Flow-based Model with Oxford flower 102 dataset

MokkeMeguru

<2020-05-29 Fri> May 29, 2020

Contents

1 Flow-based Model を使って、カラー画像を学習してみる

Flow-based Model は逆関数のある深層学習モデル、というちょっと変わったモデルです。
このモデルは GANs や VAE を始めとする生成モデルの仲間に含められることが多く、実際に画像生成分野を中心に研究が進められています。

今回は Flow-based Model を用いて Oxford flower102 dataset という画像のデータセットを学習、何らかの画像を生成してみます。

2 生成モデルと Flow-based Model

簡単に、この系統のモデルで出てくる要素は、 **画像** , **モデル** , **潜在表現** の 3 つです。

- 画像
  画像とは実画像であったり生成画像であったり、人間が視認できるアレです。

- モデル
  何かから画像を生成したり、画像の意味を解釈して行列やベクトルといったもの (すぐ後に出てくる潜在表現) にしたりする機能を持った、 **箱** です。
  小麦を突っ込んだらパンが出てくるみたいな気持ちで見て下さい。

- 潜在表現
  ベクトルや行列の形で表される、人間が視認してもちょっとわからないけど、数学や統計の目で見ると画像などの性質を読み取ることの出来るものです。潜在表現は例えば勾配法などの数学・統計的手法を用いて操作することが容易になっています。
  Flow-based Model や VAE は画像から潜在表現を作れます。そのため、画像から潜在表現を獲得、潜在表現を上手いこと調節して画像を再生成することで、イケメンが美少女になったりします。

2.1 Flow-based Model の立ち位置

Flow-based Model は VAE や GANs と比べて、画像から潜在表現をきっちり作れることが、特徴の 1 つです。この性質から、ちゃんとデータを与えて学習を行うと、そこそこに良い画像が生成できます。とはいえ GANs の方がモリモリ研究が進んでいるので、今の所 GANs のフィールドではそんなに勝てていません 。

	画像から潜在表現を作れるか	モデルの大きさ (パラメータ数)	訓練時間
VAEs	近似的に作れる	そこそこにでかい	短~並
GANs	作れない	でかい	長・安定しない
Flow-based	作れる	小さい	短

3 今回作る Flow-based Model の概要

3.1 まずは式で解釈する

x を画像、 f をモデル、 z を潜在表現とします。すると、Flow-based Model は次の式で表すことが出来ます。めっちゃシンプル。

$$z = f(x) \text{ (in training)} \tag{1}$$
$$x = f^{-1}(z) \text{ (in generating)} \tag{2}$$

次に目的関数ですが、実際の画像 x の確率 p_X を高くする (つまり実際の画像が沢山生成できるようにする)、という意味を持っています。 p_X はそのまま計算させることが難しいので、 モデル f を使って $p_Z(z = f(x))$ とします。するとここの過程は変数変換 (ref. 極座標変換) なので、ヤコビアン J_f をくっつける必要があります。そのまま対数化すると下の式になります。

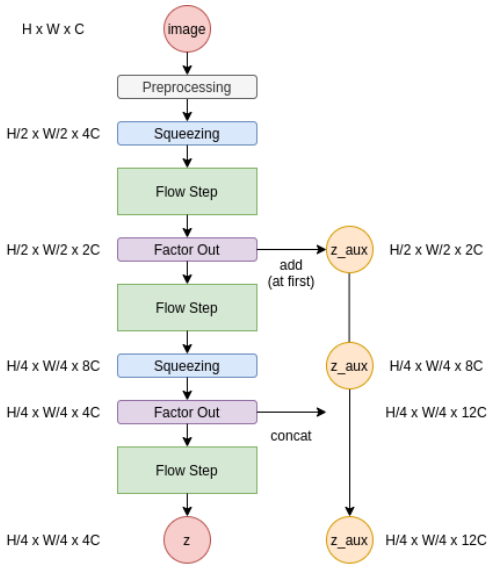
$$p_X(x) = p_Z(z = f(x))|J_f| \tag{3}$$
$$logp_X(x) = \log(p_Z(z = f(x))) + \log|J_f| \tag{4}$$

3.2 モデルの概要図

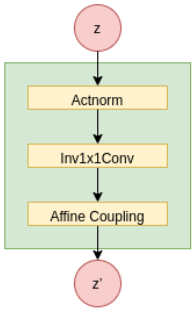
モデルはちょっと中身が複雑になっています。この Architecture が今の所 Flow-based Model の基本的な型になっています (ref. Glow, RealNVP, Flow++ etc.). H, W, C はそれぞれ処理している行列の次元数を示しており、例えば image では H (height) x W (width) x C (channel) になっています。

この図を見ると、image から z へ一直線に流れる線と、 z_{aux} というよくわからないものへ流れる線がわかります。 z_{aux} というのは、 image から z へ処理していく中で、「もう z とみなしちゃっていいでしょ」枠の行列です。これを設けることで、計算コストを抑えることが出来る他、 image から z へ流れの中で、画像の局所的な特徴を掴む部分 → 画像の全体的な特徴を掴む部分 と処理が行くようになります。

Squeezing やら Actnorm やら Inv1x1Conv, Affine Coupling あたりの話は、この辺りの要約 か ライブラリの実装ドキュメント を見て下さい。



< Simple Flow Base Model >



< Flow Step >
(Actnorm + Inv1x1Conv + AffineCoupling)

4 Oxford flower102 データセットについて

Oxford flower102 データセットは flower の名前の通り、花の画像のデータセットです。内訳は下のテーブルになっていて、画像サイズは統一されていません。実験では計算機的問題から 48 x 48 にしています。データの前処理なんかは こちら を見て下さい (画像の data augumentation や前処理は素人なので、追加したほうが良い内容など教えて頂ければ幸いです)。このデータセットは元々クラス分類のためのデータセットなので近年の **でかいデータセット** + **潤沢な計算資源** みたいなケースに比べるとかなり厳しい問題設定になっています。

	枚数
train	1020
valid	1020
test	6149

5 実際にコードを動かす

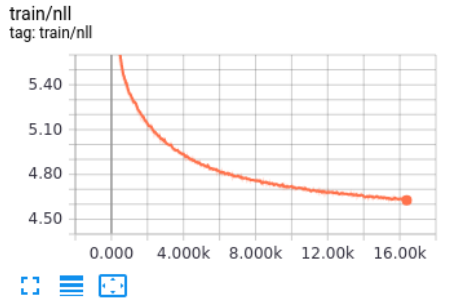
コードそのものは こちら にあります。実行はこんな感じで出来ます。
`python glow.py` 以降はオプションで、Facebook の hydra というツールでパースさせています。より詳しいオプションが見たい場合は、`python glow.py --help` としてみて下さい。

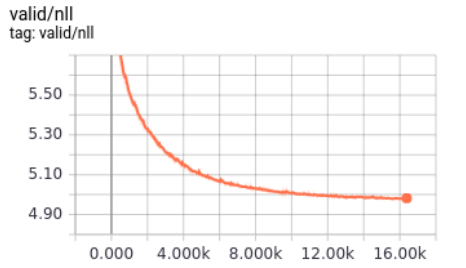
```
docker-compose build
docker-compose up -d
docker-compose exec tfgenzoo /bin/bash
tmux
python glow.py dataset=flower model.K=24 batch_sizes.0=32 batch_sizes.1=128 batch_sizes.2=12
```

尚私の行なった GPU 環境は、Quadro P6000 x 1 (4 hour) でした。GCE の無料枠か、Colab (頑張れば) で動くと思います。

6 結果の観察

outputs/<date>/<time>/ 以下に生成されたモデル、ログが出力されます。Tensorboard を用いているので、こいつを使って可視化します。
まずは損失関数から。この損失関数は、先述の通り、実際の画像に高い確率を割り当てられたか、の結果になっており、上手いこと変形して最小化の形になっています。形を見ると大分綺麗に損失が下がっていることがわかると思います。尚、損失はまだまだ下がりそうなのに訓練していないのは、計算資源的事情です。

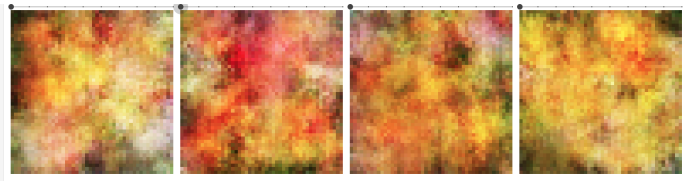




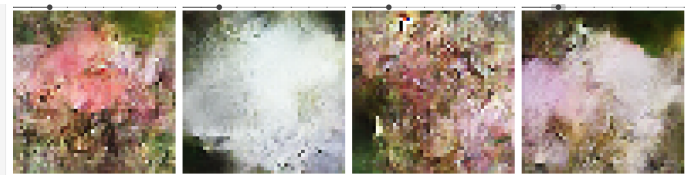
6.1　生成画像の観察

次に生成画像を数 epoch ごとに見てみてます。

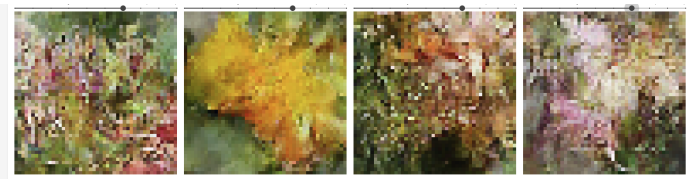
- 25 epoch
  まだテクスチャ的な画像しか見えません。



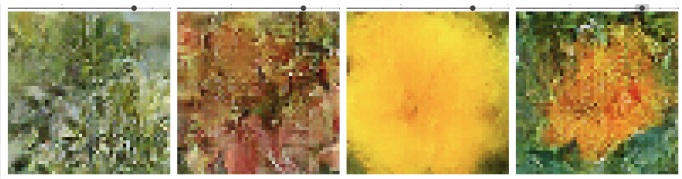
- 200 epoch
  なんとなく周りは葉っぱにしたい気持ちが見えてきます。



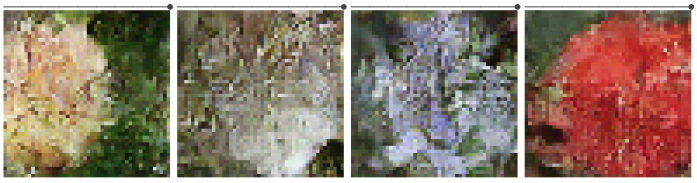
- 300 epoch
  2 番目あたりは大分それっぽい画像 (解像度がアレ) になっています。



- 400 epoch
  花弁と雌しべの領域的違いが見えてきました。



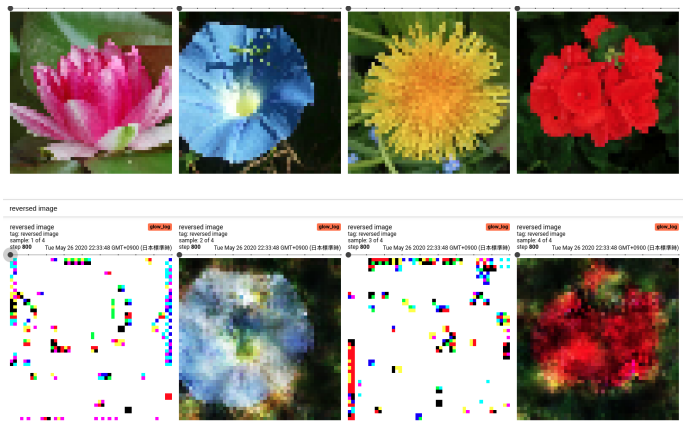
- 512 epoch
  葉っぱと花の領域がかなりはっきり見分けがつくようになっています。が花の中の方はまだまだです。



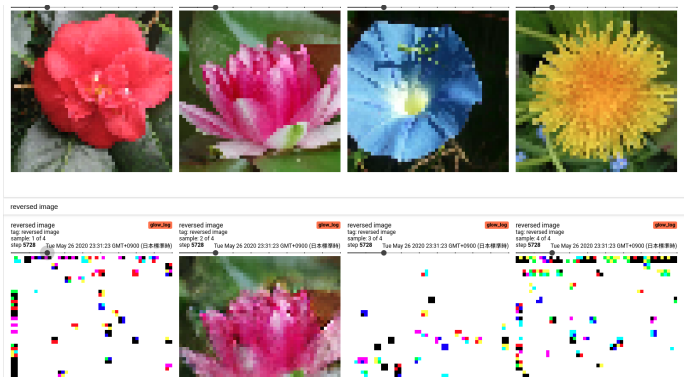
6.2 再構成画像の観察

そしてこちらは GANs と比較できないのであんまり注目されない部分ですが、再構成画像 (画像を潜在表現にして、もう一度画像にする) の結果も見てみます。

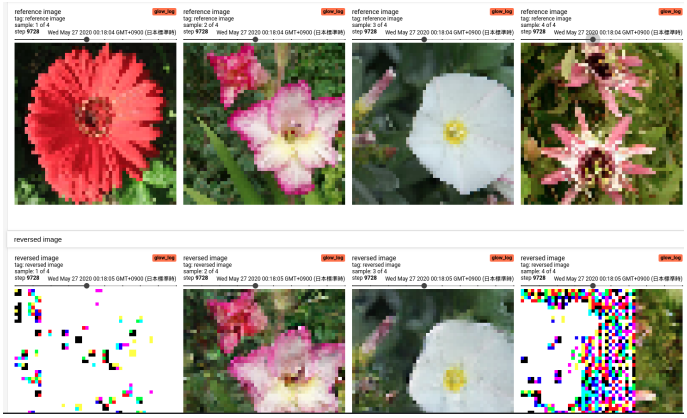
- 25 epoch
  驚くべきことに学習初期は逆関数が正しく求まらず、きちんとした再構成画像が出てきません。



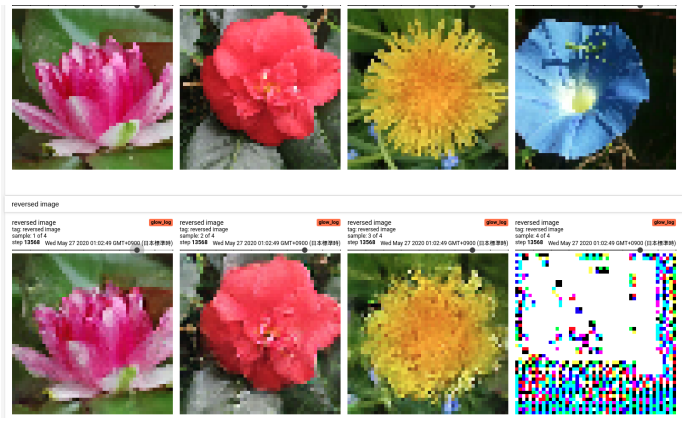
- 200 epoch
  rgb = [0, 0, 0] の飛んだ値が出てきています。rgb の範囲的に [0, 1] ([0, 255]) で値をクリッピングしているので、つまり白い部分は 値が 0 以下になってしまっていることを示しています。



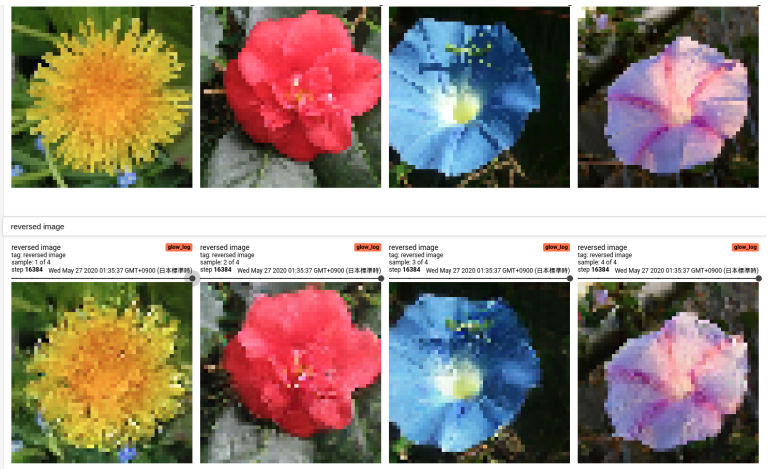
- 300 epoch
  面白いことに、4 枚目の画像は一部だけ正しい再構成画像が出来ているという不思議なことになっています。これは VAE や GANs などに比べてもかなり異質な特徴であると言えます。



- 400 epoch
  400 epoch になってもこの特徴は残っています。但し、上手く行っているものはきちんと再構成が出来ていることを確認できます。



- 512 epoch
  この当たりになると、流石に飛んだ値は出なくなっているようです。再構成としても完全一致ではないですが、良いものができているような気がします。



7　考察

7.1　生成力が弱くない？

計算資源的問題とデータ的問題を抜きにして話すならば、Data Augumentation の知見が不足していることや、内部で用いている AffineCoupling というレイヤーについて問題があると考えています。AffineCoupling レイヤーは Flow++ という最近出てきたネットワークでより良いバージョンが提案されており、ここを更新することでもう少し生成力を高められると思います。(とはいえこのレイヤーは実装がものすごく大変です。具体的には深層学習モデル中に、累積分布関数を導

入します)

7.2 生成画像がちょっとボケてない？

生成画像がボケる理由は、MLE (maximum likelihood estimation) という最適化手法を取っているためです。これは GANs の Adversarial Training と比較されることがしばしばあり、前者は画像がボケやすくなることが、後者は mode collapse という生成画像が似たようなものしか出てこないことが指摘されています。

直感的に MLE の画像がボケる理由としては、なんとなく実際の画像っぽい画像にも高い確率を与えてしまうことにあります。潜在表現は似た画像は同じような潜在表現を持つため、実際の画像に高い確率を与えてしまうと、自然とそれっぽい (ちょっとボケた) 画像も高い確率が与えられてしまいます。

7.3 再構成画像おかしくない？

再構成画像、おかしいですねぇ。ただ丸め誤差や、内部で用いている対数関数や指数関数などが悪さをしているのではないかと考えています。
また、Flow-based Model は画像の rgb が [0, 255] の離散値を ノイズをかけて 連続値にしているという理由もあると考えています。

8 GANs is better than Flow

よく言われるものとして、やっぱ GANs が最強っていうのがあります。実際に Flow-based Model の研究のかなりは vs GANs を意識したものを書いています。とはいえ、Flow-based Model は、一部のスタイル変換 (ICLR でニッチな需要だねｗ なんて言われてますが、グリザイユ画法という立派な絵画手法であったりします) や、 画像とテキストを関連付けるタスク (画像の潜在表現と文の潜在表現をエイヤッと Flow-based Model でくっつけます) なんかで強力な 力 を見せています。またモデルパラメータや計算資源使用量も比較的少なくても上手く行くようになっているので、面白い分野ではあるんじゃないかなと思います。

9 ライブラリの話: TFGENZOO

9.1 TFGENZOO について

こいつは今インターンシップを中心に開発を進めている Flow-based Model 系の生成モデルの作成を支援するライブラリです。 Tensorflow 2.x 系の上に作られており、数式とモデルの実装が出来るだけ近くなる (関連がわかりやすくなる) ように注意して作っています。使い方やドキュメントは レポジトリ や Example, あ

Figure 1: GUIDED IMAGE GENERATION WITH CONDITIONAL INVERTIBLE NEURAL NETWORKS

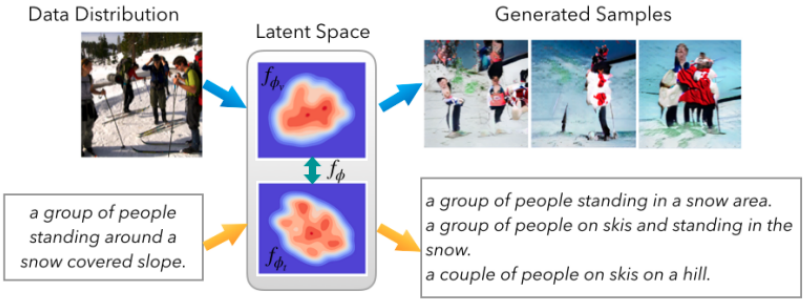


Figure 2: Latent Normalizing Flows for Many-to-Many Cross-Domain Mappings

るいは Issue を参照して下さい。

9.2 今回の実装の詳しい話

実装の詳しい話をすると、こちら がモデルファイルになっています。基本的には概要図のとおりですが、注意する点として、次の部分があります。

```
class Glow(Model):
    def __init__(self, hparams: Dict):
        super().__init__()
        self.model_params = hparams["model"]
        K = self.model_params["K"]
        L = self.model_params["L"]
        conditional = self.model_params["conditional"]
        flows = []
        flows.append(LogitifyImage())
        for layer in range(L):

            # Squeezing Layer
            if layer == 0:
                flows.append(Squeezing(with_zaux=False)) # <--- !!!
            else:
                flows.append(Squeezing(with_zaux=True))
            fml = []

            # build flow module layer
            for k in range(K):
                fml.append(Actnorm())
                fml.append(Inv1x1Conv())
                scale_shift_net = ShallowResNet(
                    width=self.model_params["hidden_width"])
                fml.append(AffineCoupling(
                    mask_type=AffineCouplingMask.ChannelWise,
                    scale_shift_net=scale_shift_net))
            flows.append(FlowModule(fml))

            # Factor Out Layer
            if layer == 0:
                flows.append(
                    FactorOut(with_zaux=False # <--- !!!
                              conditional=conditional))
            elif layer != L - 1:
                flows.append(
```

```
                    FactorOut(with_zaux=True,
                              conditional=conditional))

        self.flows = flows
```

$\text{with}_{\text{zaux}}$ というのは、モデルの概要図的に、「横に z_{aux} が見えるか」 というフラグになっています。これは z_{aux} が img → z の処理と同じように次元数を変えるために必要になっています。