The Essence of Pose

Jayant Thatte
Stanford University
Electrical Engineering
jayantt@stanford.edu

Vincent Sitzmann
Stanford University
Computer Science
sitzmann@stanford.edu

Timon Ruban
Stanford University
Electrical Engineering
timon@stanford.edu

Abstract

Convolutional Neural Networks (CNNs) are often used to extract a lower-dimensional feature descriptor from images. Though they have recently been deployed successfully to solve tasks in computer vision such as semantic segmentation, image reconstruction or depth understanding, the inner workings of CNNs are still subject to current research.

Recent research to better understand features learned by CNNs has focused on networks trained to do image classification. We perform a similar investigation, but in the realm of 3D pose estimation. We explore inversion techniques similar to those in [12, 13, 3] to invert the feature descriptor learned by PoseNet, a CNN by the Stanford Computer Vision and Geometry laboratory trained to understand the 3D pose of objects. We use a dataset consisting of several million images annotated with camera parameters.

Our results suggest that PoseNet mainly embeds information about edges that point towards the main vanishing point in the feature descriptor. Most strikingly, color and texture information are lost, while the feature descriptor retains enough information to reconstruct the rough shape of the object in the original image.

1. Introduction

The aim of this work is to find the features in a 2D image that are relevant to understand the 3D pose of a presented object. We make use of PoseNet (explained in section 2.1) developed at Stanford's Computer Vision and Geometry Lab. PoseNet was trained using street view images annotated with the corresponding camera parameters. The network embeds these images in a low-dimensional vector representation - this representation is then used to compute the pose of the presented object in the image.

In this project, we invert the vector representation produced by PoseNet to investigate what features of the original image are retained by the network in the vector embedding. Studying which features are retained and which are discarded will help in understanding what aspects of an image are really required to solve the task of 3D pose estimation. For example, we expect image colors and textures to have little or no information about the pose, whereas certain object boundaries as well as perspective and keystone effects should play a critical role.

We used two main techniques for inverting the network. They are outlined in detail in section 3. The first type inverts the network by solving an optimization problem while imposing a natural image prior and does not involve learning any parameters. It takes as input an original image and extracts its vector representation by passing it through PoseNet, then uses this feature descriptor to output a reconstructed image with the same vector representation. The second approach tries to learn a decoder to minimize the loss in image space between the original image and the reconstructed image. The algorithm takes the feature descriptor of the original image as input, passes it through an upconvolutional network and outputs the reconstructed image. We expect the reconstructed images to preserve only those qualities of the original image that give useful insights into 3D orientation, while other details should be discarded.

2. Background/Related Work

Convolutional Neural Networks have been proven to be useful for semantic understanding of images [11]. Recently, there has been a lot of effort in inverting the vector embeddings generated by CNNs to understand what the networks learn in order to solve classification tasks. [12] reconstructs the original image by analytically reversing the network, while [13] generates a typical image for each image class. Such work can give significant insights into why CNNs work so well on classification tasks [16]. [14] formulate the problem as an optimization problem and produce input saliency maps given a specific output label.

Another class of approaches is to learn an optimal decoder that recovers an input image from the vector embedding. The idea being that the reconstructed image can only be as good as the vector embedding and therefore seeing the reconstructed image can give useful insights into what input features were discarded by the network, when trained to solve a semantic task [3]. Such learned decoders are shown to be effective at inverting shallow representations such as HoG and SIFT, as well as deeper and highly non-linear representations produced by CNNs [2]. [7] argue that L2 reconstruction loss is not optimal to perform the task of network inversion and introduce the concept of Generative Adversarial Networks (GANs). Decoders trained using GANs have recently been shown to produce much sharper reconstructed images [4]. They also argue that GANs can be unstable and very susceptible to the choice of hyperparameters and give some pointers to make such networks trainable.

While almost all of the existing work in inverting CNNs has been in the area of image classification, CNNs have recently shown promising results in the areas of image morphing, 3D reconstruction and novel view synthesis [6].[5] trains a CNN to learn vector embedding corresponding to 3D representation of computer generated models and demonstrates that these vectors can then be used to synthesize novel views of 3D objects. Understanding what a deep neural network learns when trained to solve 3D tasks is therefore critical. Such understanding will help us better deploy such networks for solving problems in 3D computer vision.

2.1. PoseNet

PoseNet is part of a multi-task CNN developed at Stanford's Computer Vision and Geometry Lab [15]. It takes as input a 100x100 RGB image and produces a 512-dimensional feature descriptor that is then used in combination with the descriptor of a second image to solve three different tasks: wide baseline matching, estimating the relative camera pose between the two images and the shape (normal vectors) of the depicted object. Since these tasks can be solved using only the PoseNet feature descriptors of the two images, the PoseNet feature descriptor must embed features that encode information about the geometry of the scene.

3. Methods

In the following sections we outline two approaches for network inversion. The first formulates the inversion as an optimization problem that can be solved iteratively via a gradient descent method. The second involves training a neural network that learns to generate reconstructed images.

3.1. Inverting the feature descriptor

To investigate what information is retained after passing an image through the network we follow an approach similar to [12] to invert the feature descriptor (see Figure 1). Minimizing the euclidean distance between the feature descriptors of the original and reconstructed image can be

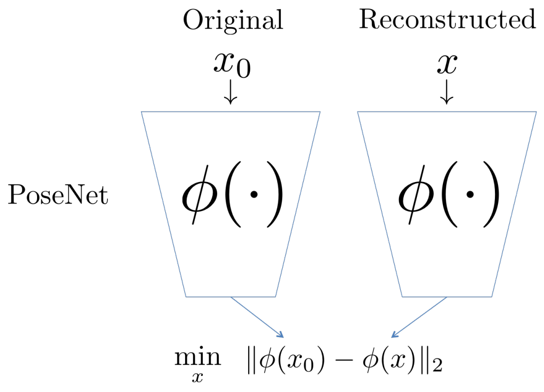

Figure 1: To invert the feature descriptor we solve an optimization problem to find a reconstructed image that has the same (i.e. as close as possible in the euclidean norm) feature descriptor as the original image.

thought of as finding an image that is indistinguishable from the original in the eyes of the network. However, because the feature descriptor is of lower dimension than the input image, many such solutions might exist. To find the visually most interpretable image, we additionally impose regularization and a natural image prior on the objective function. One popular natural image prior is total variation (TV). It encourages sparse gradients in the image and is defined as follows:

$$TV(x) = \sum_{i,j} \sqrt{(x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2} \quad (1)$$

For the regularization we follow [12] and choose the L6-norm. Combining the above yields the following loss function:

$$Loss(x) = \|\phi(x_0) - \phi(x)\|_2 + \lambda_{TV}\text{TV}(x) + \lambda_{L6}\|x\|_6 \quad (2)$$

Additionally, guided by [14], we implement "small-contribution clipping". Small-contribution clipping sets pixels to zero that contribute little to the loss function, thereby reducing unwanted background noise in the image and increasing the contrast between important features and the background. The matrix of approximated contributions C of single pixels to the loss function can be approximated by multiplying the gradient $\nabla_x Loss(x)$ of the loss function with respect to the pixel values with the pixel values x themselves, summing over all three channels R, G, B and then taking the absolute:

$$C = |\sum_{R,G,B} \nabla_x Loss(x) \circ x|$$

C is a linear approximation of how much every single pixel impacts the loss. Pixels that are within a certain percentile

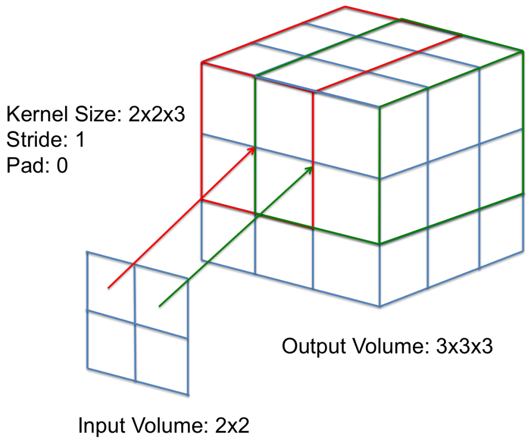

Figure 2: Upconvolution essentially works like the backward pass of a normal convolution layer. The values of the input volumes are multiplied with the upconvolution kernel and added to the output volume.

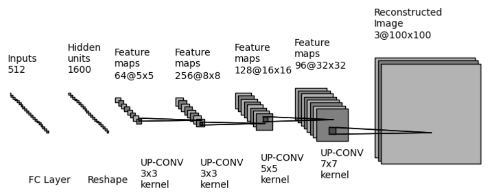

Figure 3: Basic architecture of our upconvolutional network used to invert PoseNet

of this distribution can now easily be clipped to zero.
We can find the reconstructed image by using a gradient descent method to minimize the above loss and clipping pixels with low contribution to zero in every iteration.

3.2. Upconvolutional Neural Networks

Our second approach to go from feature descriptor to a reconstructed image is via an upconvolutional network. Such upconvolutional networks can act as effective decoders for the forward network [2, 3]. In the first layer of the network the feature descriptor representing the embedding of the input through PoseNet is reshaped into a 3D volume. This 3D volume is repeatedly passed through upconvolutional layers, nonlinearities (ReLu) and batch normalization layers until we reach the original image space (of dimension 3x100x100). Figure 2 explains the workings of an upconvolutional layer.

The basic architecture of the upconvolutional network used is given in Figure 3. The architecture of the upconvolutional network is modeled after the PoseNet architecture, but in reverse order. In addition we add a batch normalization layer after each nonlinearity. Batch normalization ensures that the input distribution to subsequent layers is fixed by normalizing the mean and variance of its inputs [8]. This lets us use higher learning rate and the network is less sensitive to weight initialization. It also acts as a regularizer and eliminates the need for additional regularization like Dropout. The loss function that we use for learning the weights in the decoder network is the L2 (mean squared error) loss between the generated and the original image.

4. Dataset

Our dataset consists of 800,000 street view images depicting roadside buildings from different view points and angles provided to us by [15]. The original images are 640x640 RGB images. We resized the images to 100x100, transposed the channels to BGR and subtracted the dataset mean of every channel. We then passed all images through PoseNet and extracted 800,000 feature descriptors from the CNN to complete the dataset. For the sake of training the generative CNN we split the data into 650,000 training, 100,000 validation and 50,000 test examples. Because of the considerable size of the dataset we did not use any kind of data augmentation.

5. Experiments/Results/Discussion

5.1. Frameworks and infrastructure

All experiments were conducted on Stanford's Sherlock computing cluster on single NVIDIA Titan or GTX GPUs. The upconvolutional network was implemented using Torch [1], while we used Caffe [9] for the computations needed for the inversion of the feature descriptors extracted from PoseNet. For the experiments involving Convolutional Neural Networks, batch sizes of 64 images were used for training - in a benchmark comparing batch sizes of multiples of 64 up to 960, this, unexpectedly, proved to be the most efficient batch size. We used Adam [10], a first-order gradient-based method, to optimize all the objective functions proposed in section 3. All of the training was done on "cities" dataset, developed in Stanford Computer Vision and Geometry Lab.

5.2. Inverting the feature descriptor

Our metric for a successful inversion of the feature descriptor is the relative loss between the feature descriptor of our generated image and the descriptor of the original.

$$RelativeLoss(x) = \frac{\|\Phi(x) - \Phi(x_0)\|_2}{\|\Phi(x)\|_2 + \|\Phi(x_0)\|_2} \tag{3}$$

As a qualitative measure we looked at the visual interpretability of the generated images. To find a set of parameters that would yield the most interpretable images we

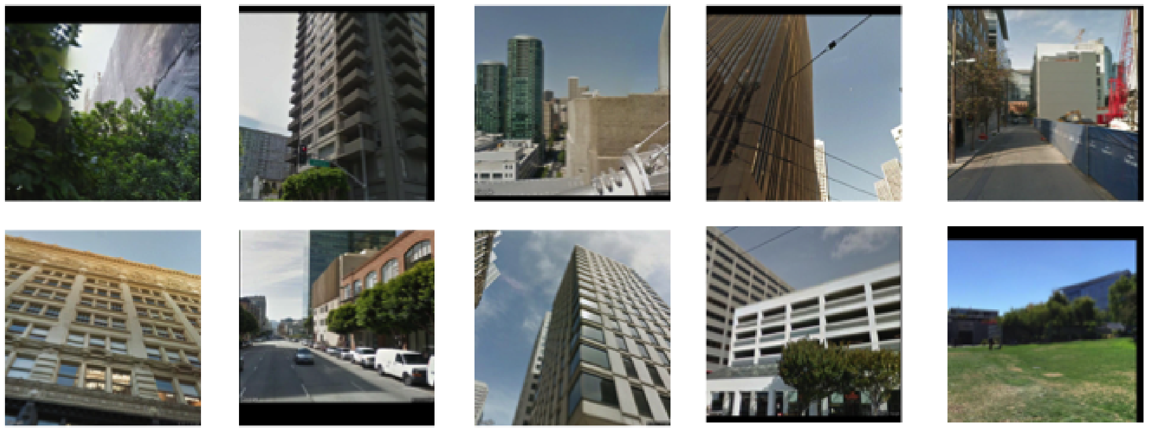

Figure 4: Ten random samples from the street view dataset.

Hyperparameter	Value
ADAM parameters	
$LearningRate$	26.60
β_1	0.90
β_2	0.99
ϵ	$1 * 10^{-8}$
Regularization parameters	
λ_{L_6}	$1.95 * 10^{-15}$
λ_{TV}	$4.00 * 10^{-6}$
$ClippingPercentile$	21.40

Table 1: Best set of hyperparameters for inverting the feature descriptor

conducted an extensive hyperparameter search. In an initial search we set the regularization strengths (λ_{TV} and λ_{L6}) to zero and took 1500 random samples for the learning rate from a log-space between 1e-10 and 1e4. For β_1, β_2 and ϵ we chose the default values suggested in [10]. The best parameter set we found yields a relative loss of 0.08.

We found the relative loss to be insensitive to the amount of high-frequency patterns and noise in the reconstructed image, which both strongly impact interpretability. Choosing the right regularization is therefore vital to find a readily interpretable image. Continuing the search for suitable hyperparameters, we looked at 2500 randomly sampled combinations of learning and regularization parameters and chose the best set of parameters by doing a visual comparison between the reconstructed images.

Figure 5 shows inversions from both the conv4 layer and the fc5 layer of six city images from the test set. It is interesting to see that the feature descriptor does not encode either color or texture information of the original image, as the reconstructed images lack these features.

The most striking feature retained in the reconstructed images are the dominant edges pointing toward the vanishing point in the original image. This often coincides with the edge between the buildings and the sky: Whenever the image exhibits such an edge, this edge is the most significant feature in the reconstructed image. The neural network also exhibits a strong preference for edges in the image that align with this major feature: this is most evident when looking at windows, where vertical edges are seldomly found in the reconstructed images, while horizontal edges are often still present. Looking back at one of the tasks PoseNet was originally trained to do, it seems to make intuitive sense that these dominating edges are an important feature to do camera pose estimation.

5.3. Upconvolutional Neural Networks

Our metric for the successful reconstruction of images from their feature descriptor is the training and validation loss of the upconvolutional neural network, on the quantitative side, and a visual analysis of the reconstructed images, on the qualitative side. Similar to subsection 5.2 we first conducted a coarse hyperparameter search sampling the learning rate randomly from a log-space. For the coarse search we only trained the network for 500 batches and evaluated the parameters on the training loss to find a range of parameters that allowed the network to converge. After narrowing down the search space we did a second hyperparameter search, this time choosing the best set of parameters based on the loss calculated from 1000 batches randomly sampled from the validation set. The best set of hyperpa-

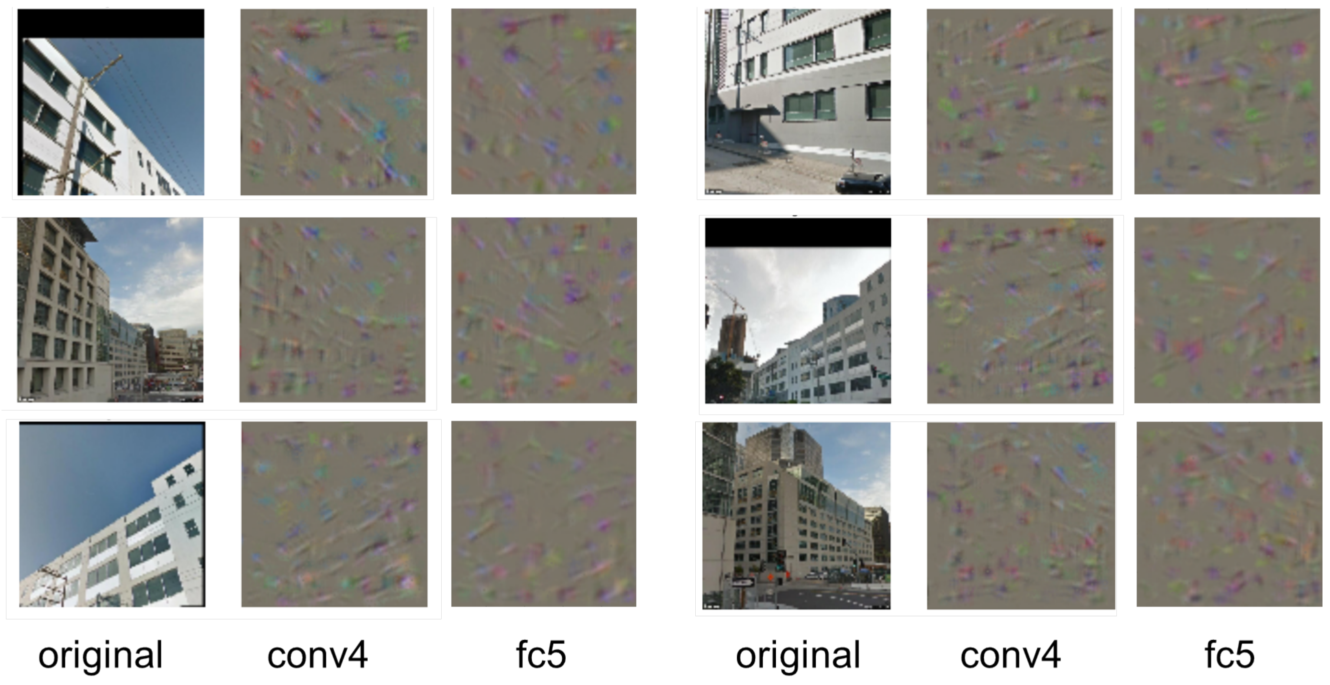

Figure 5: The original and reconstructed images side by side. fc5 are the images reconstructed from the feature descriptor extracted at the end of PoseNet, while conv4 are the images reconstructed from the feature descriptor extracted after the fourth convolutional layer. It is interesting to see that color information of the original images is lost, while the dominant edges pointing towards the vanishing point in the original image are retained.

Hyperparameter	Value
ADAM parameters	
$LearningRate$	26.60
β_1	0.90
β_2	0.99
ϵ	$1 * 10^-8$

Table 2: Hyperparameters for the upconvolutional neural network

rameters is listed in 2. We found that, as specified in [10], β_1 and β_2 had very little impact on the optimization process.

To test the capacity of our network and ensure a correct implementation we overfit the network heavily by repeatedly showing it the same images from a small excerpt of the training dataset. The results can be seen in Figure 6.

Thereafter we used the best set of hyperparameters to train the upconvolutional neural network for 140,000 mini-batches, or an equivalent of 14 epochs or 8.96 million images. Figure 8 depicts the validation and training losses over time. Both losses drop significantly in the first 1000 batches and subsequently stagnate. The two drops in the loss curve happen after manually dividing the learning rate

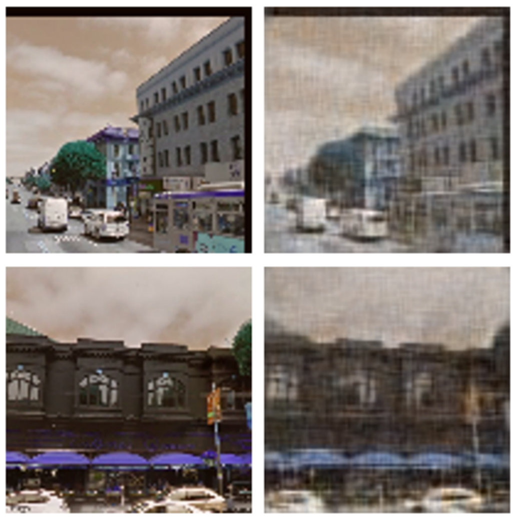

Figure 6: Two reconstructions from overfitting the upconvolutional Neural Network on a small dataset

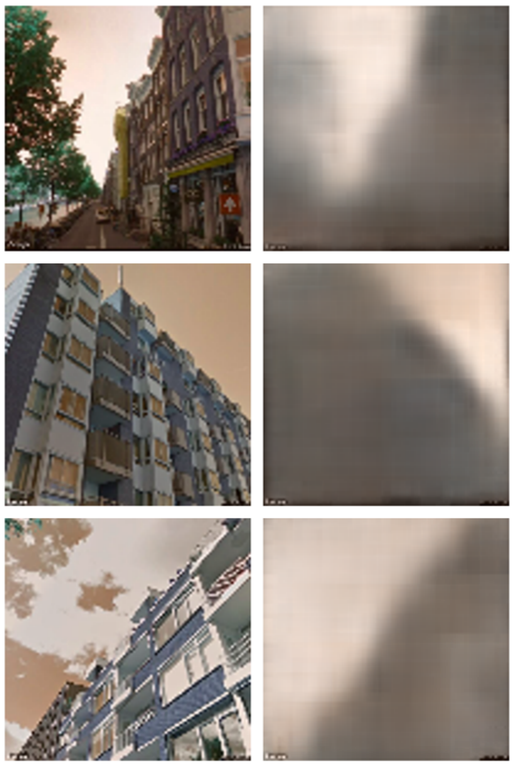

Figure 7: Inversions of city images achieved with the fully trained upconvolutional neural network

by 10. The small difference between training and validation loss is indicative of little overfitting.
The stagnation of the two losses in the observed learning curve has three likely explanations: The upconvolutional network used is not powerful enough, the optimization process is not working properly (wrong hyperparameters or weight initialization) or the network has exploited all information retained in the feature descriptor that is useful for image reconstruction.

To check for the first possibility we trained a deeper network (with additional convolutional layers of depth 256). We then did an independent hyperparameter search for this neural network and trained it on 2000 batches (past the 1000 batches where the less powerful network had already more or less converged) of imagery with the best hyperparameter set we found. We found that this second, more powerful network did not do significantly better and that adding more layers does decrease neither training nor validation loss.

We conclude that the network was indeed powerful enough. Seeing that we did an initial, extensive hyperparameter search and used learning rate decay, we thus find that the network exploited all information contained in the feature descriptors of the images.
Figure 7 depicts a choice of a few images generated by the fully trained upconvolutional neural network.

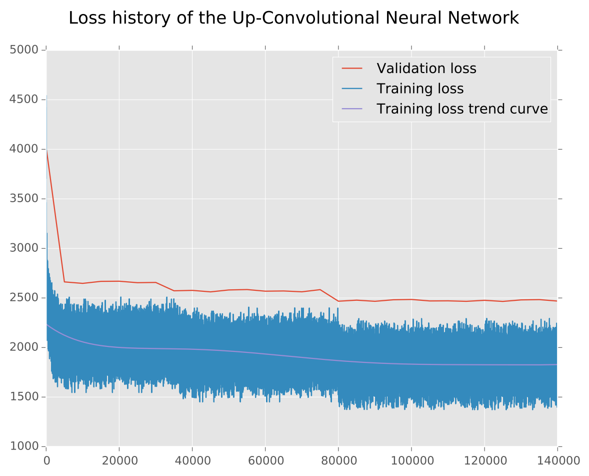

Figure 8: Plot of training and validation losses for the upconvolutional Neural Network

Building on the hypothesis that this network fully exploits the information contained in the feature descriptors these images allows for a reasonable interpretation: to detect the pose of buildings in a city, the essential information necessary is the silhouette of this building against the sky. This agrees with the results of the feature descriptor inversion that demonstrated the network's preference for the dominant edge found in the image. However, we find that the inversions produced by the upconvolutional neural network do not show any other significant structure: edges other than the shilhouette of the building against the sky are not visible, although they are clearly significant to the feature descriptor as demonstrated by figure 5. This is likely due to the L2 loss used as an objective function: The inversions produced by the optimization method suggest that the feature descriptor does not embed precise localization of every edge in the image - L2 regularization in image space will average over all possible locations and will thus likely lead to blurry images. This hypotheses motivated another experiment: The implementation of a Generative Adversarial Model.

5.4. Additional Experiments

5.4.1 Generative Adversarial Network

To reconcile the shortcomings of the L2-loss, [4] suggest to complement it with a new class of loss functions. Similar to generative adversarial networks, proposed in [7], the idea is to train a discriminator network to differentiate ”real” images from those generated by the upconvolutional neural

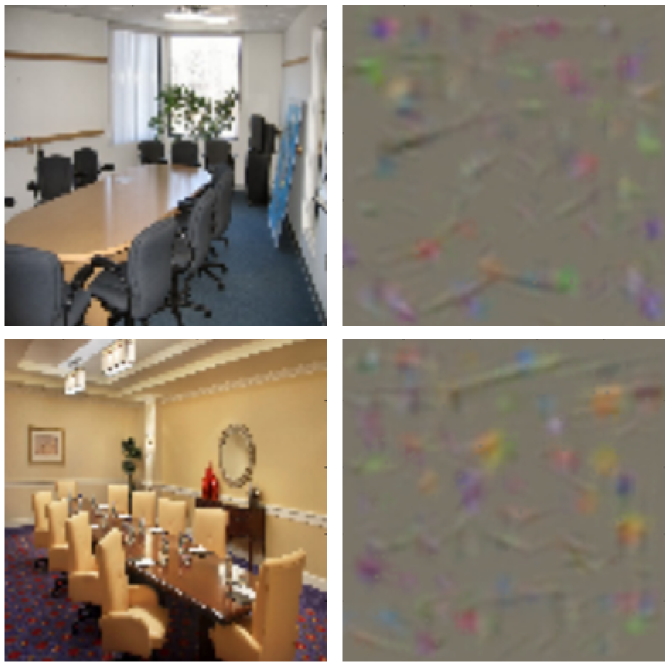
Figure 9: Sample of inversions of images of indoor scenes. These images suggest that the CNN is also able to pick up dominant edges in scenes it has not been trained on.

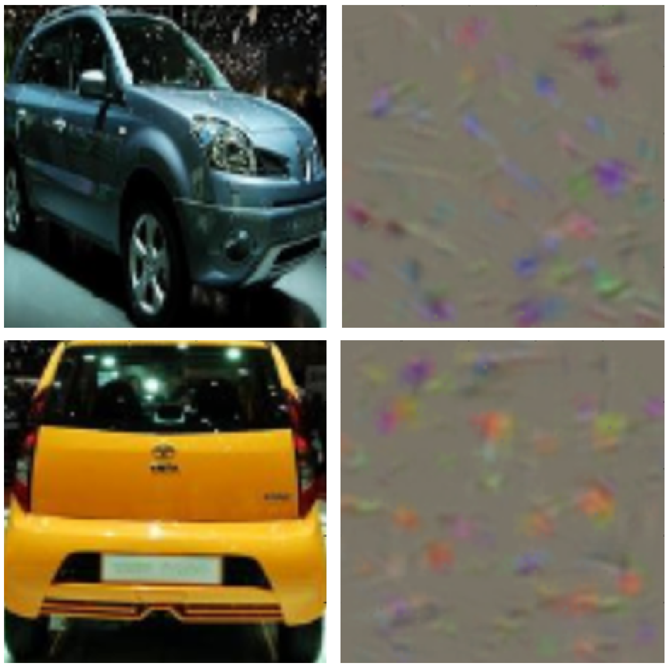
Figure 10: Sample of inversions of images of cars. These images suggest that the CNN is also able to pick up dominant edges in scenes it has not been trained on.

network. The generative network will then, in addition to minimizing the L2-loss between the reconstructed and original image, try to fool this discriminator network. If trained correctly, the loss function defined by the discriminator can be interpreted as a prior favoring realistic images and will lead to much sharper and more realistic images.

We briefly trained a generative adversarial model after hypothesizing that the upconvolutional neural network trained with the L2 objective did in fact exploit all information embedded in the feature descriptor, but did not produce sharp images due to the limitations of the L2 loss. However, we did not succeed in balancing the training of the generative network and the discriminative network - training did not converge, and due to the time constraints of this project we were not able to tune the pipeline to function properly.

5.4.2 Other image datasets

Even though the network was trained on the ”cities” dataset, we wanted to see if the network is still capable of learning edges contributing to vanishing points in completely different scenes, with different image statistic. We thus ran the same experiments on two other datasets - ”cars” and ”indoor”. Sample images from the datasets, along with the reconstructed images, are shown in figures 10 and 9.

These experiments show that the network is able to pick up dominant edges in the images that contribute to vanishing points. Further, the forward network only retains dominant edges that give perspective information, while other edges are ignored, even if they are dominant in terms of RGB contrast.

6. Conclusions/Future Work

In this project, we have investigated which information is retained in the feature descriptor learned by PoseNet with both optimization and learned upconvolution methods. We found that the most significant feature embedded by PoseNet in the feature descriptor are edges that give strong cues about object pose. As opposed to AlexNet, PoseNet does not retain significant color or texture information in its feature descriptor. Both optimization and learned upconvolution delivered relevant insights, however, optimization-based approaches lead to more detailed inversions. Future work can be directed at comparing the generated inversions with inversions of the same images obtained from AlexNet, allowing insight into what features are unique to PoseNet and thus likely to give strong cues about object pose.

7. Acknowledgments

Special thanks to Amir R. Zamir and the Computational Vision and Geometry Lab at Stanford that provided us with the PoseNet, the street view dataset and advice throughout the project.

References

[1] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[2] A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. *CoRR*, abs/1506.02753, 2015.

[3] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. 6 2015.

[4] A. Dosovitskiy and T. Brox. Generating images with perceptual similarity metrics based on deep networks. *CoRR*, abs/1602.02644, 2016.

[5] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *Proc. IEEE CVPR*, 2014.

[6] J. Flynn, I. Neulander, J. Philbin, and N. Snavely. Deepstereo: Learning to predict new views from the world's imagery. 6 2015.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[8] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[11] 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.

[12] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. 11 2014.

[13] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *Iclr*, pages 1–, 2014.

[14] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. page 12, 6 2015.

[15] A. R. Zamir. CVG Lab, CS Department, Stanford, zamir@cs.stanford.edu.

[16] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833. Springer Verlag, 2014.