OpenVINO™ を使用したモデル最適化のための合成データ生成 DL モデルのトレーニング - パート 2

この記事は、Medium に公開されている「Train a DL model for synthetic data generation for model optimization with OpenVINO — Part 2」の日本語参考訳です。原文は更新される可能性があります。原文と翻訳文の内容が異なる場合は原文を優先してください。

最近のディープラーニング・モデルはますます大きくなる傾向があります。何十億ものパラメーターを含むニューラル・ネットワークは、高負荷の作業を大規模なプロセシング・ユニットで実行することにより、さまざまなタスクで最先端のパフォーマンスを実現しています。モデル最適化の目標は、リソース要件を軽減しながら、ディープラーニング・モデルのパフォーマンスを向上させることです。

この記事は、インテルの OpenVINO™ ツールキットの後援の下に開発された、Google* Summer of Code 2022 プロジェクト「モデル最適化のための合成データ生成 DL モデルのトレーニング」に関するブログのパート 2 です (パート 1 はこちら)。プロジェクトは 2 つのパートで構成されます。パート 1 では、軽量のディープラーニング・モデルをトレーニングして合成画像を生成しました。このパート 2 では、パート 1 の事前トレーニング済みモデルを使用して、CIFAR-10 向けの合成画像のデータセットを生成します。このデータセットを、OpenVINO™ のトレーニング後の最適化ツールによるモデル最適化に使用します。その後、さまざまなコンピューター・ビジョン・モデルで、8 ビットのトレーニング後の量子化手法のパフォーマンスを評価します。

プロジェクトのページ: Google* Summer of Code (英語)、GitHub* リポジトリー (英語)

はじめに

トレーニング後のモデル最適化

トレーニング後のモデル最適化は、ハードウェア、ソフトウェア、ネットワーク・アーキテクチャーなど、さまざまなレベルの技術を利用して、ディープラーニング・モデルのパフォーマンスを向上させるプロセスです。モデルは、ハードウェア・レベルでは特定のハードウェア・アクセラレーター (TPU など) で実行するように最適化され、ソフトウェア・レベルではパフォーマンスに優れたプラグイン・アーキテクチャー (OpenVINO™ ランタイム (英語) など) で実行するように最適化されます。さらに、**ネットワーク・プルーニング**などの手法によりネットワークのサイズを削減して、推論速度を向上させます。現在は、モデルの量子化も一般的な手法の 1 つです。

モデルの量子化

この手法は、ネットワークの浮動小数点の重みを低精度のコンパクトな表現に変換するプロセスを指します。精度をほとんど損なうことなく、モデルのサイズを削減して、推論速度を向上させることができます。モデルの重みに加えて、対応する精度と一致するように演算を変換することもできます。この手法を使用すると、ディープラーニング・モデルを 16 ビットまたは 32 ビットの浮動小数点精度から 8 ビットの整数精度に変換できます。つまり、推論速度の点でモデルのパフォーマンスが 2 倍または 4 倍向上する可能性があります。

OpenVINO™ ツールキット

OpenVINO™ (英語) ツールキットは、インテルが無償で提供しているオープンソースのツールで、独自の推論エンジンを使用してディープラーニング・モデルを最適化し、インテル製のさまざまなハードウェアへデプロイすることができます。

モデル・オプティマイザー

OpenVINO™ のモデル・オプティマイザーの目的は、さまざまなフレームワーク (TensorFlow*、PyTorch*、Caffe* など) で開発されたディープラーニング・モデルを OpenVINO™ ランタイムで推論が可能なモデルの中間表現 (IR) に変換することです。生成された IR モデルは選択したターゲットデバイス向けに最適化され、モデルの精度を維持したまま推論速度が向上します。

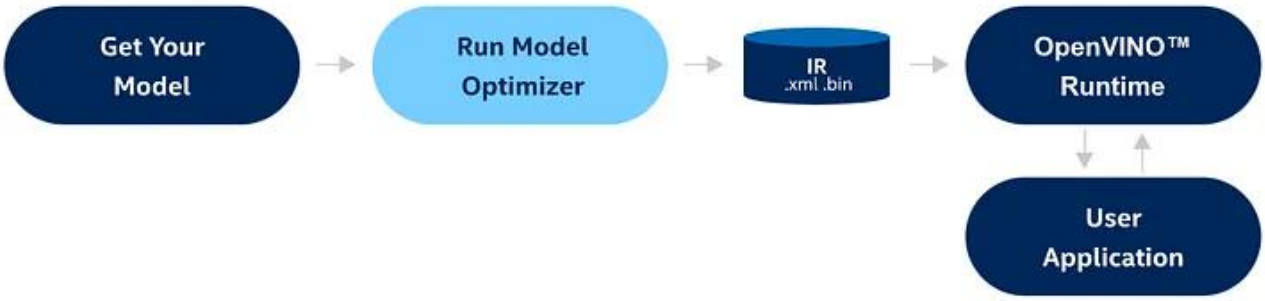

図 1: モデル・オプティマイザーのワークフロー (出典 (英語))

IR モデルは、OpenVINO™ のトレーニング後の最適化ツールを使用してさらに最適化できます。モデル・オプティマイザーの詳細は、公式ドキュメント (英語) を参照してください。

トレーニング後の最適化ツール

OpenVINO™ のトレーニング後の最適化ツールは、モデルのパフォーマンスを最適化する 2 つの量子化手法を提供します。このツールはトレーニング後に実行されるため、データセットは必要ありませんが、典型的なキャリブレーション・セット (例: 300 サンプル) が必要です。さらに、モデルを OpenVINO™ の IR 形式に最初に変換する必要があります。これらの要件を満たすと、デフォルトの量子化 (英語) アルゴリズムまたは精度を考慮した (英語) 量子化アルゴリズムを使用して、浮動小数点精度モデル (FP32 または FP16) を 8 ビット整数精度に量子化できます。前者は、ほとんどのケースで満足なパフォーマンスが得られ、注釈付きのキャリブレーション・データセットが必要ないため (注釈なしのキャリブレーション・データセットは必要です)、最初のステップとして推奨されます。一方、後者は、精度を特定の範囲に維持することに重点を置いているため、注釈付きのキャリブレーション・データセットが必要です。図 2 に最適化ワークフローを示します。

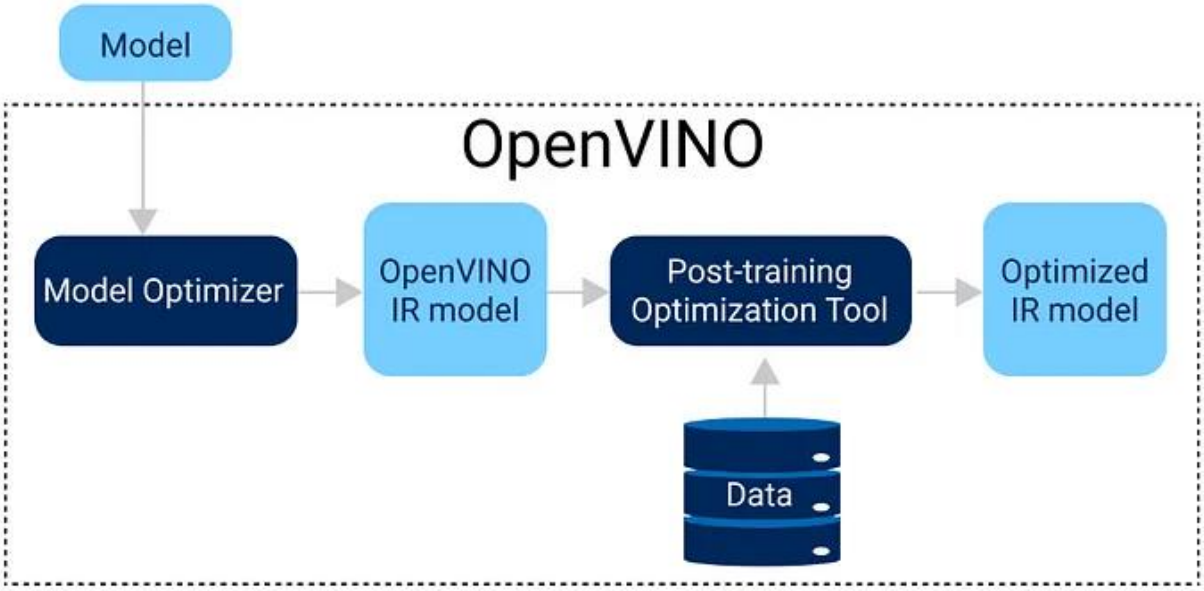

図 2: トレーニング後の最適化ツールのワークフロー (出典 (英語))

テスト

このプロジェクトのパート 1 で開発した DiStyleGAN モデルは、CIFAR-10 データセットの分布から画像を生成します。そのため、分類のために CIFAR-10 で事前にトレーニングされた PyTorch* モデルをテストしました。テストでは、OpenVINO™ のトレーニング後の最適化ツールで提供される、デフォルトの量子化と精度を考慮した量子化の両方の手法を使用しました。複数のキャリブレーション・データセットを使用して同じテストを行い、公式 CIFAR-10 テストセットの分類タスクの量子化されたモデルの結果と比較しました。

PyTorch* モデル

分類のために CIFAR-10 で事前にトレーニングされた PyTorch* モデルを量子化することにしました。これらのモデルは、GitHub* の chenyaofo/pytorch-cifar-models (英語) リポジトリーから取得しました。次のモデルを使用しました。

- ResNet20 (resnet20)
- VGG16 (vgg16_bn)
- MobileNetV2 (mobilenetv2_x1_4)
- ShuffleNetV2 (shufflenetv2_x2_0)
- RepVGG (repvgg_a2)

キャリブレーション・データセット

OpenVINO™ のトレーニング後の最適化ツールの量子化手法を使用するには、キャリブレーション・データセットが必要です。プロジェクトの目標は、DiStyleGAN で生成された合成画像をキャリブレーション・データセットとして使用することですが、ほかの 3 つのデータセットを使用した量子化も行いました。テストでは、次のキャリブレーション・データセットを使用しました。

- 公式 CIFAR-10 (英語) トレーニング・セット
- StyleGAN2-ADA (英語) で生成された合成画像
- DiStyleGAN (英語) で生成された合成画像
- GitHub* の Datumaro のリポジトリー (英語) を使用して生成されたフラクタル画像
  (上記の合成データセットは CIFAR-10 分布に近似しているため典型的なデータセットであると見なすことができますが、フラクタル画像は CIFAR-10 で事前にトレーニングされたディープラーニング・モデルの典型的なデータセットではないことに注意してください)。

各データセットから 5,000 枚、CIFAR-10 データセットのクラスごとに 500 枚の画像を使用しました。これらのサブセットはここ (英語) からダウンロードするか、リンクの手順に従って生成できます。次に、CIFAR-10 テストセットを使用して、選択した PyTorch* モデルの分類タスクの量子化手法の結果を評価しました。

結果

以下のデフォルトおよび精度を考慮した量子化アルゴリズムの結果はそれぞれ、defaultQuantization.ipynb (英語) ノートブックと accuracyQuantization.ipynb (英語) ノートブックを使用して取得したものです。テストは、インテル® Core™ i7-1165G7 CPU 上で行いました。量子化されたモデルはここ (英語) から入手できます。

デフォルトの量子化
デフォルトの量子化手法の結果を表 1 に示します。精度は公式 CIFAR-10 テストセットで計算した値です。

Search this file...

Model	Calibration Dataset	Accuracy	FPS
ResNet20 (PyTorch)	-	0.926	186.7
ResNet20 (IR)	-	0.926	1277.52
ResNet20 (8-bit quantized)	Fractal	0.9158	1590.19
ResNet20 (8-bit quantized)	CIFAR-10	0.9234	1565.45
ResNet20 (8-bit quantized)	FakeCIFAR10 (StyleGAN2-ADA)	0.922	1541.86
ResNet20 (8-bit quantized)	FakeCIFAR10 (DiStyleGAN)	0.9217	1602.92
VGG16_bn (PyTorch)	-	0.9416	54.85
VGG16_bn (IR)	-	0.9416	246.29
VGG16_bn (8-bit quantized)	Fractal	0.9351	654.41
VGG16_bn (8-bit quantized)	CIFAR-10	0.9411	680.46
VGG16_bn (8-bit quantized)	FakeCIFAR10 (StyleGAN2-ADA)	0.9401	604.65
VGG16_bn (8-bit quantized)	FakeCIFAR10 (DiStyleGAN)	0.9409	623.14
MobileNetV2_x1_4 (PyTorch)	-	0.9421	37.5
MobileNetV2_x1_4 (IR)	-	0.9421	130.38
MobileNetV2_x1_4 (8-bit quantized)	Fractal	0.937	478.11
MobileNetV2_x1_4 (8-bit quantized)	CIFAR-10	0.9414	425.89
MobileNetV2_x1_4 (8-bit quantized)	FakeCIFAR10 (StyleGAN2-ADA)	0.9416	447.95
MobileNetV2_x1_4 (8-bit quantized)	FakeCIFAR10 (DiStyleGAN)	0.9406	483.06
ShuffleNetv2_x2_0 (PyTorch)	-	0.9398	43.14
ShuffleNetv2_x2_0 (IR)	-	0.9398	228.89
ShuffleNetv2_x2_0 (8-bit quantized)	Fractal	0.1202	442.75
ShuffleNetv2_x2_0 (8-bit quantized)	CIFAR-10	0.928	404.5
ShuffleNetv2_x2_0 (8-bit quantized)	FakeCIFAR10 (StyleGAN2-ADA)	0.8695	417.81
ShuffleNetv2_x2_0 (8-bit quantized)	FakeCIFAR10 (DiStyleGAN)	0.9258	443.99
RepVGG_a2 (PyTorch)	-	0.9527	11.43
RepVGG_a2 (IR)	-	0.9527	56.77
RepVGG_a2 (8-bit quantized)	Fractal	0.5551	154.89
RepVGG_a2 (8-bit quantized)	CIFAR-10	0.5531	136.29
RepVGG_a2 (8-bit quantized)	FakeCIFAR10 (StyleGAN2-ADA)	0.5586	148.98
RepVGG_a2 (8-bit quantized)	FakeCIFAR10 (DiStyleGAN)	0.5445	135.94

default-results.csv hosted with ❤ by GitHub view raw

表 1. デフォルトの量子化の結果

精度を考慮した量子化
精度を考慮した量子化手法の結果を表 2 に示します。ShuffleNetV2 と RepVGG の 2 つのモデルをテストした結果、デフォルトの量子化手法を使用すると精度が低下することが分かりました。精度は公式 CIFAR-10 テストセットで計算した値です。推論速度 (FPS) は、対応するモデルの表 1 の値と同様であるためレポートしていません。

Search this file...

Model \ Calibration Dataset	Fractal	CIFAR-10	StyleGAN2-ADA	DiStyleGAN
ShuffleNetv2_x2_0	0.1202	0.928	0.9343	0.9388
RepVGG_a2	0.5510	0.9484	0.9481	0.9475

accuracy-results.csv hosted with ❤ by GitHub view raw

表 2: 精度を考慮した量子化の結果 (測定: 精度)

考察

テストで得られた結果に基づくと、多くのケースで、デフォルトの量子化アルゴリズムは、ディープラーニング・モデルの精度をほとんど低下させることなく、モデルの推論パフォーマンスを向上させることができます。これは特に、ResNet20、VGG、および MobileNetV2 モデルに当てはまり、典型的でないキャリブレーション・データセット (Fractal) を使用した場合でも、精度の低下はわずか 1.1% で、推論速度は、IR モデルで最大 4 倍、PyTorch* モデルで最大 14 倍になります。

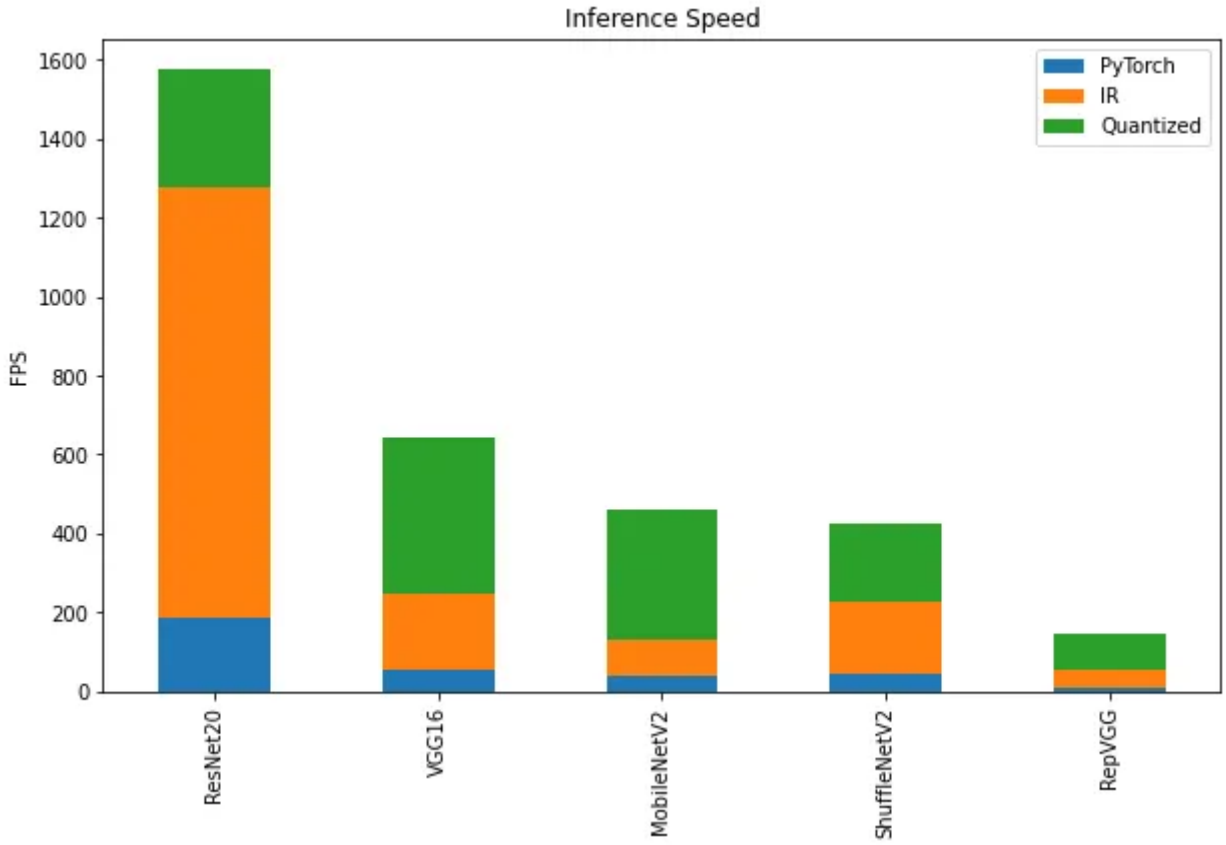

図 3: IR および量子化されたモデルでの推論速度の向上

RepVGG モデルは、デフォルトの量子化アルゴリズムを使用すると精度が大幅に低下します。公式 CIFAR-10 データセットを量子化に使用した場合でも、精度が約 42% 低下します。ShuffleNetV2 モデルは、公式 CIFAR-10 または DiStyleGAN キャリブレーション・データセットを使用した場合はそれほど影響はありませんが、StyleGAN2-ADA データセットを使用した場合は精度が 7.48% 低下します。しかし、精度を考慮した量子化アルゴリズムを使用すると、3 つの典型的なデータセット (CIFAR-10、StyleGAN2-ADA、および DiStyleGAN) で 2 つのモデルのパフォーマンスが大幅に向上します。

最後に、2 つの合成データセット、StyleGAN2-ADA と DiStyleGAN の量子化の結果が公式 CIFAR-10 データセットと同等であることに注目することも重要です。驚くべきことに、DiStyleGAN モデルは StyleGAN2-ADA を教師ネットワークとした知識蒸留によりトレーニングされたにもかかわらず、前者で生成されたデータセットを使用した量子化プロセスの方が後者を使用した場合よりも結果が優れていることがあります。

まとめ

量子化プロセスにより得られる推論速度のパフォーマンスの向上は、わずかな精度の低下を上回るほど十分に大きいことは明らかです。また、2 つの合成データセットで得られた結果、および一部のモデルの Fractal データセットで得られた結果は、公式 CIFAR-10 データセットの結果と同等でした。つまり、キャリブレーションには公式データセットを使用するという制約がなくなり、ディープラーニング・モデルで生成されたデータを OpenVINO™ ツールキットを使用したモデル最適化に使用**できる**ことになります。

OpenVINO™ ツールキットとは

AI を加速する無償のツールである OpenVINO™ ツールキットは、インテルが無償で提供しているインテル製の CPU や GPU、VPU、FPGA などのパフォーマンスを最大限に活用して、コンピューター・ビジョン、画像関係をはじめ、自然言語処理や音声処理など、幅広いディープラーニング・モデルで推論を最適化し高速化する推論エンジン/ツールスイートです。

OpenVINO™ ツールキット・ページでは、ツールの概要、利用方法、導入事例、トレーニング、ツール・ダウンロードまでさまざまな情報を提供しています。ぜひ特設サイトにアクセスしてみてください。

https://www.intel.co.jp/content/www/jp/ja/internet-of-things/openvino-toolkit.html