成蹊大学理工学研究報告
J. Fac. Sci.Tech., Seikei Univ.
Vol.56 No.1 (2019)pp.9-14

機械学習によるRAW現像技術の開発

伊東 直毅[*1]，甲斐 宗徳[*2]

Development of RAW Image Format Converter Using Deep Learning

Naoki ITO[*1], Munenori KAI[*2]

ABSTRACT：RAW development is an operation manually performed by a person in order to finish a photograph according to his / her desire. At this time, many image adjustment parameters are set, and confirmation and adjustment are repeated. The work requires much time and effort, and it is very difficult to make many photos taken into the desired images. However, if deep learning is used, learning RAW images and photographs after RAW development may create photographs of the same level as manual RAW development. In this research, we succeeded in developing a machine learning model that performs RAW development using RAW data as input. We also aimed to improve the processing speed of learning using GPGPU. As a result of comparing processing in which RAW development is performed for 100 images with a single multi-core CPU using a parallel program and processing using a GPGPU, it is shown that the latter can be significantly faster.

Keywords：RAW Development, Deep Learning, Machine Learning, Parallel Processing, GPGPU

(Received May 20, 2019)

1．はじめに

最近ではインスタグラムなどの写真共有を目的としたSNSの人気の高まりから、一眼レフに代表されるハイエンドカメラの人気が高まっている。ユーザは少しでも良い写真に仕上げようと画像の加工を行う事が恒例となっているが、一般的な画像形式であるJEPGでは、画像の加工は画質の劣化を招く。そこで、カメラが受け取った光の情報を無加工、無圧縮で記録したRAWデータを用いて好きな写真のイメージを作り出し現像することで、より高画質でレベルの高い加工を施すことが近年盛んになっている。しかしこのRAW現像では多くのパラメータを手動で設定し、完成イメージを確認しながら調整を行うため非常に多くの手間と時間がかかる。そのため大量に写真を撮影した場合、その全てにRAW現像を行うことは現実的ではない。そこで近年急速に拡大している機械学習に注目する。機械学習（AI）ではニューラルネットワークという人間の神経構造を模したアルゴリズムにより、独自の思考回路を実装可能である。近年では画像認識や自然言語処理の分野で幅広く使用されており、従来アルゴリズムでは対処不可能、または莫大な計算量を必要としていた分野に対しても人間的思考回路を使用、またGPU資源を最大限使用することによって非常に高速な学習・推論を可能としている。RAWデータを自分の意図通りの写真に仕上げるためには、ユーザが自分で加工を施す事が一番確実である。しかし近年注目されている機械学習の技術を用いて、ユーザの意図を学習してRAW現像を行う事で、自分好みの写真を作り出す事が可能ではないかと考える。AIでRAW現像を行う事が出来れば、ユーザは手間をかけずに短い時間で大量にRAW現像を行う事が可能となる。

[*1]：理工学部情報科学科学生

[*2]：理工学専攻教授 (kai@st.seikei.ac.jp)

2．RAW現像

2．1　RAW現像の詳細

RAWデータには様々な形式があるが、今回は最も汎用性のあるDNG形式からRAW現像を行うことを目的とす

る。RAWデータにはカメラメーカー毎に独自の規格があり、それぞれの規格に互換性はない。しかし多くのRAWデータはDNG形式への変換が可能であり、その際の品質ロスは理論上起こり得ないとされている。

DNGデータからRAW現像を行う際の主な処理事項を以下表 2.1 に示している。本来処理の内容はメーカー独自技術として様々な技術が使用されているが、今回の研究ではそのアルゴリズム自体に詳細に触れることはない。AIの学習に現像前のRAWデータと現像後のデータを使用し、その間に施される処理はAIの学習アルゴリズムに頼るものとし、直接画像に対し操作を行わないものとする。ただし、足りない色情報を周辺画素から補完する技術「デモザイク」はAIへの入力前に行うものとする。

表 2.1：主な画像処理事項

● デモザイク	● 明暗調整
● ノイズ除去	● 彩度・輝度補正
● ガンマ補正	● シャープネス補正
● ホワイトバランス調整	● 周辺減光補正

AIへの入力データとしては、RAWデータから各画素のRGB値を〔縦画素数×横画素数×3〕の配列へ入力し、配列へのデータの入力前にはデモザイク処理を施して足りない色情報を周辺画素から補完する。そして既存アルゴリズムによるRAW現像後のデータを教師データとして与え、入力データに対し、教師データとして与えた画像に近くなるようにニューラルネットワーク上のパラメータを近似計算により求め、学習ごとに値を更新していく。学習には初期学習と、ユーザ意図反映のための「ユーザ学習」の2段階を設ける。初期学習ではRAWデータからの基本的な画像出力を学習し、「ユーザ学習」でユーザの意図通りの学習を行うものとする。

2. 2　学習データの準備

機械学習における学習データの必要枚数に明確な定義はないが、大量の学習データが必要となる。一般的には数万という単位で学習データを用意する。今回学習データには私自身が一眼レフカメラで撮影したRAWデータを用いるが、その総数は 2500 枚である。このRAWデータに対して明るさや色味を変更し学習データを増幅させる「data_booster」というプログラムを実装し、このプログラムから学習データを 10 倍に増幅させ総数 2 万 5 千枚とした。なお、今回は 10 倍に増幅したが、実際にはより多くの増幅が可能である。しかしデータをより多く増幅させたところで写真の全体的な印象が変わるわけではなく、学習への効果が大きくなるとは言えない。学習データをより多く用意したい場合、単純に写真枚数をより多く用意する事が好ましい。

3．機械学習技術

3. 1　主な機械学習技術

機械学習の場で主に使われるプログラミング言語はPythonである。Pythonが選ばれることの理由はその文法の柔軟性にある。Pythonでは変数の型宣言が不要であり、外部ライブラリを呼び出す際に複雑な処理を記述する必要がない。他の言語に比べコード数も少なくシンプルに記述することが可能である。

Pythonで機械学習を行うにあたって、特筆すべきは「Tensorflow」[1]というライブラリである。これは機械学習に特化した専用APIであり、オープンソースソフトウェアライブラリである。TensorflowではAIの設計を容易に実現できる様々なAPIが容易されており、実行にあたっては特別な記述を一切せずに最適化されたマルチコア内並列実行が可能である。また、近年注目されているGPGPU（General Purpose computing on GPU）にも対応しており、GPUを使用した場合のパフォーマンスはCPU単体でのパフォーマンスに比べ一般に 100 倍以上であり非常に有用な技術である。いずれもGPUの利用にあたっての特別なコードの記載は不要である。

3. 2　開発要領

本研究では前述のTensorflowをPythonより利用し、Tensorflowの文法に基づきAIモデルを一から開発している。なお、これらのバージョンはPython3.5.2、Tensorflow1.12.0 を用いた。

入力されたRAWデータに対し多数の演算を行い、出力画像は種々の画像処理技術と同等程度のレベルの品質を実現するよう専用設計を行う。一般的な機械学習モデルでは入力データの情報より特徴となる点を抽出し、それを数値化することによって高いレベルの予測を行うが、今回は入力データの情報を失ってしまっては画質が落ちてしまう。従って入力データの情報をできるだけ失わずに種々の演算をこなした上で画像として出力することが求められる。

3. 3　主な機械学習技術

本研究で実際に使用した開発環境の概要を以下表 3.1 に示す。

表 3.1：主な画像処理事項

OS：	Ubuntu 16.04.5 (64bit)
CPU:	Intel Xeon(R) E5-1650 v4 @3.6GHz x6
RAM:	DDR4 2600MHz 16GB x 4
	Total:64GB
GPU:	Titan V x2
	CUDAコア: 5120
	Tensorコア: 640

4．機械学習モデルの開発

AIモデルの構築とは、入力データに対しどのような演算を行うか、その計算式を定 1 義することである。ニューラルネットワークではY=Wx+bの形を基本としており、ここでxが入力、Y が出力である。このときWの値を重み（weight）、bをバイアス(bias)と呼ぶ。そしてこの重みとバイアスが学習により更新される値である。今回は入力データに対し、同じサイズの配列を用意し、初期値として 0.9 から 1.1 の数値を乱数により代入し、入力データとの乗算を行った後にバイアスを足すという形を基本としている。以下これを「Multiplyレイヤー」とする。

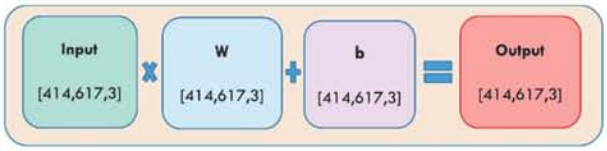


図 4.1：Multiplyレイヤー

このMultiplyレイヤーを図 4.2 の通り 4 層並列に展開し、それぞれの出力の平均をとって出力する層を「avg_x4 レイヤー」として実装する。

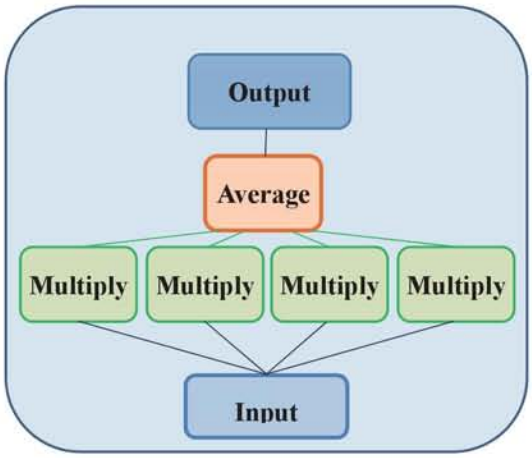
さらにavg_x4 レイヤーを 2 層並列展開して平均をとって 1 出力とする層を図 4.3 の通り 4 層実装し、その計算結果をAIの出力とした。以上でavg_x4 レイヤーを合計 8 層実装し、モデルの全体像としている。

図 4.4 にこのモデルの実装のコードを示す。

5．ユーザ意図の反映

5．1　ユーザ意図反映の仕組み

機械学習では、出力した画像に調整が必要な場合にモデルに対して微調整を行う事が出来ない。これは機械学習が大量の演算を行い、その大量の演算の極めて細かいバランスの上に出力が成り立っているためであり、モデル上のどのパラメータをどのように変更すれば期待通りの出力となるかは未知数である。



4.2：avg_x4 レイヤー

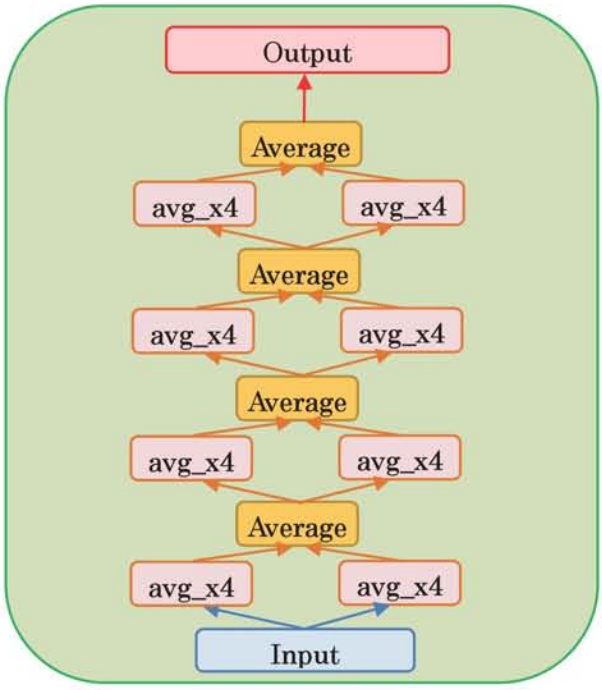


図 4.3：avg_x4 レイヤー

```
def multiply(x):
    W = tf.Variable(tf.random_normal([414,617,3], mean=1.0, stddev=0.05))
    b = tf.Variable(tf.zeros([414,617,3]))
    x = tf.math.multiply(x,W)+b
    return x
def avg_x4(x):
    with tf.name_scope("adding"):
        x1 = mlp(x)
        x2 = mlp(x)
        x3 = mlp(x)
        x4 = mlp(x)
        x1 = tf.math.add(x1,x2)
        x3 = tf.math.add(x3,x4)
        h  = tf.math.add(x1,x3)/4

        return h

def avg_x4_x2(x):
    x1 = layer_add(x)
    x2 = layer_add(x)
    h  = tf.math.add(x1,x2)/2
    return h

def model(x):
    x = layer(x)
    x = layer(x)
    x = layer(x)
    x = tf.nn.relu(layer(x))

    return x
```

図 4.4：モデル実装のコード

図 5.1：GUIプログラムの画像選択画面

そこで本研究では、ユーザがどのような画像出力を期待しているか、画像の具体例をGUIで提示しユーザが選んだ画像を学習データとして再学習するというアプローチをとっている。またユーザの負荷軽減のため、少ない学習データでも効率的に学習を行う事が出来るよう、2.3章で紹介した「data_booster」を用いて学習データ数を10倍に増幅して学習を行う仕組みとしている。

このGUIの画面の様子を図 5.1 に示す。1 枚のRAWデータを 8 種類の異なる明るさ・色味の写真にRAW現像を行い、ユーザが選んだ画像を保存し学習データとして活用する。画像の一覧は縦 2 段、横 4 列で表示し、上段が暗い画像、下段が明るい画像である。また、左側が寒色系、右に行くにつれて暖色系になるように構成している。画像の系統ごとに規則的に配置することにより、ユーザは各画像の変化をより直感的に感じることができる。また、画像の選択は画像の位置に対応したボタンを押す仕組みとなっているが、ボタンを一か所にまとめ距離を縮めたことにより、マウスの移動距離を少なくしユーザの負担を軽減している。

6．評価

6．1　学習結果

以上の機械学習モデルの構成により、初期学習として 2 万 5 千枚の学習データを用いて学習を行った。なお学習の評価の尺度として、図 6.1 に示すようにAIの出力画像と教師データの画像との全ピクセルの数値の差分をとり、その差の平均値をLossとして算出し、Lossの値を評価尺度としている。

学習回数とLossの値の推移の関係を図 6.2 に示す。

図 6.1：Lossの算出

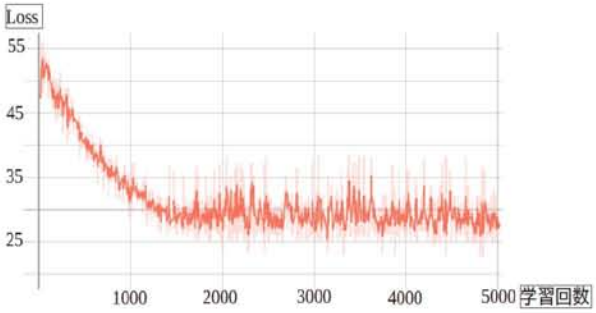


図 6.2：学習回数とLossの関係

グラフから分かるように、学習を進めるにつれてLossの値は確実に減少しており、学習が効果的に進んでいることが確認できる。学習約 1500 回でLossは 30 前後、学習 5000 回では概ね 30 以下に収まっている。実験では 5 万回まで学習を進めたが、Lossの値がこれ以上減少することはなかった。つまり 5000 回の学習でこのモデルは十分成熟したと言える。このモデルで実際に出力した画像と、入力データとして与えた画像を図 6.3 に示す。

図 6.3 は上が入力画像、下が出力画像である。入力画像と出力画像を比較すると、明るさが大きく補正されていることがわかる。入力画像に対して機械学習モデル内で多数の計算が行われた結果、このように差が生じてい

図 6.3：入力・出力画像の比較

る。一方このように暗かった画像が明るくなる例もあるが、実際には元々明るい画像がさらに明るくなってしまった例もある。AIに期待しているのは元々の明るさに対して適切な調整をすることであるが、必ずしもそうはならなかった。今回は精度が上がらなくなるところまで学習を進めたため、これ以上の精度を求めるにはモデルのレイヤーを増やす必要がある。しかしモデルを重くすればメモリ消費量が増える。今回学習に使用したGPUのメモリ 12GBでは、このモデルのメモリ消費量が限度であり、これ以上重いモデルでは実行時エラーとなったため、本研究ではこの精度が限界となった。

6. 2 追加学習とその結果

GUIプログラムによって作成した学習データ 100 枚を 10 倍に増幅し、合計 1000 枚の学習データを用いて追加学習を行った。なお、今回は検証のためにあえて暖色系の画像のみを選択し、学習が収束する事、出力が確かに暖色系の画像になることを検証するものとする。学習回数とLossの推移を以下図 6.4 に示す。

図 6.4 のグラフでは、学習済みモデルに対しての追加学習である性質上、スタートのLossが低くなっている。Lossの推移は 35 付近からスタートし、学習 1000 回で 25 程度に収束している。一度学習済みであること、学習データが少ないことから初期学習よりも少ない学習回数で収束できていると推測できる。Lossの値が減ったことで、本当に画像が暖色系になっているのかを次の図 6.5 の比較画像にて検証する。

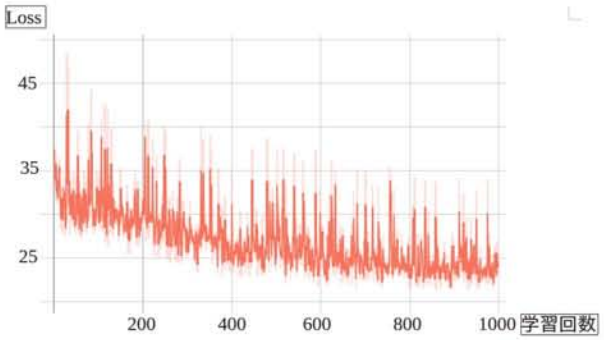


図 6.4：追加学習時の学習回数とLossの関係

図 6.5 は上が入力画像、下が出力画像である。入力画像ではピンク色の照明の光を受けた桜がやや青っぽく照らされているのに対して、出力画像では桜がオレンジ色に照らされている。学習では暖色系の画像のみを追加学習したため、モデルが暖色系の出力へと傾向を変えたことが確認できる。このように、ユーザの意図に合う学習データを 1000 枚確保し追加で学習を行う事で、ユーザの意図を反映したモデルを作りあげる事が可能であることが分かった。

図 6.5：追加学習後の入出力画像の比較

7．処理時間

今回はGPUを用いて機械学習を行っていることから、RAW現像にかかる処理時間が従来アルゴリズムによりCPU上で行った際に比べ早くなっていることが期待できる。GPU上ではメモリ上限を考慮し一度に 100 枚までのRAW現像を行う事が出来るため、CPUでのRAW現像も同様に並列プログラムで実装し、一番結果の良かった並列実装の処理時間を機械学習によるRAW現像の処理時間と比較することとする。

従来アルゴリズムによるRAW現像の並列手法については、RAW現像の処理の中身を並列化するものではなく、Python の 並 列 プ ロ グ ラ ミ ン グ 用 ラ イ ブ ラ リ「multiprocessing」を用いて同時に複数枚のRAW現像をプロセス並列で行うものとする。同時に立ち上げるプロセス数を 5 刻みで変え実行時間を計測した結果を以下図 7.1 に示す。なお今回使用するCPUは 6 コア 12 スレッドである。なお、処理時間はディスクI/Oの時間を含まず、RAW現像の処理にかかった時間のみを計測する。

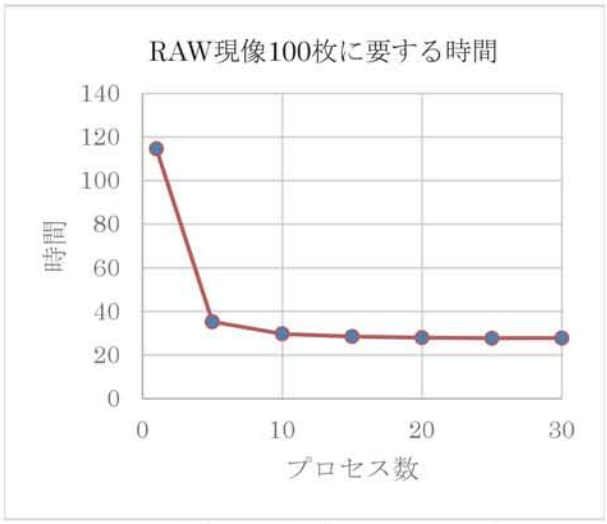


図 7.1：従来アルゴリズムによる処理時間

図 7.1 のグラフより、1 プロセスでの実行よりも複数プロセスでの実行の方が遥かに早く、その差は最大で 4.14 倍となっている。1 プロセスでは 114.56 秒かかっていた処理時間が、最速であった 25 プロセスでは 27.67 秒となっている。プロセス数 10 以上でほとんど処理速度は頭打ちとなっているが、これはCPUが 6 コア 12 スレッドのため当然の結果といえる。

次に従来アルゴリズムで最速タイムであった 25 プロセスでのRAW現像と、機械学習でRAW現像を行った際の処理時間を次の図 7.2 に示す。

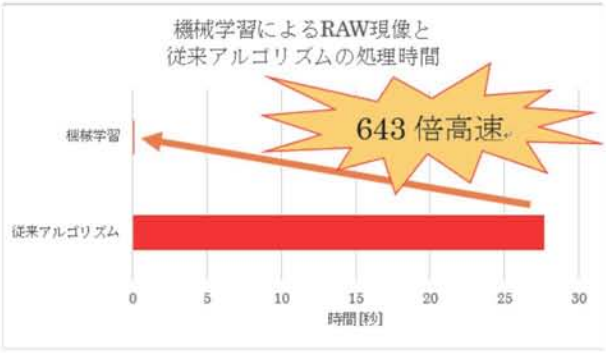


図 7.2：機械学習による処理時間

結果は圧倒的で、従来アルゴリズムの 27.67 秒に対して機械学習では 0.043 秒であり 643 倍の高速化となった。最も今回の実験ではハイエンドのTitan VというGPUを用いていることが高速化の要因として大きいが、GPUを利用できる機械学習にとって処理時間の面では利があると言える。

8．結論

RAWデータを入力としてRAW現像を行う機械学習モデルを開発することに成功した。また学習を補助するプログラム「data_booster」による学習データの増幅によってデータ数を増幅し、そのデータを用いて学習を行った結果、期待通りの学習結果が得られたため、「data_booster」によるデータ増幅に一定の効果があったと言える。

またユーザ意図の反映のため、GUIプログラムによりユーザの期待する画像例を学習データとして効率的に作成し、「data_booster」と組み合わせて追加学習を行う事でユーザ意図をモデルに反映することができることを確認出来た。

処理速度に着目すると、CPU単体で並列プログラムを用いてRAW現像を 100 枚分行った際に比べ機械学習では 643 倍高速であり、大きな成果といえる。

参考文献

[1] Tensorflow 公式サイト https://www.tensorflow.org