JCGAN: Generative Image Composition

Xuaner (Cecilia) Zhang, Qijing (Jenny) Huang
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{cecilia77, qijing.huang}@berkeley.edu

I. Problem Statement and Background

A. Problem Statement

We aim to automate image composite, an image processing technique (see an example in Figure 1) that combines multiple image sources (or patches) to create the illusion that all visual elements are part of the same scene. The goal is to generate composite images in a seamless way that humans cannot distinguish the generated image from natural images.

Specifically, three attributes are used to evaluate image realism:
1) Lighting consistency: whether colors of object and background images match
2) Spatial consistency: whether the object is inserted at a reasonable position in the image
3) Semantics: whether the composite image has objects that co-occur in real world

In terms of application of image composite. With better understanding of realism in image composite, objects can be re-rendered into the scene with higher fidelity. Given a composite image that does not look realistic, we are able to use the trained network to apply enhancements to improve the perceptual realism of the composite image.

Photo forensics in the mass media would also be a vital application. Understanding on how realistic images are composite could help detect photo splicing, which is widely used in media to mislead the public.

B. Background

To composite images, people normally use advanced image editing software such as Photoshop to create realistic composite images with careful image adjustments, which involves large amount of tedious hand annotations.

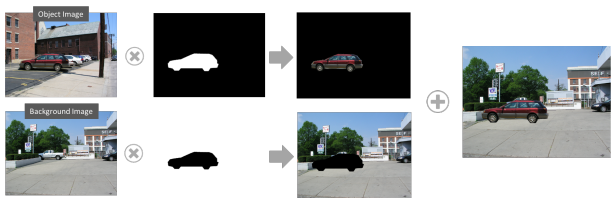

Fig. 1. standard pipeline for image composition (image source from LabelMe dataset [1])

The graphics research community has also been exploring generative models to composite image in realistic ways. Early work by Lalonde et al [2] assesses composite image realism using color compatibility by matching co-occurring color palette and matching object and background color distribution. With the success of convolutional network for feature learning, recent work by Zhu et al [3] reaches the state of the art performance for generating composite images. They used pre-trained features from the VGG16 [4] network and trained a classifier that discriminates real images from fake (generated) ones. Then they fix their trained discriminator and used optimization to iteratively enhance color compatibility. Their work produces realistic composite images, but the system is not end-to-end and requires manual optimization.

Our composite image generation task is closely related to the recent work of generative adversarial network (GAN) [5]. In the work of GAN, the network is able to generate images that lay on the manifold of real images with an adversarial loss added to the discriminator. An equilibrium state is reached when the discriminator is not able to distinguish between the generated images and real images. Applied to our task, an equilibrium state is when the generator outputs composite images that the discriminator thinks are real. This means the binary discriminator would produce an accuracy of 50%, equivalent to random guess. Another inspiration of our network architecture design is the work of Siamese network, in which the network structure is suitable for data pairs. In our task, we have a pair of image as input: one object image and one background image. We decide to share the parameters (filters) between the two streams to make their extracted features be correlated.

C. Data

We use Labelme [1] as our data source; the dataset contains images with labeled objects and corresponding annotations. In total it has 2920 images with 32164 annotated objects.

Our task is to composite objects into a background image, and thus we require dataset that has object segmentation. Labelme is a suitable dataset since it contains polygon masks that accurately segment objects.

D. Evaluation

Qualitative Evaluation There are three commonly-used metrics for evaluating generative models: average log-likelihood, Parzen window estimates, and visual fidelity of samples. According to [6], these metrics are largely independent of each other, so the generative models should be evaluated directly with respect to their intended application(s). For image synthesis, [6] suggests that a subjective evaluation based on visual fidelity of samples is appropriate. For this project, we thus decided to evaluate our models based on visual fidelity of samples. We plan to ask volunteers to participate in a

experiment to see if they can distinguish the generated images from the real one. Another potential evaluation method is to do inference on trained classification networks to compare the performance with that tested with real images. This means we use the discriminator as feature extractor. Then the classification accuracy would be a fair qualitative evaluation.

Quantitative Evaluation A quantitative measure to evaluate the color transformation is presented as follows: we compute the relative color distance between the mean pixel value of the inserted object area and the mean pixel value of the rest of the image. Denote the color distance as D, it is defined as:

$$D = \frac{\|\bar{P_{obj}} - \bar{P_{bg}}\|}{\bar{P_{bg}}}$$

The intuition behind the color distance measurement is the grey-world assumption, in which it assumes consistent and smooth reflectance (grey value) of the real world scene. Therefore, a more realistic and visually-pleasant composite should have consistent reflectance level throughout the image. The mean pixel value of the inserted object area should be similar to its background image region.

II. Approach

The input to our network are triplets: object images, background images and mask images. Object images and background images are used to learn the transformation parameters, which will be applied back to object images. The mask images are used to composite the background images with object images (the system flow can be seen in the network architecture illustration in Figure 4). Following we would discuss how we prepare these image sets.

A. Dataset filtering

Object images Labelme has noisy and wrong object labels, and many object categories have so few instances that makes it very hard for training. In order to avoid class in-balance and ensure sufficient images within one class for training, we only keep the largest object class, which is 'cars'. This sub-dataset makes the training being more efficient. Notice that by limiting the generator to only see images with cars, and to composite only car objects, we have to also craft real images (seen by the discriminator) so that the images also contain cars. Otherwise, the discriminator can easily overfit on the existence of car objects as a trivial solution to distinguish between generated and real images.

Background images For background images, we only keep the outdoor scenes and remove the indoor ones.

There are multiple objects (in our case, cars) in one image (see the left image in Figure 2), and we separate the different instances within one image by duplicating the image by the number of cars in it. We eventually filtered out 7912 car objects. Ideally, we hope to get only objects that are not occluded, but the occlusion flag in Labelme is too sparse and noisy that we decided not to use the flag in the end. This introduces potential difficulty in training since the discriminator may also take the simple cue of whether the inserted object is complete or not.

Fig. 2. example images in our car dataset

B. Pre-processing

We notice that for high-level and complicated task such as image composite, data pre-processing is crucial for training. The reason behind is that the processing should be carefully considered to avoid introducing any bias. Based on the network architecture we designed for this task (see Figure 4), one problem is how the object images be fed to the Siamese streams in the generator. Denote the background image as A and the object image as B. A car object exists in image B. To feed in the object into the network, we could either input the full image B, or only the cropped car object in B. The intuition of feeding in the full image is that the image content other than the car object in image B also effect the adjustment of the car object (e.g. the light source information could be an indicator of how the car's color should be adjusted). The reasoning of feeding in only the cropped car object in image B is that the network is able to focus on the object but not the rest of the image (similar to network attention). We experimented with both options, which will be discussed in the result section.

We processed two sizes for background and object images: 64 by 64 and 256 by 256. The smaller sized (64 by 64) are input to the Siamese streams to train for the transformation parameters. To generate more visually-pleasant images, the higher resolution ones (256 by 256) are used to generate composite images to feed into the discriminator. This means that the real images input to the discriminator are also 256 by 256.

To briefly summarize, the operations we did to process data are enumerated as follows:

For object images:

1) keep images with car objects
2) keep objects that have a relatively precise polygon mask (with more than 6 control points)
3) remove objects with either dimension that is smaller than 16
4) resize images to 64 by 64 and 256 by 256
5) normalize the images to [-1, 1]

For background images:

1) keep only outdoor images
2) remove images that are of size smaller than 256 by 256
3) resize images to 64 by 64 and 256 by 256
4) normalize the images to [-1, 1]

III. Baseline and Model

A. Baseline

Two methods are used as baseline for evaluating our model: 1. state-of-the-art [3] 2. cut-and-paste. The first method represents the state-of-the-art work for image composition. It uses CNN as a classifier to provide guiding information to image color transformation. Differing from our method, this work did not use a generative network which can automate the process of compositing images. The second method serves as the most intuitive baseline for our automated task. By cut-and-paste we meant that, we directly copy an object from one image and paste it to the same position on the image we use as background. Figure 3 shows samples from cut-and-paste images.

Fig. 3. Baseline Images

B. Model

The architecture of our network model resembles the original GAN [5] with two competing neural networks — a generator and a discriminator. The generator is trained to generate more realistic sample images while the discriminator is trained to distinguish between the generated images and the real images. We are inspired by the adversarial training technique, however, instead of learning a generative model, we learn a transformation model that is able to transform a composite image into a realistic one. More specifically, our network takes three images as input: background, object and mask. Our end goal is to learn a composite function that accounts for lighting, spatial and semantics information to apply to the object image, so that after the object image is blended to the background image, the composite image looks seamless. The generator outputs parameters as the color and spatial transformer to apply to the object. The discriminator will then take in the synthesized image, together with a training set of real images, to train a binary classifier.

Figure 4 shows the architecture of our original model. We first train the background and object images together

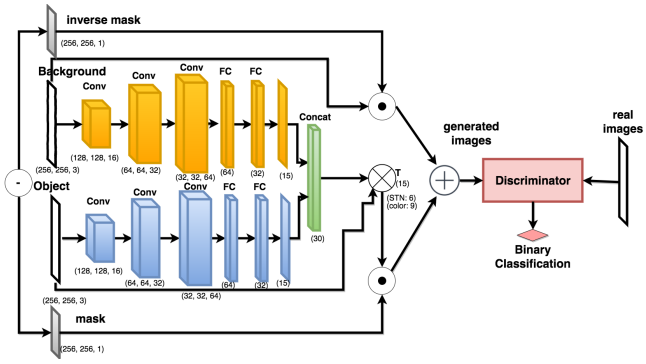

Fig. 4. JCGAN Network 256

using a network similar to the Siamese Network [7], which is commonly used to train data pairs. The images are of size 256 by 256. The Siamese net consists of two streams that can share weights. Each stream is composed of 3 convolutional layers and 3 fully connected layers (with Relu and max-pooling layers in between). The generator containing the Siamese net outputs a 3 by 3 matrix for color transformation and a vector of 6 floats for Spatial Transformer Networks [8] (STN); the weights for each layer are shared. For the discriminator, we directly use the discriminator from the DCGAN [9] with 4 convolutional layers. Sigmoid cross entropy is used as the loss function and Adam is used as the update rules for gradient descent.

We attempted to add more layers and output channels in each layer to extract more features. However, the network size is limited by the our GPU memory, so we decided to modify our generator network to take 64 x 64 images as input. The new model is shown in Figure 5. For the object images, we cropped the cars from its original 256 x 256 images instead of training with full images. We assumed that the background of the object image contains few important information. The background images are directly resized to 64 by 64. By reducing the size of images, the number of output feature maps of each convolutional layer and the number of images per batch can be increased. Aside from that, we also added in a l2 norm of the generated color filter to prevent color number overflow/underflow.

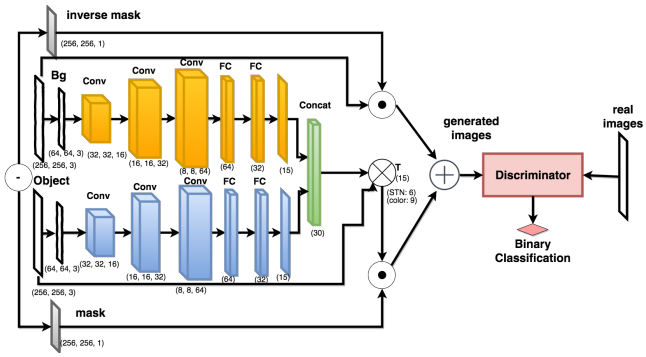

Fig. 5. JCGAN Network 64

Code for JCGAN can be found at: https://github.com/hqjenny/JCGAN.

IV. Experiments

This section gives a description of three separate experiments to verify the effectiveness of the network.

We realize that it is hard to consider the color and spatial transformation together without knowing how each could work effectively. If one of the two tasks is not trained well, the discriminator could easily pick up certain artifact to tell the generated images as fake. For example, if the car is not positioned at a reasonable position, the discriminator would distinguish real from fake images by only looking at the position of car being reasonable or not. This would prevent it from learning color transformation as desired. Therefore we separate the task of learning color and spatial transformations. During the learning of color transformation, we ensure the position of object to be correct by inserting the object into its original image background, and perturb the color of the objects to make it incompatible with the background.

A. Color Transformation

Given the fact that there is no end-to-end network architecture that is trained to learn the color transformation, we divide our task into two sub-tasks: 1) whether the network is able to interpret the real/fake loss back propagated from the binary classifier to learn the color transformation. 2) whether the network can be generalized to learn a color transformation that leads to compatibility between the object and background image pair. In the following we will elaborate how we approach the two sub-tasks:

Experiment 1 To solve the first task, we did an 'overfit' experiment. The goal is to test whether the supervision on the binary classifier can be transferred to learning a color transformation. Specifically, we only use a small subset (1600 images) of the entire training data. We perturb the color of the object image, and use its original background as the background image. Most importantly, we input the non-perturbed image to the discriminator to offer a trivial solution to the network. The network should be able to figure out that as long as the composite is made identical with the original image, the discriminator would be fooled. And our goal is to see whether the goal of making the composite image identical to its original image can be transferred to learning a color transformation that transforms the object's color back to its original.

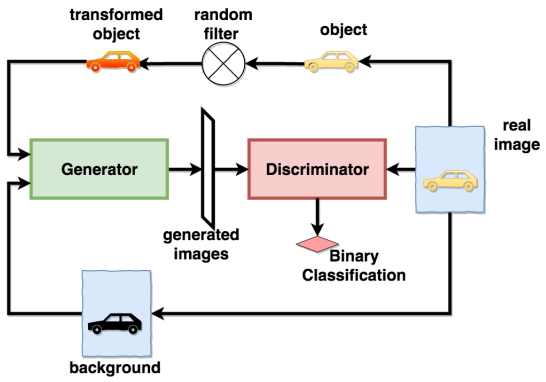

Fig. 6. Experiment 1

Experiment 2 After making sure that the network is able to use the discriminator's supervision to learn a good color transformation, we proceed to our next task of network generalization. Instead of offering the discriminator the original image as a trivial solution, we feed in random real images. The goal is to see whether the network can generalize to produce color transformation that makes the object image compatible with the background image.

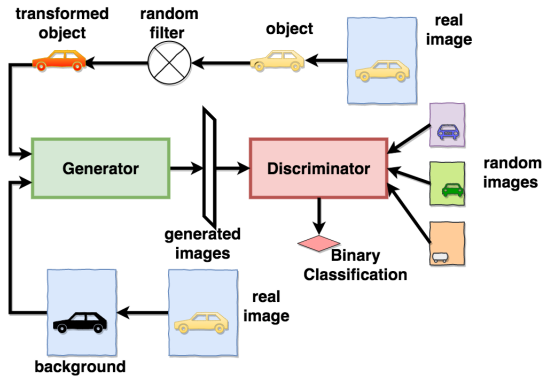

Fig. 7. Experiment 2

V. Results

A. Color Transformation

Experiment 1 Figure 8 shows the training results from experiment 1. The cars in the initial statem shown in the left image of Figure 8 are transformed with a random color filter. The color of cars are intentionally made to look unnatural. After 100 epochs of training, the color of most cars in the images is recovered to be close to their original colors, as shown in the right image in Figure 8. The results are used as a sanity check for our network to verify that it is able to learn the color transformation with light supervision, which is the loss propagated from the binary discriminator.

Experiment 2 When we first started experiment 2, we ran into a numerical overflow issue. Once the norm of the color transformation gets large, the pixel values overflow and the objects turn into mosaic-like images as shown in the right image of Figure 9. To prevent overflow/underflow, we added a L2 regularization on the color filter to bound the filter range. The regularization is added to both data processing and training: during data preparation, we normalized the color filter used to perturb the object images at the initial state, and during training, we also applied a l2_reg on the generated color filter as a L2 loss. The final state 10 shows a more stable result we got after adding the L2 regularization.

Fig. 8. Experiment 1: (LEFT) Epoch 0, the Initial State (RIGHT) Epoch 100, the Final State

Fig. 9. Experiment 2 - overflowed: (LEFT) Epoch 0, the initial State (RIGHT) Epoch 100, the final state

Fig. 10. Experiment 2 - regularized: (LEFT) Epoch 0, the Initial State (RIGHT) Epoch 10, the Final State

cut-n-paste	ours	state-of-the-art
1.24	3.89	4.12

TABLE I. USER STUDY ON IMAGE REALISM.

Quantitative Evaluation From the reflectance level described earlier, we compute the average color distance among 50 randomly selected generated images, and compared with the average color distance among those original composite images. A large color distance indicates a big color discrepancy while smaller color distance refers to more consistent color. The average distance for generated images is **0.0899**, while the original perturbed images has an average distance of **0.6635**. This indicates a big improvement of our color-transformed images.

Performance Comparison As discussed earlier, conducting user study is an effective way to evaluate generative models. We thus did a user study on 20 users to evaluate the realism of our composite images. Specifically, we present each image and let the user to rate it with a number in between 1 and 5, 1 being the least realistic and 5 being the most realistic. Three sets of images are presented to the users: randomly perturbed images at their initial state, images trained after 10 epochs, images photoshopped with careful manipulation. The images are selected randomly and presented in random order. An example of images from the described three sets is shown in Figure 11. The averaged user rating of 20 images is listed in Table I.

Fig. 11. (LEFT) image from the random initial state (MIDDLE) image trained using our network at epoch 10 (RIGHT) image from [3]

Both our trained composite images and the results from [3] have higher user evaluation score than the images from initial state. Although the score from [3] is higher than our result, their approach requires further optimization while ours is an end-to-end system.

B. Spatial Transformation

STN is a point-wise affine transformation as shown in equation 1:

$$\begin{bmatrix} x_i^s \\ y_i^s \end{bmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{bmatrix} x_i^t \\ y_i^t \\ 1 \end{bmatrix} \tag{1}$$

θ is the 6-bit STN filter learned from the network. We are still working on tuning the network for learning the correct spatial transformation. Figure 12 shows the sample images we got from one result. Currently the network can transform the shape of the objects, but barely change their position. We found that the shifting factors θ_{13}, θ_{23} are normalised and need to be rescaled to the image size in order to move the object significantly. The learning rates of the geometrical transformation factors θ_{11}, θ_{12}, θ_{21} , θ_{22}, and of the shifting factors θ_{13}, θ_{23} might also need to be tuned separately.

Moreover, we can see that the mask of car in the second image is not detailed enough, which may cause the discriminator to overfit on the mask. Thus, more data pre-processing might be needed to filter out the objects with bad mask.

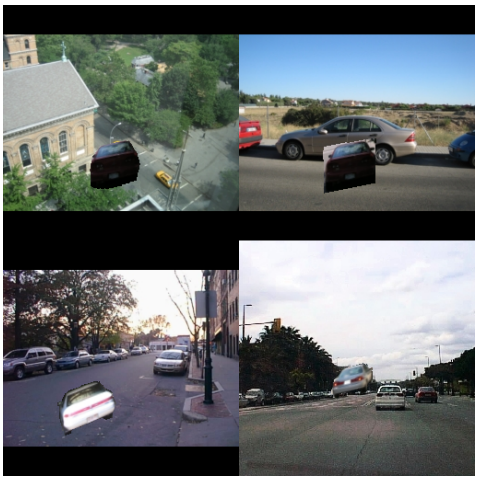

Fig. 12. Spatial Transformation: Epoch 100 (Final State)

C. Full Network

We combine the color and spatial transformer network and train them end-to-end. There are 7920 object and background pairs in total, 90% are for training and 10% for testing. Discriminator loss is set to cross-entropy loss as proposed by the original GAN paper.

Learning curve The equilibrium state of a GAN is a point at which neither the discriminator nor generator can improve their individual loss functions. However, there is no guarantee that the equilibrium state is stable, or proof of the existence of that point. This makes it hard to read out improvements and convergence purely based on the training curve. That said, we examined the training curve of our training and have found some reasonable patterns that serve as a hint for convergence.In Figure 13, we plot the loss history for both the generator and the discriminator. Loss of the discriminator is further divided into loss for real images (labeled 1) and loss for fake/composite images (labeled 0). Although the training curves are locally unstable, there is a promising trend of loss decay for the generator. Notice that a desired training curve for the discriminator is a relatively high loss for fake images. At the beginning of the training, when the generator is very weak, the loss for fake images is expected to be very low. As the generator is trained to be stronger, the generated images are more like real images and could be labeled wrongly as 'real'. However, even if we see noticeable improvements on composite images based on appearance, we did not see an correlated pattern on the discriminator loss. Though the loss on real and fake images are becoming more balanced, we found it hard to interpret network performance based on the learning curves.

Composite results We visualize the result by plotting a random batch of composite images using test image pairs, and

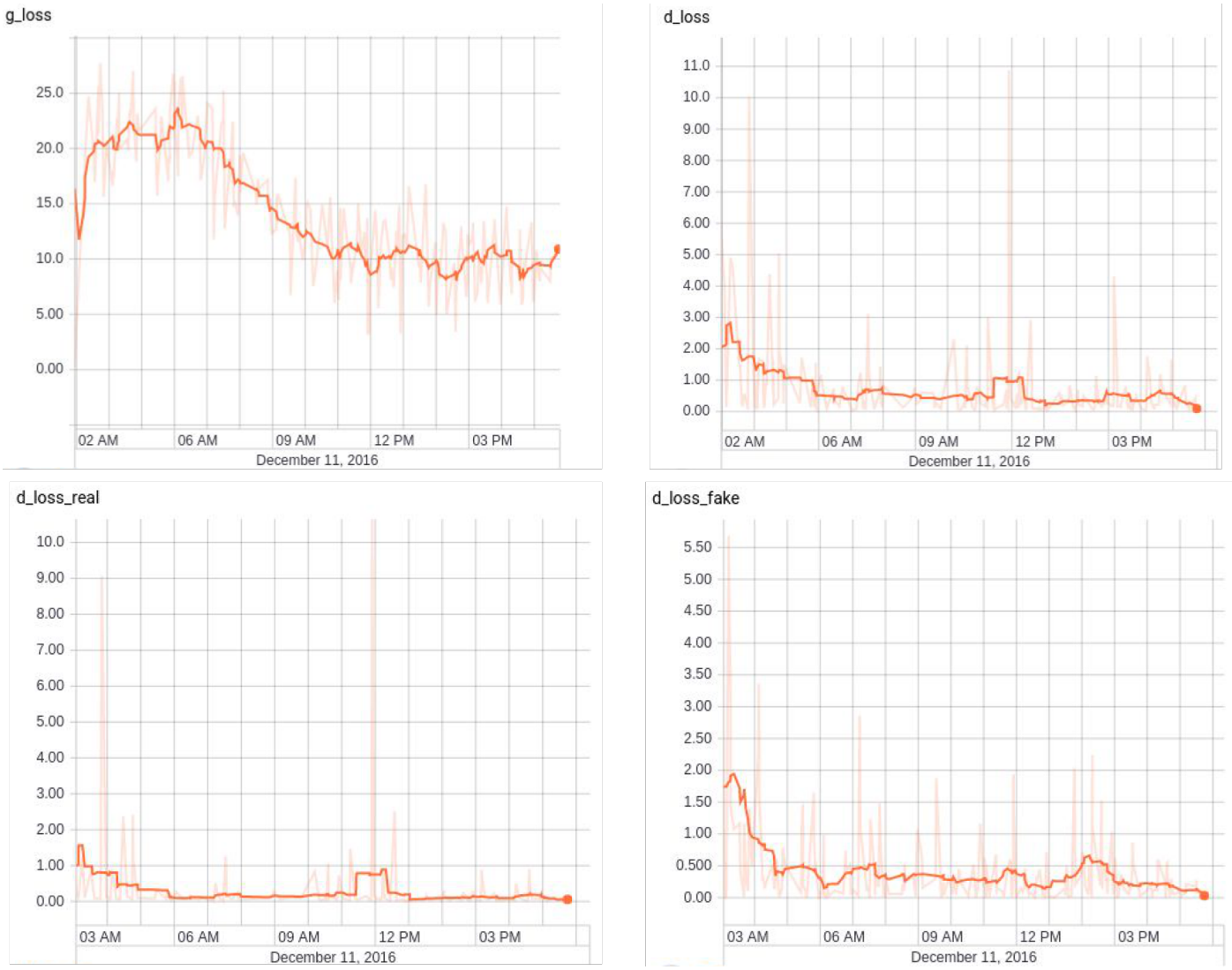

Fig. 13. Generator and Discriminator Loss

compared their initial and final epoch states. From the initial state shown in the left image of Figure 14, we can see some obvious color artifacts. While in the right image, we are able to see improved composites with normal colors.

Camouflage phenomenon An interesting result we've found is that as we train the network longer, it tends to 'hide' the object by washing out its colors, as seen from the result of epoch 39. The inserted car objects are detected as the 'alien' object by the network, and are colored with similar colors of its surroundings. For example, row 1 column 4 colored the inserted object's color with similar color of the sky; row 4 column 2 colored the car with similar color as the background building.

One explanation to this 'Camouflage phenomenon' is that the network tries use the count of car objects as a cue to tell whether an image is real or fake. All the real images seen by the discriminator contain only one car object, while all the composite images have one extra car object inserted into the original image. Thus the network tries to smooth out the inserted object to make it seem 'real'. A potential way to avoid this network bias is to replace car object instead of inserting an extra one. However, due to time limit, we leave this as future work.

Fig. 14. Test: testing results (LEFT) Epoch 0, the Initial State (RIGHT) Epoch 11, the Final State

Fig. 15. Test: testing results (LEFT) Epoch 11, the (RIGHT) Epoch 39

VI. Tools

We used a C++-based Python APIs called Tensorflow as our development tool. It auto-derives the gradient of the network for back propagation, which helps to improve our productivity. Compilation errors will pop up if there is no gradient can reach specific variables. This feature prevents us from designing dysfunctional layers. Differing from the layer-specific libraries (e.g. Caffe), the network in Tensorflow is constructed as data-flow graph, which enables inter-layer optimizations to improve the speed. Besides, Tensorflow includes a graph visualization library called Tensorboard. With the visualization support, it is easier for the users to understand and tune the network. Figure 16 shows a top-level data-flow graph of our network generated by Tensorboard. By inspecting the inputs and outputs of nodes on the generated graph, we were able to debug the network in a more straightforward way. Lastly, Tensorflow also supports multiple GPUs, which allows more computing resources to be used to accelerate the training process. Albeit currently its performance is not optimal compared to other computational graphic engines, we envision to be more powerful due to the the fast growing community of contributors. We were able to find the open-source Tensorflow projects (DCGAN [10], Siamese [11] and STN [12]) on GitHub to use for reference. The drawback of the tool is that the learning curve can be steep for people who are not familiar with dataflow-based programming models. [13]

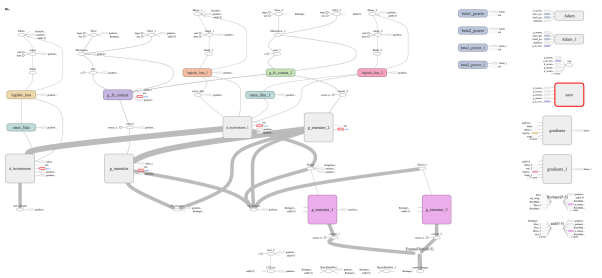

Fig. 16. TensorBoard Graph

VII. Lessons Learned

GAN is a class of models for training generative model inspired by game theory. Fascinated by its novel architecture, we designed a brand new GAN-like network for image composition. There are three parts that are particularly challenging in this project: 1. the lack of guarantee for convergence 2. the lack of human-guided information 3. Complexity of training GAN. Unlike improving upon existing network, there is no previous work on using GAN for image composition. Besides, training GAN is similar to finding a Nash equilibrium, which is known to be a difficult problem. However, part of our purpose for this project is to better understand how to design a network and to see how powerful a GAN can be.

During the process, we learned that the gradient flow crucial to the results of the network. Every layer should be designed with careful thoughts and calculations. For example, for the spatial transformation, simply shifting the images is inefficient for the gradient flow, so we decided to add a learnable STN layer instead. The fact that our network is able to propagate the information from the discriminator to the generator is interesting considering only binary labels are provided. It also indicates that part of the information we intend to extract is successfully propagated with the gradient descent in our design.

Another important part to consider in the design is to feed the appropriate data to the network. For a large number of objects, the entropy in the distribution can be too large for the network to learn. To limit the complexity of the project, we chose car as the only object for generating images. However, at the beginning of training, we fed random real images, with or without cars, to the discriminator. The discriminator loss reached close to 0 after a few epoch and the quality of generated images was poor. We suspected that it is because the discriminator overfitted on the existence of cars. We then adjusted our design to feed the real images with cars to the discriminator instead and were able to generate more realistic images.

One concern of our current network design is network bias. One potential cause of such bias comes from data pre-processing. The way we construct the dataset, for example, by applying synthetic color perturbation using an affine transformation, may lead the network to only learn a structured way of color correction (also an affine transformation). Another cause comes from the way we feed in data and train the network. As we have seen from the experiments above, when we composite images by inserting an extra object into the background image, the network can pick up the cue of the extra count of object in the image, instead of learning the color correction as desired. Ultimately, the network might not be deep enough to capture all the information. Our experiments with STN currently show that the network is unable to learn the right geometric transformation. However, the network size is limited by our GPU memory size now. The Nvidia GeForce GTX 780 GPU we use only has 3GB of memory. Besides, partitioning our network graph between CPU and GPU might lead to slower training speed due to communication overhead.

Furthermore, despite the results from experiments 1 and 2 recovered to a normal color, the overall loss of the discriminator and generator is not monotonically decreasing. This brings up another issue – the set of features the network used for real images identification might not be the same information used by humans. This brings an ambiguous termination condition to the training.

Lastly, controlled experiments proved to be a powerful method to verify the effectiveness of the design. Without the controlled experiments, the results of the current network look very confusing. Since there is a lot of information (lighting, tone, geometry, semantics and etc.) to learn, poorly generated images provide no information on what the network can learn and what the network fails to learn. Therefore, by controlling the variation of the input images, we were able to troubleshoot the problems of the design and adjust the network for learning different features.

In the future, we plan to conduct experiments on spatial transformation and design a deeper network to improve the quality of results.

VIII. Team Contributions

Xuaner (Cecilia) Zhang is responsible for data processing using the Labelme API and network architecture design. She modified and tested modified versions of JCGAN model in Tensorflow, and conducted quantitative evaluations.
Percentage of contribution: 50%

Qijing (Jenny) Huang is responsible for developing the JCGAN model in Tensorflow. She wrote the code for the network using the DCGAN project as reference, added Siamese network and STN at the generator side. After the network is built, she debugged and tuned the network using Tensorboard visualization, performed controlled experiments, and collected results.
Percentage of contribution: 50%

References

[1] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, "Labelme: a database and web-based tool for image annotation," *International journal of computer vision*, vol. 77, no. 1-3, pp. 157–173, 2008.

[2] J.-F. Lalonde and A. A. Efros, "Using color compatibility for assessing image realism," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[3] J.-Y. Zhu, P. Krahenbuhl, E. Shechtman, and A. A. Efros, "Learning a discriminative model for the perception of realism in composite images," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3943–3951.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, 2014, pp. 2672–2680.

[6] L. Theis, A. van den Oord, and M. Bethge, "A note on the evaluation of generative models," *ArXiv e-prints*, Nov. 2015.

[7] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[8] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 2017–2025. [Online]. Available: http://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, vol. abs/1511.06434, 2015. [Online]. Available: http://arxiv.org/abs/1511.06434

[10] T. Kim, "Dcgan in tensorflow," https://github.com/carpedm20/DCGAN-tensorflow.

[11] "Tensorflow-siamese," https://github.com/jeromeyoon/Tensorflow-siamese.

[12] "Spatial transformer network," https://github.com/tensorflow/models/tree/master/transformer.

[13] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative study of caffe, neon, theano, and torch for deep learning," *CoRR*, vol. abs/1511.06435, 2015. [Online]. Available: http://arxiv.org/abs/1511.06435