Unsupervised Learning of Object Landmarks through Conditional Image Generation using Transposed Convolutions

Kartik Ramesh Jain
University of Central Florida
Orlando, USA
kjain@knights.ucf.edu (4810577)

Abstract

This project was undertaken primarily to re-implement "Unsupervised Learning of Object Landmarks through Conditional Image Generation" [6] as part of an assignment for the course CAP 6412: Advanced Computer Vision. As shown later on, this has been successfully accomplished. Incremental improvement is also made to original architecture by changing a network in the architecture. Experiments are performed on the CelebA dataset [7] and 2D renderings of cars, planes, and chairs from the ShapeNet dataset [2]. The reported results include both qualitative visualization and quantitative metrics.

1. Introduction

The paper "Unsupervised Learning of Object Landmarks through Conditional Image Generation" (referenced as "base paper" from here on) learns landmarks without using any keypoint information by only changing the image viewpoint or using two different video frames. The base paper used a self-supervised approach in that it converts the keypoint learning problem to a image reconstruction problem. One of the two perturbed images are passed through a bottleneck which aims to encode an image using only it's geometrical lankmarks (i.e. encode spatial location of the keypoints). Here, as the training progresses, if the network learns to reconstruct clearer images, it means that the bottleneck was able to extract meaningful landmarks which are finally represented as keypoints.

We will get into the implementation details later but one thing to be noted here is the base paper's decoder/renderer structure uses interpolation for image generation/reconstruction. My architecture (referenced as "updated architecture" from here on) instead uses transposed convolutions to render the image. Motivation and details of this implementation are discussed in Section 3 - Method.

2. Related Work

Most of this section follows work up until the base paper: Since we have turned the keypoint extraction problem to an image reconstruction problem using only the geometry of that object, a common idea is to use Generative Adversarial Networks (GANs) based on a given condition (which in this case is the other perturbed image). This conditioning constraint in the latent space of GANs [3], although may be useful in GANs, simplifies the problem of reconstruction so much so that only a simple reconstruction loss is enough to train the network effectively.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 124, 124]	4,736
BatchNorm2d-2	[-1, 32, 124, 124]	64
ReLU-3	[-1, 32, 124, 124]	0
Conv2d-4	[-1, 32, 124, 124]	9,248
BatchNorm2d-5	[-1, 32, 124, 124]	64
ReLU-6	[-1, 32, 124, 124]	0
Conv2d-7	[-1, 64, 62, 62]	18,496
BatchNorm2d-8	[-1, 64, 62, 62]	128
ReLU-9	[-1, 64, 62, 62]	0
Conv2d-10	[-1, 64, 62, 62]	36,928
BatchNorm2d-11	[-1, 64, 62, 62]	128
ReLU-12	[-1, 64, 62, 62]	0
Conv2d-13	[-1, 128, 31, 31]	73,856
BatchNorm2d-14	[-1, 128, 31, 31]	256
ReLU-15	[-1, 128, 31, 31]	0
Conv2d-16	[-1, 128, 31, 31]	147,584
BatchNorm2d-17	[-1, 128, 31, 31]	256
ReLU-18	[-1, 128, 31, 31]	0
Conv2d-19	[-1, 256, 16, 16]	295,168
BatchNorm2d-20	[-1, 256, 16, 16]	512
ReLU-21	[-1, 256, 16, 16]	0
Conv2d-22	[-1, 256, 16, 16]	590,080
BatchNorm2d-23	[-1, 256, 16, 16]	512
ReLU-24	[-1, 256, 16, 16]	0

Total params: 1,178,016

Figure 1. Image Encoder: For a given image of size $128 \times 128 \times 3$ the four block (eight convolutions) reduced it to a $16 \times 16 \times 256$ tensor.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 124, 124]	4,736
BatchNorm2d-2	[-1, 32, 124, 124]	64
ReLU-3	[-1, 32, 124, 124]	0
Conv2d-4	[-1, 32, 124, 124]	9,248
BatchNorm2d-5	[-1, 32, 124, 124]	64
ReLU-6	[-1, 32, 124, 124]	0
Conv2d-7	[-1, 64, 62, 62]	18,496
BatchNorm2d-8	[-1, 64, 62, 62]	128
ReLU-9	[-1, 64, 62, 62]	0
Conv2d-10	[-1, 64, 62, 62]	36,928
BatchNorm2d-11	[-1, 64, 62, 62]	128
ReLU-12	[-1, 64, 62, 62]	0
Conv2d-13	[-1, 128, 31, 31]	73,856
BatchNorm2d-14	[-1, 128, 31, 31]	256
ReLU-15	[-1, 128, 31, 31]	0
Conv2d-16	[-1, 128, 31, 31]	147,584
BatchNorm2d-17	[-1, 128, 31, 31]	256
ReLU-18	[-1, 128, 31, 31]	0
Conv2d-19	[-1, 256, 16, 16]	295,168
BatchNorm2d-20	[-1, 256, 16, 16]	512
ReLU-21	[-1, 256, 16, 16]	0
Conv2d-22	[-1, 256, 16, 16]	590,080
BatchNorm2d-23	[-1, 256, 16, 16]	512
ReLU-24	[-1, 256, 16, 16]	0
Conv2d-25	[-1, 10, 16, 16]	2,570

Total params: 1,180,586

Figure 2. Pose Encoder: For a given image of size $128 \times 128 \times 3$ the four block (eight convolutions) reduced it to a $16 \times 16 \times K$ tensor where $K = 10$ in this case.

While there are several papers using GANs to solve this reconstruction problem, on the other end of this spectrum, we see KeypointNet [8] which aims to learn 3-dimensional keypoints in the latent space for 3D objects rendered as multiple 2D images. The base paper, in contrast, aims to learn 2D keypoints i.e., it does not make any predictions about the depth of a keypoint.

3. Method

We will discuss the base paper's architecture and it's accompanying code [5] in TensorFlow in detail and try to replicate them as closely as possible to the original implementation.

3.1. Pre-processing

CelebA dataset images [7] were resized to 128×128 to serve as x_0. For each sample, two images are then generated x and x' by applying perturbations on the sample. The base paper uses random Thin-Plate-Spline (TPS) [1] as perturbations and so we have used them as well. The TPS code is referenced from [4].

In an addition experiment we have used Random Rotations with a maximum angle of 20.0 degrees. These serve as an effective augmentation technique without deforming the image. Black boundaries which serve as noise in the image reconstruction process are removed as well since the same is done in the base paper by means of Boundary Discounting.

For the ShapeNet dataset, each object (cars/planes/chair in our case) has it's `.tfrecords` file which contains multiple different views of a single object as 2D renderings and has numerous such objects. We have referenced the dataset loading code from KeypointNet's [8] Github repository. This code was modified to work with TensorFlow Version

2.0 and was ultimately made part of a PyTorch dataloader to work with my model written in PyTorch. After these views are obtained, one of them is used as x and another as x'. It is to be noted that in this experiment, perturbations are not carried out as we already have multiple views of the same object.

3.2. Image Encoder

Image encoder network takes in x as input and encodes it to a $16 \times 16 \times C$ tensor by means of 4 sequential blocks shown in detail in figure 1. Each block contains two convolution layers with batch normalization applied before ReLU activation. Each block doubles the previous number of channels starting from 32.

3.3. Pose Encoder

Pose encoder network takes K for the number of landmarks and image x' as input and encodes it to a $16 \times 16 \times K$ tensor as shown in figure 2. The first four sequential blocks are exactly same in structure to Image Encoder's architecture (figure 1). Only a $1 \times 1-$convolution is added at the end to reduce the output to K channels.

Further, for each of the K channels, softmax score is calculated along the rows as well as the column separately to identify the (x, y) coordinate pair for each channel in the $16x16$ space. This single super bright spot is converted into a region representing a Gaussian distribution for each channel. In the code, this is done by convolving with a Gaussian filter of fixed standard deviation as shown in this PyTorch thread.

3.4. Decoder/Renderer

The decoder takes in a tensor of concatenation from the Image Encoder and the Pose Encoder's output as input: $16 \times 16 \times (C + K)$ as shown in detail in figure 3. This decoder uses only convolution layers for learning while the actual up-sampling is done using Bi-Linear interpolation with a scale factor of two. The channels are halved after every upsample and kept at a minimum of 8 (in the code [5]) until the final resolution of 128×128 is achieved. After that, a final convolution layer reduces the number of channels to 3.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 256, 16, 16]	613,120
BatchNorm2d-2	[-1, 256, 16, 16]	512
ReLU-3	[-1, 256, 16, 16]	0
Conv2d-4	[-1, 256, 16, 16]	590,080
BatchNorm2d-5	[-1, 256, 16, 16]	512
ReLU-6	[-1, 256, 16, 16]	0
Upsample-7	[-1, 256, 32, 32]	0
Upsample-8	[-1, 256, 32, 32]	0
Conv2d-9	[-1, 128, 32, 32]	295,040
BatchNorm2d-10	[-1, 128, 32, 32]	256
ReLU-11	[-1, 128, 32, 32]	0
Conv2d-12	[-1, 128, 32, 32]	147,584
BatchNorm2d-13	[-1, 128, 32, 32]	256
ReLU-14	[-1, 128, 32, 32]	0
Upsample-15	[-1, 128, 64, 64]	0
Upsample-16	[-1, 128, 64, 64]	0
Conv2d-17	[-1, 64, 64, 64]	73,792
BatchNorm2d-18	[-1, 64, 64, 64]	128
ReLU-19	[-1, 64, 64, 64]	0
Conv2d-20	[-1, 64, 64, 64]	36,928
BatchNorm2d-21	[-1, 64, 64, 64]	128
ReLU-22	[-1, 64, 64, 64]	0
Upsample-23	[-1, 64, 128, 128]	0
Upsample-24	[-1, 64, 128, 128]	0
Conv2d-25	[-1, 32, 128, 128]	18,464
BatchNorm2d-26	[-1, 32, 128, 128]	64
ReLU-27	[-1, 32, 128, 128]	0
Conv2d-28	[-1, 3, 128, 128]	867

Total params: 1,777,731

Figure 3. Decoder/Renderer: Bilinearly upsamples the image until the final resolution is reached.

3.5. Modification: Decoder - Transposed Conv

This is the network that is updated to hopefully improve the re-construction process. The motivation: since upsampling has no learnable parameters involved, it is limited by the information it receives as input. Therefore, the burden of actually 'learning' to reconstruct images fall on the other convolution layers.

It should be noted that there are several ways of increasing height and width of a given tensor using learnable layers. One of them is adding high padding to output increased height and width after a convolution layer. While this

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 256, 16, 16]	613,120
BatchNorm2d-2	[-1, 256, 16, 16]	512
ReLU-3	[-1, 256, 16, 16]	0
Conv2d-4	[-1, 256, 16, 16]	590,080
BatchNorm2d-5	[-1, 256, 16, 16]	512
ReLU-6	[-1, 256, 16, 16]	0
ConvTranspose2d-7	[-1, 256, 32, 32]	1,048,832
BatchNorm2d-8	[-1, 256, 32, 32]	512
ReLU-9	[-1, 256, 32, 32]	0
Conv2d-10	[-1, 128, 32, 32]	295,040
BatchNorm2d-11	[-1, 128, 32, 32]	256
ReLU-12	[-1, 128, 32, 32]	0
Conv2d-13	[-1, 128, 32, 32]	147,584
BatchNorm2d-14	[-1, 128, 32, 32]	256
ReLU-15	[-1, 128, 32, 32]	0
ConvTranspose2d-16	[-1, 128, 64, 64]	262,272
BatchNorm2d-17	[-1, 128, 64, 64]	256
ReLU-18	[-1, 128, 64, 64]	0
Conv2d-19	[-1, 64, 64, 64]	73,792
BatchNorm2d-20	[-1, 64, 64, 64]	128
ReLU-21	[-1, 64, 64, 64]	0
Conv2d-22	[-1, 64, 64, 64]	36,928
BatchNorm2d-23	[-1, 64, 64, 64]	128
ReLU-24	[-1, 64, 64, 64]	0
ConvTranspose2d-25	[-1, 64, 128, 128]	65,600
BatchNorm2d-26	[-1, 64, 128, 128]	128
ReLU-27	[-1, 64, 128, 128]	0
Conv2d-28	[-1, 32, 128, 128]	18,464
BatchNorm2d-29	[-1, 32, 128, 128]	64
ReLU-30	[-1, 32, 128, 128]	0
Conv2d-31	[-1, 3, 128, 128]	867

Total params: 3,155,331

Figure 4. Modified Decoder: Uses blocks of convolutions and transposed convolution to upsample tensors to a final $128 \times 128 \times 3$ image.

method achieves upsampling to some extent, it adds a lot of useless information in the form of padding which makes the training converge slower. Instead, we use transposed convolutions which are naturally built as a reverse operation of convolution.

Although it is a known fact that Transposed Convolutions are responsible for checkerboard artifacts (e.g. see this popular Distill Pub article), it is to be noted that my decoder is not entirely comprised of ConvTranspose but rather spaced between Conv layers to only 'learn' correct upsampling through training (see figure 4).

3.6. Perceptual loss

Referred from base paper's code [5], VGG16 architecture is used to calculate the perceptual loss. The input to this network is x' and $\hat{x}'$. Their code uses linear combination of outputs from the following layers in VGG: `[input, conv1_2, conv2_2, conv3_2, conv4_2, conv5_2]`.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 128, 128]	1,792
ReLU-2	[-1, 64, 128, 128]	0
Conv2d-3	[-1, 64, 128, 128]	36,928
ReLU-4	[-1, 64, 128, 128]	0
MaxPool2d-5	[-1, 64, 64, 64]	0
Conv2d-6	[-1, 128, 64, 64]	73,856
ReLU-7	[-1, 128, 64, 64]	0
Conv2d-8	[-1, 128, 64, 64]	147,584
ReLU-9	[-1, 128, 64, 64]	0
MaxPool2d-10	[-1, 128, 32, 32]	0
Conv2d-11	[-1, 256, 32, 32]	295,168
ReLU-12	[-1, 256, 32, 32]	0
Conv2d-13	[-1, 256, 32, 32]	590,080
ReLU-14	[-1, 256, 32, 32]	0
Conv2d-15	[-1, 256, 32, 32]	590,080
ReLU-16	[-1, 256, 32, 32]	0
MaxPool2d-17	[-1, 256, 16, 16]	0
Conv2d-18	[-1, 512, 16, 16]	1,180,160
ReLU-19	[-1, 512, 16, 16]	0
Conv2d-20	[-1, 512, 16, 16]	2,359,808
ReLU-21	[-1, 512, 16, 16]	0
Conv2d-22	[-1, 512, 16, 16]	2,359,808
ReLU-23	[-1, 512, 16, 16]	0
MaxPool2d-24	[-1, 512, 8, 8]	0
Conv2d-25	[-1, 512, 8, 8]	2,359,808
ReLU-26	[-1, 512, 8, 8]	0
Conv2d-27	[-1, 512, 8, 8]	2,359,808
ReLU-28	[-1, 512, 8, 8]	0

Total params: 12,354,880

Figure 5. Relevant part of VGG: Apart from input, the volume used for norm are at layer numbers `[4, 9, 14, 21, 28]` in the figure.

The norm we have implemented is L2 since that the norm written in paper and used by the code. Here, ignoring the input, the conv layers can be seen in figure 5. As shown in experiments later, we try different linear combinations (by using the weighted sum scores used by the code) of these norms to observe the change.

3.7. Training

Following all the training tricks used in the code, we have similarly used the following:

Adam optimizer with learning rate starting from `1e-3` and reducing by a factor of 2 every

time the training loss stops decreasing (for a history of 100 iterations). Learning rate is clipped to a minimum value to avoid vanishing gradients problem. Additionally, since weight decay of `5x1e-4` and gradient clipping to `1.0` is also used by the code, we use those as well.

We train the model on a subset of CelebA dataset and for 100 epochs and 60 epochs in the two experiments explained below and use the other for testing. For ShapeNet, we used two views per object and trained the model for 100 epochs for each of the three objects.

4. Experiments

A series of experiments conducted on the CelebA and the ShapeNet datasets involve training and testing on two models: Upsample and ConvTranspose. As the name suggests, Upsample is the base paper's implementation of the decoder and the ConvTranspose is my modification to the decoder.

4.1. Quantitative evaluation

As seen from figures 6 the reconstruction loss is shown to reduce further for the ConvTranspose model compared to the original Upsample model. This trend continues when using the code [5] weights for the linear combination of VGG loss as shown in figure 7. This was expected as the upsampling layer now has weights that can be learned to improve the image quality over training iterations.

4.2. Qualitative evaluation

Shown in figure 8 are visualized keypoints (K = 10) for both the models with four random samples and their corresponding reconstructions. The keypoints have been plotted on the image x' by scaling the co-ordinate values in the K 16×16 landmark channels. Since we see that the updated model is performing as good, if not better, as the base model, the further visualizations are shown only from the updated model's results.

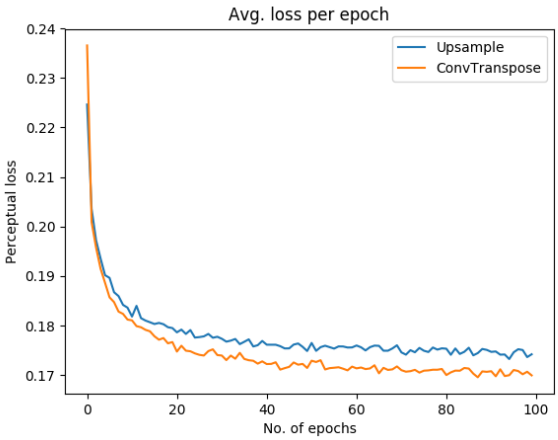

Figure 6. Trained for 100 epochs using perceptual loss for base as well as modified network.

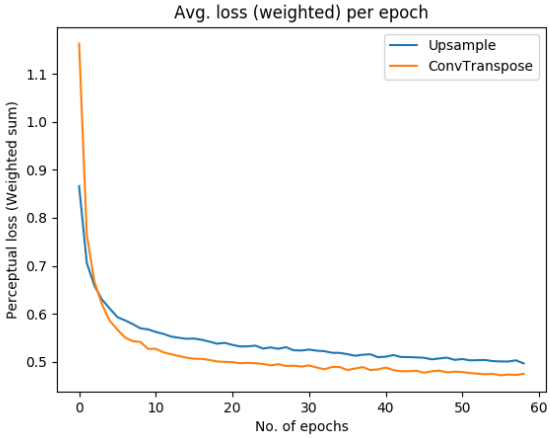

Figure 7. Trained with norm weights = `[100.0, 1.6, 2.3, 1.8, 2.8, 100.0]` for `[input, conv1_2, conv2_2, conv3_2, conv4_2, conv5_2]` respectively as referred from [5].

Shown in figures 10, 11, and 12 are three samples each of chairs, cars, and planes respectively. As seen from the figures, the reconstruction is relatively poor and, as a result, the identified landmarks are not very accurate. We further expand on this in the next section.

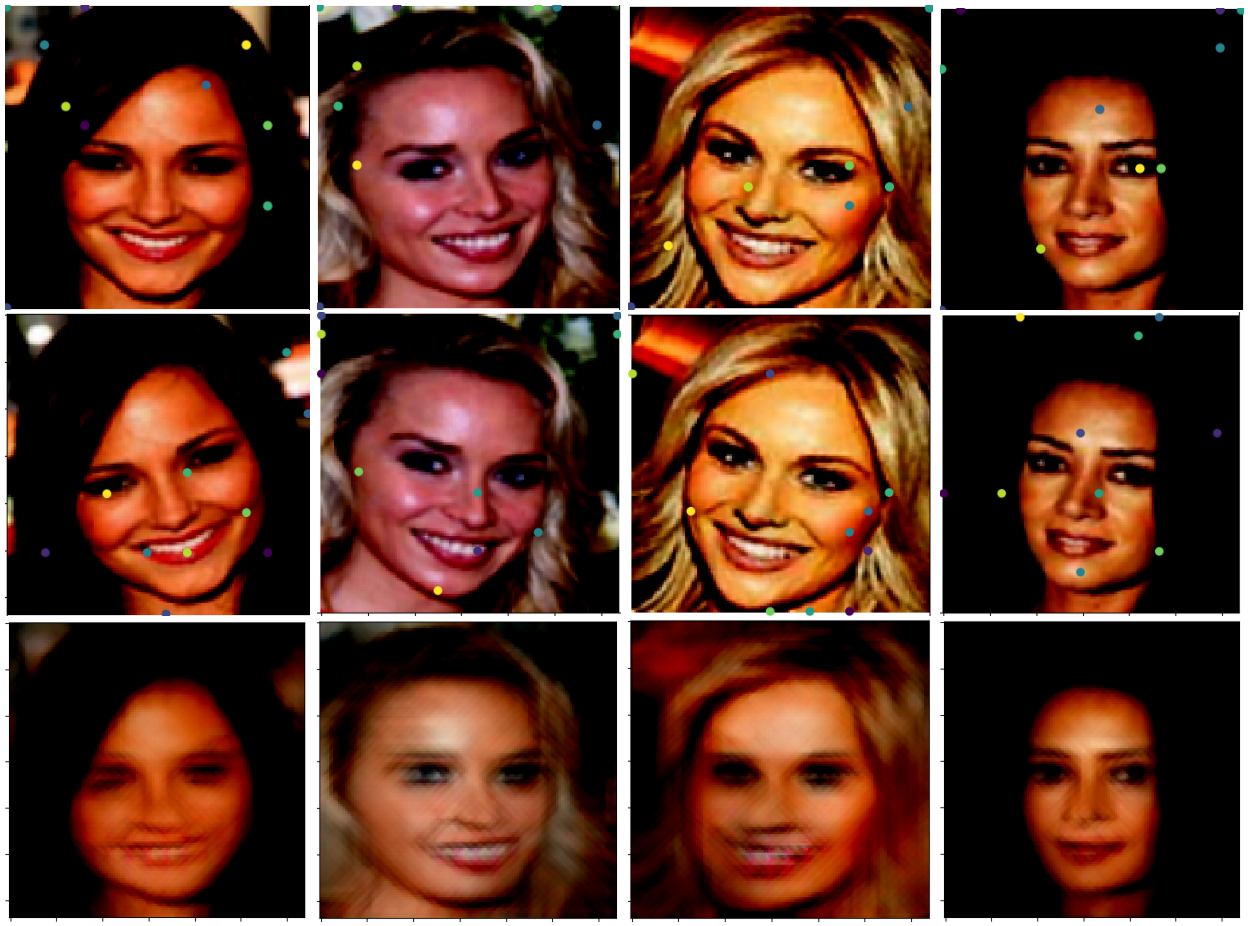

Figure 8. Visualized keypoints (K = 10) on four samples from the base model (top row) and the updated model (middle row) - best viewed in color. Bottom row shows updated model's corresponding reconstructions ($\hat{x}'$).

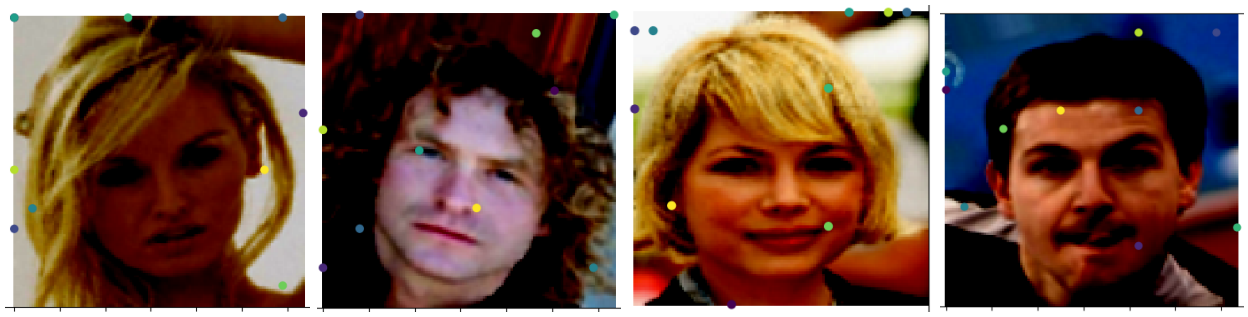

Figure 9. From left to right: Test samples after 10, 1000, 5000, and 10000 steps (iterations).

5. Discussion

From figure 8, we see, some of the landmarks have learnt to point to a part of the face while the others have not i.e. they have started settling on irrelevant regions like corners. One possible reason might be that, as seen from the visualizations, only five points were enough at this stage to reconstruct the face. If the training were to progress even further, the reconstruction

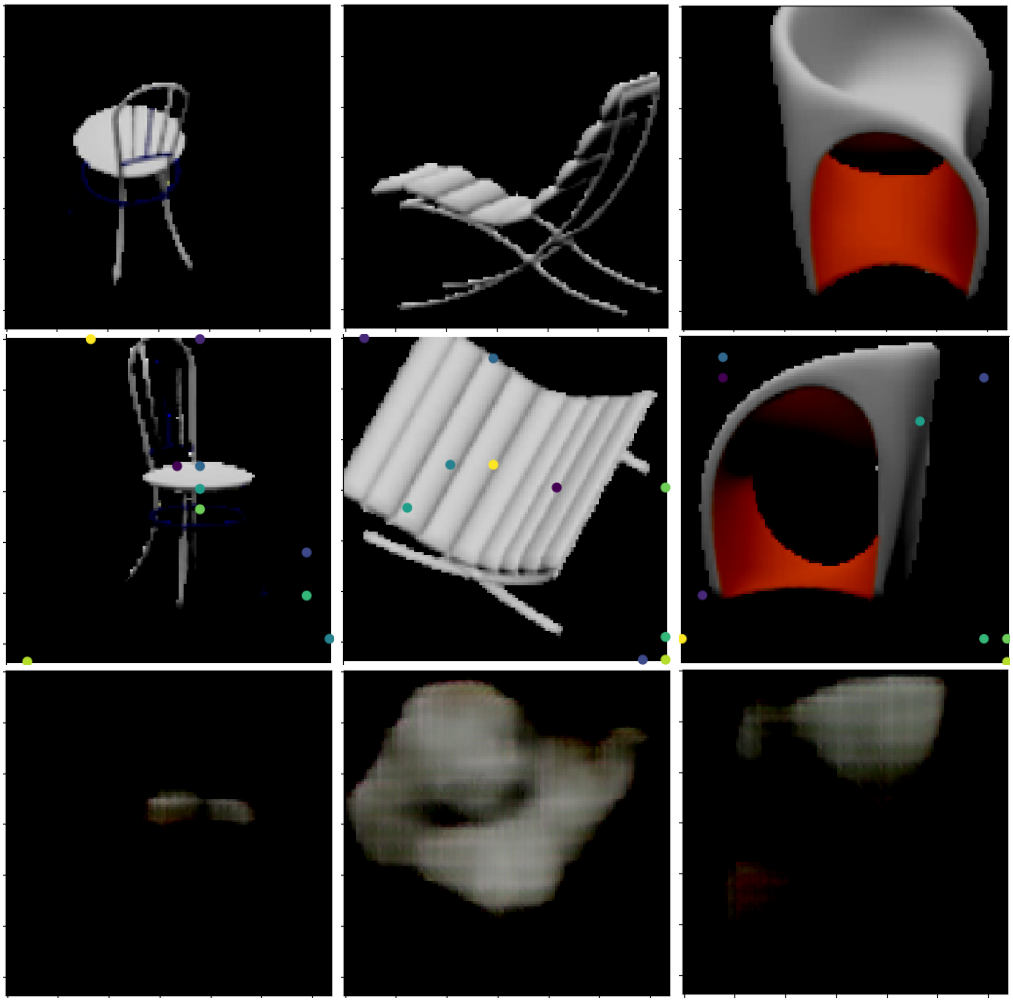

Figure 10. Top row: (x). Middle row: (x') with keypoints visualized (K = 10) on three samples for chairs from ShapeNet. Bottom row: corresponding reconstructions $(\hat{x}')$.

loss (which will be reduced even further) will make those other landmarks fall on relevant face regions as well. This can be corroborated by the visualizations seen in figure 9 where we see more and more landmarks falling on test faces as the training progresses.

For the ShapeNet results, from these individual experiments, we see that object changes significantly between views which makes this a much more difficult problem than the previous case (random perturbations). Third column from chairs (figure 10) shows one of the worst outputs from the model. Further tests (figure 11 and 12) were performed only for five landmarks (K = 5). As we can see from the images, the background color (black) is mostly dominant is all of the images which, as we interpret it, makes the reconstruction more difficult i.e., reconstruction looks smudgy and doesn't have sharp features. Even after training with 100 epochs, the results did not improve after 60 epochs suggesting that either the model needs to be further improved and/or the data needs to be better pre-processed to predict accurate landmarks.

It was observed in all my experiments that, regardless of K, the reconstructed image was heav-

Figure 11. Top row: (x). Middle row: (x') with keypoints visualized (K = 5) on three samples for cars from ShapeNet. Bottom row: corresponding reconstructions $(\hat{x}')$. Sample from the first column has similar views while the other two samples show huge view difference.

ily conditioned on x rather than x' which means the keypoints were not as considered as important in reconstruction as they should have been. A possible solution to this could be to add regularization like dropout on the image encoder part to make the conditioning less strong with the trade-off being sharpness of the reconstructed image.

6. Conclusion

The paper [6] was successfully implemented and modifications were made that added minor improvements to the results. Transposed convolutions, when used in conjunction with other layers, can construct higher quality images compared to interpolation.

References

[1] F. L. Bookstein. Principal warps: Thin-Plate splines and the decomposition of deformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11:567585, 1989.

[2] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015.

[3] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. pages 2172–2180, 2016.

[4] C. Heindl. https://github.com/cheind/py-thin-plate-spline.git.

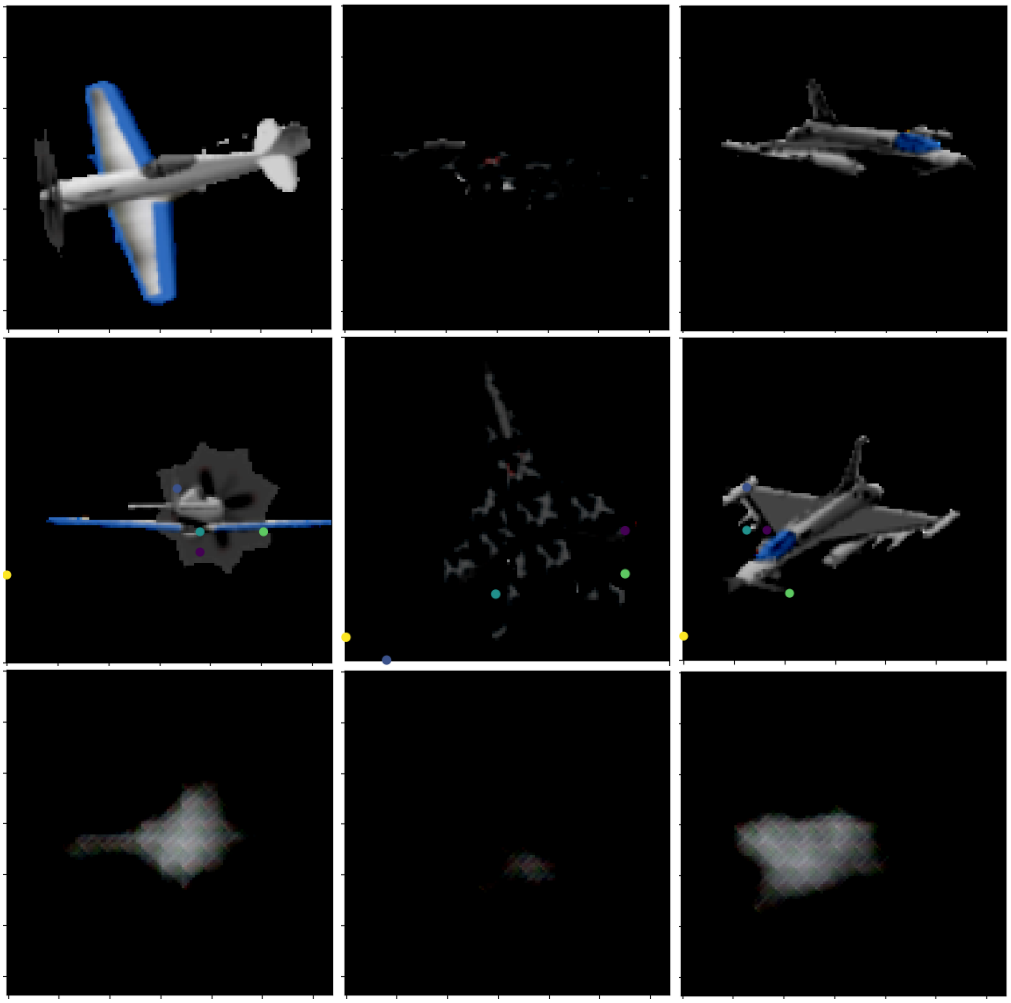

Figure 12. Top row: (x). Middle row: (x') with keypoints visualized (K = 5) on three samples for planes from ShapeNet. Bottom row: corresponding reconstructions $(\hat{x}')$. Sample from the first column has a drastic view change while the second column has object which is difficult to perceive.

[5] T. Jakab, A. Gupta, H. Bilen, and A. Vedaldi. https://github.com/tomasjakab/imm.

[6] T. Jakab, A. Gupta, H. Bilen, and A. Vedaldi. Unsupervised learning of object landmarks through conditional image generation. pages 4016–4027, 2018.

[7] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. 2015.

[8] S. Suwajanakorn, N. Snavely, J. J. Tompson, and M. Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. pages 2059–2070, 2018.