Enhanced View-Synthesis by Appearance Flow

Shradha Agrawal
A53105993
sha015@eng.ucsd.edu

Sudhanshu Bahety
A53209213
sbahety@eng.ucsd.edu

Jean Choi
A53214201
jsc078@eng.ucsd.edu

Ajitesh Gupta
A53220177
ajgupta@eng.ucsd.edu

Nimish Srivastava
A53222968
n2srivas@eng.ucsd.edu

Abstract

In this work we explore an approach to synthesize new images of same object or scene from unseen viewpoints. This can be helpful for 3D reconstruction of an object when we do not have images of that object from various viewpoints. This could be applied to photo editing programs like Adobe Photoshop to manipulate objects in 3D instead of 2D and help create full virtual reality environments based on historic images or video footage.

Catering to this task, we extend the recent work of Zhou et al[1] on View-Synthesis from Appearance flow, drawing inspiration from Single-view to multi-view paper by Tatarchenko et al[2]. We believe that the current appearance flow system[1] lacks the capability to hallucinate pixels which are not present in the input view, as pointed out by the authors themselves.

In this work we display the viewpoint transformed synthesized images and empirical results for the $L2$ reconstruction loss on the viewpoint transformed images and the error statistic measuring the role of correlated information between different viewpoints. We have also described our network architecture in this report and plan to make our implementation in Keras[7] available for public use along with an instance of the dataset we have used.

1 Introduction

Human perception is not bound by the limited viewpoints to fully imagine a known object. When we see an object from a particular viewpoint, we can hallucinate the object from different viewpoints because we had previously seen different viewpoints of objects similar to this. Experiments in psychophysics show that humans perceive an object 3-dimensionally even when they look at a 2D image of it, and excel at the task of mentally rotating it to a new viewpoint[5].

On the other hand, neural networks have not yet reached good standards of accomplishing this task of generating novel viewpoints of an object based on semantic information of similar objects and transformations. They are good at learning features in the service of a task, but their generalization ability is hugely limited by the training data and procedure used. We refer and extend the works of Zhou et al[1] and Tatarchenko et al[2] and create a neural network trained to cater to the task of novel view point generator which gives a sharp output and extrapolates on the information of occluded pixels in the input view, building concurrently on the strengths of both [1] and [2].

To this end we propose a network architecture similar to the one used in [1], but with a modification to the final layer to better hallucinate. The final layer is appended with 3 more RGB channels which are trained as an autoencoder in a pretraining stage. The network is then trained as a whole, i.e. 5 channels in the last layer, 3 RGB autoencoder and 2 appearance flow channels which are used to

create the final output after bilinear sampling from the input image and using the 3 RGB channel output for occluded pixels. The warped input image ensures sharpness of reconstruction as pointed out in [1] and the use of autoencoder information builds upon the lost information from occluded pixels.

The authors of [1] got inspiration for the Bilinear layer from the work done in [7], where the authors propose a new Spatial Transformer Layer. This Spatial Transformer Layer is a differentiable layer which can thus be added to any network, and is used to warp input features into a desired output feature map. The authors in [7] use it rectify their images, in order to make their networks spatially invariant to the images orientation in terms of rotation, translation and scale.

For experimental purposes, we only work with a small part of the ShapeNet[3] dataset, as procured from the authors of [1]. This segment contains 700 chair classes, each rendered with 504 different viewpoints and serves the purpose of this task well. We plan to extend our observations on a more general set of images.

2 Background

The major source of inspiration behind this work is the View Synthesis by Appearance Flow work done by Zhou et al[1]. In this work they use a convolutional neural network to learn the representations of an image in a low-dimensional convoluted space. Then combined with the required viewpoint transformations, they deconvolve their representation to generate an appearance flow field. [1] Using this appearance flow, which tells where to steal a given pixel in the output from the input, they warp the input image to generate a view transformed synthesis. This output is shown to be sharp in [1], but it fails in generating information about the occluded pixels.

The work of Oh et al[6] shows the early attempts at generating different views from a single image. They posed it as an optimization and computer vision problem rather than a learning problem. Since then we have had various advances in learning and deep neural networks. The work of Tatarchenko et al[2] poses this as learning problem in form of training an encoder-decoder network to learn to predict transformed images of a view given RGBD data. They were able to tackle the problem faced in [1], as they generate view transformed images with information about occluded pixels. The drawback of this work was that their reconstructions were blurry and lacked details of the object. Another limitation to application of this work[2] is lack of availability of ground truth depth maps for the image datasets which are required for their network.

Our approach is targets building upon the strengths of [1] for generating a sharp view transformation and to fill up the occluded pixel information, seeking inspiration from [2]. To simplify things a little further, we plan to use the same architecture as in [1] to train an encoder decoder network by appending 3 RGB channels in the final layer. This not only simplifies the problem, but with the use of a stage-wise training regime we can ensure that the appearance flow network gets a better set of starting weights and that while it is being trained, the autoencoder is also fine-tuned simultaneously. The simultaneous fine-tuning of autoencoder adds towards maintaining fidelity of the network towards the task, i.e. unlike the recent works in neural networks, where the network is not exactly aware of the task, our autoencoder backpropagates the information pertinent to the task of viewpoint transformation simultaneously, while the network learns a slightly more obscure representation, the appearance flow, to serve the task better.

2.1 Autoencoder

An autoencoder was originally deployed to reduce the dimensionality of networks in the works of Hinton and Salakhutdinov[4]. For data with high dimensions, the network can get very complex. Thus to represent the same data with less dimensions, a network with a central layer that reconstructs an original representation with lesser dimensions was proposed.

[1] Appearance flow field is not similar to optical flow. Although the appearance flow field tells us a pixel correspondence like the optical flow, but this pixel correspondence has nothing to do with how the object has moved. The appearance flow gives information about how an output pixel information can be stolen from a similar looking input pixel and is not bound to establish any object correspondence as in optical flow.

3 Model

3.1 Dataset

In original work[1], the authors use 7,499 cars and 700 chairs from ShapeNet[3] dataset, where each image class has 504 different viewpoints. The cars dataset was more than a 135GB in size and as such was beyond the scope of this project. Thus we decided to go ahead with just the chairs dataset which was around 8GB in size with 350,000 chair images.

These images were sampled from 700 different chair designs each having 504 viewpoint variations in terms of azimuth and elevation angles. The azimuth difference varies from 0° to 355° and is quantified into 19 bins of 20° separation between each successive bin. As in [1], we also conduct experiments just with transformations on the azimuth, keeping elevation between input and view transformed output same.

We collected the above mentioned dataset from the authors of [1]. They had rendered this data from the original ShapeNet[3] dataset.

3.2 Resources

Parts of the code are made public by the authors at `https://github.com/tinghuiz/appearance-flow`. Their code for running experiments is still incomplete. Due to ease of extension and faster convergence we have re-implemented the architecture in Keras[7] with a tensorflow backend.

To train the models we have used AWS resources provided for this project.

We would like to take this opportunity to thank the authors of [1] for sharing their rendered dataset. This not only saved us time on rendering the data from ShapeNet, but also AWS resources which would have been exhausted to an extent for this task.

3.3 Architecture

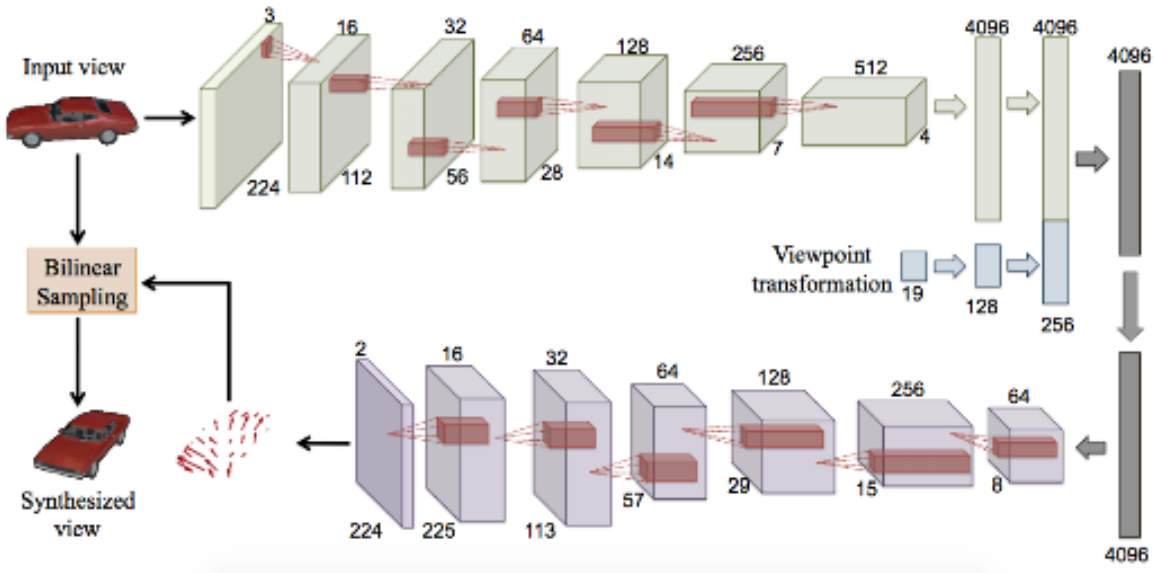


Figure 1: This figure illustrates the original architecture of the network[1].

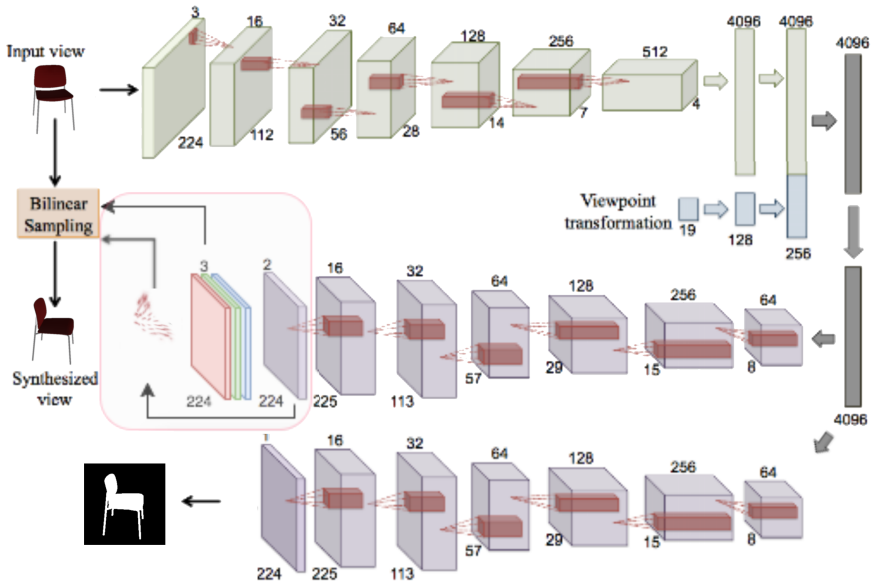


Figure 2: This figure illustrates the proposed architecture of the network[2].

The original network architecture as shown in Figure 1 has three major components:

1. Input view encoder – extracts relevant features (e.g. color, pose, texture, shape, etc.) of the query instance (6 conv + 2 fc layers).
2. Viewpoint transformation encoder – maps the specified relative viewpoint to a higher-dimensional hidden representation (2 fc layers).
3. Synthesis decoder – assembles features from the two feature encoders, and outputs the appearance flow field that reconstructs the target view with pixels from the input view (2 fc + 6 deconv layers).

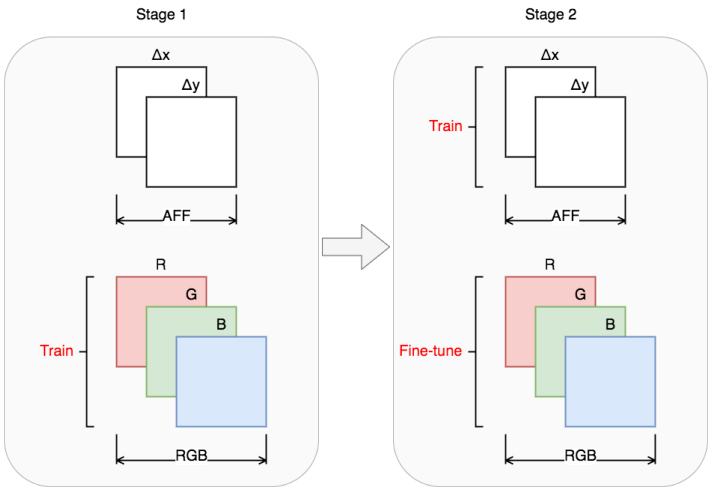


Figure 3: This figure illustrates a magnified portion of last layer in the proposed architecture. The top 2 channels represent displacements in x and y respectively. Together they make up appearance flow field i.e. AFF. The bottom 3 channels represent R, G, and B respectively.

All the convolution, fully-connected and fractionally-strided/up-sampling convolution (uconv) layers are followed by rectified linear units except for the last flow decoder layer.

Layer Name	Stride	Kernel Size	Activation	Output Size	#Filters	Previous Layer
Convolution2D 1	2	3x3	ReLU	112x112	16	Input Image(224x224x3)
Convolution2D 2	2	3x3	ReLU	56x56	32	Convolution2D 1
Convolution2D 3	2	3x3	ReLU	28x28	64	Convolution2D 2
Convolution2D 4	2	3x3	ReLU	14x14	128	Convolution2D 3
Convolution2D 5	2	3x3	ReLU	7x7	256	Convolution2D 4
Convolution2D 6	2	3x3	ReLU	4x4	512	Convolution2D 5
Flatten 1	-	-	-	8192	-	Convolution2D 6
Conv_Dense	-	-	-	4096	-	Flatten 1
Dropout 1	-	-	-	4096	-	Conv_Dense
View_Dense 1	-	-	-	128	-	View Input(19x1)
Dropout 2	-	-	-	128	-	View_Dense 1
View_Dense	-	-	-	256	-	Dropout 2
Merge	-	-	-	4352	-	Dropout 2 & Dropout 1
Deconv_Dense	-	-	-	4096	-	Merge
Reshape 1	-	-	-	8x8x64	-	Deconv_Dense
Deconvolution2D 1	2	3x3	ReLU	15x15	256	Reshape 1
Deconvolution2D 2	2	3x3	ReLU	29x29	128	Deconvolution2D 1
Deconvolution2D 3	2	3x3	ReLU	57x57	64	Deconvolution2D 2
Deconvolution2D 4	2	3x3	ReLU	113x113	32	Deconvolution2D 3
Deconvolution2D 5	2	3x3	ReLU	225x225	16	Deconvolution2D 4
Deconvolution2D 6	1	3x3	ReLU	225x225	5	Deconvolution2D 5
Resize Layer	-	-	-	224x224	5	Deconvolution2D 6

Table 1: The current architecture being used by us. For masking layer, we have used the output of original set of convolution layer and create a separate series of deconvolution layer for mask prediction with the only difference in output layer where it has only 1 channel instead of 5 channels.

In the original network, the input are an image and the desired viewpoint vector. The output is the appearance flow field which consists of 2 channels, each channel representing the displacement in either x axis or y axis. The network we are proposing extends this network to include RGB channels, to better hallucinate the pixels that are not available in the input image.

The major modification is illustrated in the Figure 2. In the first stage of model training, only the RGB channels are trained using the autoencoder for reconstruction of input images. In other words, the 3 channels are trained with input images as both input and target images. This stage of pretraining is needed to initialize the weights with reasonable values. The pretraining is for obtaining good initial weights for the task of better hallucinating the pixels of the occluded portions of an object. Then in the next stage, we train the entire network with previously trained RGB channels.

During the first stage, the appearance flow field is not trained but only the RGB channels. In the second stage, the appearance flow field is trained with the RGB channels, fine-tuning the previously trained weights.

3.4 Input-Output Representation

The input to the network is a RGB image and a one-hot encoded 19 class vector for the azimuth transformation between input and output images. The RGB image is passed through 6 convolutions and then the one hot vector is appended to it and passed to the fully connected layers in the network.

The outputs of our network are a 5 channel concatenated output of the appearance flow field and the 3 channel autoencoder output, and a binary output mask of 1 channel. The 5 channel output is further passed through a bilinear sampling layer to generate the desired viewpoint transformed image.

3.5 Objective Function

The network has two training stages, thus we propose a consistent training objective across all training stages. It can be expressed as a linear combination of RGB reconstruction loss from the 3

new channels and the original loss terms.

The original loss is the L1 pixel error as described in [1]. We denote it as $\mathcal{L}_{\mathcal{PE}}$

$$\mathcal{L}_{\mathcal{PE}} = \sum_{\{I_s, I_t, T\} \in D} ||I_t - g(I_s, T)||_p$$

where D is the set of training tuples, g($\cdot$) references the encoder-decoder CNN, $|| \cdot ||_p$ is the L_p norm, I_t is the output viewpoint transformed image, I_s is the input image and T is the transformation.

For the flow-field predicting encoder-decoder CNN, g($\cdot$) can be more formally defined as:

$$g^{(i)}(I_s, T) = \sum_{q \in \{neighborhood\, of\, (x^{(i)}, y^{(i)})\}} I_s^q (1 - |x^{(i)} - x^{(q)}|)(1 - |y^{(i)} - y^{(q)}|)$$

where q is the 4-neighborhood around $(x^{(i)}, y^{(i)})$ where $(x^{(i)}, y^{(i)})$ are the appearance flow field components. This is also known as differentiable image sampling with a bilinear kernel, and its (sub)-gradient with respect to the CNN parameters could be efficiently computed [1].

For training mask stream, we use binary cross entropy loss denoted by $\mathcal{L}_{\mathcal{M}}$. We use a linear combination of both $\mathcal{L}_{\mathcal{PE}}$ and $\mathcal{L}_{\mathcal{M}}$ as our overall training objective, defined as:

$$\mathcal{L} = \mathcal{L}_{\mathcal{PE}} + 0.1 * \mathcal{L}_{\mathcal{M}}$$

Later in our experiments, we figured out that network was learning background as well which is not required and thus we proposed to use masked L1 loss instead of L1 for $\mathcal{L}_{\mathcal{PE}}$.

4 Experiments and Results

4.1 Network Training Details

We used ADAM solver with $\beta_1 = 0.9$ and $\beta_2 = 0.99$, initial learning rate of 0.0001, batch size of 128 and samples per epoch 281230. Also since we had approximately 350,000 images, it was impossible for us to load all images at once in memory due to limited RAM of 61 GB on AWS p2.xlarge instance. Thus we implemented a data generator in Keras which load input for one batch in memory at one time. This data generator randomly samples tuple $< I_s, I_t, M_t, T >$ from the training data where I_s is input image, I_t is target image (I_t is chosen such as it is of same model and same elevation as of I_s), T is one hot encoded 19D vector representing azimuth transformation between I_s and I_t and M_t is mask output of the target image.

4.2 Pretraining the Autoencoder

In order to test the hallucination and generalization capabilities of autoencoders we conducted 2 training experiments using different splits on the dataset.

4.2.1 Our Train and Test Split

For this part of experiments, we train our 3 channel autoencoder with a train and test split over viewpoints of the chair classes. That is, we don't use certain elevations as input across all our model when training and then generate those elevations at the test time.

The autoencoder performs almost as well as on the split suggested by the authors discussed in the next subsection. This experiment affirms our expectations of the autoencoder being able to learn viewpoint variations as well.

In Figure 4 we illustrate that the autoencoder trained on a different set of views does well in generating a representation of unseen viewpoints of the same model of chairs for some examples. The first column shows an input image from the training set, the second column shows the autoencoder output for this image, the third shows an input image from the test set and the fourth shows the autoencoder output for this test set image.

These images were generated after training the model for 42 epochs. The loss and accuracy functions are shown in Figure 5. We see that the graphs smooth out and the accuracy reaches 95.35% .

Figure 4: Illustration of (a) an instance from the training set and (b) it's RGB channel auto-encoder output after 30 epochs (c) an instance from the test set (unseen viewpoint) and (d) it's RGB channel auto-encoder output after 30 epochs

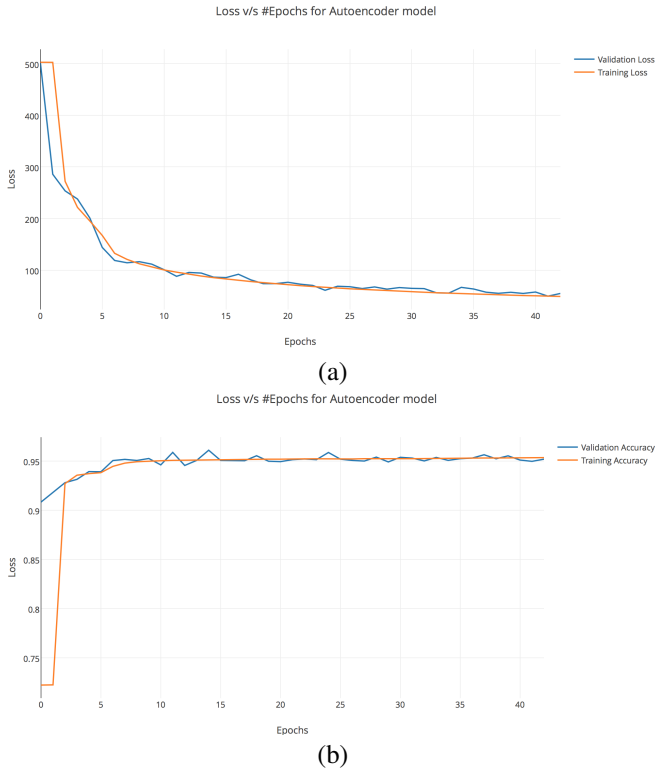
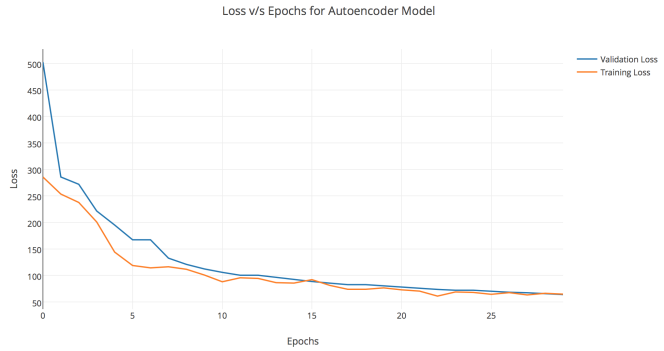


Figure 5: Illustration of (a) convergence of train(orange) and validation(blue) loss (b) convergence of train(red) and validation(blue) accuracy

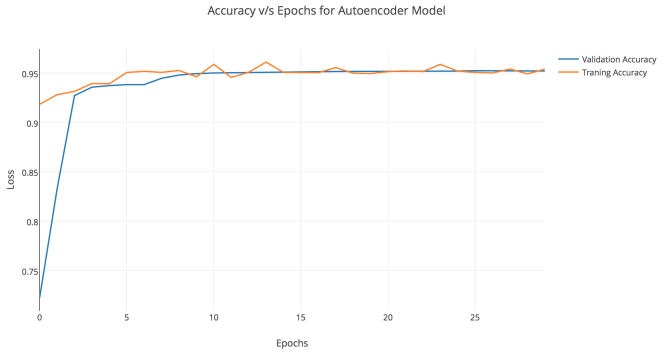
4.2.2 Keeping the Original Train Test Split as in [1]

In this section we discuss our results from the pretraining the network, in which only the 3 RGB channels were trained. For this we use a similar split as in [1], where we keep 80% of the different chair models in the training set and 20% in the test set. We conducted this experiment not only because this was recommended in [1] but also to see that our network generalizes over unseen chairs.

For this experiment, we train the 3 channel autoencoder upto 30 epochs with a train-test split of 80-20. With this we were able to achieve a validation accuracy of upto 95.19 as in Figure6%.



(a)



(b)

Figure 6: Illustration of (a) convergence of train(orange) and validation(blue) loss (b) convergence of train(red) and validation(blue) accuracy

In Figure 7 we display our RGB channel autoencoder output for different chair types. We probe the model after every 5th epoch by using Keras[7] checkpoints. If the model has improved in terms of validation loss, we save the weights. We use these saved weights to generate output from the test set we have. Results of convergence of the 3 channel autoencoder along with a few outputs are shown in Figure7 and our observations are stated below.

- The first chair is a sample from the training set. It is notable to observe here that the network's representation does not change much for this chair with epochs. This shows that the network is learning good representations for what it has seen in a few epochs.
- The second chair has mostly whites and grays. This was not a part of our training set, but the network was able to learn a consistent representation in very few epochs and further improves on finer details like the exact shade of gray on the armrest.
- The third chair is a challenging case for the chair class and was not present in the training split. Again it is notable how well the network learns the representation of such challenging cases.
- The fourth, fifth and the sixth images show that the color channels are still converging. The convergence is observed more strongly on the green and blue channels, with the blue channel converging the fastest and the red one still far from convergence even after 30th epoch.

- The last three images show that the the network is also trying to learn the fine details like texture in the third and second last images, and finer elements of design in the armrest like in the last image while the red color channel is also converging.

Figure 7: Illustration of (a) original input and 3 RGB channel auto-encoder output for some chair types (b) 20 epochs, (c) 25 epochs and (d) 30 epochs

4.3 Encoder-Decoder for View Transformation

For this experiment we trained an Encoder-decoder network for producing a view transformed images from a given input image. For this we gave an input consisting of an image along with a 19 length view transformation vector, encoding the view transformation in terms of the azimuth difference. We conducted this experiment in order to test the validity of our proposed idea, as to whether an encoder-decoder network can learn to hallucinate occluded parts of an image.

A similar effort was done in [2] where the authors used the original images along with the depth maps to predicted view transformed images given transformation parameters. We differ from them in the fact that we just use the RGB values of the chairs and do not provide the depth maps.

We trained this network for L2 as well as L1 losses. For L2 losses we ran upto 40 epochs over the whole chair dataset and for L1 we ran for 21 epochs.

4.3.1 Network Architecture and Training

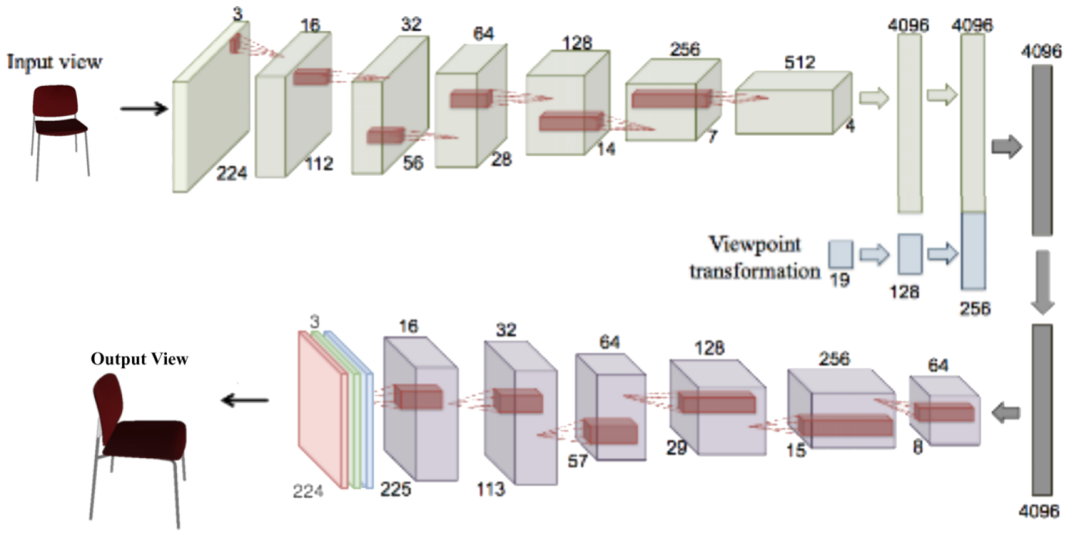


Figure 8: Sub-network architecture for Encoder-decoder for View Transformation

We have illustrated the network sub-architecture in Figure 8. The components are as described in the Network Architecture section of this report. We pass an input image, concatenate the one hot 19 vector as discussed and the network generates a view transformed image.
For training, we limited the permissible view transformations to the first 9 bins of the 19 bin vector, i.e. transformations only between 0° to 180° from the input view.

4.3.2 Using L2 Loss

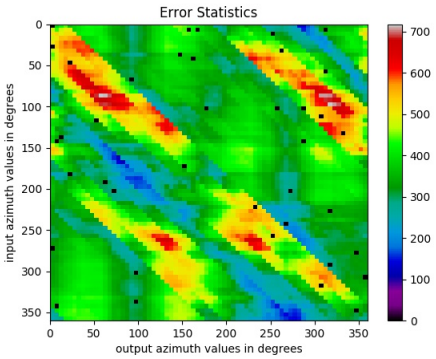


Figure 9: Error Statistics of Encoder-decoder for View Transformation using L2 loss

In Figure 9 we have displayed the error statistics measure for this network using L2 loss in the final layer. Our network does well in generating view transformations even outside the range it was trained for as is evident with the green regions corresponding to lower error on the error statistics plot.

We observed a higher error along the diagonal, which corresponds to image pairs with smaller view transformations. It is important to note here that our network is only trained for around 40 epochs and these errors might converge further with more training.

We have generated these results for a randomly chosen chair type from the test set, and keeping the elevation fixed to a randomly chosen value between the input and output instances. This result was derived for 250,000 input output pairs of images chosen as mentioned above.

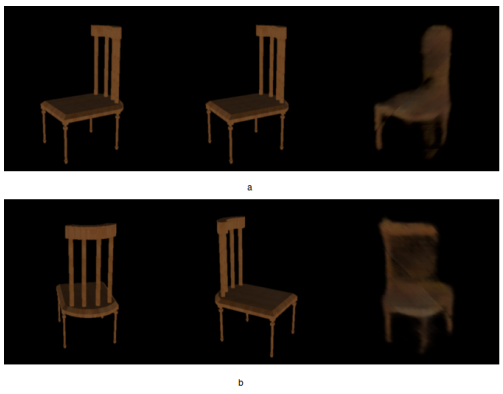


Figure 10: (a) Input image(first column), Output image(second column) are set the same. Third column: network output (b) Input(first column) with occluded regions required for output(second column), Third column: network output

The results in Figure 10 were generated after training the network for 40 epochs. We can see that the network has learnt the shape and color correspondences well and is converging on learning the effects of viewpoint transformation.

Although these outputs are very blurry right now. To get sharper outputs, we move with training the same network with L1 loss. These results are discussed in the next sub-section.

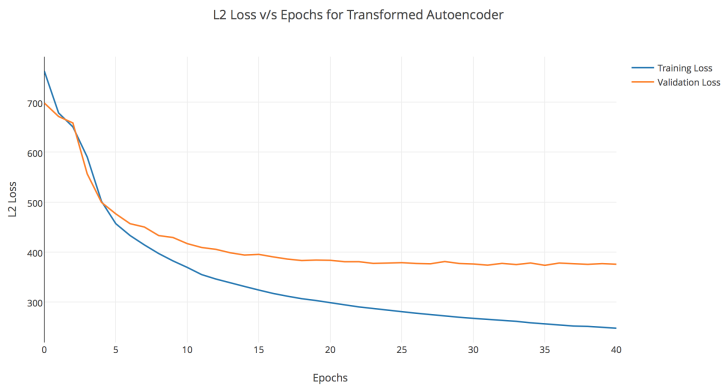


Figure 11: Convergence of L2 loss on training and validation set

Figure 11 shows convergence of L2 loss on the training and validation set for the Encoder-Decoder for View Transformation network. These indicate a smooth convergence, which can be improved upon with further training.

4.3.3 Using L1 Loss

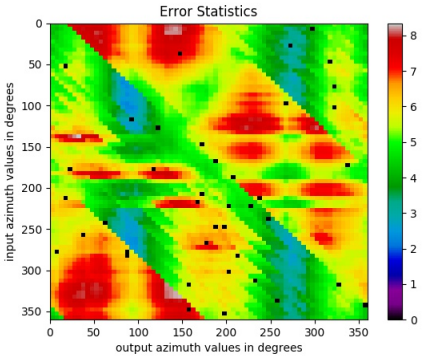


Figure 12: Error Statistics of Encoder-decoder for View Transformation using L1 loss

Here we can see that the error drops to lower values corresponding to input output pairs with smaller transformations (along the diagonal). The scale of this figure is different than that for L2 loss and it can be observed that the L1 reconstruction error is bound in a very small range even for input output pairs with huge transformations.

Also, since this error is bound in a small range and very close the error for input output pairs with smaller transformations, we can highlight the generalization capabilities of this network architecture.

We have generated these results for the same chair type used for L2 loss above but changing the elevation to a randomly chosen value and keeping it fixed between the input and output instances. This result was derived for 250,000 input output pairs of images chosen as mentioned above.

Figure 13: (a) Input image(first column), Output image(second column) are set the same. Third column: network output (b) Input(first column) with occluded regions required for output(second column), Third column: network output

These images were generated from a test chair model, only after 11 epochs of training. A good convergence on shape and color can be observed already and the output only misses on fine details like legs. This is observable even when the input and output images are view transformed as in (b). Another important point to note here is the observation on occluded pixels in the input, which were learned by the network as in (b) where part of the seat of chair was occluded by the backrest, but was recovered correctly in the view transformed network output.

As expected, the outputs are sharper in this case as compared to using L2 loss. This sharpness is achieved even after the 11th epoch as compared to around 40 epochs of the same network with L2 loss as illustrated in Figure 10.

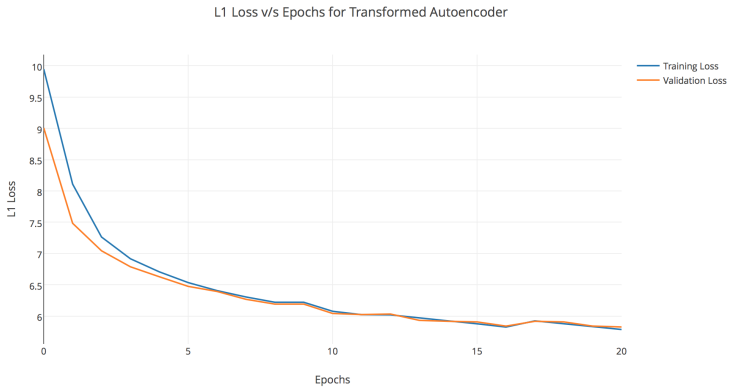


Figure 14: Convergence of L1 loss on training and validation set

Figure 14 shows convergence of L1 loss on the training and validation set for the Encoder-Decoder for View Transformation network. These indicate a smooth convergence, which can be improved upon with further training.

4.4 Our proposed method

We combined the ideas from appearance flow paper[1] and our proposed extension using view transformation encoder-decoder network. As discussed previously, the appearance flow restricted to pixels present in the input image, would not be able to work well in cases where the transformations would require producing parts of the object which were occluded in the original view.

The first task as part of this was to pretrain the network in an autoencoder style to learn a good weight initialization. Following that we built our proposed network and loaded the weights learned using the autoencoder pretraining. Then we run the training. While training, we randomly sampled sets of 2 images, each from the same class and with different azimuth angles as input and target.

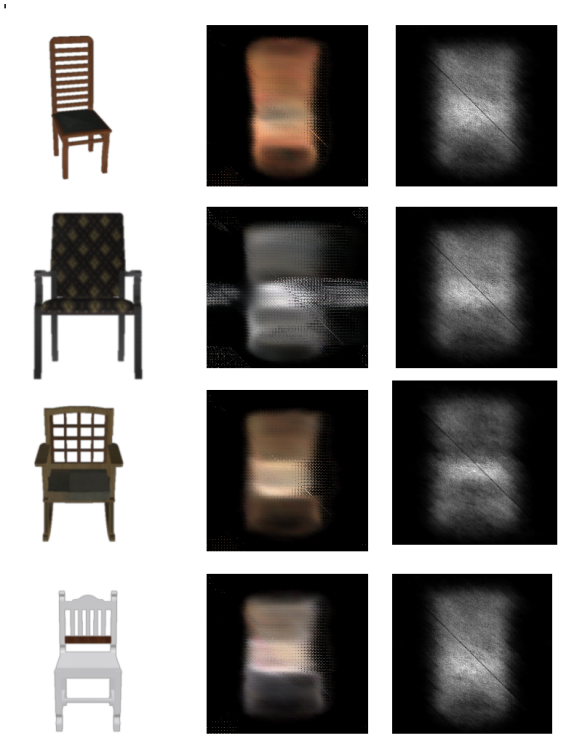


Figure 15: The transformed output produced using our proposed network. The left is the input, the center is the transformed view, with the predicted mask in the last column

Experiment	**Training loss**	**Validation loss**	**Train accuracy**	**Validation accuracy**
Autoencoder	48.98	61.06	96.76	95.19
EDVT (L2, 40 epochs)	248.90	376.6922	90.56	90.21
EDVT (L1, 20 epochs)	5.7873	5.8275	39.93	13.51
Replication	12.34	11.23	94.3	93.32
Proposed network (ongoing)	–	–	–	–

Table 2: Table of results for various experiments performed. We are still training transformed autoencoder, replication and proposed network; EDVT: Encoder-Decoder for View Transformation

4.5 Masked L1 loss

Figure 16: This figure shows results when transformed autoencoder is trained with Masked L1 loss.

4.6 Filter Visualization

In order to inspect what the network was learning we performed a visualization of the filters in the convolution layers. For this we performed experiments similar to those performed by Google in their research blog[9]. They do repeated feedfoward passes to calculate the losses and gradients across the various filters. In each pass the input is updated rather than the network itself using these losses and gradients. This helps to learn what the features the filters are looking for.

For our experiments we used two kinds of input images, one belonging to a chair in the dataset and another one is just a random noise created with the same seed each time. We do 500 passes on each image and use median filtering on the outputs to remove noise. We visualized 4 convolutional layers in the network namely the 3rd, 4th, 5th and 6th convolutional layers. Note that the output of each convolutional layer is first activated using ReLu else the results would be bad.

In Figure 18 we see that filters are trying to detect low level patterns at the lower level i.e. for the 3rd and 4th convolutional layer. As we move forward towards the 5th and 6th layers, the features become more coarser and abstract. From Figure 17 we can see that the 5th and 6th layers are trying to detect the rod like structures in the chair back. This network was learning through a masked loss function and hence was only concentrating on the loss in the chair region.

In Figure 20 we can see that the network is not able to learn any good features as there was no mask provided to it. The results from the masked loss network clearly are much better than this. Same is visible from Figure 19 as we see that it activating for almost everything in sight in the lower layers.

In Figure 22 we can see that the network still learned better than the non-masked L1 error based network. But the features are still not as clear as in the one from the masked L1 loss function network.

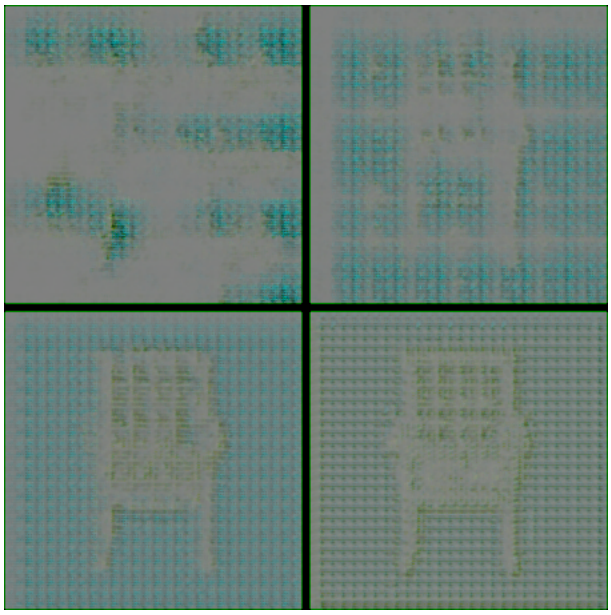


Figure 17: This figure shows visualization of convolution layer filters of transformed autoencoder trained with MaskedL1 loss with a chair as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

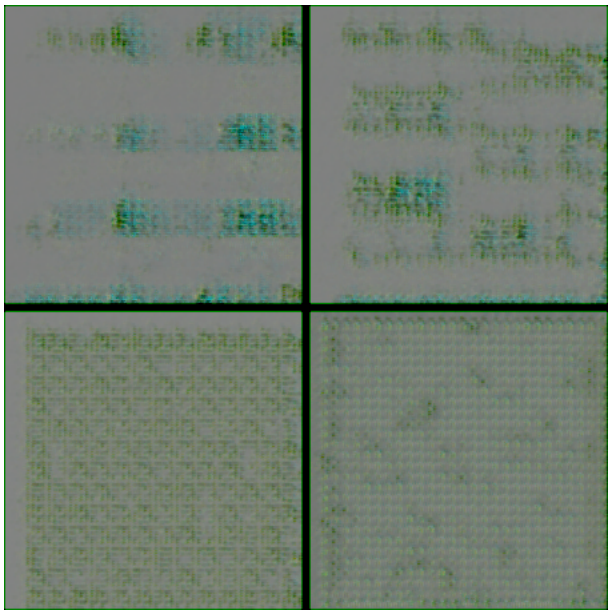


Figure 18: This figure shows visualization of convolution layer filters of transformed autoencoder trained with MaskedL1 loss with noise as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

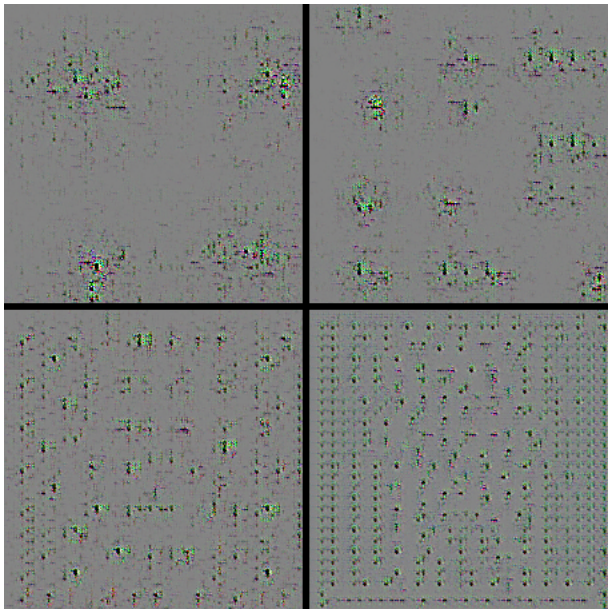


Figure 19: This figure shows visualization of convolution layer filters of transformed autoencoder trained with L1 loss with a chair as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

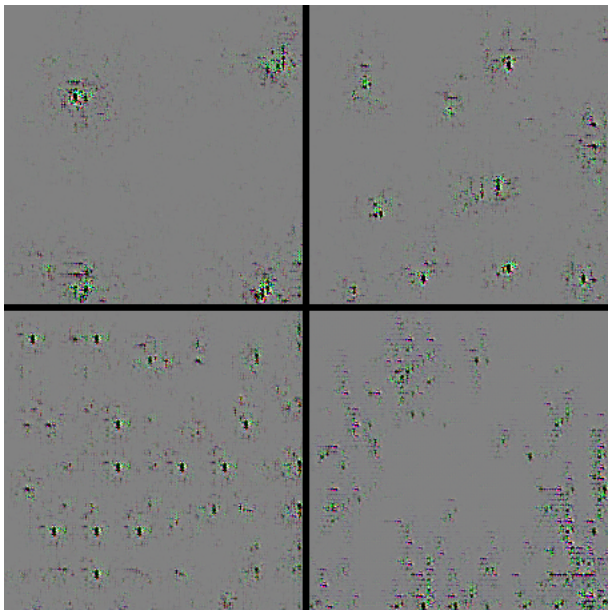


Figure 20: This figure shows visualization of convolution layer filters of transformed autoencoder trained with L1 loss with noise as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

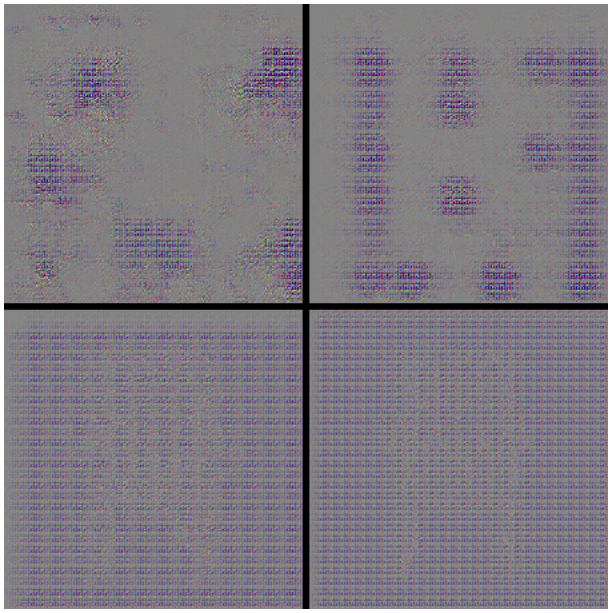


Figure 21: This figure shows visualization of convolution layer filters of transformed autoencoder trained with L2 loss with a chair as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

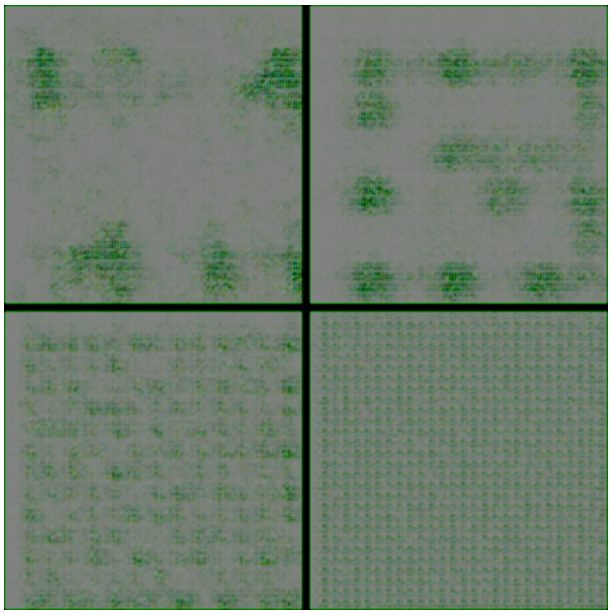


Figure 22: This figure shows visualization of convolution layer filters of transformed autoencoder trained with L2 loss with noise as input. Top left shows mean of weights of all filters of 6th convolution layer, top right for 5th, bottom left for 4th and bottom right for 3rd convolution layer.

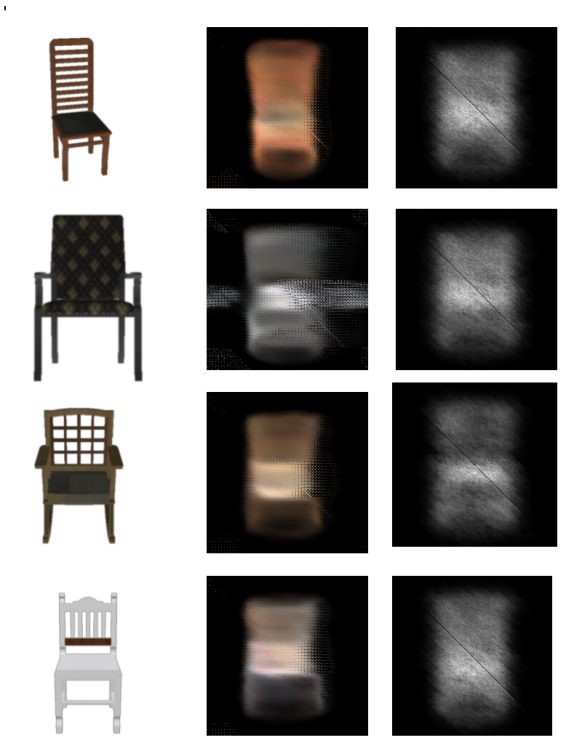


Figure 23: The transformed output produced using our proposed network. The left is the input, the center is the transformed view, with the predicted mask in the last column

4.7 Activation Visualization

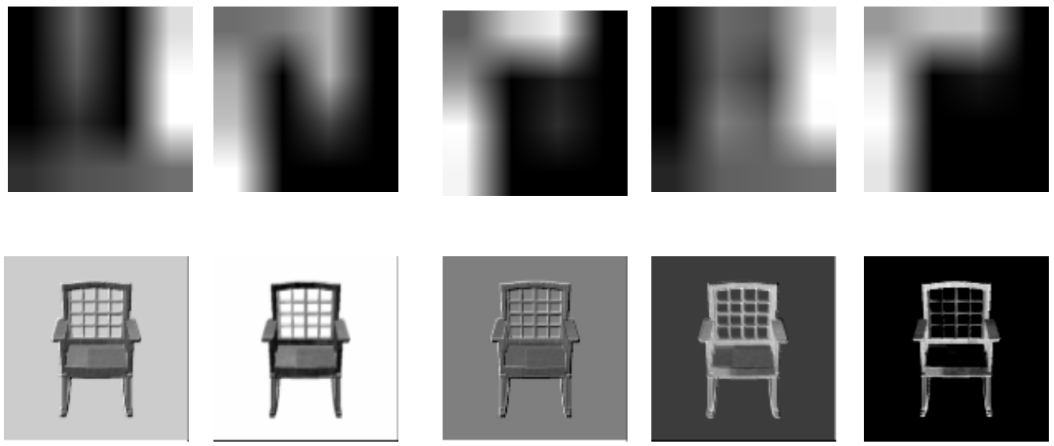


Figure 24: The above figure shows the activations of different convolution layer filters. The first row shows the activation of five different filter activation of convolution layer 6. The second layer shows the activation of five different filter activation of convolution layer 1.

We visualized the activations of the 1st and 6th convolution layer of our synthesized transformed autoencoder. The visualizations can be seen in Figure 24. The first row shows the activations of convolution layer 6. The output presented is pixelated because the original output was of size 4x4. We can see that convolutional layer 6 is able to learn shapes like back of the chair (Figure 24 row 1, image

4), legs (Figure 24 row 1, image 2), arm rest(Figure 24 row 1, image 3, 5). The first convolutional layer as shown by row 2, is able to learn standard features like color contrast, sharpness, edges.

5 Discussion and Conclusion

5.1 Pretraining the Autoencoder

We observe good reconstructions for the autoencoders trained on both the training splits. In Figure 7 we see the outputs from an autoencoder learnt with a disjoint set of chair styles between training and testing data. In Figure 4 we see outputs on an autoencoder trained using splitting such that there is a disjoint set of viewpoints in the training and testing data. We see that both these autoencoders are producing good reconstructions which improve progressively with the number of training epochs. The images of the chairs produced in each column belong to an autoencoder trained over 20, 25 and 30 epochs respectively, going from left to right, with the first column corresponding to the input image. Thus we can infer that the autoencoders are able to learn good representations of the chairs in both the cases, regardless of whether they haven't seen either the viewpoint or the chair itself previously.

5.2 Encoder-decoder for View Transformation

In Figure **??** we can see the results of the encoder-decoder network we trained for producing view transformed outputs of the given image. We see that the outputs are a little blurry but produce reasonable transformations on the input image. The blurry outputs were expected beforehand, since the Encoder-decoder network is essentially learning an efficient dimensionality reduction on the given dataset. This removes the high frequency features which make the image sharper and more detailed. Also contributing to this is the fact that the network uses an L2 loss function to Apart from this, the shapes are not exactly like those in the ground truth because this is a very complex task for a simple autoencoder to learn. This is evident from the results of the Tatarchenko et al[2]. Even after using depth maps, they produce blurry outputs which do not have the features present in the expected image.

5.3 Our proposed method

Our proposed is currently not working as we had expected it to be. We have thought of a couple of reasons for the same -

- We have used the same deconvolution branch for the flow and regeneration channels. This might lead to poor features maps as both processes need to have different kinds of features. We plan to mitigate this by having different deconvolution branches for flow and image regeneration channels by the time of the end presentation.
- The network might not have been trained with enough epochs yet, as we just had time to train it with 10 epochs.

6 Concept and/or Innovation

- For the first and second experiments, we implemented the encoder-decoder network. This was an analysis over the results obtained in [2]. We analyzed whether the autoencoder network would be able to regenerate chairs from both missing views and missing chair styles or not.
- For the third experiment, we have based our network on that of Tatarchenko et al[2]. However, we did not use depth images apart from RGB images, as they have done for their training. We have just used RGB images for our training in that experiment. We analyzed whether the encoder-decoder network alone would be able to generate view transformed images. This was essential to our task as we wanted to supplement the ideas in [1] with hallucination capabilities from encoder-decoder network architecture such as this one.
- For the final experiment, we have created a novel modified architecture from the works of Zhou et al[1] by adding the RGB channels alongside the flow channel for generating view transformed reconstructions as we concluded that it would help the failure cases of [1].

- We also implemented the novel Bilinear-sampling layer in Keras, proposed by the authors in [1]. They had implemented a similar layer in Caffe for their network but their implementation was iterative. Our implementation on the other hand is optimized for batch processing using matrix operations.
- We also found a better method to initialize parameters of our novel architecture. Rather than just starting with a bunch of random initial weights, we pretrain our network in an auto-encoder style to learn good representations of the chairs themselves. We can see that the initialization works well as the initial loss is pretty low for our network.

7 Acknowledgements

We would like to thank Prof. Gary Cottrell for helping us contact the original authors of the paper and for also providing us with valuable data of the project. We would also like to thank the TAs - Mainak Biswas and Utkarsh Simha for helping us out with various issues we faced. Also we'd like to thank Tinghui Zhou and Shubham Tulsiani for providing us with the dataset and information about their implementation.

References

1. Zhou, T., Tulsiani, S., Sun, W., Malik, J. and Efros, A. View Synthesis by Appearance Flow. Computer Vision – ECCV 2016, pp.286-301. (2016).
2. Tatarchenko, M., Dosovitskiy, A., Brox, T.: Single-view to multi-view: Reconstructing unseen views with a convolutional network. arXiv preprint arXiv:1511.06702 (2015).
3. Chang, Angel X., et al. Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015).
4. Hinton, G. Reducing the Dimensionality of Data with Neural Networks. Science, 313(5786), pp.504-507. (2006).
5. Shepard, R. and Metzler, J. Mental Rotation of Three-Dimensional Objects. Science, 171(3972), pp.701-703. (1971).
6. Oh, B.M., Chen, M., Dorsey, J., Durand, F.: Image-based modeling and photo editing. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. (2001).
7. Max Jaderberg, Karen Simonyan, Andrew Zisserman and Koray Kavukcuoglu - Spatial Transformer Networks - In CoRR 2015. (2015).
8. Chollet, François, Keras, `https://github.com/fchollet/keras`. (2015).
9. Google Deep Dream Research Blog `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`

Team Contribution

Shradha Agrawal implemented all network architectures including autoencoder, transformed autoencoder, replication network of [1] and proposed network with 5 channels in Keras and wrote code for training and running the experiments. She also ran various experiments including autoencoder, transformed autoencoder and replication network. She played a crucial part in procurement of data from the authors of previous work.

Sudhanshu Bahety implemented the bilinear sampling layer in a more efficient way than the original work. He also coded the implementation for activation visualization and helped in running the autoencoder experiments. He complied the results after running experiments for the activation visualization task and helped ad-hoc in generating good quality plots for convergence of our models.

Jean Choi worked towards all the aspects related to image processing, like mask generation on the fly, visualization of the network architecture and compilation and presentation of the results in this report as well as the presentation. She also ran autoencoder experiments in the pre-training stage of the network and provided crucial inputs for implementation of bilinear sampling.

Ajitesh Gupta worked towards understanding and completing the missing pieces of code provided by the authors of previous work in caffe. He worked towards making the bilinear sampling code portable to tensorflow and helped in implementing it efficiently. He wrote the code for the deep dream visualization and ran those experiments. He also created the masked L1 Loss function based transformed autoencoder network and ran the training experiments on that network.

Nimish Srivastava worked towards procuring the data from authors of previous work. He contributed towards making the image masks on the fly. He also wrote code for generating data batch-wise on the fly as loading the entire dataset was beyond the scope of computation. He implemented the error statistics experiments in code, ran experiments for autoencoder training, Encoder Decoder for viewpoint transformation training, and error statistics of viewpoint transformed images.

Everyone contributed towards the presentation and compiling our observations in this report.

8 Appendix

As the code is very big, we are providing link to our repository here:
https://bitbucket.org/shrads93/cse-253-appearance-flow-keras

You should take code from branch 'develop'.

To clone this, you can execute command
git clone https://shrads93@bitbucket.org/shrads93/cse-253-appearance-flow-keras.git