Art'Em – 絵画風加工を VR で実現する: パート 4

この記事は、インテル® デベロッパー・ゾーンに公開されている「Art'Em – Artistic Style Transfer to Virtual Reality Week 14 Update」の日本語参考訳です。

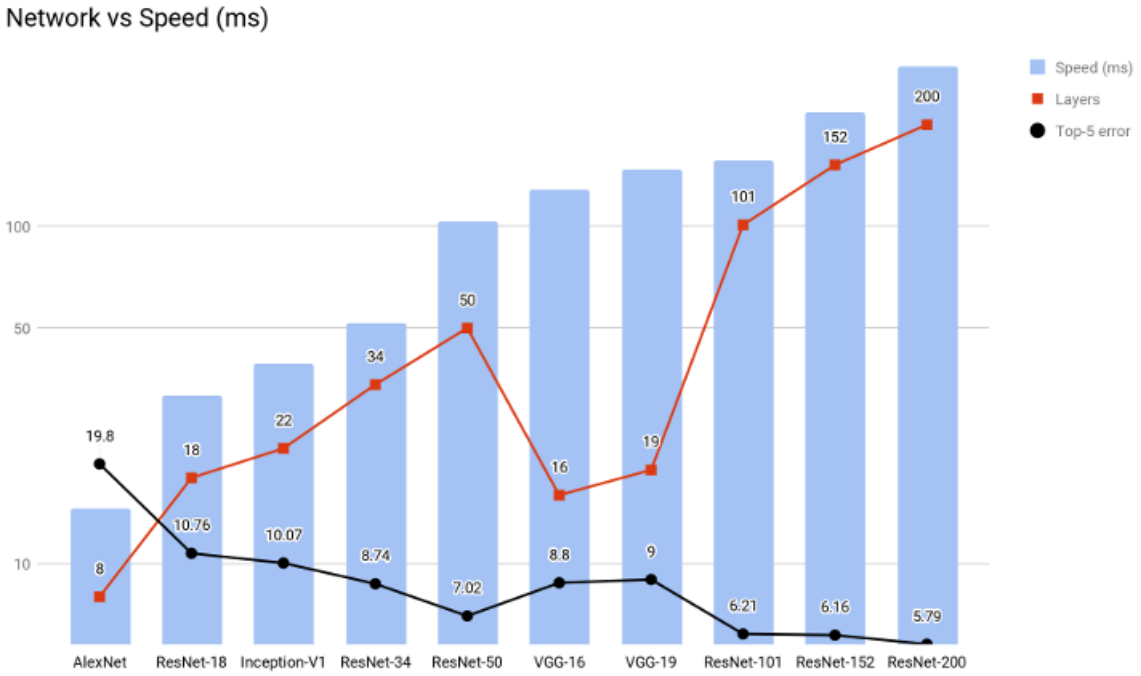
Art'Em (英語) は、バーチャル・リアリティー (VR) で絵画風加工を利用できるようにするアプリケーションです。低精度ネットワークを利用することで、スタイライゼーション (様式化) を高速化することを目的としています。

パート 3[1] では、低精度ネットワークの実装アプローチと畳み込みの仕組みの概要を取り上げました。ここでは、CUDA (Compute Unified Device Architecture) を使用して畳み込み操作を高速化する方法を示し、絵画風加工を高速化するその他のアプローチについて説明します。

最適化

こちらのブログ[2] (英語) から、100 行未満で最適化ベースのニューラル絵画風加工アルゴリズムを作成する方法が分かります。最適化手法により、インテル® Nervana™ プラットフォーム上で訓練を 10,000 回繰り返したところ、以下の結果が得られました。

最適化ベースの絵画風加工手法は、前述のように非常に高品質な結果をもたらしますが、処理に長い時間を要します。ネットワーク速度を見ただけで、独自の最適化では VGG16 の速度が 100ms をはるかに下回ることがないことに気付きました。つまり、最適化ベースの絵画風加工では、繰り返し回数が増えるのに伴って、処理時間も増えます。



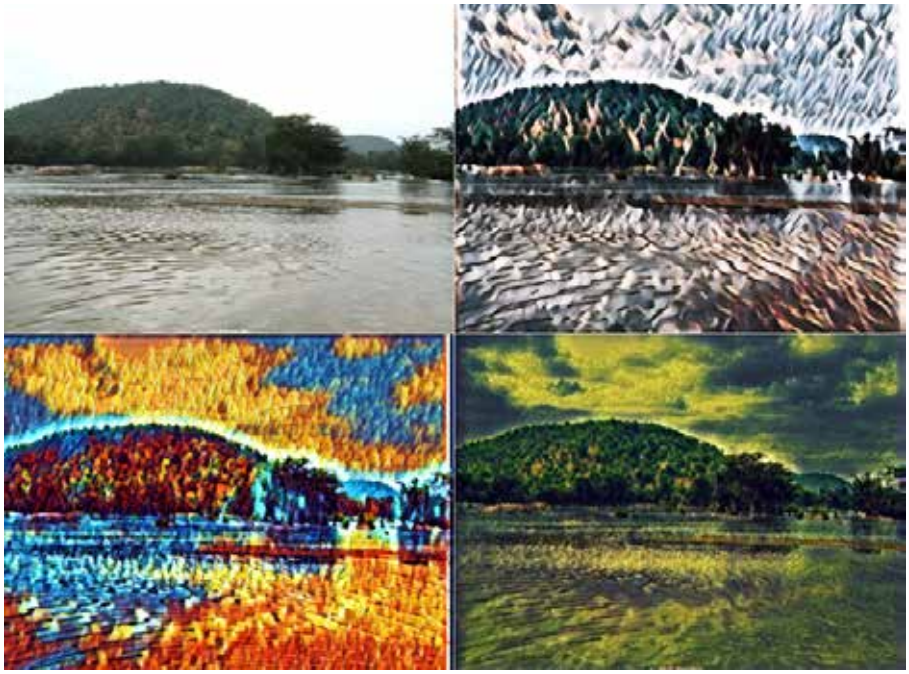
注: 速度は Pascal* Titan X + cuDNN 5.1 上でのフォワードパスとバックワード・パスの合計時間

上のグラフから、精度とネットワーク速度のバランスが良いのは ResNet-34 であることが分かります。しかし、ResNet-34 でも妥当な絵画風加工速度を達成できません。スタイライゼーションに対する繰り返しアプローチは、非常に低速です。こちらの論文[3] (英語) は、「ワンショット」の絵画風加工を提供する生成ネットワークを示唆しています。

論文[3] (英語) では、それぞれのコンテンツイメージを特定のスタイルにスタイライズする生成ネットワークの存在を大まかに仮定しています。生成ネットワークは、最適化によるスタイライゼーションと比較すると、視覚効果と多様性が劣りますが、絵画風加工処理を大幅に高速化できます。特定のスタイル向けに、ネットワークは Microsoft* COCO データセットで訓練されています。これにはいくつかの明らかな欠点がありますが、その 1 つは、新しいスタイルに適応するため、ネットワーク全体を MS-COCO データセットで再訓練する必要があることです。これについては、後述する「アダプティブ・インスタンスの正規化」で触れます。

スタイライゼーション

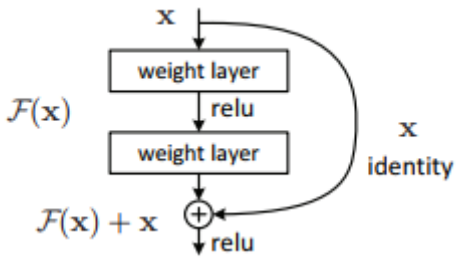
Logan Engstrom[4] の研究を参考にして、私は、生成ネットワークを用いて単純な高速絵画風加工を実装しました。実装にはオリジナルのネットワークを使用しましたが、私の最終的な目的は固定解像度を処理することなので、変換ネットワークのみを使用して、それをイメージローダーと組み合わせることが適切であると考えました。



最適化したコードを GTX 1070 グラフィックス・プロセシング・ユニットで実行したところ、フル VR 解像度で 1 秒あたり 5 ～ 6 フレームになりました。サイズを小さくすると速度が向上します。ここで超解像度が役立ちます。コードは、通常、ウェブカメラからの入力を受け取り、それをスタイライズします。同じアルゴリズムを YouTube* 動画と PIL ImageGrab ユーティリティーで実行した出力が以下の動画です。ここでは、出力の解像度を下げています。

モデルを調整するとより高速な結果が得られます。これらのモデルの訓練は、インテル® Nervana™ DevCloud で実行されます。

Logan Engstrom[4] の生成ネットワークは、3 つの畳み込み層、5 つの残差ブロック、3 つの転置畳み込み層を使用します。このモデルを簡素化することで、より高速な結果が得られる可能性があります。

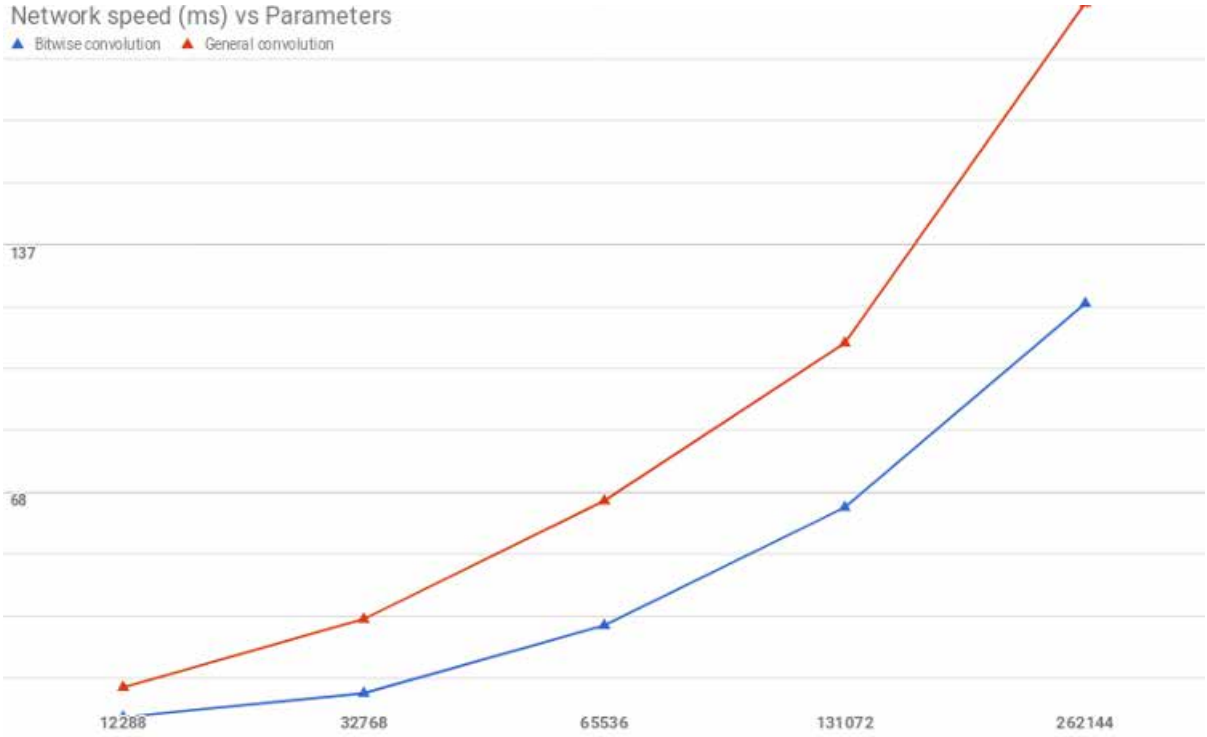


残差ブロック (左) は、通常の CNN 面のスケーリング問題に対応します。例えば、CNN の深さが増すと、過剰適合に関係なく結果が低下します。ただし、残差ブロックを追加し過ぎると、「恒等写像」(出力 = 入力) になります。

転置畳み込みの詳細は、こちら[5] (英語) を参照してください。

スループット

CUDA プログラミングで XNOR 畳み込みを実装するため、私は、バイナリー (2 進) 形式の畳み込みと一般的な完全精度の畳み込み用に単純なカーネルを設計しました。並列化はチャネルごとに行われていますが、チャネル畳み込み自体は並列化されていないことに注意してください。利用可能な最大グリッドサイズが利用できる場合、スピードは数倍向上します。以下のベンチマークは、2 つの手法の違いを示しています。カーネルが最適化されていれば、同様のスケーリングが得られるでしょう。



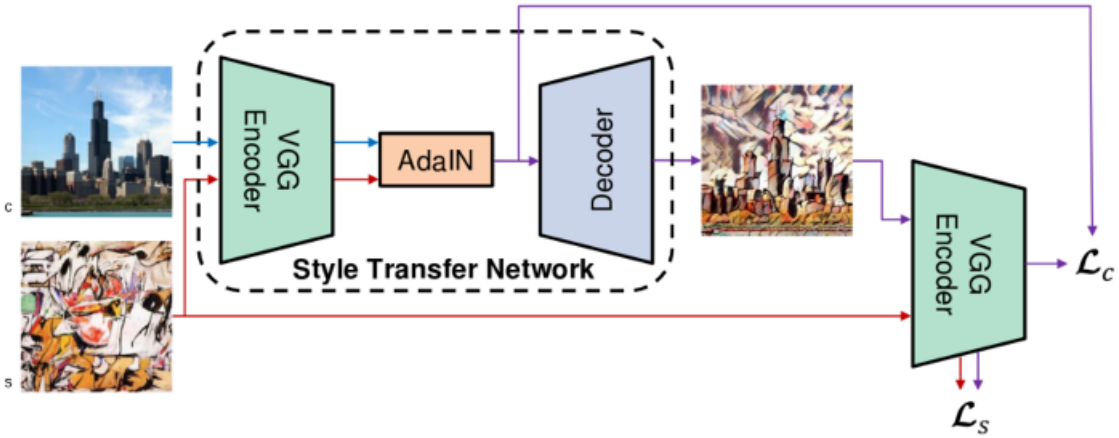
しかし、コードをディープラーニング・フレームワークに統合しようとしたところ、私は多くの技術的な問題に直面しました。そのため、完全精度の畳み込みと比較した XNOR 畳み込みの効果をベンチマークすることにしました。結果は、XNOR-net の行列乗算ほど畳み込みの複雑性が軽減されていないため、全結合ネットワークほど顕著ではありませんでした。それでも、ビット単位の畳み込みによる大幅なスピードアップが確認できました。

このアプローチは、ネットワークのパフォーマンスを大幅に向上できる可能性がありますが、まだ実装に至っていません。完全精度の生成ネットワークをインテル® Nervana™ DevCloud で訓練することは現実的な選択肢と言えます。

アダプティブ・インスタンスの正規化

絵画風加工の初期実装では、インスタンスの正規化と生成ネットワークを使用しました。その際、大きな制限となったのが、それぞれの生成ネットワークは、1 つの絵画風加工にのみ対応できることでした。この制限は、VR かどうかに関係なく、ユーザー・エクスペリエンスに大きく影響します。

最近、素晴らしい研究論文[6] (英語) ですべてのスタイルに適合する方法が見つかりました。それは、VGG エンコーダー、アダプティブ・インスタンスの正規化ブロック、デーコーダーの 3 つのコンポーネントで構成されます。以下の図は、この手法を要約したものです。



$$\text{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y) \qquad \mathcal{L}_s = \sum_{i=1}^{L} \|\mu(\phi_i(g(t))) - \mu(\phi_i(s))\|_2 + \sum_{i=1}^{L} \|\sigma(\phi_i(g(t))) - \sigma(\phi_i(s))\|_2 \qquad \mathcal{L}_c = \|f(g(t)) - t\|_2$$

$\mu(x)$ Mean
$\sigma(y)$ Standard deviation
ϕ_i Layer in VGG19 used to compute style loss

t Adaln(f(c), f(s))
f(t) VGG Encoder
g(t) Decoder

AdaIn は、エンコーダーからのコンテンツ入力を、スタイルイメージのチャネル全体の平均と分散に応じて変換します。

ネットワークは、MS-COCO データセットと Wikiart* イメージのデータセット[7] (英語) で訓練されています。事前に訓練されたネットワークを使用することで、妥当な結果が得られます。同様の結果は、こちら[8] (英語) にもあります。このモデルの実装は、単一スタイルと比較して、非常に時間のかかるものになりました。スタイライズされたモデルはそれぞれ約 20MB であったため、素早くスタイライズするためにインスタンスの正規化という概念を使用するほうが良い選択肢でした。

参考資料:

[1] https://www.isus.jp/machine-learning/art-em-3/
[2] https://quirkyai.wordpress.com/2017/12/17/artistic-style-transfer-in-100-lines-in-python-and-tensorflow/ (英語)
[3] https://arxiv.org/pdf/1701.02096.pdf (英語)
[4] https://github.com/lengstrom/fast-style-transfer (英語)
[5] https://github.com/vdumoulin/conv_arithmetic (英語)
[6] https://arxiv.org/pdf/1703.06868.pdf (英語)
[7] https://www.kaggle.com/c/painter-by-numbers (英語)
[8] https://github.com/xunhuang1995/AdaIN-style (英語)

前のステップ: パート 3
次のステップ: パート 5

コンパイラーの最適化に関する詳細は、最適化に関する注意事項を参照してください。