ケーススタディー – 画像認識モデルのトレーニングにインテル® Deep Learning SDK を使用する

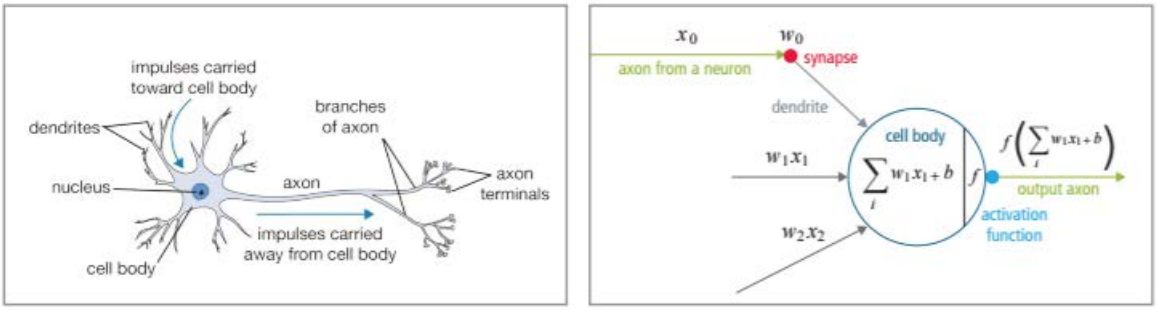
この記事は、インテル® デベロッパー・ゾーンに公開されている「Case Study – Using the Intel® Deep Learning SDK for Training Image Recognition Models」の日本語参考訳です。

概要

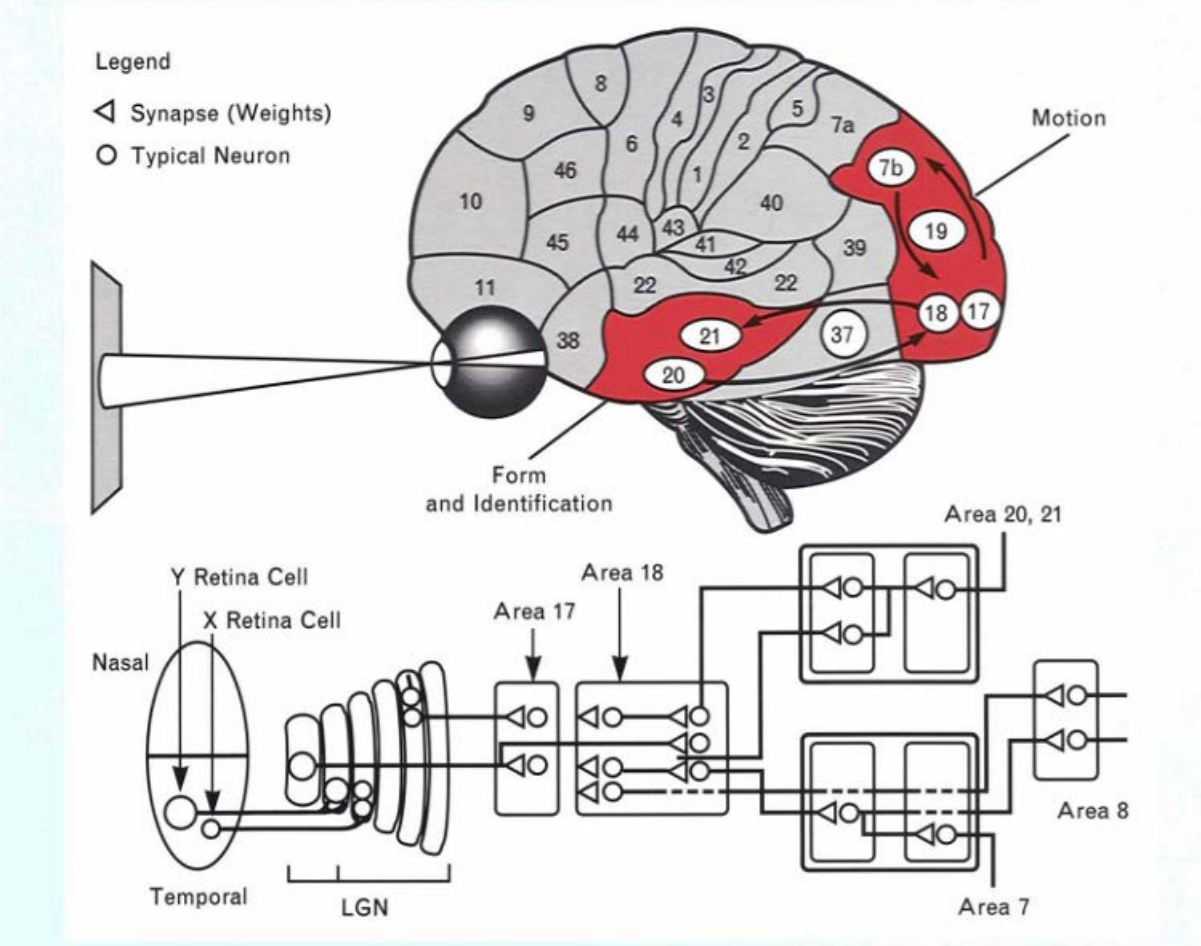
インテル® Deep Learning SDK は、ディープラーニング・ソリューションの開発、トレーニング、配置を支援する、データ・サイエンティストとソフトウェア開発者向けの無料のツールセットです。トレーニング・ツールと配置ツールが含まれており、ディープラーニング・ワークフローにおいて個別または一緒に使用することができます。このケーススタディーでは、有名な手書き数字認識用画像認識トポロジーの 1 つである LeNet を調査し、インテル® アーキテクチャー向けに最適化された Caffe* 上でトレーニング・ツールを使用して MNIST (Mixed National Institute of Standards and Technology) データセットのセットアップ、チューニング、トレーニングを行う方法を示します。このケーススタディーは、データ・サイエンティスト向けです。

人間の視覚システムと畳み込みニューラル・ネットワーク

インテル® Deep Learning SDK の使用法を説明する前に、人間の視覚システムの仕組みと、コンピューター・ニューラル・ネットワークの設計との関連性を理解しておくと良いでしょう。神経細胞は、脳の基本計算ユニットです。樹状突起から入力を受け取り、すべての入力の計算が特定のしきい値を超えると、つながれた神経細胞に出力を送ります。数学的には、次のように表すことができます [1]。



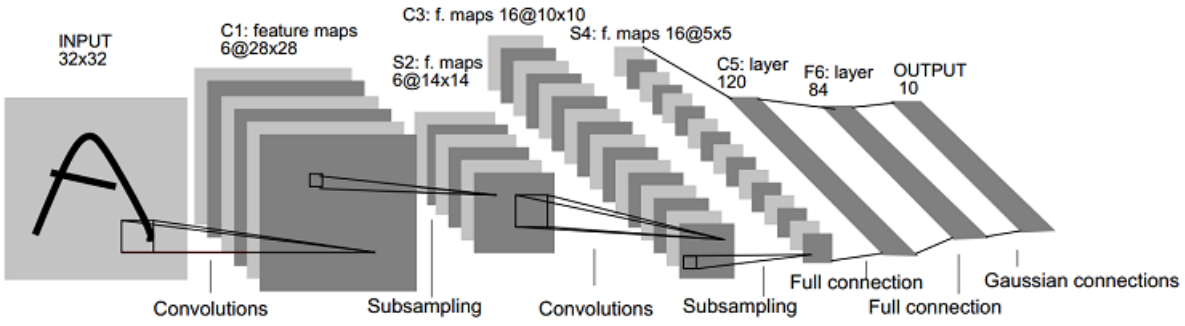
これは、非常に単純化したモデルです。実際には、人間の脳は入力信号を、特徴抽出、特徴検出、分類を行う視覚野の複数の層で処理します。特徴抽出は、視覚野の細胞で処理されます。視覚野の細胞は、視野 (受容野) のオーバーラップする領域のサブサンプリングを行うため、入力イメージのフィルターとして機能します。特徴検出は大脳皮質野 17、18、19 で、分類は 20 と 21 でそれぞれ行われます。以下の図を参照してください [2]。処理情報は、前方に送られるとともに、画像が正しく認識されるまで前の層へ後方伝播されます。



畳み込みニューラル・ネットワーク (CNN) では、畳み込み層が特徴検出器の役割を果たし、全結合層が分類器の役割を果たします。特徴抽出はランタイムに行われます。複数の入力セットを同時に CNN に渡し (ミニバッチ)、各反復で重みを調整することで、各フォワードパスの特徴をまとめ、後方伝播時に特徴を細かくチューニングします。次に有名な手書き数字認識用 CNN である LeNet トポロジーについて説明します。

LeNet トポロジー

以下は、[3] で公開されている LeNet-5 アーキテクチャーです。



トポロジーには、入力のほかに 7 つの層があります。畳み込み層とサブサンプリング層がそれぞれ 2 つ、2 つの全結合層、1 つの出力分類層が続きます。

最初の畳み込み層 C1 には、6 つの 28 × 28 次元の特徴マップがあります。カーネルサイズ 5 × 5、ランダムな重み、定数バイアスが選択されています。特徴マップは 6 つあるため、トレーニング可能なパラメーターの合計は 156 です (6 つの特徴マップ * (5 * 5 + 1[バイアス項]))。特徴マップは、オーバーラップする領域で 1 ピクセルずつ移動してスキャンされるため、最初の層の結合は合計 122,304 になります。問題の複雑さに応じて、各層の神経細胞が急速に多様化します。計算ユニットの数を減らすため、サブサンプリングを使用します。層 S2 には、14 × 14 の 6 つの特徴マップがあります。対応する特徴マップ C1 の 2 × 2 ピクセルをサンプリングし、4 つの入力を追加して、それらにトレーニング可能な係数とバイアスを掛けることで、次元を減らします。サブサンプリング (プーリング) 層の 2 × 2 領域はオーバーラップしません。そのため、S2 では、トレーニング可能なパラメーターは 12 ((1 つの係数 + 1 つのバイアス)* 6 つの特徴マップ)、結合は 5,880 になります。

次に、層 C3 について見てみましょう。10 × 10 の 16 の特徴マップがあります。以下の表は、S2 と C3 の間でピクセル数を減らした方法を示しています [3]。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

S2 から C3 へ非対称のピクセルを選択することで、合理的な結合数を保持しつつ、異なるピクセルを持つそれぞれの特徴マップが異なる特徴を抽出できるようにしています。S4 は、S2 と同じ概念です。

最後に、全結合層について見てみましょう。これらは分類器なので、正しい出力を抽出するため、前の層で抽出された特徴はすべて、入力のマッチングに使用されます。C1、S1、C2、S2 では、最初のパスでランダムな重みとバイアスを選択します。入力を CNN からの実際の出力と比較すると、精度は想定どおり非常に低くなります。ここでは、ほとんどの入力に対し、予測された出力が提供されたラベルと一致するように、モデルの精度を向上することが目標です。そのためには、費用関数に勾配降下法を利用します。最も簡潔に表すと、次のような式になります [4]。

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

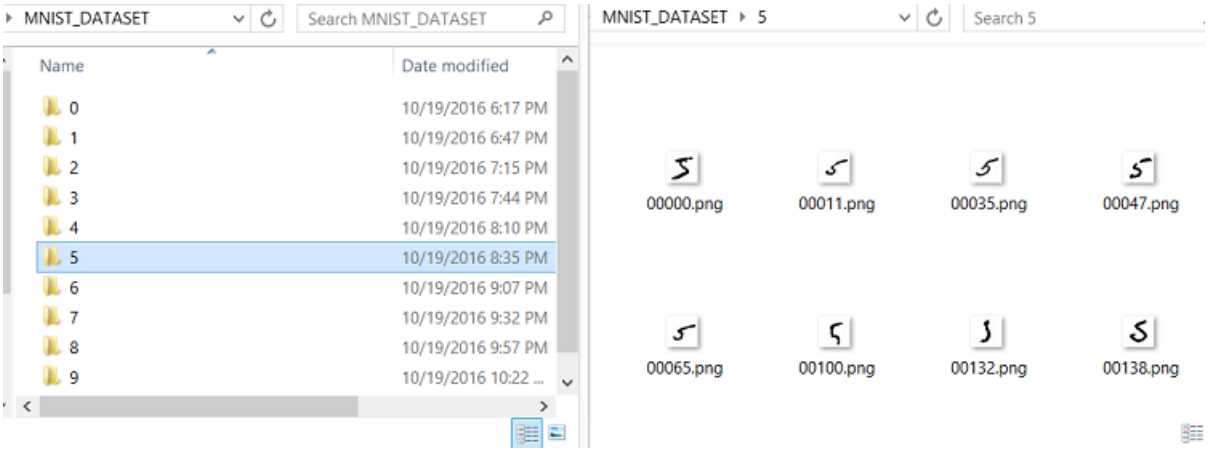
C は費用関数で、選択した重みとバイアスの関数です。重み "w"、バイアス "b" で抽出された特徴を使用して、任意の入力 "x" に対する実際の出力 "y" とすべての入力 "n" に対する予測される出力 "a" の間の差を最小に抑えることが目標です。

Caffe* で畳み込み、プーリング、全結合層を作成する方法と、インテル® Deep Learning SDK を使用して勾配やその他のハイパーパラメーターを選択する方法については後述します。

MNIST データセット

MNIST データセットは、7 万個の手書き数字のグレースケール画像のリポジトリーです [5]。各画像は 28 × 28 次元です。国勢調査局の職員と高校生によって収集された 2 つの NIST データセットから作成されています。データの変動率を上げるため、最終的な MNIST コレクションは、それぞれのデータセットからトレーニング用に 3 万画像ずつ、テスト用に 5 千画像ずつ使用しています。

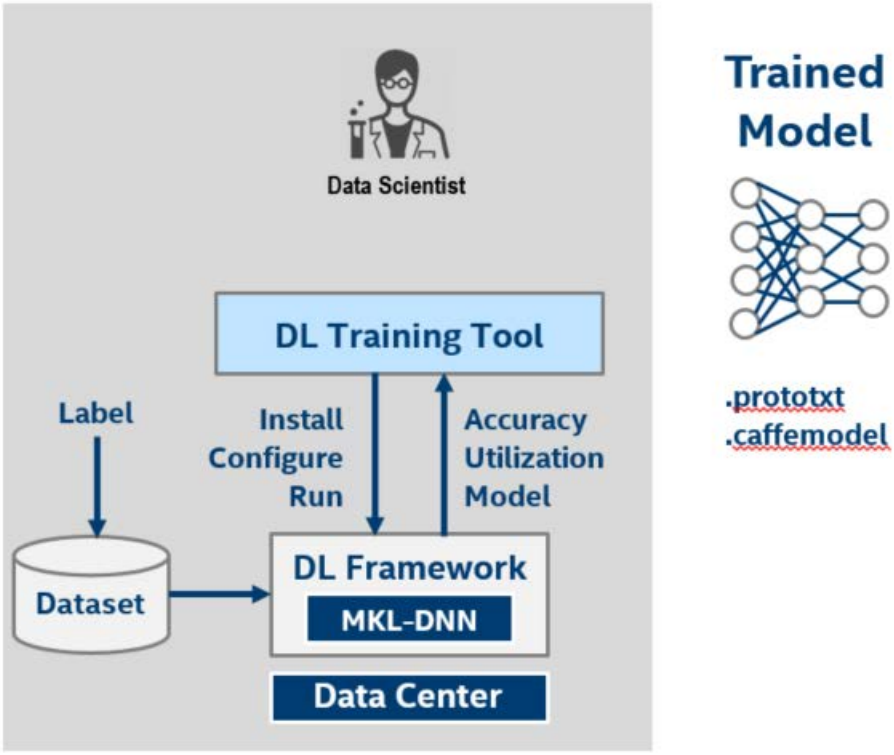
こちら (英語) からデータセットを取得できます。ここでは、前述の LeNet トポロジーと MNIST データセットを使用して、インテル® Deep Learning SDK に含まれるトレーニング・ツールの使い方を説明します。最初に、各フォルダー内に対応する画像を配置し、カテゴリーラベル (0、1…9) がディレクトリー構造の最上位になるようにデータを処理する必要があります。



準備ができたら、インテル® Deep Learning SDK を使用して、モデルのトレーニングを行います。インテル® Deep Learning SDK をまだインストールしていない場合は、こちらから (英語) ダウンロードできます。インストーラーには、Windows® 版と Mac* OS 版があります。インストーラーは、バックグラウンドで Docker* を使用して Ubuntu* 14.04 以上のマシンにインテル® Deep Learning SDK をインストールします。トレーニング・プロセスは、インテル® アーキテクチャー向けに最適化された Caffe* フレームワークで実行します。詳細は、[7] を参照してください。インテル® Deep Learning SDK のトレーニング・ツールは、パラメーターを操作し、トレーニング・プロセスを視覚化するためのグラフィカル・ユーザー・インターフェイス (GUI) を備えています。

インテル® Deep Learning SDK を使用してモデルをトレーニングする

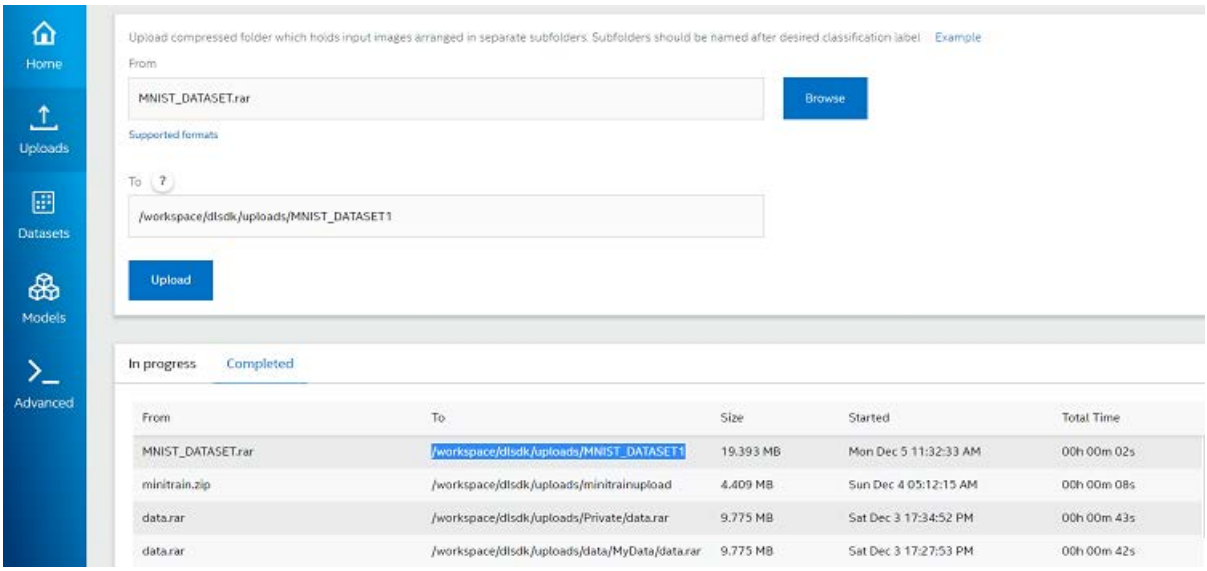
インテル® Deep Learning SDK の主な利点の 1 つは、モデルを簡単にトレーニングできることです。データ・サイエンティストは、簡単にトレーニング・データを準備し、可能な場合は既存のトポロジーを使用したり、必要に応じて新しいモデルを設計して、自動化されたテストと高度な視覚化を利用してモデルをトレーニングできます。トレーニング・ツールは、これらすべての利点を提供するとともに、一般的なディープラーニング・フレームワークを簡単にインストールできるようにします。以下は、トレーニング・ツールのスナップショットです。



既存のトポロジーからトレーニングされたモデルを生成するためのステップについて見てみましょう。最初に、トレーニング・ツールをインストールしたデバイスの IP アドレスとポートを使用して、トレーニング UI を起動します。ポート番号を指定しない場合、デフォルトで 8080 になります。インストール時に作成したユーザー名とパスワードを入力して、インターフェイスにログインします。

ステップ 1: データセットをアップロードする

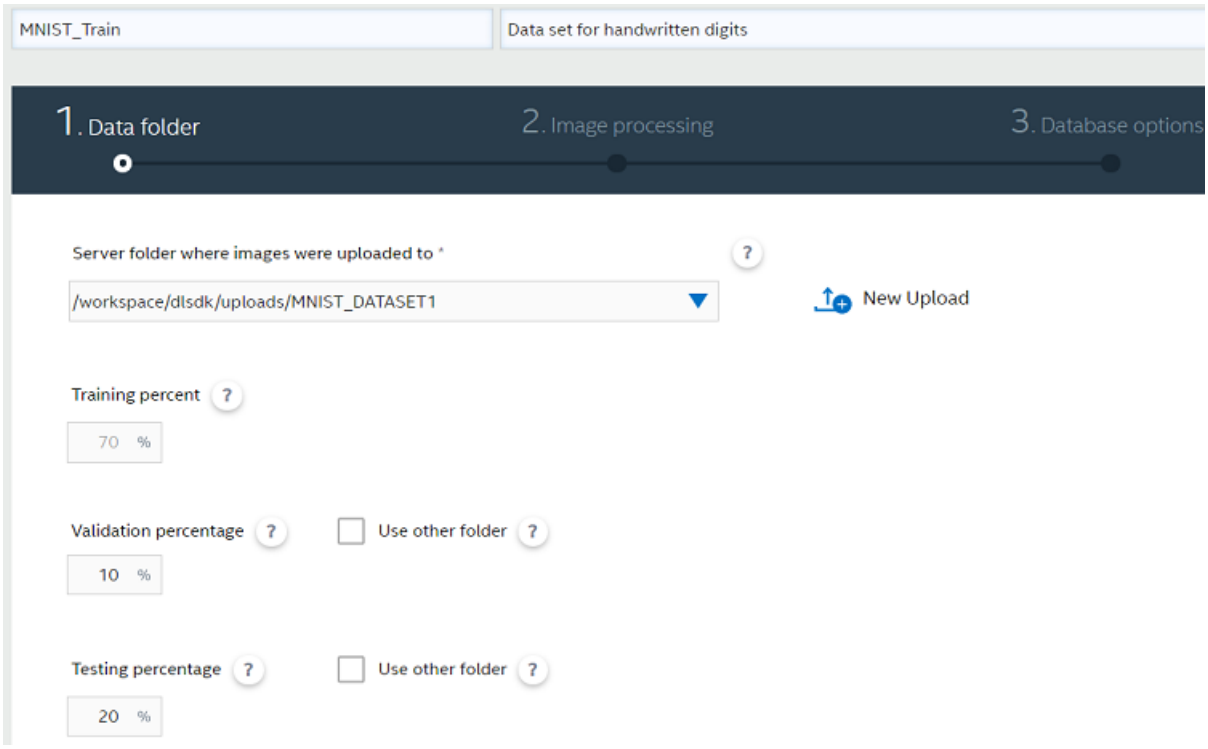
左の **[Uploads]** タブから、前述のラベル付きフォルダーと関連イメージを含むデータセットの zip ファイルを選択します。フォルダーパスを選択して、**[Upload]** をクリックします。完了すると、**[Completed]** セクションからアップロードしたデータセットのパスが取得できるようになります。



ステップ 2: 新しいデータセットを作成する

トレーニング/検証の割合 (%) とデータ拡大を選択する

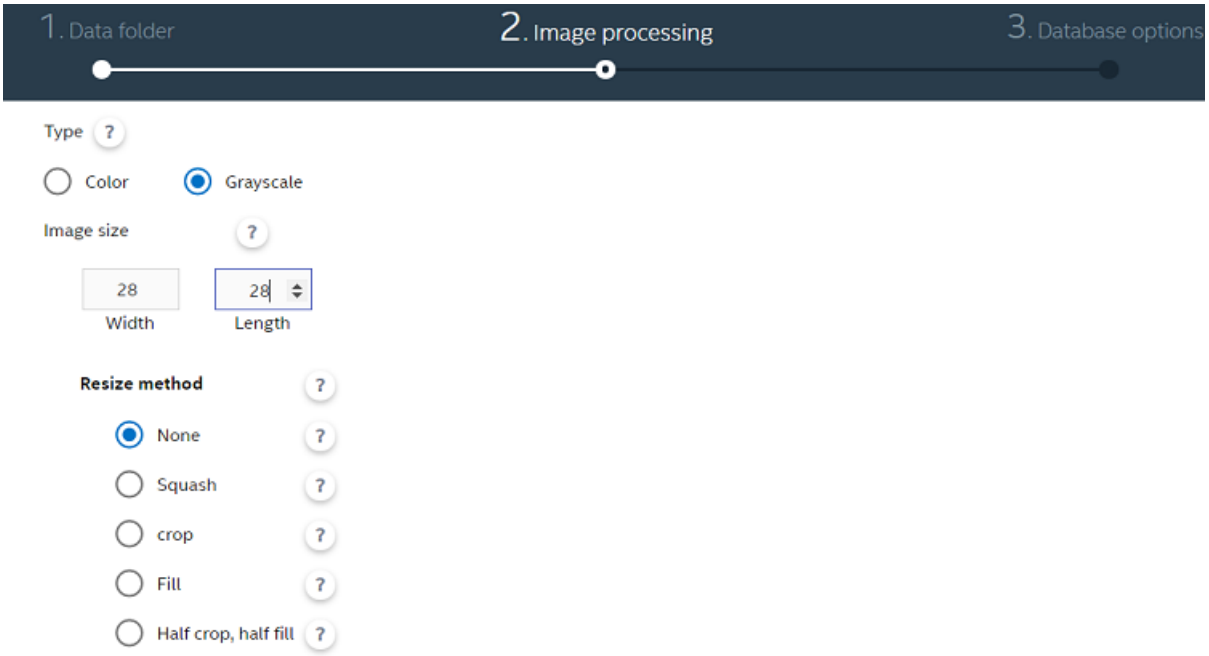
[Datasets] タブで、**[New Dataset]** をクリックします。データセットの名前を指定し、ドロップダウン・リストからアップロード済みのデータセットを選択します。トレーニング、検証、テストで使用するデータの割合 (%) を指定できます。



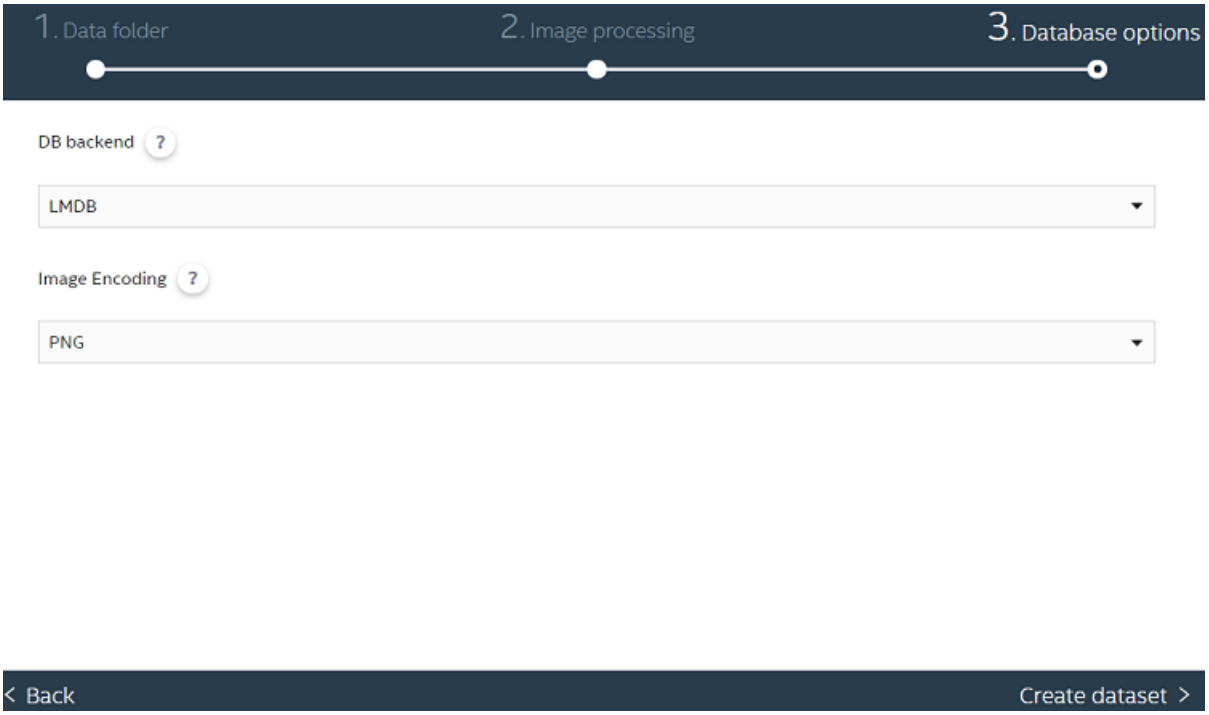
トレーニング・プロセスの効率は、データセットに含まれるデータの多様性に依存します。ベースのデータセットを変更せずに、変換を適用することでモデルへの入力を拡大することができます。拡大手法には、回転、双方向シフト、拡大/縮小、ミラーなどがあります。各手法の詳細は、[8] を参照してください。

データをプロセスする

ここで使用した MNIST データセットは、28 × 28 ピクセルのグレースケール画像であるため、次の設定を使用しました。データのサイズ変更が必要な場合は、利用可能なオプションのいずれかを使用します。各オプションの詳細は、ユーザーガイドを参照してください。

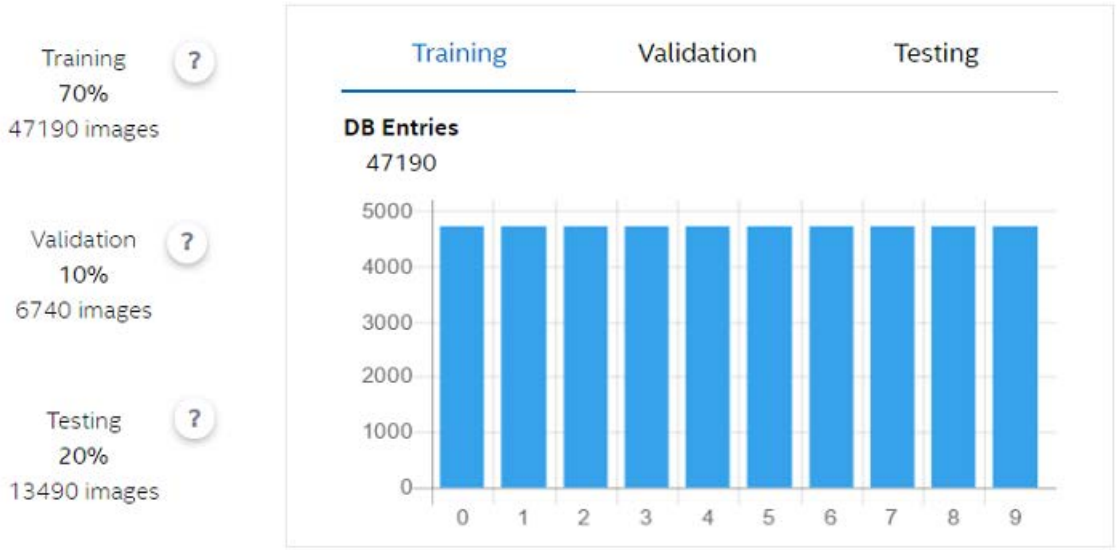


データベース・バックエンドとイメージ形式を選択する



データセットを作成する

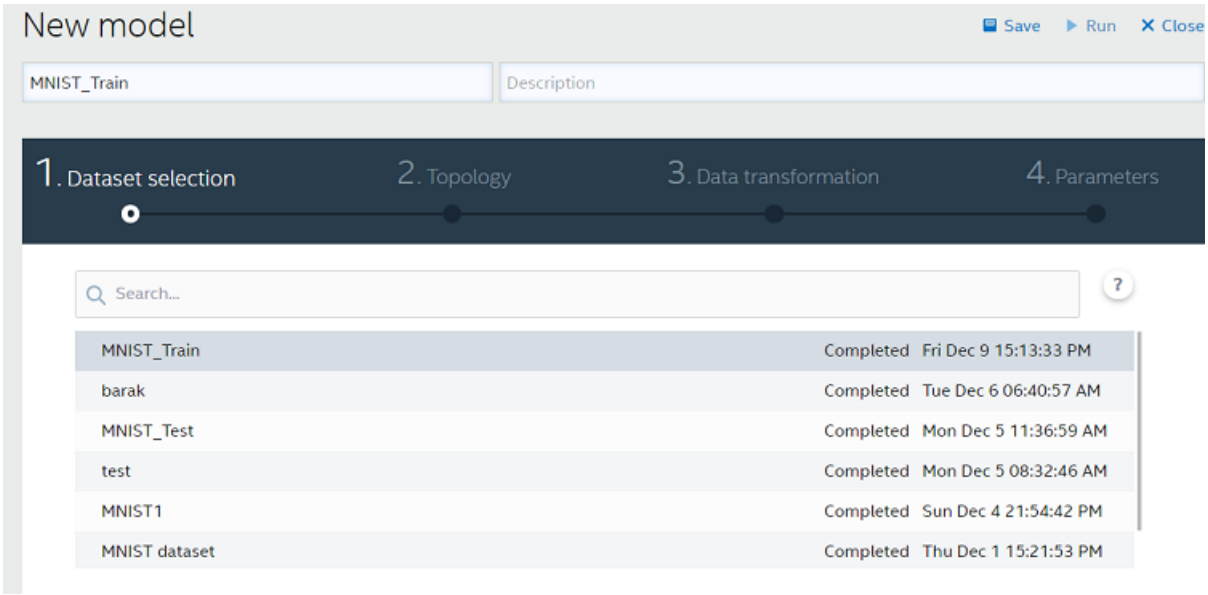
[Create Dataset] をクリックします。プロセスが完了すると、トレーニング・データセットとテスト・データセットの各ラベルのデータ数を視覚化できます。



ステップ 3: モデルを作成する

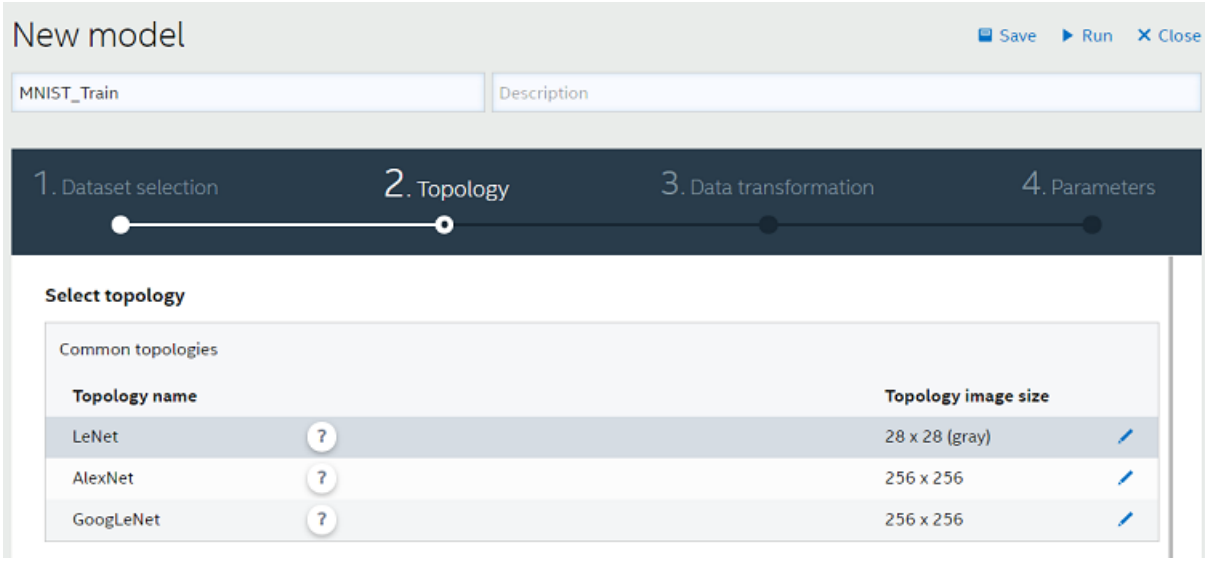
データセットを選択する

左の **[Models]** タブから、**[New Model]** をクリックして、モデルの名前を指定します (ここでは、データセットと同じ名前にしていますが、別の名前を指定することも可能です)。



トポロジーを選択する

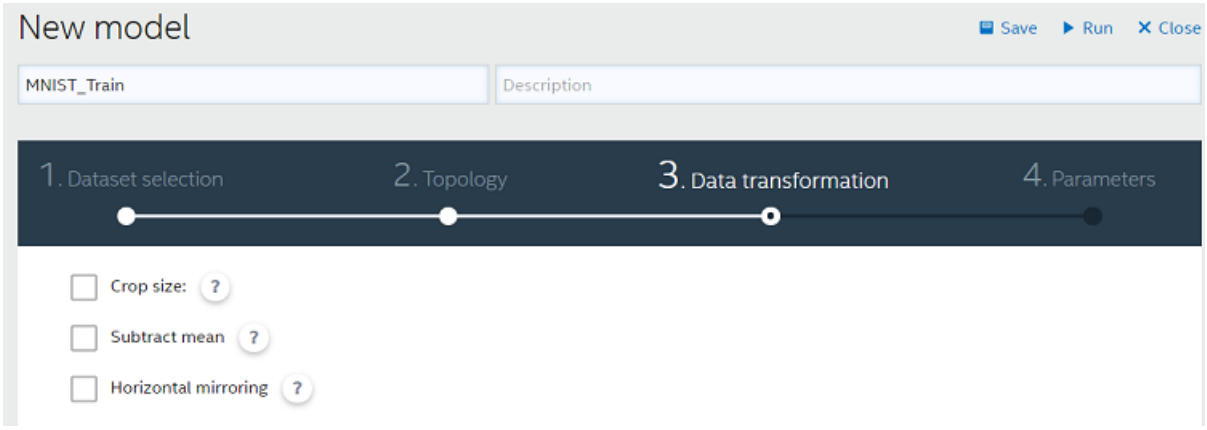
トレーニング・ツールは現在、LeNet、AlexNet*、GoogLeNet* の 3 つの画像認識トポロジーをサポートしています。ここでは、手書き数字認識用モデルのトレーニングに適した LeNet を選択します。ベースデータ (畳み込み層のチャネル数、プーリング層、全結合層など) は、背後で動作しているインテル® Distribution for Caffe* から取得できます。異なるモデルのトレーニングでは、例えば Cifar 100 や ImageNet からのカラー画像を使用する場合、AlexNet* または GoogLeNet* を使用することができます。



また、後述するように、LeNet、AlexNet*、GoogLeNet* をベースに新しいカスタムトポロジーを作成することもできます。

データ変換を実行する

トレーニング・プロセス時にランダムなデータが必要な場合、データセットのローデータを変更せずにデータ変換操作を行うことができます。データ変換ステップについては、ユーザーガイドで詳しく説明されています。ここでは、デフォルトの設定を使用します。



ハイパーパラメーターを選択する

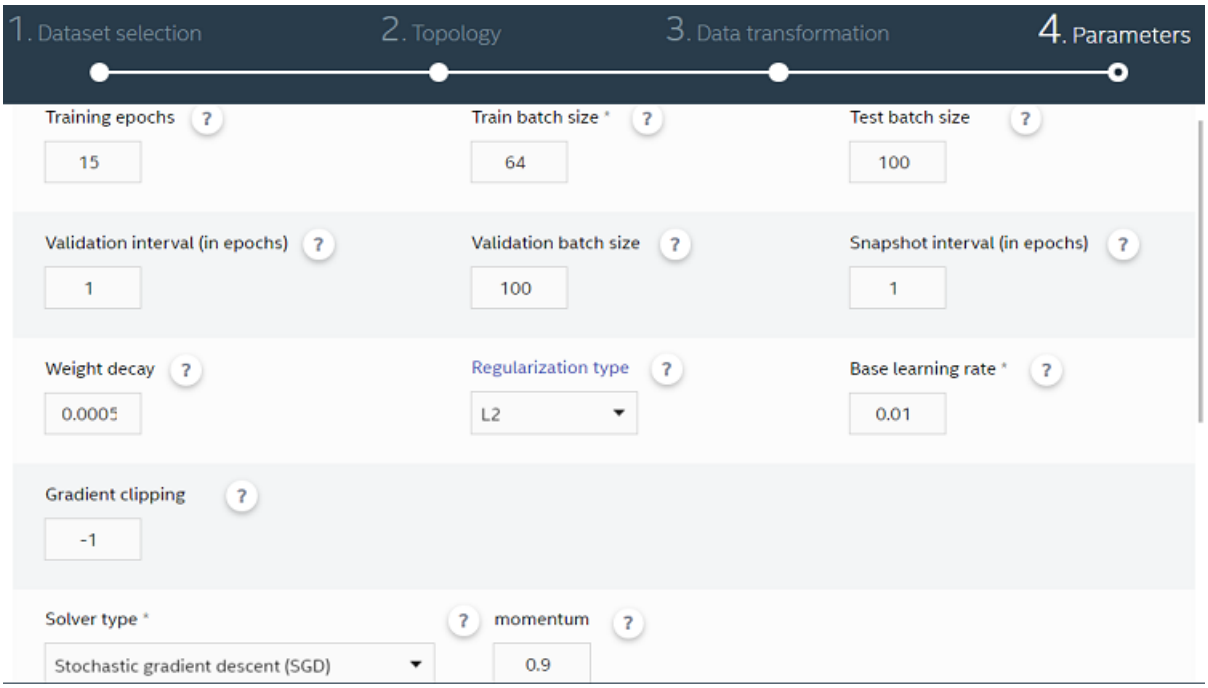
モデルを細かくチューニングするための設定を選択します。以下は、重要なパラメーターの一部です。

エポック (epoch): トレーニングにおいて、データセットのすべてのデータファイルを使用して CNN のすべての層を通る 1 つの完全パスを実行する回数を指定します。

バッチサイズ: 多くの場合、CNN データベースは大きいです。トレーニングを効率良く行うため、データベースを “x” 個のイメージのバッチに分割します。1 つのバッチが CNN を通る完全パスを反復と呼びます。大きなバッチサイズは、パラメーター更新プロセスの分散を軽減しますが、多くのメモリーを使用します。そのため、2 つのバランスをとることが重要です。

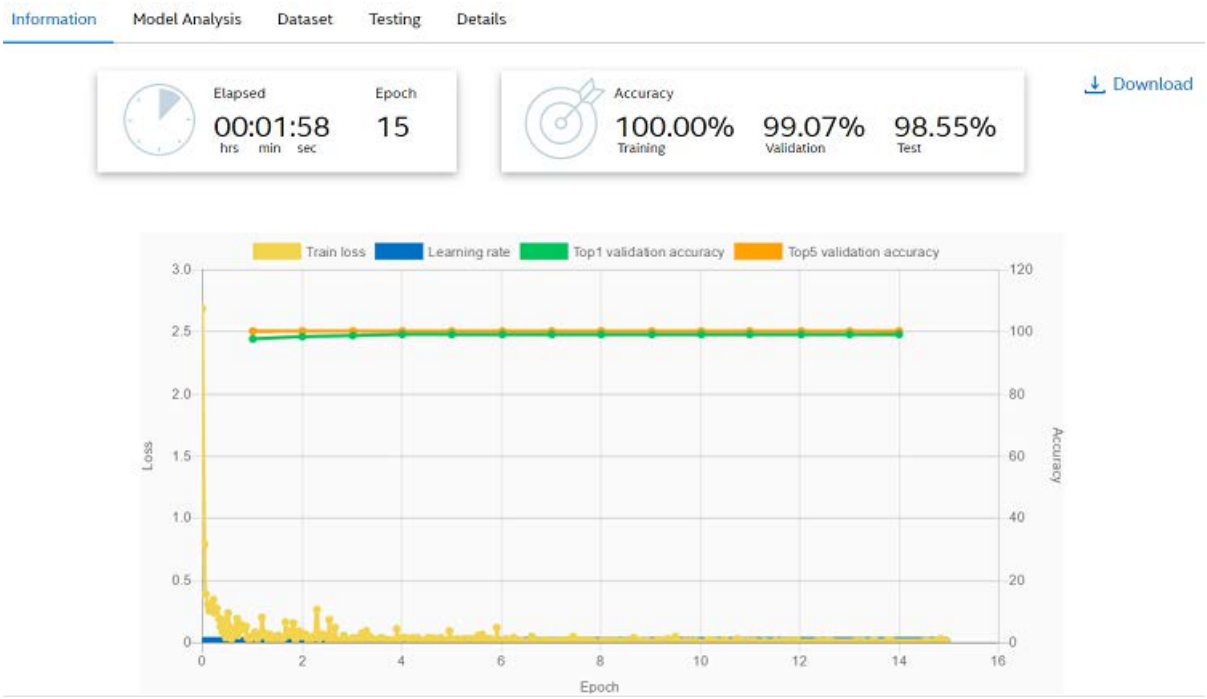
ベース学習率: 前述の勾配降下法アルゴリズムでは、最初のパスで重みとバイアスがランダムに選択されるため、精度が低くなります。グローバル最小値への収束を見つけるため、アルゴリズムのベース学習率を変更します。学習率を大きくすると高速に収束しますが、学習率が大きすぎるとモデルは収束しません。一方、学習率を小さくすると、グローバル最小値への収束が遅くなります。

ソルバーの種類: デフォルトは確率的勾配降下法です。連続するバッチで入力データセットのランダムなサンプルを使用して、より迅速にグローバル最小値への収束を取得します。ユーザーガイドにあるその他のオプションも使用できます。



これでモデルを実行する準備が整いました。実行が完了すると、実行の各エポックの精度と損失に加えて、最終トレーニング、検証、テストの精度を確認できます。

これでトレーニングは完了です。後でトレーニング・ツールによって生成された Caffe* ファイルのリストが必要になるため、**[Download]** をクリックしてファイルを保存します。次のセクションでは、これらのファイルについて、ここまででカバーした概念と紐付けて説明します。



Caffe* モデルファイルを理解する

ダウンロードした model.zip には、CNN を理解するのに必要なファイルがすべて含まれています。これらのファイルは、カスタムトポロジーを作成したり、モデルをデバッグする場合に重要です。また、配置プラットフォームでリアルタイム・データを使用してモデルを検証する際にも使用されます。アーカイブに含まれるいくつかのファイルについて詳しく見てみましょう。

最も重要なファイルは、モデルのアーキテクチャーを含む train_val.prototxt です [9]。

データ層

データ層について詳しく見てみます。

```
name: "MNISTLeNet"
layer {
    name: "MNISTFinal」
    type: "Data」
    top: "data」
    top: "label」
    include {
        phase: TRAIN
    }
    transform_param {
        mirror: false
        scale: 0.00390625
    }
    data_param {
        source: "/workspace/dl-genie/jobs/datasets/ca04ddcd-66b2-422d-8de8-
dcc1ad38cfd4/train.txt_LMDB」
        batch_size: 64
```

```
        backend: LMDB
    }
}
```

上記のコードは、“top” という 2 つのブロブを生成します。1 つはデータ用で、もう 1 つはラベル用です。“include” セクションは、作成するデータ層が TRAIN フェーズ用であることを示します。検証用にも同じ構造が必要になるため、train_val.prototxt ファイルには別のデータ層 (“include” セクションで TEST と示された) があります。次に、すべてのピクセルが 0 から 1 の範囲内になるようにスケーリングします (ここでは、1/256 = 0.00390625 を使用します)。そして、データソースがデータセットが生成される Docker* コンテナーをポイントするように設定します。バックエンドは LMDB に、バッチサイズはトレーニング・ツールの説明で示したとおり 64 に、それぞれ設定します。

畳み込み層

```
layer {
    name: "conv1"
    type: "Convolution"
    bottom: "data"
    top: "conv1"
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    convolution_param {
        num_output: 20
        kernel_size: 5
        stride: 1
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant"
        }
    }
}
```

前述の LeNet トポロジーから、最初の畳み込み層は 28 × 28 のバイナリーイメージを受け取ることが分かっています。これは “bottom:” パラメーターで指定します。このパラメーターは、データブロブが入力であることを示します。“param” セクションは、この層で学習可能な重みとバイアス・パラメーターの学習率の調整を指定します。重みの学習率 (lr_mult: 1) は、ハイパーパラメーターで示した値に設定します。バイアスの学習率は、重みの学習率の 2 倍に設定します。経験上、これらの設定により優れた収束率が得られます。

次に、畳み込み層のチャネル数を定義します。これは、“num_output” で定義されます。このチャネル数は、前述の LeNet トポロジーとは異なります。Caffe* フレームワークの LeNet トポロジーは、前述の LeNet アルゴリズムとやや異なります。ただし、概念は似ています。kernel_size は 5 × 5 ピクセル (LeNet-5 と同じ) に設定します。これは、“stride” で一度に 1 ピクセルずつ移動して、28 × 28 の入力画像をスキャンするのに使用されます。前述のとおり、重みとバイアスは最初の反復でランダムに選択され、トレーニングが進むにつれこれらのパラメーターは特徴を細かくチューニングする方法を学習します。ここでは、weight_filler は、一様分布 [-scale, scale] から重みをサンプリングする “Xavier” に設定します (scale = sqrt (3/ n)、n= 入力の数)。Caffe* で利用

可能なその他のフィルターについては、こちら (英語) を参照してください。次に、bias_filler を constant value = 0 に設定します。

サブサンプリング/プーリング層

```
layer {
    name: "pool1」
    type: "Pooling」
    bottom: "conv1」
    top: "pool1」
    pooling_param {
        pool: MAX
        kernel_size: 2
        stride: 2
    }
}
```

サブサンプリング層の定義は簡単です。入力として前の畳み込み層を受け取り、2 × 2 ピクセルの領域を取得し、“stride” で 2 ピクセル移動して、最大プーリングを実行します。オーバーラップはありません。サブサンプリングにより、特徴マップのサイズは、最初の畳み込み層のチャネル数の半分になります。

train-val.prototxt ファイルには、さらに 50 チャネルの 2 つ目の畳み込み層と 2 つ目のプーリング層がリストされた 2 つのセクションがあります。これらは上記と似ているため、ここでは説明しません。

全結合層

```
layer {
    name: "ip1"
    type: "InnerProduct"
    bottom: "pool2"
    top: "ip1」
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    inner_product_param {
        num_output: 500
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant"
        }
    }
}
layer {
    name: "relu1」
    type: "ReLU」
    bottom: "ip1」
```

```
    top: "ip1」
}
layer {
    name: "ip2」
    type: "InnerProduct」
    bottom: "ip1」
    top: "ip2」
    param {
        lr_mult: 1
    }
    param {
        lr_mult: 2
    }
    inner_product_param {
        num_output: 10
        weight_filler {
            type: "xavier"
        }
        bias_filler {
            type: "constant」
        }
    }
}
```

Caffe* では、“InnerProduct” は全結合層を指します。ここで注目すべき変更点は、チャネル数を 500 に設定していることです。正規化線形関数 (ReLu) は、要素単位で関数 f(x) = max(0,x) を実行します。最後の全結合層は 10 の信号を出力します。これは、バイナリー信号です。出力 n (例えば n= 3) の活性度は、トレーニングされたモデルが入力を 3 と予測したことを示しています。

損失層

```
layer {
    name: "loss」
    type: "SoftmaxWithLoss」
    bottom: "ip2」
    bottom: "label」
    top: "loss"
}
```

前述のように、CNN では、重みとバイアスを調整して将来の特徴抽出と予測を改善できるように、モデルの精度に関する情報が後方伝播されます。損失層は、入力としてラベルブロブと現在の予測を受け取り、損失を計算して、後方伝播時にデータを戻します。

層の定義には、特定のルールを追加することができます。例えば、トレーニング時のみ精度層を定義するには、次のようにルールを追加します。

```
layer {
   name: "accuracy_top5"
    type: "Accuracy"
    bottom: "ip2"
    bottom: "label"
```

```
    top: "accuracy_top5"
        include {
        phase: TEST
        }
        accuracy_param {
        top_k: 5
    }
}
```

solver.prototxt ファイルについて見てみましょう。このファイルには、トレーニング・ツールのインターフェイスで定義したハイパーパラメーターがすべて含まれています。これらのパラメーターを変更して、トレーニングへの影響を確認することができます。

```
# トレーニング/テスト用ネット・プロトコル・バッファーの定義
net: "/workspace/dl-genie/jobs/models/ce7c60e9-be99-47bf-b166-
7f9036cde0c8/train_val.prototxt"
# test_iter はテストするフォワードパスの数を指定
# MNIST では、バッチサイズ 100 とテスト反復数 100 を使用
# テスト用画像 10,000 すべてをカバー
# ネットワークのベース学習率、モーメンタム、重み減衰
base_lr: 0.01
momentum: 0.9
weight_decay: 5.0E-4
# 学習率ポリシー
lr_policy: "inv" gamma: 1.0E-4
power: 0.75
# 100 反復ごとに表示
display: 31
# 最大反復数
max_iter: 12645
# 中間結果のスナップショット
snapshot: 843
snapshot_prefix: "/workspace/dl-genie/jobs/models/ce7c60e9-be99-47bf-b166-
7f9036cde0c8/snapshot"
# ソルバーモード: CPU or GPU
solver_mode: CPU
type: "SGD"
```

インテル® Deep Learning SDK は、デフォルトでは CPU 上でトレーニングを実行します。Caffe* を手動で設定する必要はありません。この SDK を使用せずにインテル® アーキテクチャー向けに最適化された Caffe* でトレーニングを実行する場合は、こちらの記事 (英語) の手順に従ってください。

これで Caffe* モデルファイルの解釈方法が分かったため、トレーニング・モデル・インターフェイスを使用してトポロジーをカスタマイズする方法を紹介します。

インテル® Deep Learning SDK を使用してトポロジーをカスタマイズする

新しいモデルの作成プロセスを振り返ってみましょう。新しいモデル名を作成し、データセットを選択したら、**[Topology]** タブに移動して既存のトポロジーの中から 1 つを選択します。この例では、LeNet トポロジーを選択します。次に示す**編集**アイコンをクリックします。

選択したモデルの train_val.prototxt ファイルがテキストボックスに表示されます。

ここで、畳み込み、プーリング、全結合層のパラメーターを変更することができます。この新しいモデルの名前を入力し、新しいトポロジーを保存します。保存後、作成したカスタムトポロジーのリストが利用可能になります。

これで、前述の残りのトレーニング・プロセスに進むことができます。完了後、トレーニング・ツールからダウンロードしたモデルファイルを使用して、デプロイメント・プラットフォームでリアルタイムに予測を行うことができます。

参考資料 (英語)

[1] 画像認識に畳み込みニューラル・ネットワークを使用する

[2] 汎用画像認識向けニューラル・ネットワーク・アーキテクチャー

[3] ドキュメント認識に勾配ベースの学習を適用する

[4] ニューラルネットを使用して手書き数字を認識する

[5] MNIST データベース

[6] インテル® Deep Learning SDK

[7] インテル® Deep Learning SDK – トレーニング・ツールのインストール・ガイド

[8] インテル® Deep Learning SDK – トレーニング・ツールのユーザーガイド

[9] MNIST と Caffe* で LeNet をトレーニングする

[10] インテル® アーキテクチャー向けに最適化された Caffe* でディープ・ニューラル・ネットワークをトレーニングして配置する

著作権と商標について