強力なNormalization手法、GauGAN (SPADE) を読む

MokkeMeguru

<2019-04-24 Wed>

Contents

1 導入

GauGAN [1] というのは 2019 年画像生成系の分野を賑わせた新たな Normalization (正規化) 手法についての論文で、NVIDIA の研究成果になります。website [2] でそのデモを体験することが出来、日本でも深層学習をやっている人たちが盛り上がっていたイメージです。
あと 実装が **PyTorch** なので 大変読みやすく、バグがないです。

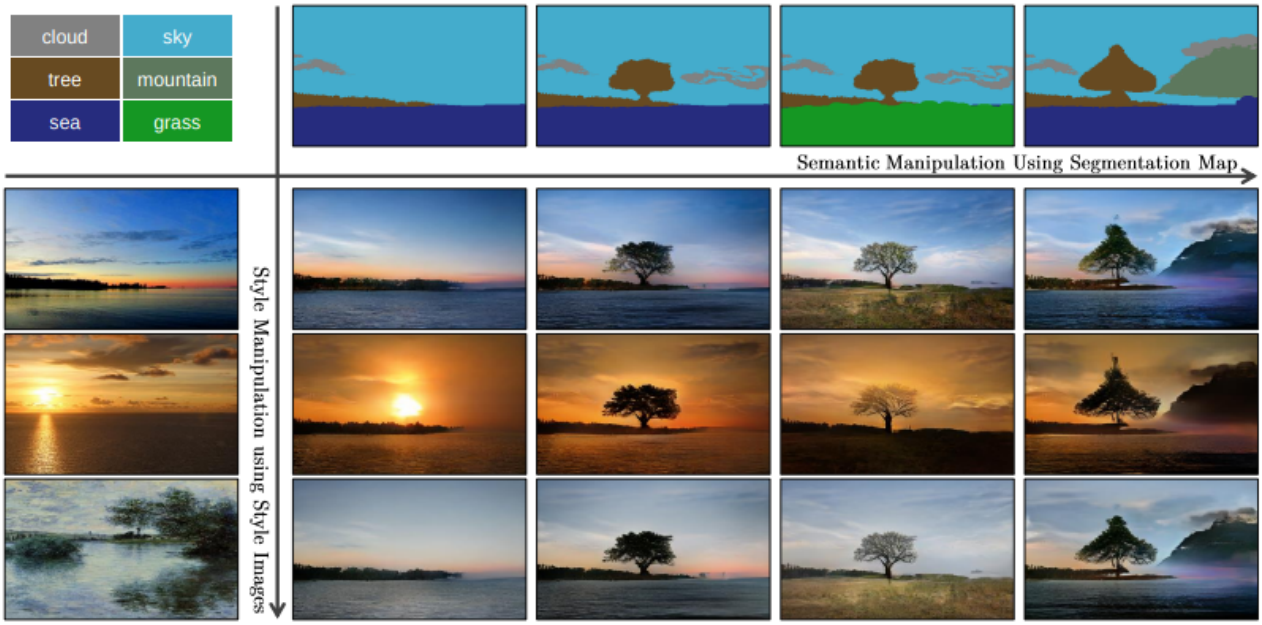


Figure 1: Our model allows user control over both semantic and style as synthesizing an image. The semantic (e.g., the existence of a tree) is controlled via a label map (the top row), while the style is controlled via the reference style image (the leftmost column). Please visit our website for interactive image synthesis demos.

[1]: Semantic Image Synthesis with Spatially-Adaptive Normalization

[2]: Github

2 GauGAN の取り組む問題

GauGAN が取り組むタスクは Semantic Image Synthesis というものです。こは簡単に言ってしまうと こんな感じの画像を作りたい というイメージから 写真のような画像 を生成することを指し、パッと Web の背景画像を作りたいとか、ゲームの画像を作りたいとかいった場面で役に立ちます。

本研究の Semantice Image Synthesis において、入力は概形を描いたセグメンテーション画像 (色分けした画像) であり、出力は写真のような画像、となっています。関連研究としては、例えば入力が単語であったりする場合があります。

本研究で用いたデータセットの一つとして COCO-Stuff dataset があります。これは写真データと、そのセグメンテーション画像がペアになっています。本来的には COCO-Stuff は写真データ -> セグメンテーション画像、というふうな学習を目的としたデータセットなのですが、本研究ではその逆を行なっている点に注意して下さい。

3 Normalization とは何か

Normalization (正規化) とは雑に言うと $\boldsymbol{x}$ の値域を $\boldsymbol{x'}$ へ変換することを指します。 変換前後の次元的に言うと、 $f : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{H\times W\times C}$ です。
画像データ $\mathbb{R}^{N\times H\times W\times C}$ (N : バッチサイズ、 H : 画像の高さ、 W : 画像の幅、 C : 画像のチャンネル) を例にすると、次のようなものが例として挙げることが出来ます。

- BatchNormalization

  C について $\mu = 0\,, \sigma = 1$ への正規化をします。このとき平均・分散の求め方は $\mu_B = \dfrac{1}{NHW}\Sigma X_{n,h,w}$ $\sigma_B^2 = \dfrac{1}{NHW}\Sigma(X_{n,h,w} - \bar{X})^2 \in \mathbb{R}^C$ のようになります。(バッチサイズ N が存在していることに注意)

  正規化の式について簡単に取り上げると、一枚画像 x の C 軸について $\hat{x_i} = \dfrac{1}{\sigma_B}(x_i - \mu_B), x_i \in \mathbb{R}^C$ となります。(但し実装や性能向上のために、実際はもっと複雑な式が用いられます。)

- InstanceNormalization

  C について $\mu = 0\,, \sigma = 1$ への正規化をします。但しこのときの平均・分散の求め方は $\mu_I = \dfrac{1}{HW}\Sigma x_{h,w}$ $\sigma_I^2 = \dfrac{1}{HW}\Sigma(x_{h,w} - \bar{x})^2$ のようになります。Batch Normalization が画像データ群全体で C が $\mu = 0, \sigma^2 = 1$ となるようにしているのに対して Instance Normalization が 一枚の画像について $\mu = 0, \sigma^2 = 1$ に正規化している点が主な違いです。

- ActNorm

  ActNorm は Glow[3] で提案された Normalization で、これは まず 初期バッチ $X_B = x_1, x_2, ..., x_n$ について、次の式に従うようにして C について $\mu = 0\,, \sigma = 1$ への正規化を行います。

  $$\mu_{init} = \frac{1}{NHW}\Sigma X_{n,h,w}, \sigma_{init}^2 = \frac{1}{NHW}\Sigma(X_{n,h,w} - \bar{X})^2 \in \mathbb{R}^C$$

  ActNorm の $\mu_{init}, \sigma_{init}^2$ は、初期バッチで初期化された後、特に C について正則化をするという制約をかけられず、ただの訓練パラメータとして用いられます。つまりこの Normalization は $\mu = 0\,, \sigma = 1$ への変換ではないという点に注意して下さい。

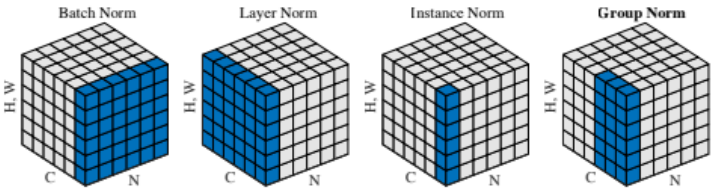


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

正規化について直感的な図を Group Normalization [4] から引用しましょう。例えば BatchNormalization の場合、青い部分がシュッとなって $\mu_{B_i}, \sigma^2_{B_i}$ となります。これが C 個出来るので、 $\mu_B \in \mathbb{R}^C$ です。Instance Normalization は同様に考えると $\mathbb{R}^{CB}$ ですが、バッチ軸についてはまとめ上げられるので $\mathbb{R}^C$ となります。

[3]: Glow: Generative Flow with Invertible 1x1 Convolutions

[4]: Group Normalization

4 SPADE (Spatially-Adaptive (DE)normalization)

GauGAN では Normalization について新たな手法 SPADE (Spatially-Adaptive (DE)normalization) を提案しました。
特にこの Normalization は Conditional Normalization の一種としてみなされます。Conditional Normalization の関連手法としては Conditional Batch Normalization や AdaIN を挙げることが出来ます。これらは外部のデータを用いた手法であり、この手順は (1) BatchNormalization などの手法で 平均 0, 分散 1 へ正規化を行い x を獲得 、(2) 外部のデータを用いてアフィン変換 $ax + b$ を行う、というものになっています。先行研究でのアフィン変換のパラメータ a, b はベクトルであったりスカラーであったりいろいろですが、SPADE ではここにセグメンテーション画像を用いました。

4.1 SPADE のコンセプト

> their normalization layers tend to “ wash away ” information contained in the input semantic masks.
> — quoted from page 2 line 1

SPADE のコンセプトは、 **BatchNormalization らが "wash away(洗い流す)" する内容を復元する** ということです。そして彼らは復元する情報源として **セグメンテーション画像** を使いました。

直感的な説明をしましょう。例えばセグメンテーション画像からハワイの海岸の画像を生成しようとするとき、海の部分と砂浜の部分を同じように平均 0, 分散 1 にされてしまうと (ここで Batch Normalization が C について正規化されているという点を思い出してみましょう) 情報落ちてない？となるわけです。ここで Conditional Normalization をして情報補完をしてみよう→どうやって補完する？→そういえばセグメンテーション画像なんてものがあるな、みたいな感じに発想を進めていくことが出来ます (いや彼らがそう思っているかは知りませんが)。

下の画像が SPADE のレイヤーの概要です。確かに (1) BatchNormalization (2) セグメンテーション画像から γ, β を用いてアフィン変換、をしていますね。

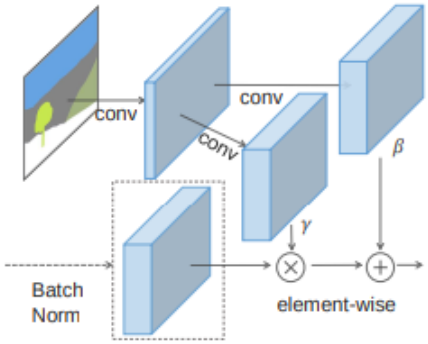


Figure 2: In the SPADE, the mask is first projected onto an embedding space and then convolved to produce the modulation parameters γ and β. Unlike prior conditional normalization methods, γ and β are not vectors, but tensors with spatial dimensions. The produced γ and β are multiplied and added to the normalized activation element-wise.

4.2 SPADE の細かい話

次に細かい話として SPADE の入力と出力を示しましょう。前提として、SPADE は BatchNormalization に合わせて モデルの複数ヶ所に適用されるので、それぞれの SPADE を i で区別します。

SPADE の入力はセグメンテーション画像で、セグメンテーションラベルは one-hot vector になっており 、つまりこれがいわゆるセグメンテーション画像の C になります。これが $H^i \times W^i$ 個あるので、結局 SPADE の入力は　$m \in (\mathbb{L}^{H^i \times W^i} = \mathbb{R}^{H^i \times W^i \times C^{i'}})$ となります。 β^i, γ^i をそれぞれ後述する式で求めます。
それはそれとして、BatchNormalization Layer から μ^i, σ^i をそれぞれ用意しておきます。

BatchNormalization Layer に入ってくる Tensor h^i と $\mu^i, \sigma^i, \beta^i, \gamma^i$ から h^i -> [BN -> SPADE] -> $h^{i'}$ は 次の式で表すことが出来ます。

$$h^{i'} = \gamma^i \frac{h^i - \mu^i}{\sigma^i} + \beta^i$$

ここであれ？って思える人は深層学習の実装に向いています。そう、この式ですと次元数があんまりよくわかんないです。なので、詳しく次元数を書いてみます。

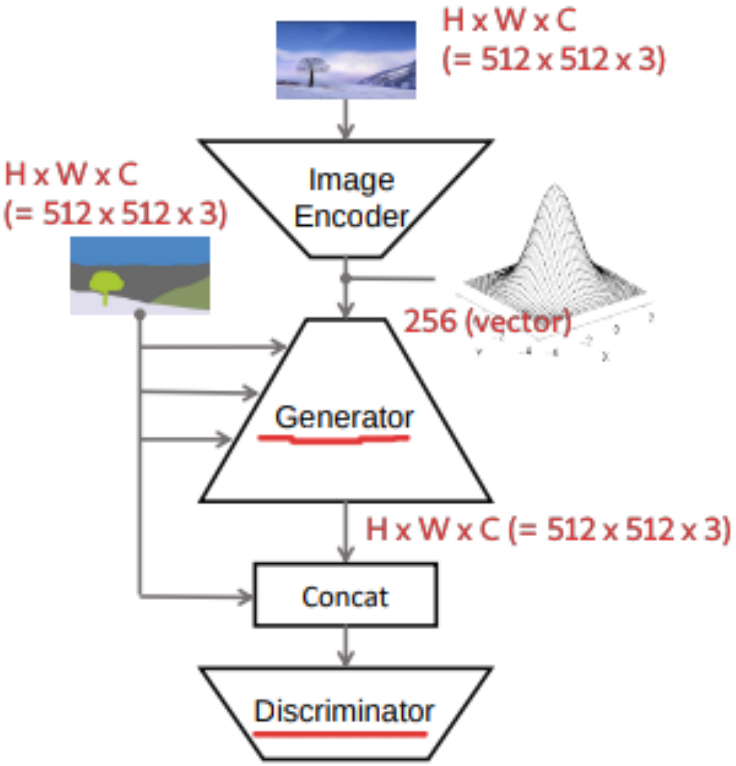
$$
\begin{aligned}
, where & \\
h^i &\in \mathbb{R}^{N \times H^i \times W^i \times C^i} \\
\mu^i, \sigma^i &\in \mathbb{R}^{C^i} \\
\gamma^i, \beta^i &\in \mathbb{R}^{H^i \times W^i \times C^i}
\end{aligned}
$$

つまり Batch Normalization が C^i について正規化が行われているのに対して、SPADE は H^i, W^i, C^i について正規化が行われています。

雑な発想ですと、セグメンテーション画像をそのままペッと $\frac{h^i - \mu^i}{\sigma^i}$ へ貼っているイメージでしょうか。こうすることで Batch Normalization で落としてしまったであろう情報を復元できるというわけです。

5　モデルの全容

本研究で特にセグメンテーション画像を使っている部分は一般的な GANs でいう Generator と Discriminator の部分なので、これらについて概形→詳細、と詰めて見てみましょう。



5.1　Generator

最終的な出力が $N \times H \times W \times C = 256 \times 512 \times 512 \times 3$ の画像となる Generator を下に引用します。

`SPADE ResBlk(K)` は入力を $N \times H^i \times W^i \times C^i$ の入力を受け取り $N \times H^i \times W^i \times K$ を出力します。そして `Upsample(2)` は $N \times H^i \times W^i \times K$ を入力として $N \times 2H^i \times 2W^i \times K$ を出力とします。例として、上から 1 つ目の `SPADE ResBlk(1024)`, `Upsample(2)` は、 $N \times 4 \times 4 \times 1024$ を入力として $N \times 8 \times 8 \times 1024$ を出力とします。(以降バッチサイズ N を省略)

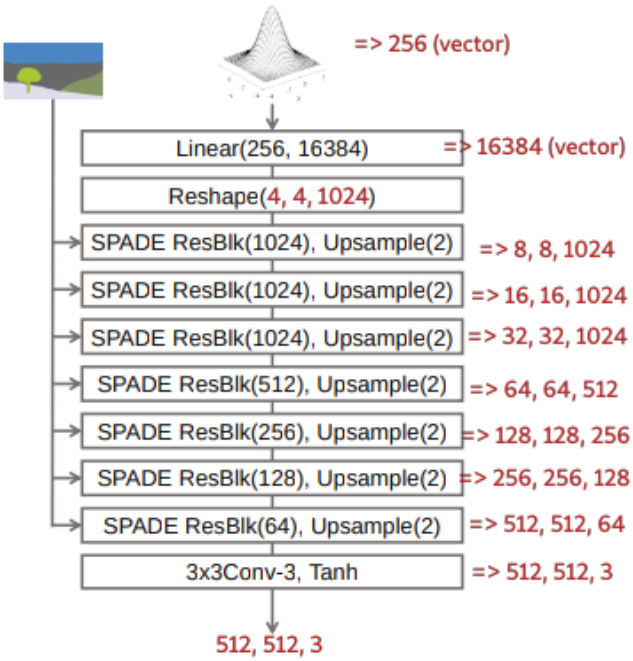


Figure 12: SPADE Generator. Different from prior image generators [22, 48], the semantic segmentation mask is passed to the generator through the proposed SPADE ResBlks in Figure 11.

5.1.1 SPADE ResBlk

SPADE ResBlk については、下に引用される図で説明します。ResNet を元にしていますが、入力と出力の次元数が異なっている点に注意して下さい。簡単な構造は ResNet と変わっていませんが、正規化層がそのまま SPADE に置き換わっており、SPADE のための入力であるセグメンテーション画像が外部から与えられていることがわかると思います。

`SxSConv-K` は カーネルサイズ S フィルタサイズ K の畳み込みを示しており、$H \times W \times C$ の Tensor を入力として $H \times W \times K$ の Tensor を出力とします。

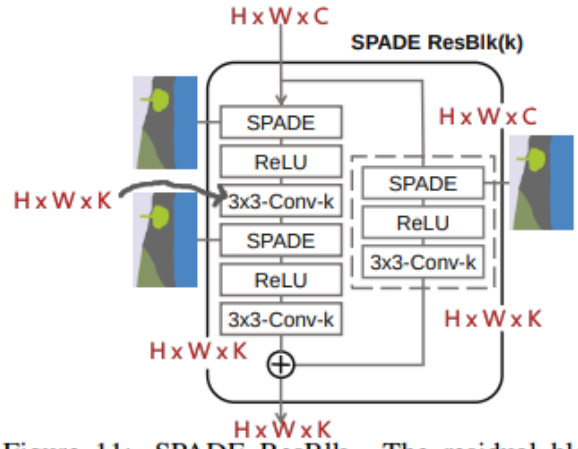


Figure 11: SPADE ResBlk. The residual block design largely follows that in Mescheder *et al.* [36] and Miyato *et al.* [39]. We note that for the case that the number of channels before and after the residual block is different, the skip connection is also learned (dashed box in the figure).

convolutional layers of the discriminator. The details of the discriminator architecture is shown in Figure 13.

5.1.2 SPADE

SPADE そのものについては、下に引用される図で説明します。ここで注意してほしいのは、SPADE そのものは **入力と出力で次元数が変わらない＝通常の Normalization と同じ** という点です。すると問題になるのが、**セグメンテーション画像をどう扱うか** です。

セグメンテーション画像は一枚の $H' \times W' \times C' = 512 \times 512 \times C'$ で共通であり、これは SPADE に入ってくるデータの次元数 $H^i \times W^i \times C^i$ とは異なります。C については畳み込みレイヤーでどうとでもなるのですが (3x3-Conv-k で変えられる) H, W についてはそうはいきません。なので SPADE では Resize を使って H, W の変換を行います。

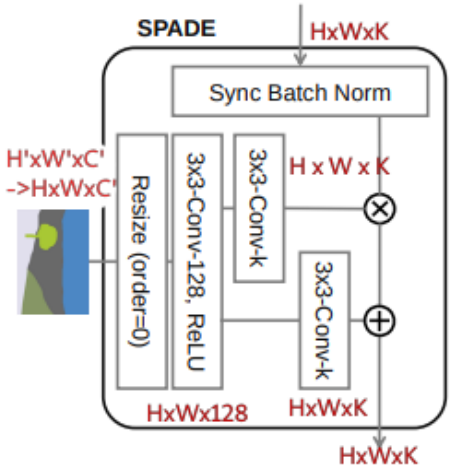


Figure 10: SPADE Design. The term 3x3-Conv-k denotes a 3-by-3 convolutional layer with k convolutional filters. The segmentation map is resized to match the resolution of the corresponding feature map using nearest-neighbor downsampling.

5.2 Discriminator

Discriminator は Pix2PixHD と同等の機能を持っています。この Discriminator の図がわかりにくいので簡単に Pix2PixHD から引用を行いながら説明を行います (というか既存の解説記事はこのあたりの話雑すぎませんか？)。

Discriminator への入力は (1) セグメンテーション画像と実際の画像のペア (真) (2) セグメンテーション画像と生成画像のペア (偽) に なっており、これらを区別することが Discriminator の役割になっています。

次に Discriminator の構造ですが、これは multi-scale Discriminator となっており、通常の GANs の Discriminator が複数入っています。

> To address the issue, we propose using multi-scale discriminators. We use 3 discriminators that have an identical network structure but operate at different image scales.
> We will refer to the discriminators as D1, D2 and D3. Specifically, we downsample the real and synthesized highresolution images by a factor of 2 and 4 to create an image pyramid of 3 scales. The discriminators D1, D2 and D3 are
> then trained to differentiate real and synthesized images at the 3 different scales, respectively.
> Although the discriminators have an identical architecture, the one that operates at the coarsest scale has the largest receptive field.
> It has a more global view of the image and can guide the generator to generate globally consistent images. On the other hand, the discriminator at the finest scale encourages the
> generator to produce finer details. This also makes training the coarse-to-fine generator easier, since extending a lowresolution model to a higher resolution only requires adding
> a discriminator at the finest level, rather than retraining from scratch.
> Without the multi-scale discriminators, we observe that many repeated patterns often appear in the generated images.
>
> — quoted from Pix2PixHD page 4 second column

つまり Concat した画像を (1) そのまま (2) 半分のサイズに downsample (実装上は average pooling (カーネルサイズ = 3)) (3) 更に down sample した画像 A, B, C についてそれぞれ Discriminator D1, D2, D3 を適用するよ、と言っているわけです。つまり 下の図は (1) についての構造を説明していて、 (2) については Concat の下に down sample が加わります。また **本論文で用いている Discriminator は D1 と D2 のみである点 に注意して下さい** 。

また 下の図の IN というのは Instance Normalization を示しています。つまり `Conv, IN, LReLU` というのは 畳み込みレイヤー → Instance Normalization → Leaky ReLU (活性化関数) という流れを示しています。更に本論文 (実装も) ではすべての Conv の手前に Spectral Normalization が用いられています。つまり Conv -> Spectral Norm -> Instance Norm -> LReLU という流れになります。(これは 実装を眺めている感想ですが、最後の Conv には Spectral Norm も Instance Norm も入っていません。)

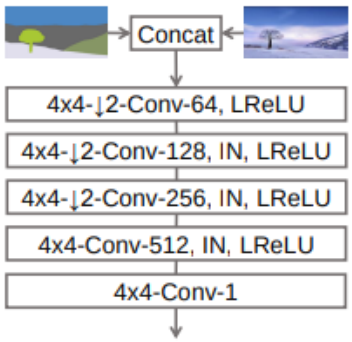
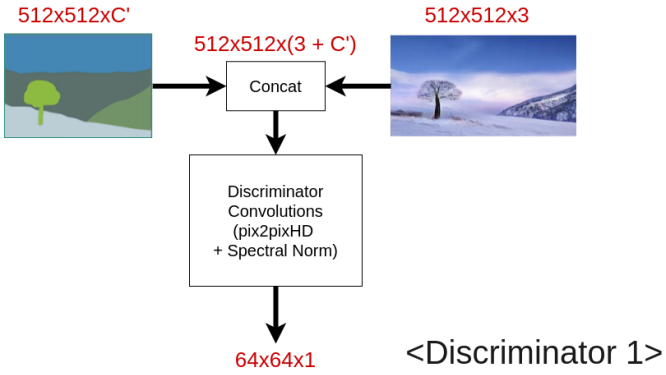
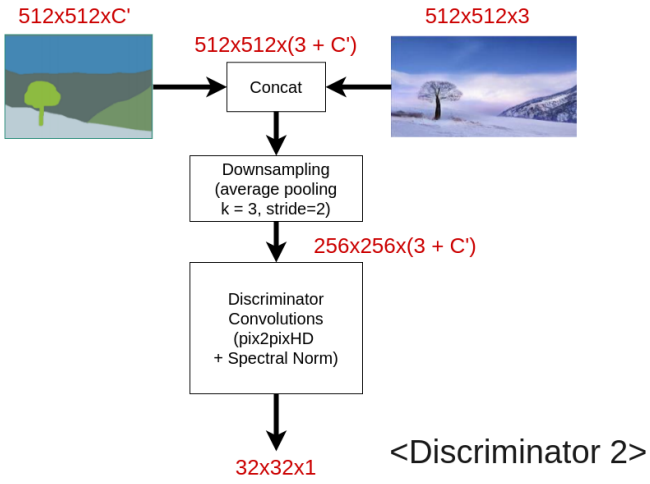


Figure 13: Our discriminator design largely follows that in the pix2pixHD [48]. It takes the concatenation the segmentation map and the image as input. It is based on the PatchGAN [22]. Hence, the last layer of the discriminator is a convolutional layer.





6 訓練・推論手法

We train the generator with the same multi-scale discriminator and loss function used in pix2pixHD [48] except that we replace the least squared loss term [34] with the hinge loss term [31,38,54].
— quoted from page 3 column 2

訓練は通常の GANs と同様に Discriminator と Generator の戦いによって行われます。損失関数を簡単に書くと次のようになります。

推論は正規分布からサンプルした 256 個の値をベクトルにしたものと、ユーザが描く or 与えられるセグメンテーション画像を用いて、写真のような画像を出力する、という仕組みになっています。このため、最初の方に図に登場した Encoder は不要になります。

1. Encoder の損失 (推論時に正規分布からサンプル出来るようにするため)

$$L_{KLD} = D_{KL}(q(z|x)||p(z))$$

2. GANs の損失 (一般的な GANs の損失)

$$\min_{G} \max_{D_1,D_2} \Sigma_{k=1,2} L_{GAN}(G, D_k)$$

但し本論文では Pix2PixHD とは違い $L_{GAN}(G, D)$ は Hinge 関数を用いた 次の式で表します。但し $L_{GAN}(\hat{G}, D)$ は Discriminator のパラメータ更新時、 $L_{GAN}(G, \hat{D})$ は Generator のパラメータ更新時の損失関数になります。Appendix で議論されていますが、Hinge 関数を用いたほうが評価が向上するようです。

$$\begin{aligned} L_{GAN}(\hat{G}, D) &= E_{x\sim q_{data}(x)}[min(0, -1 + D(x))] + E_{z\sim p(z)}[min(0, -1 - D(\hat{G}(z)))] \\ L_{GAN}(G, \hat{D}) &= -E_{z\sim p(z)}[\hat{D}(G(z))] \end{aligned}$$

ちなみに Pix2PixHD での $L_{GAN}(G, D)$ は次の式です。

$$L_{GAN}(G, D) = E_{x\sim q_{data}(x)}[logD(x)] + E_{z\sim p(z)}[log(1 - D(G(z)))]$$

3. Feature Matching Loss (Pix2PixHD で提案された損失)
つまり Discriminator に実際の画像を入れたときの各層の出力と、生成画像での出力を近づかせるための損失です。

$$L_{FM}(G, D_k) = E_{s,m}\Sigma_{i=1}^{T}\frac{1}{N_i}[||D_k^{(i)}(s, x) - D_k^{(i)}(s, G(s))||_1]$$

4. Perceptual Loss (Perceptual Losses for Real-Time Style Transfer and Super-Resolution より)
Pix2PixHD で追加されている (論文の 4. Results 10 行目あたりにしれっと書いてある)
クラス分類モデルの重みって画像のコンテンツを理解することが出来ているのでは？というモチベーションで、画像生成の際の教師のような形にして用いる。

$$\lambda\Sigma_{i=1}^{N}\frac{1}{M_i}[||F^{(i)}(x) - F^{(i)}(G(s))||_1]$$

7 実験

COCO-dataset などなどで訓練し、その評価を Pix2PixHD やいくつかのモデルで比較しました。またどの部分が強い影響を与えているのかを比較するために Pix2PixHD に提案手法のいくつかを組み込んだモデルで性能の向上度合いを比較しました。

評価手法は、mIoU(mean Intersection over-Union, 生成画像と実際の画像の領域的一致度。おそらく生成画像をセグメンテーションして、元のセグメンテーションとどのくらい一致しているのか、という測り方です) と accuracy(生成画像と実際の画像との差)、FID(生成画像群と元の画像群がどのくらい似ているのか) の 3 つです。多分 FID だけで良いのでは？という気もしますが結構いろいろな面から眺めていますね。

8　結果

Pix2PixHD よりも良い性能が達成できました。

	COCO-Stuff			ADE20K			ADE20K-outdoor			Cityscapes		
Method	mIoU	accu	FID	mIoU	accu	FID	mIoU	accu	FID	mIoU	accu	FID
CRN [6]	23.7	40.4	70.4	22.4	68.8	73.3	16.5	68.6	99.0	52.4	77.1	104.7
SIMS [43]	N/A	N/A	N/A	N/A	N/A	N/A	13.1	74.7	67.7	47.2	75.5	**49.7**
pix2pixHD [48]	14.6	45.8	111.5	20.3	69.2	81.8	17.4	71.6	97.8	58.3	81.4	95.0
Ours	**37.4**	**67.9**	**22.6**	**38.5**	**79.9**	**33.9**	**30.8**	**82.9**	**63.3**	**62.3**	**81.9**	71.8

Table 1: Our method outperforms the current leading methods in semantic segmentation (mIoU and accu) and FID [17] scores on all the benchmark datasets. For the mIoU and accu, higher is better. For the FID, lower is better.

またサンプリングベクトルとセグメンテーション画像から複数の画像を生成した場合の結果は次のようになります。

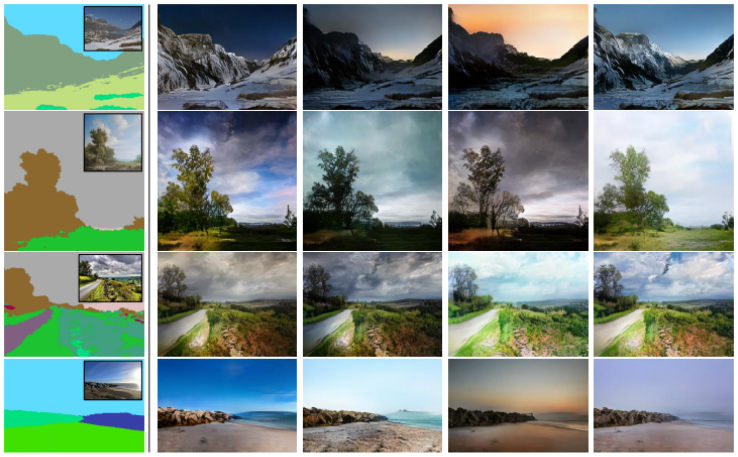


Figure 9: Our model attains multimodal synthesis capability when trained with the image encoder. During deployment, by using different random noise, our model synthesizes outputs with diverse appearances but all having the same semantic layouts depicted in the input mask. For reference, the ground truth image is shown inside the input segmentation mask.

9　論文のアブストラクトの和意訳

セグメンテーション画像を与え写真のような画像が生成するモデルで有効な、シンプルかつ強力なレイヤーである `SPADE` を提案するよ。以前の手法ではセグメンテーション画像をモデルに入れて出力に写真のような画像を取り出す、というものが一般的だったけど、これだと途中の層にある `Normalization` レイヤーがセグメンテーション画像の意味情報 `(Semantic information)` を”洗い流してしまう”傾向があるんだ。だから `SPADE` によって `Normalization` レイヤの出力を変換することを提案するよ。
複数のデータセットで実験をしたら、視覚的な良さや入力のセグメンテーションの反映度について先行研究に比べて良い性能を示していることがわかったよ。更にこの手法はセグメンテーションとスタイルのそれぞれをユーザが制御できるようになっているんだ。コードは `Github` にあるよ。

10　読んだ感想とか

Normalization が落としてしまう情報、というのに注目した面白い研究だと思います。そういえば ResNet も残差をくっつけているので落としてしまう情報、という観点では似ているのかな、と思いました。

この論文のすごいところは、NVIDIA のインターン生が First Author になっていたことだと思っています。画像生成の研究では概して大量の計算機が必要になるので、彼のムーブは最適解ではありますが、よく通ったなぁっていう顔をしています。

あとは個人的な直感ですが、Multi-Scale が最近のはやりなのかなぁと思っています。僕は GANs よりも VAE/Flow 系が好みなんですが、最近の VQ-VAE2 なんかも確率空間への写像を Multi-Scale にしていましたし、 Flow-based Model の新しめの手法である FFJORD や Glow なんかも Multi-Scale Architecture を用いています。もっとふわっとした直感としては、VAE の正規分布に従わせる項って KL-collapse やらなんやらで最近 (自然言語処理界隈系から) 良い話を聞いていないなぁと思いました。

5.3 Single-scale vs. Multi-scale FFJORD

Crucial to the scalability of Real NVP and Glow is the multiscale architecture originally proposed in Dinh et al. (2017). We compare an single-scale encoder-decoder style FFJORD with a multiscale FFJORD on the MNIST dataset where both models have a comparable number of parameters and plot the total NFE–in both forward and backward passes–against the loss achieved in Figure 6. We find that while the single-scale model uses approximately one half as many function evaluations as the multiscale model, it is not able to achieve the same performance as the multiscale model.

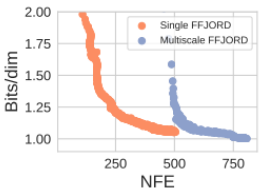


Figure 6: For image data, multiscale architectures require the ODE solver to use a greater number of function evaluations (NFE), but these models achieve better performance.

6 Scope and Limitations