Classifying U.S. Houses by Architectural Style Using Convolutional Neural Networks

Chris Pesto
Stanford University
cpesto@stanford.edu

Abstract

Automatic understanding of architectural style using computer vision has myriad applications. However, little research has been done on this specific topic compared to general object and scene classification. In this work I used deep convolutional neural networks to classify houses from a set of commonly-recognized U.S. house architectural styles, as well as localize the houses in the image. I used a new dataset created entirely from Zillow.com images, ensuring the applicability of the work to real-world real estate listing imagery.

1. Introduction

In the past few years, the explosion of available visual data and computational power has led to convolutional neural networks (CNNs) repeatedly breaking long-standing benchmark thresholds in image classification and localization. But while CNNs have reached human or even greater-than-human performance on these tasks in general benchmarks, there is a still considerable room for examining specific applications.

The architectural styles of the buildings in which people live and work are a defining part of everyday life, even if not always the subject of considerable thought. They are a major part of the history and identity of a place. As such, automated recognition of architectural style at a large scale through computer vision has applications in historical research, surveying and urban planning, and even the procedural generation of believable 3D models of a place.

Within the realm of automated architectural style classification, I chose the specific problem of classifying U.S. houses according to several common styles. U.S. house styles have a rich historical and regional background – for example, see Faragher [1] for a detailed look at the history of two of the styles classified in this work, Ranch and Craftsman (Bungalow). Additionally, according to a 2015 McKinsey report [2], the U.S. real estate industry is only moderately digitized, and in recent years there has been a major rise in startups focused on real estate. This field is ripe for the application of recent neural network-based advances in computer vision.

In this work, I created a new dataset of U.S. house images entirely from Zillow.com [3], a popular real estate marketplace. The dataset includes the style (from among 5 common U.S. house styles) and bounding box coordinates for the primary house in each image. I compare different convolutional neural network architectures for classifying the images by style and localizing the primary house (identifying its bounding box coordinates).

2. Related Work

The field of automatic architectural style identification is not extremely well developed, and in particular the amount of published research since the deep learning explosion is especially limited.

2.1. Non-deep learning architectural style identification

Shalunts, et al. [4] published work in 2011 on classifying the architectural style of building façade windows as Romanesque, Gothic, or Baroque using a small (400 image) dataset and a gradient directions-based approach. That same year, Mathias, et al. [5] published work using an SVM on 4 classes to classify architectural style, with a specific focus on "inverse procedural modeling" – using imagery to create a generative procedural model for 3D graphics. Shalunts, et al. noted that there was no automatic system for classifying façade images by architectural style, and Mathias, et al. noted little work had been done on architectural style identification.

In "What Makes Paris Look Like Paris?", Doersch, et al. [6] used a nearest-neighbor technique to find patches of architectural style characteristic of cities (windows, lamps, etc.). Goel, et al. [7] made their own dataset of 25 European monuments in 5 architectural styles, classified the images, and developed an unsupervised method to identify characteristic features for styles.

Zhang, et al. [8] broke images of buildings into "blocklets" to recognize architectural styles, and used

feature vectors to classify with an SVM. Xu, et al. [9] developed a 25-class dataset from Wikimedia, and used a model involving HOG that classified with Multinomial Latent Logistic Regression. Their model was able to find usages of multiple styles on the same building in a single image. Notably, they included "American Craftsman" (one of the house styles used in this work) as a class. Both groups noted the absence of a publicly-available dataset for architectural style recognition.

In 2015, Lee, et al. [10] published work using a large dataset of nearly 150k Google Street View images of Paris, combined with a real estate cadaster map to date building façades and discover the evolution of architectural elements over time. Their approach used HOG descriptors of image patches to find features correlated with a building's construction time period.

2.2. Deep learning architectural style identification

This field has not caught up with deep learning very well yet. In 2016, Obeso, et al. [11] used CNNs to classify Mexican historical buildings as Prehispanic, Colonial, or Modern.

2.3. Geo-localization

Geo-localization – determining where an image or video was taken – can be reliant on recognizing architectural style. For a recent general discussion of geo-localization that includes a mention of CNNs, see Zamir, et al. [12]. They note the specific challenge that "the subtle architectural styles that indeed encode a lot of information about the location are typically lost in feature extraction."

Zhang, et al. [13] developed a system to recognize the user's indoor location on the MIT campus using a phone image.

2.4. Urban surveying

Automatically identifying features of a city from imagery is an active field related to detection of building architectural style.

Arietta, et al. [14] used Google Street View images to develop models for automatically identifying non-visual attributes (crime statistics, etc.) of an area. They compared HOG+color and CNN features. They were also able to determine the "visual boundary" of neighborhoods – specifically, their system identified visual elements of the Mission neighborhood in San Francisco, such as Victorian-style houses (another house style used in this work). Porzi, et al. [15] and Ordonez and Berg [16] both similarly worked on predicting the perceived safety of city areas depicted in Google Street View images. Porzi, et al. tried both non-deep learning (HOG, SVM) and CNN-based approaches, and Ordonez and Berg used both non-deep-learning-derived and CNN-derived features with an SVM classifier and regression.

In 2016, Liu, et al. [17] used Baidu Street Map imagery of Beijing to develop a model for automatically assessing street façade quality and street wall continuity, using both non-deep learning and CNN-based approaches.

2.5. Architectural drawings

Strobbe, et al. [18] took a very different approach to architectural style classification. They automatically converted 2D floor plans into graph representations and then classified the graphs with a 1-class SVM as belonging or not to a particular corpus by a particular architect.

3. Methods

I compared 3 main CNN [19] architectures for doing joint classification and regression: an end-to-end trained baseline model, and models based on ResNet-18 and ResNet-34 [20] as feature extractors. All 3 models terminate in 2 separate output layers for classification and regression.

I used the PyTorch deep-learning framework [21] (in conjunction with the numerical computation library NumPy [22]) to develop my models and data pipeline. I used PyTorch's ResNet implementations and pretrained model weights.

3.1. Baseline network

My baseline architecture starts with 2 stacked convolutional segments of 4 layers each: a 5x5 convolutional layer with 32 output channels and a stride of 2, Batch Normalization [23], ReLU activation [24], then 2x2 max pooling with a stride of 2.

Convolutional layers slide a filter of a given size over all spatial patches on their input and apply an inner product with their weights, in this case with a "stride" of 2, meaning to slide the window by 2 pixels between each patch. In this case the convolutional filters will apply an inner product over a region of size 5x5xC_{in}, where C_{in} is the number of input channels – 3 for the first segment's convolutional layer, since it receives the 3 RGB channels of the image for each spatial location (image pixel). The number of channels C_{out} – 32 for both of these layers – indicates the number of such filters (with distinct weights) that will be used in that layer, and determines the depth of that layer's output.

Batch Normalization is a technique that allows the network to learn how to normalize data – i.e., rescale it to have a mean of 0 and a variance of 1 – at each layer,

reducing the network's sensitivity to the scale of parameters and allowing it train more easily and quickly. The ReLU activation simply applies the function $f(x) = \max(0, x)$ element-wise to its inputs, and introduces the necessary nonlinearity for a neural network to function. Finally, max pooling simply slides a window, in this case 2x2, over each spatial patch of its input and outputs the max over that window. With a stride of 2, it converts an input of size WxHxC to $((W-2)/2+1)$x$((H-2)/2+1)$xC. This simply reduces the size of the input to the next layer, reducing the computational complexity of training and evaluation.

The output from the 2nd of the convolutional segments is flattened to a 1-dimensional vector and fed into two segments that are identical except for their final number of outputs. Both start with a 1024-output fully-connected layer, followed by Batch Normalization, ReLU activation, then a final fully-connected layer. (A fully-connected layer means a standard neural network layer in which each output is the inner product of a learned weight vector with the input vector.) The first segment outputs classifier scores, and so has a number of outputs equal to the number of classes (5 for this experiment). The second segment outputs bounding box coordinates – the x and y coordinates of the top-left corner, and the x and y coordinates of the bottom-right corner – and so has 4 outputs.

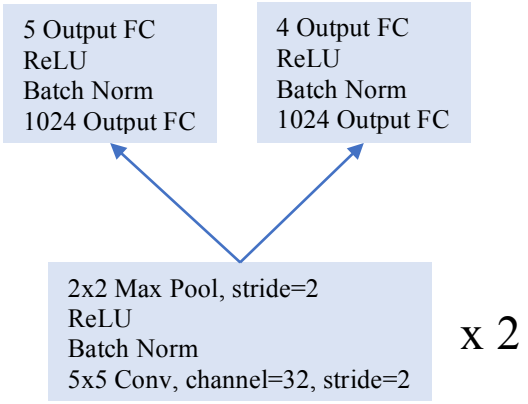

Figure 1: Baseline model architecture

3.2. ResNet feature extractors

In addition to the baseline network, I compared two more powerful network architectures, ResNet-18 and ResNet-34, which are respectively 18- and 34-layer versions of ResNet. ResNet is a state-of-the-art architecture for very deep CNNs that introduces shortcut connections. These connections pass the input directly around a block of layers and sum it with the output of the block. The idea is to allow the network to learn the "residual" function of each shortcut block – i.e., the difference between the input and the desired output of the block – rather than directly convert the input to desired output in the block. This technique produces extremely powerful models, which have won multiple competitions in classification and detection [25].

For my experiments, I used the two ResNet architectures as feature extractors. That is, I used ResNet models whose weights had been pretrained on the very large ImageNet dataset [26], and did not optimize those weights during training. Instead, I replaced the final fully-connected output layer on the two ResNets with two segments of my own, one for classification scores and one for bounding box coordinates, and only trained the weights in those segments. They received the flattened output of the pretrained ResNet prior to its final fully-connected layer. The idea of feature extraction is that a network pretrained on such a large dataset of images should be good at converting any input images into a general set of discriminative features that are useful for classification. Those features can then be fed into new classification layers that are trained for a specific new task.

As with the baseline model, these segments are identical except for the number of outputs in their final fully-connected layers. They consist of: a 4096-output fully-connected layer, Batch Normalization, ReLU activation, a 1024-output fully-connected layer, ReLU activation, and a final fully-connected layer. The final layer has 5 outputs (the number of classes in my experiment) for the classification segment and 4 outputs (1 for each coordinate) for the bounding box segment.

These particular fully-connected layers (4096, then 1024, then 4 coordinate outputs) were inspired by the regression network in Sermanet, et al. [27], which discusses using CNNs for both classification and localization.

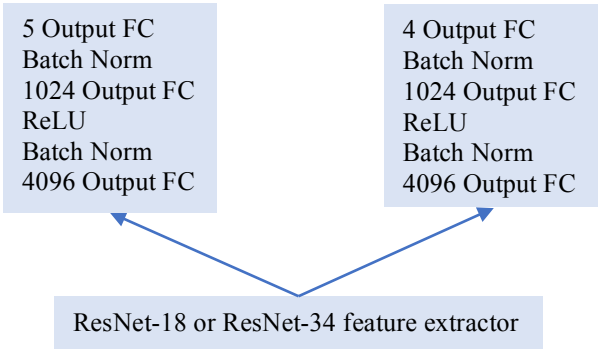

Figure 2: ResNet model architectures

3.3. Two-step training with cropping

Relatively recently, Jaderberg, et al. [28] published work on a Spatial Transformer Network which allows a network to learn spatial manipulation of data within the network. This lets it, for example, crop out the relevant patch for

classification from among background clutter. As an additional experiment beyond learning classification and localization simultaneously during the same training, I tried a similar but less sophisticated version of this general technique. I found my trained networks were very good at localizing houses in their input images, and decided to leverage this to try to improve the classifier performance.

I used the following two-step procedure with the ResNet-34 model (which I found to be the most effective classifier in my experiments):

1. Train the model on localization, only computing gradients with respect to the bounding box segment loss and only optimizing that segment's weights.
2. Train the model on classification of the image regions identified by the bounding box outputs, only computing gradients with respect to the classification segment loss and only optimizing that segment's weights. I did this via two passes of all images through the model, as follows:
    a. Perform one pass to determine the bounding box coordinate outputs for each image.
    b. Crop out the identified region of each image, and put it on a black background.
    c. Perform a second pass through the network of the new images containing only the cropped-out regions.

Evaluation of this technique against the validation set was done using the same two-pass procedure described under the 2nd list entry above.

3.4. Loss functions

I used 2 different loss functions for the 2 output segments of all 3 networks.

For the classification segment, I used the standard softmax cross-entropy loss:

$$L_{i_{classification}} = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

Equation 1: Softmax cross-entropy loss

Where f_j denotes the output classification score for class *j*, and f_{y_i} denotes the output classification score for the correct class for training example *i*.

For the bounding box regression problem, I used the following loss:

$$L_{i_{localization}} = \sum_j \left[\frac{\left(f_j - y_{i_j}\right)^2}{s}\right]$$

Equation 2: Bounding box regression loss

Where *j* ranges over the 4 coordinates of the bounding box (top-left x, top-left y, bottom-right x, bottom-right y), f_j denotes the output for coordinate *j*, y_{i_j} denotes the ground truth for coordinate *j* of training example *i*, and *s* is the size of the input image (256, 512, 768, or 1024 in my experiments). This loss essentially translates to "the fraction of the image size that each coordinate was off, squared, and summed over all 4 coordinates." It was both quick to calculate and comparable across different input image sizes, which was useful to me as I found input image size to be a major factor in classification accuracy.

The total loss for each training example was:

$$L_{i_{total}} = L_{i_{classification}} + L_{i_{localization}}$$

Equation 3: Total loss

Each type of loss – classification, localization, and total – was averaged over all training examples to produce a single scalar value for backpropagation.

3.5. Training

For all networks, I trained using standard SGD (stochastic gradient descent). I used the Adam optimizer [29] update rule with a learning rate of 0.01 (I found the more common 0.001 to train slowly in my experiments). I used a mini-batch size of 8 for all experiments, which was determined by my memory constraints while training the ResNet-34 architecture with 1024x1024 images.

As indicated in the preceding network descriptions, I did not use any explicit regularization such as dropout [30]. In early testing it did not seem effective at staving off the inevitable overfitting with my relatively small training set. Additionally, I made heavy use of Batch Normalization, which, in the words of the original paper, "also acts as a regularizer, in some cases eliminating the need for Dropout."

I typically trained for 4 epochs, as the model converged quickly on the relatively small dataset. I only saved the model weights at the end of each training epoch when the averaged total loss $L_{i_{total}}$ on the validation set was smaller than the lowest value seen at the start of training or at the end of the preceding epochs.

4. Dataset

4.1. Size and distribution

My dataset consists of 2,500 total images, distributed among 5 different classes (house architectural styles). There are 500 images from each of the 5 classes.

I randomly chose and removed 100 images (20%) of each class as a test set. Furthermore, I divided the remaining 400 images per class into 4 folds of 100 images (20%) each, shuffled first to avoid splitting the images in the order I collected them (this was done dynamically at runtime using the Python random shuffle function with a set random seed so the order, and hence split, was consistent). Numbering these folds 0-3, I was then able to do 4-fold cross-validation with each fold as the validation set.

4.2. Collection

The images were all collected manually from Zillow.com, almost entirely from home listings on the site, meaning the dataset is representative of real-world real estate listing photography (including many low quality, clearly non-professional photos). I personally evaluated whether each image belonged to each architectural style. I chose only images at or near ground level, but allowed heavy occlusion of the house. Not all images contain a single house, but all have a clear "primary" house. Occasionally I included a 2nd (and very rarely, a 3rd) photo of the same house, from a different angle and/or under different lighting conditions. As a rule, I chose images showing most or all of the front façade (potentially from a large oblique angle), though in some cases whether the pictured side was the "front" could be ambiguous.

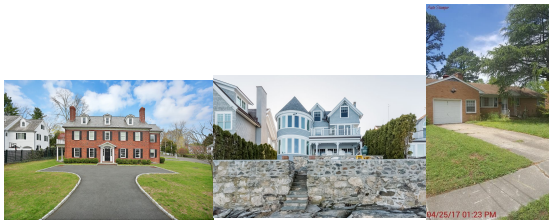

Figure 3: Example images from the dataset

4.3. Labeling

In addition to collecting 500 images from each category, I manually labeled all of the images with ground-truth bounding boxes around the "primary" house. I did not include detached guest houses, garages, or (somewhat arbitrarily) the open areas of carports.

To make this labeling tractable, I made a web app using the popular JavaScript libraries React [31] and Babel [32]. It loaded all the images in a class and let me quickly label the bounding boxes, which I could then export as JSON.

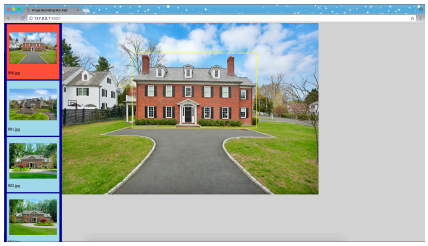

Figure 4: Labeling web app

4.4. Preprocessing

I used the popular Python image-processing library Pillow [33] to resize the images, which were of varying aspect ratios, into squares. I resized them to fit into a square of the desired size, preserving aspect ratio, and placed them in the top-left corner of a black background square. I did this for 4 different square sizes: 256, 512, 768, and 1024.

Figure 5: Example processed images

4.5. Normalization and augmentation

At runtime, images were dynamically transformed using PyTorch's ToTensor transform, which scaled the RGB pixel values from 0-255 to the range 0.0-1.0.

I then normalized each RGB channel using the precomputed mean and standard deviation for the training set corresponding to the current fold, i.e.:

$$p_{channel_{normalized}} = \frac{p_{channel} - \mu_{channel}}{\sigma_{channel}}$$

Equation 4: Normalized pixel values

Where μ denotes mean and σ denotes standard deviation. (The mean and standard deviation were computed for each possible training set over the original images, not the processed images with black backgrounds.)

I also used data augmentation, including the left-right flipped version of each image (and the left-right flipped version of its bounding box) during training, providing an effective 600 training examples per class. This was computed at runtime, so it could be generated dynamically for the training set corresponding to each fold. I also

experimented with adding top-bottom flipped images, but in preliminary testing they provided worse performance (probably since swapping the places of the sky and ground makes it more difficult to identify houses). No data augmentation was used on the validation set of each fold, or the test set.

4.6. Dataset statistics

Below are the image sizes (in pixels) of the original images for the training sets corresponding to each fold:

Fold	μ Width	μ Height	σ Width	σ Height
0	987	682	89.4	63.6
1	988	681	88.4	63.6
2	986	680	91.7	64.9
3	989	681	84.5	62.5

Figure 6: Training set dimension statistics

And the following are the computed means and standard deviations for each channel in the training sets corresponding to each fold (after applying PyTorch's ToTensor transform), which were used for normalization:

Fold	μ R	μ G	μ B	σ R	σ G	σ B
0	0.495	0.520	0.469	0.253	0.245	0.301
1	0.491	0.517	0.468	0.255	0.246	0.302
2	0.493	0.519	0.469	0.254	0.246	0.302
3	0.491	0.518	0.469	0.254	0.246	0.302

Figure 7: Training set color statistics

5. Results

For all experiments I tracked total loss, classification loss, localization loss, and fraction of examples classified correctly for the current validation fold. I computed these statistics at the start of training and at the end of each epoch. To conserve space, I only reproduce a subset of these numbers below (averaged across all folds when applicable).

5.1. Image size experiments

Input sizes of 256x256 are fairly common in image classification tasks, but I discovered that this was inadequate for this task. Recognition of house styles can involve understanding small features of the exterior. To study this, I performed 4-fold cross validation of all 3 models across 4 different image sizes: 256, 512, 768, and 1024. I trained for 4 epochs for each fold. The tables below are the total loss and fraction correct, averaged across the 4 folds, for the epoch of each fold with the lowest total loss (i.e., the epoch at which I would normally save weights during training).

	256	512	768	1024
Baseline	1.15	1.24	1.28	1.25
ResNet-18	0.74	0.56	0.51	0.56
ResNet-34	0.72	0.53	0.55	0.52

Figure 8: Best total loss, averaged across folds

	256	512	768	1024
Baseline	0.556	0.550	0.512	0.516
ResNet-18	0.737	0.811	0.816	0.808
ResNet-34	0.749	0.823	0.812	0.817

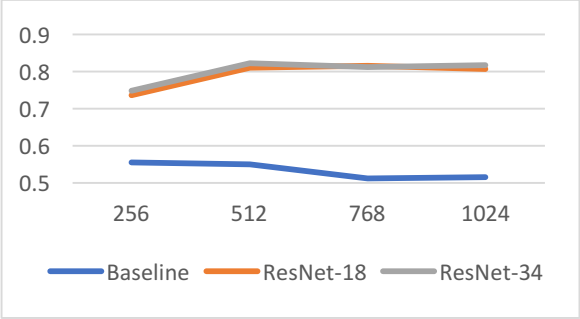

Figure 9: Fraction correct at best total loss, averaged across folds, table and graph

	256	512	768	1024
Baseline	0.0225	0.0229	0.0248	0.0251
ResNet-18	0.0234	0.0187	0.0155	0.0146
ResNet-34	0.0238	0.0180	0.0155	0.0149

Figure 10: Localization loss at best total loss, averaged across folds

The best averaged localization loss in this table is 0.0155, for 768 pixel images. Under the localization loss formula described previously, if you assume each coordinate is off by an equal amount, this implies an average error on each of the 4 coordinates of $768 \times \sqrt{0.0155 / 4} = 47.8$ pixels, or 6.22% of the image width/height.

Notably, the fraction correctly classified increased dramatically for the ResNet models going past size 256 images, while it decreased with size for the baseline model. I suspect that the baseline model capacity is simply too small at those sizes, whereas the ResNet results are reflective of the underlying reality – that 256 size images are too small to preserve enough information.

5.2. Two-step training with cropping

As mentioned, I also experimented with a 2-step training process in which I leveraged the model's ability to find houses to try to eliminate background clutter for the classifier. I tested this approach using 4-fold cross validation with the ResNet-34 model and 512-size images.

Unfortunately, as is evident by comparing the table below to Figures 8 and 9, this resulted in a worse model. Though it is still interesting it could classify 67.1% of examples correctly using only the bounding box identified regions.

	Total Loss	Fraction Correct
ResNet-34	0.89	0.671

Figure 10: Best total loss and fraction correct at best total loss, averaged across folds, using two-step training with cropping

5.3. Test set results with best model

Since it provided nearly the best total loss, and provided the best fraction correct, I chose ResNet-34 with 512x512 images as my best model to use in evaluating with the test set. The results below are for a freshly-trained model, and compare the validation set results (using fold #3) to the test set results (using fold #3's normalization values). The results include the classifier's fraction correct and the standard IoU (intersection over union) assessment of the bounding box identification.

	Total Loss	Fraction Correct	IoU
Val	0.51	0.842	0.706
Test	0.56	0.798	0.710

Figure 11: ResNet-34 model with size 512 images, evaluated on validation and test sets

To further understand the performance of the model, I visualized the performance of the classifier with a confusion matrix and the performance of the bounding boxes with images. The bounding box images show the "best 4," the images with the lowest bounding box loss in that class, and the "worst 4," the images with the highest bounding box loss in that class. The images on top show the ground-truth boxes (in teal), and the images on the bottom show the bounding box outputs (in yellow).

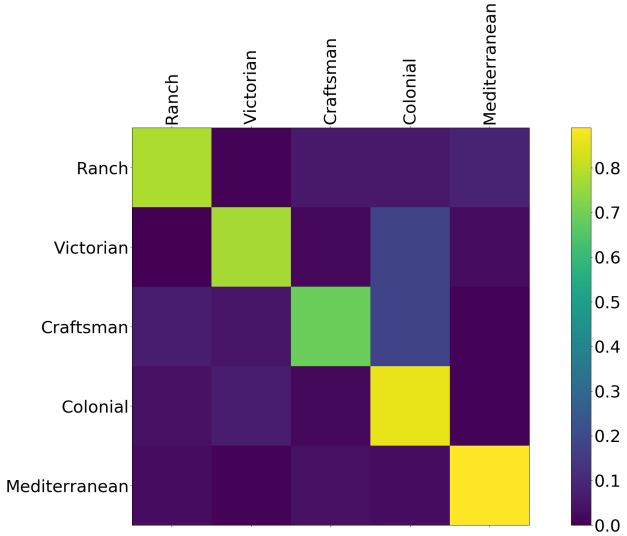

Figure 12: Confusion matrix for ResNet-34 model with size 512 images evaluated on test set

Figure 13: Bounding boxes for ResNet-34 model with size 512 images evaluated on test set

One thing the bounding box images make clear is that while the model is very good at identifying houses, it is not good at identifying the boundaries of the primary

house. Bounding boxes will span multiple buildings, including multiple houses.

The code used to generate the confusion matrix was based on example code on the PyTorch website [34]. The confusion matrix and bounding box images were generated using Matplotlib [35].

5.4. Network visualization

As a final way to understand the network, I produced saliency maps as well, based on the CS231N Assignment 3 code [36]. These show the highest absolute value of the gradient across the RGB channels of each pixel in the image, backpropagating an artificial loss equal to the classifier score for the correct class. In a sense, it is the pixels the network thinks have the greatest influence on the correct class for each image.

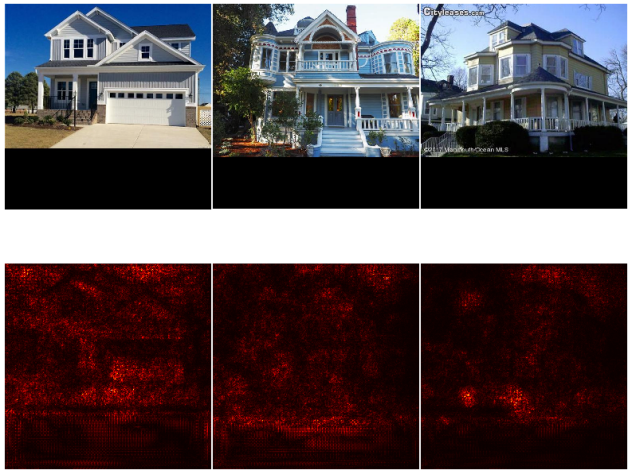

Figure 14: Saliency maps for ResNet-34 model with size 512 images evaluated on test set

These examples are all taken from among the 6 images with the highest correct-class score among the correctly-classified images of their respective classes. These saliency maps show that the network is capable of identifying the house, but also reveal shortcomings of the network. The first example clearly highlights the prominent garage – definitely an important feature for classifying a house's style. The second example shows an unfortunately common excessive focus on the black background at the bottom of the image. And the third example shows a strange almost complete focus on the shrubs in front of the house. It is unclear if that is an artifact of overfitting, or whether prominent shrubs in front of the house are in fact a salient feature for identifying Victorian houses.

6. Conclusion

This work shows that it is completely possible to classify U.S. house architectural styles with CNNs. I was able to achieve a correct classification rate of 79.8%, and a 0.710 intersection-over-union localization score on the test set, using ResNet-34 as a feature extractor with 512x512 images. I uncovered that typical 256x256 images are inferior for this task, probably because it is reliant on small features that do not show up adequately on low-quality/distant real-life real estate listing images that have been resized so small.

Importantly, this work used real-world real estate listing photography, indicating it is strong enough to apply to real-life applications.

Future work could include trying additional feature extractor networks, networks pretrained on datasets other than ImageNet, and deploying a system of this sort in a real service to improve filtering and ranking.

References

[1] Faragher, John Mack. "Bungalow and ranch house: The architectural backwash of California." Western Historical Quarterly 32.2 (2001): 149-173.

[2] Manyika, et al. "Digital America: A tale of the haves and have-mores." *McKinsey & Company | Global management consulting*. McKinsey Global Institute, Dec. 2015, http://www.mckinsey.com/industries/high-tech/our-insights/digital-america-a-tale-of-the-haves-and-have-mores.

[3] *Zillow: Real Estate, Apartments, Mortgages & Home Values*. Zillow, Inc., https://www.zillow.com/.

[4] Shalunts, Gayane, Yll Haxhimusa, and Robert Sablatnig. "Architectural style classification of building facade windows." International Symposium on Visual Computing. Springer Berlin Heidelberg, 2011.

[5] Mathias, Markus, et al. "Automatic architectural style recognition." ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences 3816 (2011): 171-176.

[6] Doersch, Carl, et al. "What makes Paris look like Paris?." Communications of the ACM 58.12 (2015): 103-110.

[7] Goel, Abhinav, Mayank Juneja, and C. V. Jawahar. "Are buildings only instances?: exploration in architectural style categories." Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing. ACM, 2012.

[8] Zhang, Luming, et al. "Recognizing architecture styles by hierarchical sparse coding of blocklets." Information Sciences 254 (2014): 141-154.

[9] Xu, Zhe, et al. "Architectural style classification using multinomial latent logistic regression." European Conference on Computer Vision. Springer International Publishing, 2014.

[10] Lee, et al. "Linking Past to Present: Discovering Style in Two Centuries of Architecture." 2015 IEEE International Conference on Computational Photography. IEEE, 2015.

[11] Obeso, Abraham Montoya, et al. "Architectural style classification of Mexican historical buildings using deep convolutional neural networks and sparse features." Journal of Electronic Imaging 26.1 (2017): 011016-011016.
[12] Zamir, Amir R., et al. "Introduction to Large-Scale Visual Geo-localization." Large-Scale Visual Geo-Localization. Springer International Publishing, 2016. 1-18.
[13] Zhang, Fan, et al. "Indoor Space Recognition using Deep Convolutional Neural Network: A Case Study at MIT Campus." arXiv preprint arXiv:1610.02414 (2016).
[14] Arietta, Sean M., et al. "City forensics: Using visual elements to predict non-visual city attributes." IEEE transactions on visualization and computer graphics 20.12 (2014): 2624-2633.
[15] Porzi, Lorenzo, et al. "Predicting and understanding urban perception with convolutional neural networks." Proceedings of the 23rd ACM international conference on Multimedia. ACM, 2015.
[16] Ordonez, Vicente, and Tamara L. Berg. "Learning high-level judgments of urban perception." European Conference on Computer Vision. Springer International Publishing, 2014.
[17] Liu, Lun, et al. "A machine learning method for the large-scale evaluation of urban visual environment." arXiv preprint arXiv:1608.03396 (2016).
[18] Strobbe, Tiemen, et al. "Automatic architectural style detection using one-class support vector machines and graph kernels." Automation in Construction 69 (2016): 1-10.
[19] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.
[20] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.
[21] *PyTorch*. Facebook, Inc., http://pytorch.org/.
[22] Walt, Stéfan van der, S. Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation." Computing in Science & Engineering 13.2 (2011): 22-30.
[23] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).
[24] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." Aistats. Vol. 15. No. 106. 2011.
[25] Li, Fei-Fei, et al. "Lecture 9: CNN Architectures." *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford University, 2 May 2017, http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf.
[26] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.
[27] Sermanet, Pierre, et al. "Overfeat: Integrated recognition, localization and detection using convolutional networks." arXiv preprint arXiv:1312.6229 (2013).
[28] Jaderberg, Max, Karen Simonyan, and Andrew Zisserman. "Spatial transformer networks." Advances in Neural Information Processing Systems. 2015.
[29] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[30] Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv preprint arXiv:1207.0580 (2012).
[31] *React – A JavaScript library for building user interfaces*. Facebook, Inc., https://facebook.github.io/react/.
[32] McKenzie, Sebastian. *Babel · The compiler for writing next generation JavaScript*. https://babeljs.io/.
[33] Clark, Alex, and Contributors. *Pillow: the friendly PIL fork*. https://python-pillow.org/.
[34] "Classifying Names with a Character-Level RNN – PyTorch Tutorials documentation." *PyTorch*. Facebook, Inc., http://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html#evaluating-the-results.
[35] Hunter, John D. "Matplotlib: A 2D graphics environment." Computing In Science & Engineering 9.3 (2007): 90-95.
[36] "Assignment #3." *CS231n: Convolutional Neural Networks for Visual Recognition*. Stanford University, http://cs231n.github.io/assignments2017/assignment3/.