Generative Adversarial Networks (GAN), part1

Alexander Binder

University of Oslo (UiO)

May 5, 2021

UiO : Department of Informatics
University of Oslo

Learning goals

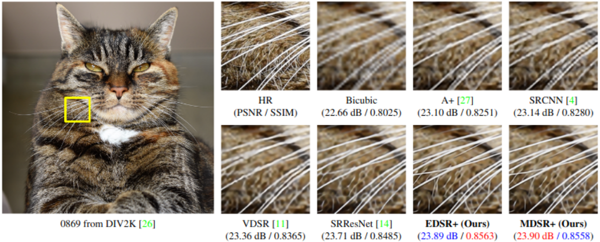
- today: none!
- this is not exam stuff, but outlook stuff

Where do we start ?

GAN inpainting from the last lecture

Superresolution

Problem: given a low-res image y, learn to interpolate higher resolution variant x

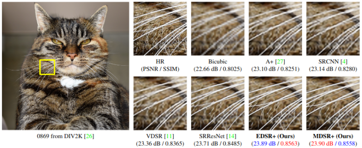


from https://arxiv.org/abs/1707.02921

Simple idea:

- take high res images x, apply some blur operator k: $y = k(x)$,
- then train a segmentation-type network on pairs or HR/LR: (x, y)
- runs into an ugly problem. Guess?

Problem: given a low-res image y, learn to interpolate higher resolution variant x



Simple idea:

- take high res images x, apply some blur operator k: $y = k(x)$,
- then train a segmentation/GAN-type network on pairs (x, y)
- runs into an ugly problem. Guess?
- training will overfit to the blur kernel k and not generalize to real images, where the LR image is created with a different kernel than your k used to generate training data.
- next idea?

Problem: given a low-res image y, learn to interpolate higher resolution variant x

next idea:

- try to solve optimization problem to learn the blur kernel involved in creating the LR image x and the LR image

$$x, k = \mathrm{argmin}_{k,x} \|y - (x \otimes k) \downarrow_s \| + \phi(x)$$

where $\downarrow_s$ is a standard bilinear downsampling operation with stride s

- can do a 2 step decomposition:

$$\begin{aligned} k &= M(x) \text{ estimate kernel first} \\ x &= \mathrm{argmin}_{x} \|y - (x \otimes k) \downarrow_s \| + \phi(x) \end{aligned}$$

Superresolution

Problem: given a low-res image y, learn to interpolate higher resolution variant x

next idea:

- try to solve optimization problem to learn the blur kernel involved in creating the LR image x and the LR image

$$x, k = \mathrm{argmin}_{k,x} \|y - (x \otimes k) \downarrow_s \| + \phi(x)$$

- can do a 2 step decomposition:

$$\begin{aligned} K &= M(x, \hat{y}) \\ x &= \mathrm{argmin}_x \|y - (x \otimes k) \downarrow_s \| + \phi(x) \end{aligned}$$

  where $\hat{y}$ is an intermediate HR image estimate

- can iterate this:

$$\begin{aligned} k_{i+1} &= \mathrm{argmin}_k \|y - (x_i \otimes k) \downarrow_s \| + \phi(x_i) \\ x_{i+1} &= \mathrm{argmin}_x \|y - (x \otimes k_{i+1}) \downarrow_s \| + \phi(x) \end{aligned}$$

A well performing not too complicated architecture:

https://arxiv.org/abs/2010.02631

- interpolates invisible information, invents structures

- ...

Neural networks to interpolate new views from scenes.

The original paper:
https://arxiv.org/abs/2003.08934

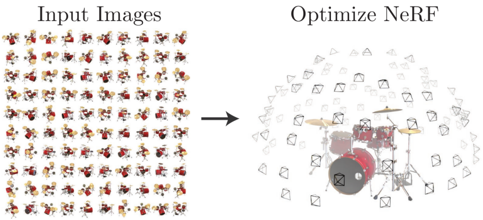
improvements:

10x faster, better
https://arxiv.org/abs/2007.11571

speed vs quality tradeoff, FPS $\gg 1$:
https://arxiv.org/abs/2103.10380

What is the problem?

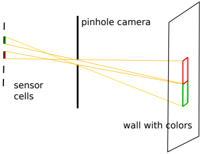
Given many views of a scene from different angles,



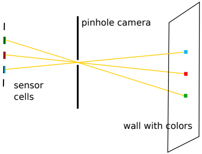
interpolate a view from a new viewing angle, possibly at a high resolution:

What is an image ?

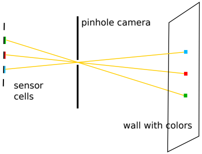
- colors received on sensor elements as an average of many incoming rays



- idealized model: color is received by a single ray.



- Question: How can we model the color arriving on a single ray? Literature on Ray-Casting/Ray-Tracing.



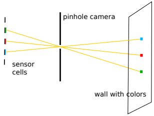
model ray as: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ where $\mathbf{o}$ is origin of the ray (camera sensor position) and $\mathbf{d}$ is the direction vector of a ray.

Then one common model for simplified modeling of color arriving on a ray:

$$C(r(t)) = \int_{\mathbf{r}(t)\in \text{ray}} \text{``probability that ray reaches until''}\ \mathbf{r}(t) * color(\mathbf{r}(t), \mathbf{d})\ldots$$
$$* \text{``density of material at''}\ \mathbf{r}(t)dt$$
$$= \int T(t)c(\mathbf{r}(t), \mathbf{d})\sigma(\mathbf{r}(t))dt$$

idea:

- $T(t)$ – probability that ray reaches until $\mathbf{r}(t)$ (no blocking surface/particle between sensor position $\mathbf{o}$ and $\mathbf{r}(t)$)



model ray as: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ where $\mathbf{o}$ is origin of the ray (camera sensor position) and $\mathbf{d}$ is the direction vector of a ray.

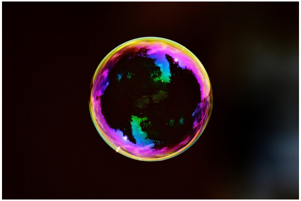
Then one common model for simplified modeling of color arriving on a ray:

$$C(r(t)) = \int T(t)\mathbf{c}(\mathbf{r}(t), \mathbf{d})\sigma(\mathbf{r}(t))dt$$

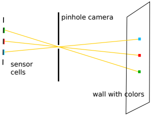
idea:

- $T(t)$ – probability that ray reaches until $\mathbf{r}(t)$ (no blocking surface/particle between sensor position $\mathbf{o}$ and $\mathbf{r}(t)$)
- $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ what color is emitted at location $\mathbf{r}(t)$ in direction $\pm\mathbf{d}$

- $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ what color is emitted at location $\mathbf{r}(t)$ in direction $\pm\mathbf{d}$



The history of physical valuables is a history of non-lambertian materials



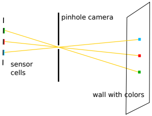
model ray as: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ where $\mathbf{o}$ is origin of the ray (camera sensor position) and $\mathbf{d}$ is the direction vector of a ray.

Then one common model for simplified modeling of color arriving on a ray:

$$C(ray) = \int T(t)\mathbf{c}(\mathbf{r}(t), \mathbf{d})\sigma(\mathbf{r}(t))dt$$

idea:

- $T(t)$ – probability that ray reaches until $\mathbf{r}(t)$ (no blocking surface/particle between sensor position $\mathbf{o}$ and $\mathbf{r}(t)$)
- $\mathbf{c}(\mathbf{r}(t), \mathbf{d})$ what color is emitted at location $\mathbf{r}(t)$ in direction $\pm\mathbf{d}$
- $\sigma(\mathbf{r}(t))$ material density



model ray as: $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$
Then one common model for simplified modeling of color arriving on a ray:

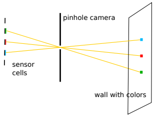
$$C(ray) = \int T(t)\mathbf{c}(\mathbf{r}(t), \mathbf{d})\sigma(\mathbf{r}(t))dt$$

- $\sigma(\mathbf{r}(t))$ material density
  - proportional to the difference in probability that ray will be stopped in a small interval $[t_1, t_2]$ passing from $\mathbf{r}(t)$ to $\mathbf{r}(t + \delta t)$
- model assumption:

$$dT(t) = T(t) * (-\sigma(\mathbf{r}(t)))$$

$$\Rightarrow T(t) = \exp(-\int_{t_0}^{t} \sigma(\mathbf{r}(t))dt)$$

- higher density σ, lower probability that ray goes through



$$C(ray) = \int T(t)\mathbf{c}(\mathbf{r}(t), \mathbf{d})\sigma(\mathbf{r}(t))dt$$

How to learn something in it ??

1. Approximate integral by sum over points on the ray. The points will be sampled according to some distribution.

$$C(ray) \approx \sum_i T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

$$T_i = \exp(\sum_{k=1}^{i-1} -\sigma_k \delta_k)$$

2. This is a differentiable function of $\sigma_i, \mathbf{c}_i$.

1. Approximate integral by sum over points on the ray. The points will be sampled according to some distribution.

$$C(ray) \approx \sum_i T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

$$T_i = \exp(\sum_{k=1}^{i-1} -\sigma_k \delta_k)$$

3. Train a neural network to predict $\mathbf{c}(\mathbf{r}, \mathbf{d}), \sigma(\mathbf{r})$ at every point $\mathbf{r}$ and every direction $\mathbf{d}$

in practice: train a neural network to predict $\mathbf{c}(\mathbf{r}, \mathbf{d}), \sigma(\mathbf{r})$ on 5-d input $\mathbf{r} = \mathbf{u}(\mathbf{x}, \mathbf{d})$ (position, viewing direction)

What loss ?

$$L = \sum_{\mathbf{r} \in \mathcal{R}} \| \widehat{C}_c(\mathbf{r}) - C(\mathbf{r}) \|_2^2 + \| \widehat{C}_f(\mathbf{r}) - C(\mathbf{r}) \|_2^2$$

where $\widehat{C}_c$ is from a coarse prediction network and $\widehat{C}_f$ is from a fine prediction network

What models - coarse and fine ?

$$C(ray) \approx \sum_i T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i))$$

$$T_i = \exp(\sum_{k=1}^{i-1} -\sigma_k \delta_k)$$

What models - coarse and fine ?

- 8 Layer MLP (= fully connected layers): 5d input, output is 3+1 dims
- output $\widehat{C}_f(\mathbf{r})$ of the coarse network is used to sample points for the fine network. How ?

$$\widehat{C}_f(\mathbf{r}) = \sum_i T_i \mathbf{c}_i (1 - \exp(-\sigma_i \delta_i)) =: \sum_i \mathbf{c}_i w_i$$

Normalize all weights w_i for a given ray. This gives a piece-wise constant distribution of intervals along the ray. Then use the normalized weights $\hat{w}_i$ to sample points $\mathbf{r_k}$ along the same ray for the fine network

They use some important tricks

- position encoding using a set of sine-cosine waves with increasing frequencies.

$$\gamma(p) = \Big(\sin(2^0\pi p), \cos(2^0\pi p), \sin(2^1\pi p), \cos(2^1\pi p), \\ \ldots, \sin(2^{L-1}\pi p), \cos(2^{L-1}\pi p) \Big)$$

  represent a point by its value over a set of waves. Authors: higher dim representation, allows better to represent higher frequency functions as it is an explicit mapping of the position in a set of frequencies
- the positional encoding is similar to the one in transformers!
- sampling of points along rays: not uniform, but randomly from intervals

Fig4 in the paper ...

- big idea: model color along a ray as a differentiable function. train some internal function to it.
- trained with images of one single scene
- quality and eff res bounded by number of images (see paper Table 2 for an ablation study)
- allows to sample at any resolution, at any view
- is an interpolation method (as machine learning always does!!). Thus it invents content, but statistics learned from one single scene.

- disadvantage: very slow (18 hours per scene to train?)
- papers which speed it up:
  10x faster, better
  https://arxiv.org/abs/2007.11571
  speed vs quality tradeoff, FPS $\gg$ 1:
  https://arxiv.org/abs/2103.10380

https://arxiv.org/abs/2007.11571

- space can be subdivided regularly into cubes (voxels)


- given an initial density estimate $\sigma(\cdot)$ over a set of cubes
- continue to estimate only on those subset of cubes with sufficiently large density $\sigma(\cdot)$
- split cubes and refine further
- focus sampling only on spaces

https://arxiv.org/abs/2007.11571

- see Fig 2 in the paper for the quick idea: do a hierarchical voxel tree to cover the non-empty space with voxels. Have one neural net per voxel, but with shared parameters across all voxels. – see also sec A.2
- why voxels ? 1. can be used efficiently in hierarchical tree structures. 2. intersection of rays to voxels in such hierarchical trees can be computed fast.
- prediction network a single MLP: Fig 9.

comes with some new tricks

- making the voxels smaller: from time to time divide one voxel into 8, then prune those voxels of the 8 smaller ones with too low density σ over sampled points within

$$\text{prune vertex if } \min_{s\sim vertex} (0,1] \ni \exp(-\sigma(g(s))) > \gamma$$
$$\Leftrightarrow \sigma(g(s)) < \exp(-\gamma)$$

- use a more complex representation $g(p)$ of a point p along a ray: take the feature vectors of the 8 vertices of a voxel $\widetilde{g}(p_1), \ldots, \widetilde{g}(p_8)$, get $\widetilde{g}(p)$ as trilinear interpolation of their values: $\chi(\widetilde{g}(p_1), \ldots, \widetilde{g}(p_8))$, then apply positional encoding on top of that.
- the feature vectors of the 8 vertices is a generic 32-dim embedding of the position.

Where to extend this to ?

- other input modalities, eg. depth-only data or RGB-D
- deformations for shape editing while adapting existing texture to fit the edited shape https://arxiv.org/abs/2011.13650
- put the human in the loop partially to save time in the end

Many other settings. Example:

- got a large number of classes (e.g. 1000), but each class has maybe only 20 labeled samples? Example: tagged data ... consider few-shot learning.
  - train a relative prediction: query sample is most similar to which of the support sample sets?
  - train a model which randomly drawn sets of classes
  - learn a discriminative, class-agnostic similarity instead of a fixed-classes classifier
  - For a conceptually easy paper see https://arxiv.org/abs/1703.05175
- semi-supervised learning: make use of unlabeled data together with labeled data
- be distrustive of results demonstrated on MNIST/Fashion-MNIST/CIFAR-10 and other low-res,low variation datasets.

Questions?!