model, we get the image below.

Maximum Activation Image



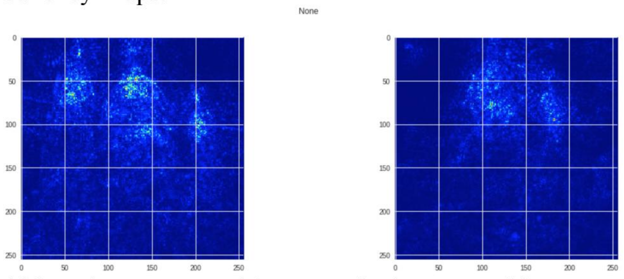
We see in the above image that there seems to be some patterns that resemble the domes of the cathedral and the pointed spires on top of them. You can especially see this resemblance to the dome in the quadrants around 100,150, and 150,150. However, you can also see some swirls and some things that look like spires and domes from different angles, or shapes, or sizes. These most likely come from the image augmentation that we are doing to our training set. We are retraining the filters within the VGG16, so that way the weights can best reflect and accurately represent our dataset, so those augmented images should be having their effect on the weights and filters, and as such, are being represented here on the maximum activation image.

As I stated before, when we tried to do the maximum activation image without retraining the weights of every block of the VGG16. We ended up getting an image that closely resembled images from the ImageNet. This is because the weights were initialized from the ImageNet dataset, and therefore the maximum activation image will be representative of the ImageNet dataset as opposed to the dataset we were using.

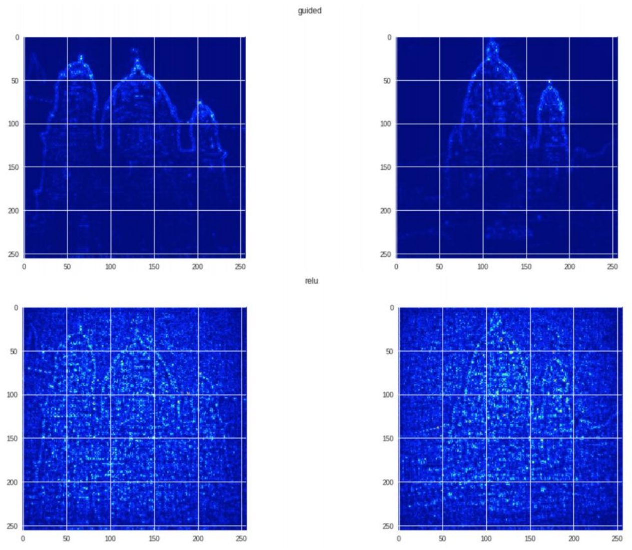
Saliency maps are ways for us to see which pixels of the image that the model is using in order to make a classification. We compute the gradient of output category with respect to input image. This should tell us how output category value changes with respect to a small change in input image pixels. All the positive values in the gradients tell us that a small change to that pixel will increase the output value. Hence, visualizing these gradients, which are the same shape as the image should provide some intuition of attention. [4]. This can all be represented by the following expression

$$\frac{\partial output}{\partial input}$$

The saliency map is a good way to visualize which regions of the image would cause the most change to the output, if they were changed. i.e. it gives us a way to see which pixels or parts of the original image are the most "important" in coming to a classification or a conclusion. Below we have the saliency maps of the two example images. In order to produce these images, we had to switch the final dense layer in which we produce the outputs from a softmax activation to a linear activation. It would have been suboptimal to get the derivative of the output if we had left it as a softmax layer. The first two saliency maps are using no modifier for the backpropogation. If we look at the saliency maps with no backpropogation modifiers, we see that the pixels that are most represented are those around the domes and spires of the cathedral. Therefore, these pixels will be the ones that are most impactful on making the final predictions and classifications.
Saliency Maps:

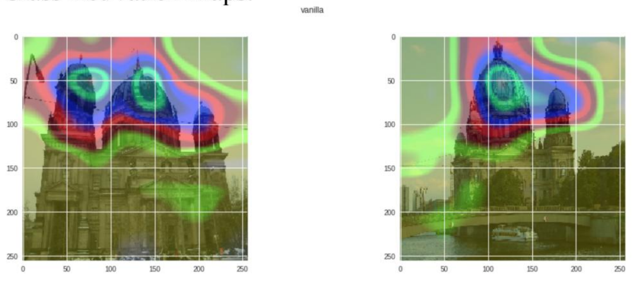


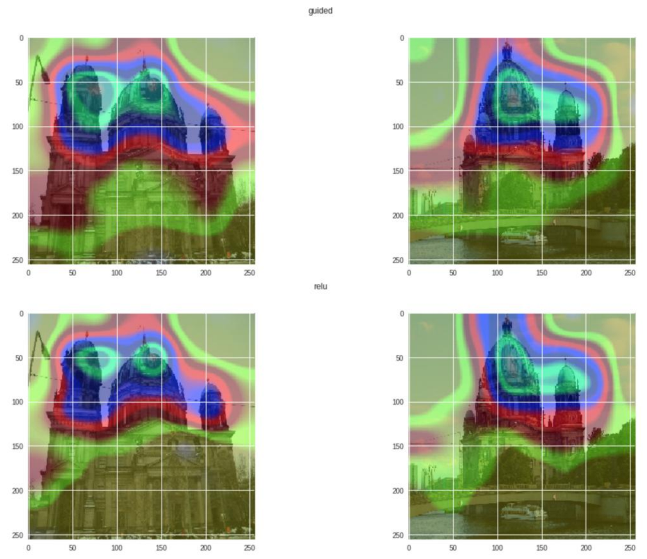
Although we can see this pattern in the maps without backpropogation modifiers, we can see it more clearly in those saliency maps that do contain backpropogation modifiers. The most likely reason is, that we had optimizers that make it much more complex in terms of determining the direct impact that each pixel would have on the output. The two backpropogation modifiers we can utilize are "guided" and "relu". Guided in this case modifies backpropagation to only propagate positive gradients for positive activations, 'relu' modifies backpropagation by zeroing negative gradients and letting positive ones go through unchanged. [4].



We can see the saliency maps with the guided and relu modifiers have a clearer outline of the original Berlin Cathedral, and we can see the outline of the domes and the spires is the part of the original images that have the most impact on the classification.

Our next visualization will not be reliant on the gradient of the output, unlike the previous one. The class activation maps visualize the attention over the penultimate convolutional layer in respect to the input. The intuition is to use the nearest Convolutional layer to utilize spatial information that gets completely lost in Dense layers. Those areas which are most important are represented by a higher "heat map" designation, while those that are garnering least attention from the penultimate convolutional layer will be designated as unimportant by the activation maps. We will also plot three different pairs of Class Activation Maps, the first pair being that without any backpropogation modifiers, the second being the "guided" backpropogation modifier, and the third being the "relu" backpropogation modifier.
Class Activation Maps:

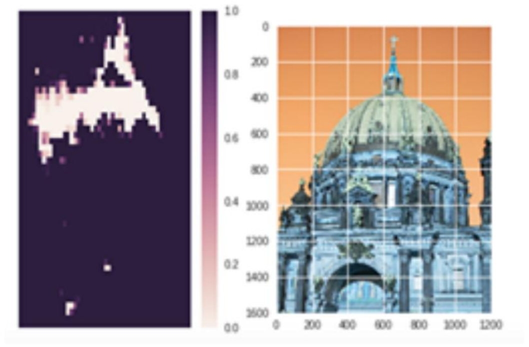




We see in the above maps that the areas that have the greatest activation are the spires and dome, particularly the area of the larger dome central to the Berlin Cathedral. We also found that the backpropogation modifiers tended to allow for the smaller dome to have more attention than it would in the original image. The "guided" backpropogation modifier also seemed to allow for a greater area of maximum attention, whereas the "ReLU" backpropogation modifier seemed to narrow the area of greatest attention on the original dome, but still extend that area to include parts of the smaller dome.

4.5. Occlusion Sensitivity

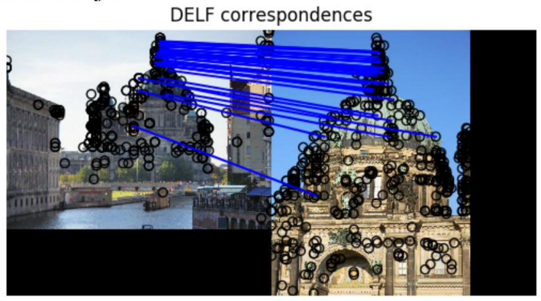
Our method systematically occludes different portions of the input image with a black square and gets a prediction from the model for the occluded image (Zelier et al., 2013). Representing the probabilities calculated after occlusions at different points as a heat map can help illuminate which areas are crucial to the classifier. Lower probability in an area means that occluding that portion greatly reduces the probability of the image being correctly classified and is therefore crucial to the model identifying correctly. To visualize occlusion sensitivity, we adapted open source code available on Github [6].



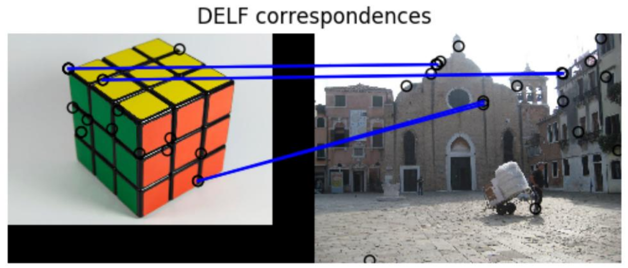
Smaller strides and occlusion windows proved to be both computationally intensive and ineffectual in highlighting high-level features. Larger strides and occlusion windows however reduced the precision of our analysis. Tuning the stride and occlusion window across various values, we concluded that a window of 10x10 pixels with a stride of 5 on an image of size 256x256 provided the best results. We see that the area that is most necessary to make a classification being that of the dome and the spires, which seems to make sense in comparison to our class activation maps and saliency maps, since the dome and spires were the most pronounced there as well.

4.6. DELF

On the advice of the CS230 project TA, who was advising my two project partners, we tried to see if we could somehow implement DELF and use it as part of our project. DELF compares local features between images and finds the correspondences that occur within those. Below we can see two example images of the Berlin Cathedral. They have 39 inliers, which means 39 local features that correspond between them. Despite the fact the images are taken from two different angles and from two different distances, we can still tell they are of the same subject.



We thought that we could use the similarities of local features as a way to distinguish between non-landmarks and landmarks, but as the image below shows, DELF can find inliers between two completely unrelated images. So even though it is fewer inliers than the above image (6 as opposed to 39). It still is not distinctive enough to utilize DELF as part of our classification process.



According to these tests, it seemed somewhat unreliable for the task of discriminating between non-landmarks and landmarks. Although we had thought that we could use DELF to classify to non-landmark images, there seem to be too many inliers found between non-similar images to accurately use it as a classification tool at this point. Disregard clutter

5. Conclusion & Future Work

As stated earlier in our project, the VGG16 performed the best given both the larger dataset we used, and the smaller dataset we used. If we could accurately implement DELF in a computationally efficient way so that we can classify if something is not a landmark. We can create a dataset of non-landmarks, and use DELF to find in-line features to compare the landmark category that the non-landmark has been categorized as, and then compare local features to try and then show that it is actually not a landmark and not a member of that class. There should be ways to implement DELF in such a way that we are able to deal with the possibilities of non-landmark images interacting with our classifiers. If we can integrate that into our models, most likely our VGG16, then we should be able to make an extremely robust landmark classifier. If we also could, we would like to run the VGG16 on the entire original dataset, for many epochs, so that way we could get very well trained weights, that wouldn't be over-fitted to any dataset.

6. References:

[1] Abdelfattah, Abdellatif. "Image Classification Using Deep Neural Networks - A Beginner Friendly Approach Using TensorFlow." Medium, Medium, 27 July 2017, medium.com/@tifa2up/image-classification-using-deep-neural-networks-a-beginner-friendly-approach-using-tensorflow-94b0a090ccd4.

[2] Araujo, André, and Tobias Weyand. "Google-Landmarks: A New Dataset and Challenge for Landmark Recognition." Google Research Blog, Google, 1 Mar. 2018, research.googleblog.com/2018/03/google-landmarks-new-dataset-and.html.

[3] Genthial, G., Moindrot, O., & Nair, S. (2018, January 24). Introducing the Project Code Examples. Retrieved from https://cs230-stanford.github.io/project-code-examples.html

[4] Kotikalapudi, R. (2017). Keras Visualization Toolkit. Retrieved from https://raghakot.github.io/keras-vis/

[5] Kaiser L. (2018) Tensorflow. DELF: DEep Local Features. Retrieved from: github.com/tensorflow/models/tree/master/research/delf

[6] Ludwig O. (2016) Sensitivity to Occlusion-Keras. Retrieved from github.com/oswaldoludwig/Sensitivity-to-occlusion-Keras-

[7] Eddins, Steve. "Network Visualization Based on Occlusion Sensitivity." Mathworks Blog. December 15, 2017. blogs.mathworks.com/deep-learning/2017/12/15/network-visualization-based-on-occlusion-sensitivity/.

[8] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." arXiv preprint (2016).

[9] Erhan, Dumitru, Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Visualizing higher-layer features of a deep network." *University of Montreal* 1341, no. 3 (2009): 1.

[10] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[11] Hinton, G. E., Osindero, S., and The, Y. A fast learning algorithm for deep belief nets. Neural Computation, 18:1527–1554, 2006.

[12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[13] Li, Yunpeng, David J. Crandall, and Daniel P. Huttenlocher. "Landmark classification in large-scale image collections." In Computer vision, 2009 IEEE 12th international conference on, pp. 1957-1964. IEEE, 2009.

[14] Noh, Hyeonwoo, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. "Large-Scale Image Retrieval with Attentive Deep Local Features." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3456-3465. 2017.

[15] Philbin, James, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. "Object retrieval with large vocabularies and fast spatial matching." In Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on, pp. 1-8. IEEE, 2007.

[16] Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks:

visualising image classification models and saliency maps (2014)." *arXiv preprint arXiv:1312.6034* (2013).

[17] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15, no. 1 (2014): 1929-1958.

[18] Torralba, Antonio, Rob Fergus, and Yair Weiss. "Small codes and large image databases for recognition." In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, pp. 1-8. IEEE, 2008.

[19] Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on computer vision*, pp. 818-833. Springer, Cham, 2014.

[20] Zhou, Bolei, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. "Learning deep features for discriminative localization." In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pp. 2921-2929. IEEE, 2016.

[21] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12, no. Oct (2011): 2825-2830.

[22] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).