Rotating Objects with Deep Feedforward Networks

July 23, 2019

Given an image of an object, we would like to generate an image of that object rotated around some axis, mostly vertical. This problem is often called "view synthesis" because we can think of it as changing the view transformation of the camera.

In this document, we will discuss a series of paper that culminates in the paper by Park et al. [2]. We start with the paper by Tatarchenko et al. [5], which is probably the first and simplest paper that solves this problem with a feedforward network. While the paper can infer rough shapes from input images, the textures are too blurry to call the results high quality.

We then take a detour to discuss the Spatial Transformer Network (STN) paper [1], which enables simple image transformtion (translating, scaling, rotating, cropping, and warping) inside a network. The paper itself does not solve the view synthesis problem, but its spetial transformation unit is a tool that can be used to implement optical flow inside a feed forward network. The technique is used by the Appearance Flow Network (AFN) paper to copy pixels of the input image to the synthesized view [6]. AFN produces textures of higher quality than those produced by Tatarchecko et al., but they are still not accurate enough.

Lastly, Park et al.'s Transformation-grounded View Synthesis Network (TVSN) extends AFN by adding a visibility mask and a second network to fill the holes left by occluded pixels and fix any artefacts caused by appearance flow. IMHO, the paper is interesting because it combines two ideas—reusing pixels through appearance flow and using GANs to generate fine-grained details—to produce better result.

1 A Straightforward Solution

- The input to the network is a pair $(\boldsymbol{x}, \boldsymbol{\theta})$ where $\boldsymbol{x}$ is an image of an object and θ is the desired viewpoint. Here, $\boldsymbol{\theta}$ is a vector $(\theta^{\mathrm{az},\theta^{\mathrm{el}}})$ of azimuthal and elevation angles, respectively.
- The output of the network is an image $\boldsymbol{y}$ where $\boldsymbol{y}$ is the image with the object in it viewed from viewpoint specified by $\boldsymbol{\theta}$. The paper also produces $\boldsymbol{d}$,the depth map of $\boldsymbol{y}$. However, we are not interested in the depth map part.
- Note the the viewpoint associated with $\boldsymbol{x}$ is not given to the network; it must infer this itself.
- The network has a encoder-decoder architecture.
  - The encoder consists of three parts, which we will call A, B, and C.
    - A is a 6-layered purely convolutional network that converts the input image $\boldsymbol{x}$ to a vector $\boldsymbol{a} \in \mathbb{R}^{1024}$.
    - B consists of 2 fully connected layers. It converts the viewpoint $\boldsymbol{\theta}$ to a vector $\boldsymbol{b} \in \mathbb{R}^{64}$.
    - C also consists of 2 fully connected layers. It converts the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$ to a vector $\boldsymbol{z} \in \mathbb{R}^{1024}$. This is the hidden representation of the desired view.

    Writing out the encoder as an equation, we have:

    $$\boldsymbol{z} = C(A(\boldsymbol{x}) \oplus B(\boldsymbol{\theta}))$$

    where $\oplus$ denotes vector concatenation.

  - The decode is a 6-layered purely convolutional network that converts $\boldsymbol{z}$ to a 128×128 3-channel image.

- Unfortunately, the paper *does not* specify the loss function it uses. (WTF!) From the code,[1] it seems that it uses L^2 loss for the image and the L^1 for the depth map.
- The training data is prepared from the ShapeNet dataset. The authors use 7496 cars and 6742 chairs. Each object is rendered from 36 azimuthal angles and 5 evaluation angles, so each object has 180 viewing directions.
- As started earlier, the paper achieves good overall shapes, but the textures are too blurry to be called high quality reconstructions.

2 Spatial Transformer Network

We take a break from view synthesis to discuss a generally useful technique.

- A **spatial transformer** is a differentiable module which applies a spatial transformation to a feature map during a single forward pass.
  - The transformation is conditioned on the particular input, producing a single output feature map. In other words, each example given to the transformation gets its own associated output.
  - For multi-channel inputs, the same transformation is performed to each channel independently.
- A spetial transformer has three parts:
  - The **localization network** takes in the input feature map and outputs the parameters of the spatial transformation that should be applied to the feature map.
  - The **grid generator** generates a set of points where the input map should be sampled from the parameters of the transformation outputted by the localization network.
  - The **sampler** samples the input map according to the grid points to produce the output.
- The localization network:
  - Takes as input the feature map $U \in \mathbb{R}^{C \times H \times W}$.
  - Outputs θ, the parameters of the transformation $\mathcal{T}_\theta$ to be applied to the feature map.
  - It can take any form but should include a final regression layer to produce the parameters θ.
- The spatial transformation outputted by the localization network can take many forms. One of them is the 2D affine transformation:
$$A_\theta = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix},$$
  which has six parameters.
- Note that the transformation transforms a *target point* in the image plane of the output to a *source point* to the image plane of the input. As such, the points transformed are always regular grid points in the output image plane. For example, the point-wise transformation according to A_θ is given by:
$$\begin{bmatrix} x^s \\ y^s \end{bmatrix} = A_\theta \begin{bmatrix} x^t \\ y^t \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{bmatrix} x^t \\ y^t \\ 1 \end{bmatrix}$$
  where (x^s, y^s) denotes the output source point and (x^t, y^t) denotes a target point.

[1] `https://github.com/lmb-freiburg/mv3d/blob/master/nobg_dm.py`

- The paper used normalized coordinates. In other words, $-1 \leq x^t, y^t, x^s, y^s \leq 1$ when the points are within the bounds of the input or the output.
- Let the output pixel lies on a regular grid $G = \{G_i\}$ where $G_i = (x_i^t, y_i^t)$. Naturally, let $(x_i^s, y_i^s) := \mathcal{T}_\theta(G_i)$ be the corresponding source point generated by the spatial transformation.
- Let V denote the output map, and let V_i^c denote the value of Channel c at G_i. We define V_i^c to be the convolution of the input map with a sampling kernel k centered at G_i:

$$V_i^c := \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y)$$

  where Φ_x and Φ_y are parameters of the sampling kernel k.
- Any kernels can be used as long as the subgradients can be defined for any x_i^s and y_i^s values.
- Example kernels:
  - The *neareast neighbor kernel*:

$$V_i^c := \sum_n^H \sum_m^W U_{nm}^c \delta(\lfloor x_i^s + 0.5 \rfloor - m) \delta(\lfloor y_i^s + 0.5 \rfloor - n)$$

  - The *bilinear sampling kernel*:

$$V_i^c := \sum_n^H \sum_m^W U_{nm}^c \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|)$$

    The paper [1] has formula for partial derivatives with respect to all the parameters.
- The spatial transformer can be inserted into any part of a neural network. Potential benefits include:
  - It can speed up subsequent modules because it can downsample the feature maps.
  - It can increase the performance of the overall network because it enables the network to learn to transform the feature maps.
  - For some tasks, the output of the localization network can be fed to subsequent modules for further processing.
  - Multiple spatial transformers can be used in parallel to process multiple objects separately.

3 Appearance Flow Network (AFN)

- The insight of the paper is to use the pixels of the input view to construct the output view. This approach works if the object is rotated by not too much or if the newly visible parts are similar to the ones already visible.
- The input to the network is a source image $\boldsymbol{I}_s$ of an object and a relative transformation T, which is a generally a rotation.
- The output is the image $\boldsymbol{I}_t$ of the object in $\boldsymbol{I}_s$ transformed by T.
- Let g denotes our desired network, the paper seeks to minimize the L^p loss between the real target image and the one produced by the network:

$$\mathcal{L} := E_{(\boldsymbol{I}_s, T, \boldsymbol{I}_t) \sim p_{\text{data}}} [\|\boldsymbol{I}_t - g(\boldsymbol{I}_s, T)\|_p].$$

- Internally, the network first produces a dense flow field $\boldsymbol{f}$. For each pixel i, the value of $\boldsymbol{f}^{(i)}$ is a 2-dimentional vector $(x^{(i)}, y^{(i)})$, specifying the position from the source image to sample from. Using bilinear interpolation, the output at Pixel i is given by:

$$g^{(i)}(\boldsymbol{I}_s, T) := \sum_{q \in \mathrm{neigbors}(x^{(i)}, y^{(i)})} I_s^{(q)} \max(0, 1 - |x^{(i)} - x^{(q)}|) \max(0, 1 - |y^{(i)} - y^{(q)}|),$$

  and this is directly lifted from [6].

- The network that produces the flow field $\boldsymbol{f}$ has the encoder-decoder artchitecture. The details are as follows:
  - The flow field generator network has 3 parts:
    * The *input view encoder* transforms the input image to a 4096-dimensional vector. It uses 5 convolutional layers followed by 2 fully connected layers.
    * The *viewpoint transformation encoder* transforms the given viewpoint to a 256-dimensional vector. It uses 2 fully connected layers.
    * The *synthesis decoder* concatenates the output of the last two parts and transfrom the result to a dense flow field. It uses 2 fully connected layers to transformed the $(4096 + 256)$-vector to a 4096-vector. Then, the vector is passed through 6 (upsampling) convolution layers to generate the dense flow field.
- The paper also has a network that predicts the foreground mask of the target image. It has the same architecture as the flow-field network above but produces a binary image. The network is trained with the cross entropy loss.
- The technique can be extended to take advantage of multiple input views.
  - This is done by having the network produce a confidence map C as an extra channel to the output.
  - Given m input images, we use the network to compute m output images $g_1(\boldsymbol{I}_s, T)$, $\dots$, $g_m(\boldsymbol{I}_s, T)$ and m confidence maps $\boldsymbol{C}_1$, $\dots$, $\boldsymbol{C}_m$.
  - The final output image is the weighted average of the outputs, using the confidence values as the weights:

$$g^{(i)}(\boldsymbol{I}_s, T) := \sum_{j=1}^{m} \left(\frac{\boldsymbol{C}_j^{(i)}}{\sum \boldsymbol{C}_j^{(i)}} \right) g_j^{(i)}(\boldsymbol{I}_s, T).$$

- The textures generated by this technique is of higher quality than that of the Tatarchenko et al. paper. However, since it only copy pixels from visible parts of the model, the textures of the occluded parts are not very accurate.

4 Transformation-Grouded View Synthesis Network (TVSN)

- The paper's premise is to decompose view systhesis into a three-step process:
  - Move pixels in the input view that are also visible in the output view to their new positions.
  - Remove the remaining pixels from the image.
  - Previously unseen pixels are hallucinated into existence.
- The paper's network consists of two subnetworks:
  - The **Disocclusion-aware Appearance Flow Network** (DOAFN) takes as input:

    * the input view image, and
    * the desired relative transformation (20° to 340° in increments of 20°, encoded a 17-vector.

  It outputs:

    * A latent representation of the input image coupled with the transformation.
    * An image of the new view where only the pixels that are shared with the input view are moved to their new positions, and the missing new pixels are left blank.

  - The **View Completion Network** (VNF) takes as input the outputs of the previous network and outputs the image of the new view. It has two jobs:
    * Generate pixels that the previous network did not.
    * Fix any artefects in the previous network's image.

4.1 Disocclusion-aware Appearance Flow Network (DOAFN)

- The DOAFN is very similar to the AFN. It has an encoder-decoder architecture.
- Details of the encoder:
  - Seven convolutional layers convert the input view image into a 2048 dimensional latent vector.
  - Two fully connected layers transform the one-hot vector of the relative view to a 256 dimensional latent vector.
  - The above two latent vectors are concatenated and then transformed by two fully connected layers into a 2048-vector.
- Details of the decoder:
  - The decoder is a purely convolutional network consisting of 7 layers. However, the last layer has three heads, producing three outputs:
    * The dense flow field $\boldsymbol{F}$.
    * The visibility mask $\boldsymbol{M}_{\text{vis}}$ that tells which pixels in the new view comes from the old view.
    * The background mask $\boldsymbol{M}_{\text{bg}}$ that tells which background pixels in the input view would remain unchanged in the output view.
  - The output of the first convolutional layer is an image block of size $512 \times 4 \times 4$ is taken as the latent representation of the rotated object. It is fed as an input to the VFN.
- The final image that the DOAFN outputs is created from $\boldsymbol{F}$, $\boldsymbol{M}_{\text{vis}}$, and $\boldsymbol{M}_{\text{bf}}$.
  - First, the flow field is used to bilinearly sample from the source image $\boldsymbol{I}_s$. Let us call this sampled image $\boldsymbol{I}_{\text{afn}}$.
  - The sampled image is mulitplied to the visibility mask. The resuilt is then addded to the extracted background:

$$\boldsymbol{I}_{\text{doafn}} = \boldsymbol{I}_{\text{afn}} \otimes \boldsymbol{M}_{\text{vis}} + \boldsymbol{I}_s \otimes \boldsymbol{M}_{\text{bg}}.$$

- The DOAFN can be trained independently without the rest of the network. The training data are generated as follows:
  - The groundtruth background mask can be obtained from the rendering of the object. However, one needs to be careful that this is the intersection of the background masks of *both* the input view and the output view. We can use the cross entropy loss with it.

  - To be the most correct, the visiliby map can be generated by shadow mapping. We may assume that there's a light source located at the camera position of the original view, the visible pixels in the new view are those that receive light when viewed from the new view. However, the paper determines visible pixels by whether the (old) normal vector is facing towards the camera in the new view. This is not entirely correct, though. Again, we can use the cross entropy loss with this.
  - The image $\boldsymbol{I}_{\text{afn}} \otimes \boldsymbol{M}_{\text{vis}}$ can be generated from the 3D models. We can use any L^p loss with it.
  - Again, it was not clear to me what loss functions are used to trained the DOAFN. This was another WTF moment in this series of paper.

- The paper trains DOAFN first before the VFN.

4.2 View Completion Network (VFN)

- The VFN's architecture is that of U-Net [3]. The only difference is that the $512 \times 4 \times 4$ latent representation from the DOAFN is inserted in one of the bottleneck layers. See the paper for more details.
- The paper says the architectural choice has the following advantages:
  - Since the network is conditioned on the latent representation of DOAFN, it can generate content that have consistent attributes with the input view.
  - Since the output image of DOAFN is already in the target view, the pixels in the final output image are aligned with the output images. Hence, we can effectively use U-Net style skip connections.
- Perhaps the most elaborate part of the VFN is the loss function.
  - There is an **advesarial loss** term. While training the VFN, we also train a discriminator D that distinguished between the real outputs and the outputs generated by the whole network, which we will call G. The loss of the discriminator is the standard GAN loss:
    $$\mathcal{L}_D := -E[\log D(\boldsymbol{I}_t)] - E[\log(1 - D(G(\boldsymbol{I}_s, T)))].$$
    The adversarial loss for the generator is given by:
    $$\mathcal{L}_{\text{adv}} := -E[\log D(G(\boldsymbol{I}_s, T))].$$
    Note that the expections are based on the random variable $(\boldsymbol{I}_s, T, \boldsymbol{I}_t) \sim p_{\text{data}}$.
  - The paper uses **feature matching** to stabilize training [4]. Let F_D denote the vector formed by the activations of the first three layers of the discriminator. The paper train the generator to minimize:
    $$\mathcal{L}_{\text{fm}} := E\Big[\|F_D(\boldsymbol{I}_t) - F_D(G(\boldsymbol{I}_s, T))\|_2^2\Big].$$
    Note, however, this is not the exact feature matching loss in the Salisman et al. paper because the expectation is pulled output the L^2 norm. This loss is more similar to the perceptual loss below.
  - It uses **perceptual loss** to match the image contents. Let $\boldsymbol{F}_{\text{vgg}}$ denote the vector formed by the activations of the first three layers of VGG16, and The perceptual loss is given by:
    $$\mathcal{L}_{\text{per}} := E\Big[\|F_{\text{vgg}}(\boldsymbol{I}_t) - F_{\text{vgg}}(G(\boldsymbol{I}_s, T))\|_2^2\Big].$$
  - Lastly, the paper also minizes the L^1 **difference** between the generated output and the ground truth and the **total variation loss** of the generated output.
    $$\begin{aligned}\mathcal{L}_{\text{diff}} &:= E[\|\boldsymbol{I}_t - G(\boldsymbol{I}_s, T)\|_1] \\ \mathcal{L}_{\text{tv}} &:= E\Big[\sum_{i,j}(G^{i+1,j}(\boldsymbol{I}_s, T) - G^{i,j}(\boldsymbol{I}_s, T))^2 + (G^{i,j+1}(\boldsymbol{I}_s, T) - G^{i,j}(\boldsymbol{I}_s, T))^2\Big]\end{aligned}$$

  - The complete loss is given by:

$$\mathcal{L}_{\text{vfn}} := \mathcal{L}_{\text{adv}} + \alpha\mathcal{L}_{\text{fm}} + \beta\mathcal{L}_{\text{per}} + \gamma\mathcal{L}_{\text{diff}} + \lambda\mathcal{L}_{\text{tv}}.$$

    The paper uses $\alpha = 1$, $\beta = 0.001$, $\gamma = 1$, and $\lambda = 0.0001$.

- After we training the DOAFN, we train the VFN with the DOAFN fixed. Then, we fine-tune both networks end-to-end. The paper observes though that fine-tuning does not results in much improvement.

- Insights from the Results section:

  - Better textures can be assessed with L_1 difference from the ground truth.
  - All methods, even the baselines, achieved similar SSIM values.
  - If one replaces DOAFN with just simple AFN in the full network, it seems that both methods perform similarly with respect to the L_1 and SSIM metrics. However, AFN's artefacts can propagate to the final image while the use of DOAFN rules this out with the visibility map.
  - It seems the VFN needs both the adversarial loss and the VGG16 perceptual loss to achieve high quality results. VGG16 loss alone yields blurry output, and adversarial loss alone yields color and details inconsistencies.

References

[1] Jaderberg, M., Simonyan, K., Zisserman, A., and kavukcuoglu, k. 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.

[2] Park, E., Yang, J., Yumer, E., Ceylan, D., and Berg, A. C. Transformation-grounded image generation network for novel 3d view synthesis. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017* (2017), pp. 702–711.

[3] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR abs/1505.04597* (2015).

[4] Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. *CoRR abs/1606.03498* (2016).

[5] Tatarchenko, M., Dosovitskiy, A., and Brox, T. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision (ECCV)* (2016).

[6] Zhou, T., Tulsiani, S., Sun, W., Malik, J., and Efros, A. A. View synthesis by appearance flow. In *European Conference on Computer Vision* (2016).