



@ksasao 2019年05月26日に更新



NSSOL Advent Calendar 2018 10日目

# Windows PCかつオフライン環境でお気軽に画像認識する ML.NET 1.0 対応版

[Windows](#) [C#](#) [機械学習](#) [転移学習](#) [ML.NET](#)

## TL;DR

Windows, つよいGPUがない, インターネットに画像をアップできない, アドミン権限がないなどの環境でも動作する、転移学習を利用した画像認識アプリを .NET Framework (Windows Forms) で作成しました。

アプリケーションをダウンロードし、分類したい名前を付けたフォルダに数枚ずつ画像を用意するだけで短時間で簡単に学習ができます。Pythonなどをインストールする必要もありません。色々なものを分類して最近の画像認識の精度を体感してみてください。

アプリケーションは、Microsoft がオープンソースで開発を進めている[ML.NET](#) (ML.NET単体では Linux, Macなどでも動きます) を利用しています。実装されている TensorFlow API を利用して、学習済みの Inception モデルを読み込み、画像特徴を抽出し、[Stochastic Dual Coordinate Ascent\(SDCA\)](#) を利用した多クラス分類器を用いて転移学習をします。

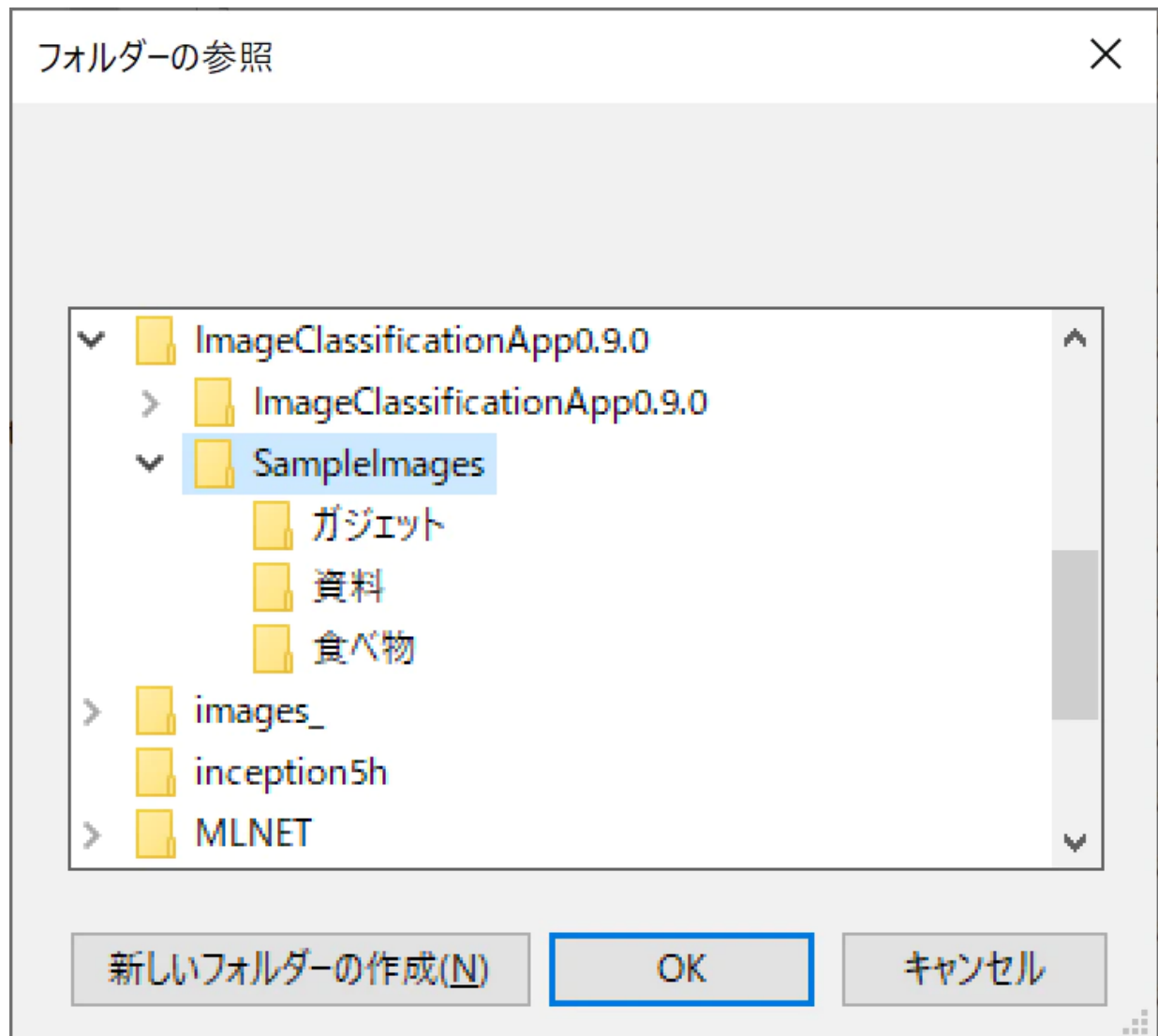
- [ソースコード](#)
- [アプリ](#) (Windows 10/x64/4GB以上のメモリ/Core i 程度のCPU/.NET Framework 4.6.1以降で動作)

# アプリケーションの利用方法

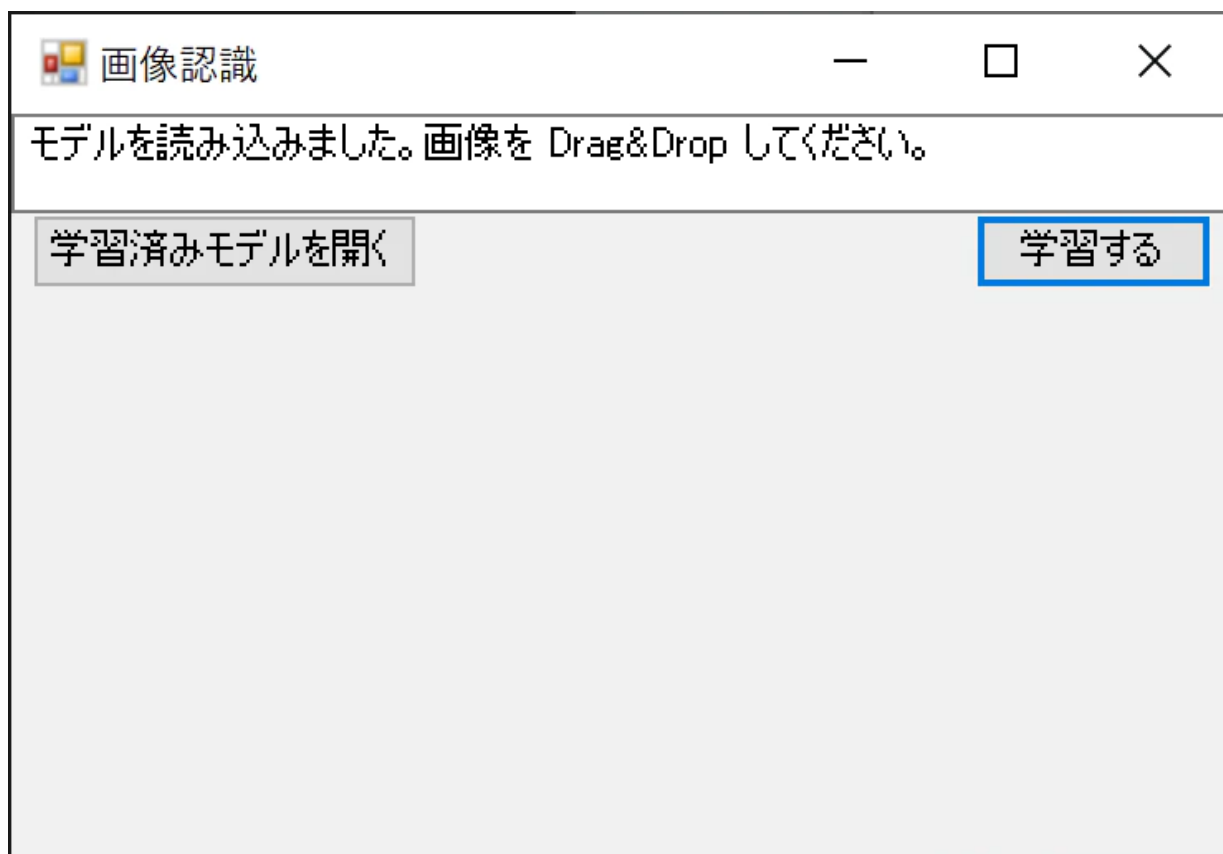
1. **アプリ**をダウンロードして展開し、

ImageClassificationApp1.0.0\ImageClassificationApp.exe を起動します。

2. 画像を学習するには、[学習する] のボタンをクリックし、開いたダイアログから画像を分類済みのフォルダ(下図の場合は SampleImages フォルダ)を選択してOKをクリックします。

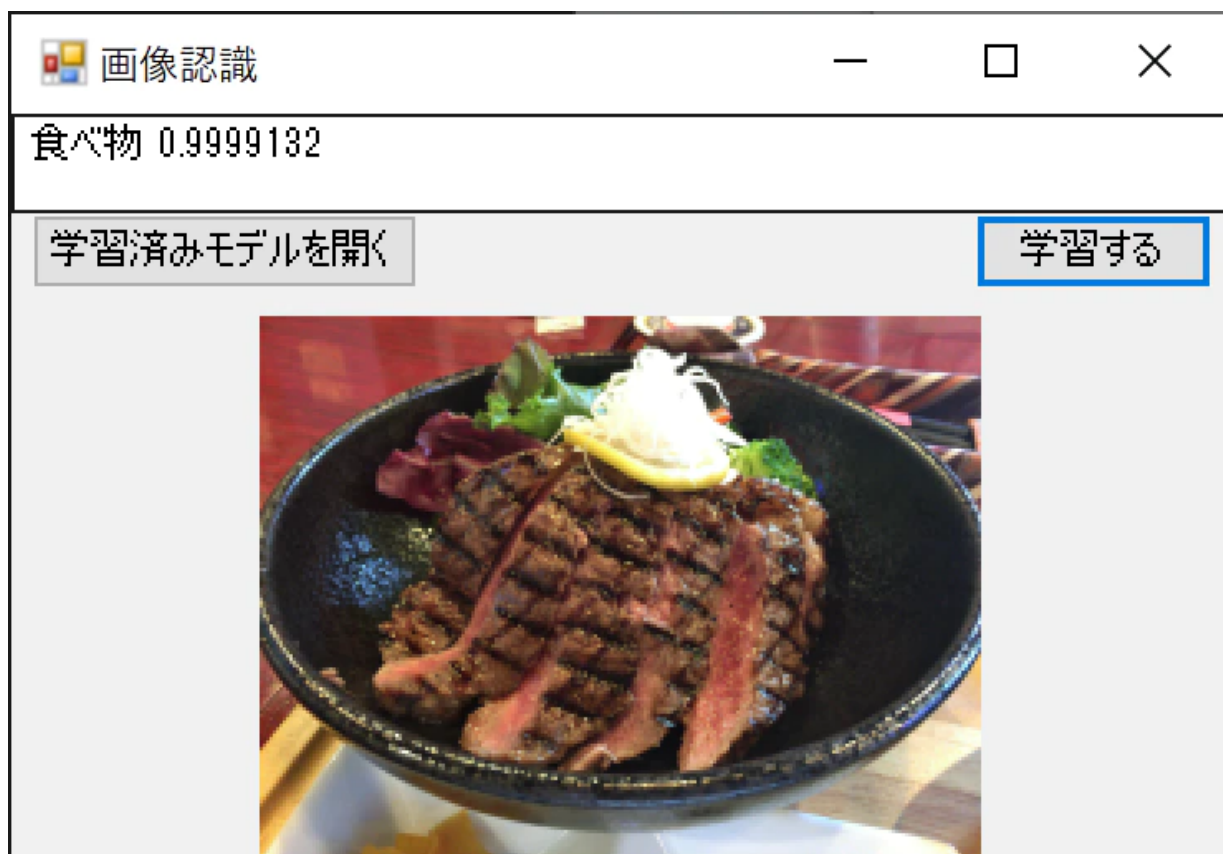


### 3. 10秒から1分程度で学習が完了します。



下図のように表示されたら画像を Drag & Drop すると認識結果と確信度(0~1, 1が最

も高い)が表示されます。



4. あらかじめ学習したモデルを利用することも可能です。先ほど画像を学習したフォルダに `imageClassifier.zip` というファイルが生成されていますので、それを読み込んでみてください。学習済みモデルのファイル名は適宜変更可能です。また学習に利用した画像も不要です。



## .NET 開発者向けの解説

# ML.NET とは

---

ML.NET は Microsoft がオープンソースで進めている .NET 開発者向け機械学習フレームワークです([GitHub](#))。クロスプラットフォームで開発されており、Windows, Linux, macOS で動作します。Windows Hello, Bing Ads PowerPoint デザインアイデアなどでも使われています。

類似のものとして、[Windows Machine Learning](#) がありますが、こちらは Windows 10 以降の OS に特化したものとなります。API等に互換性はありません。

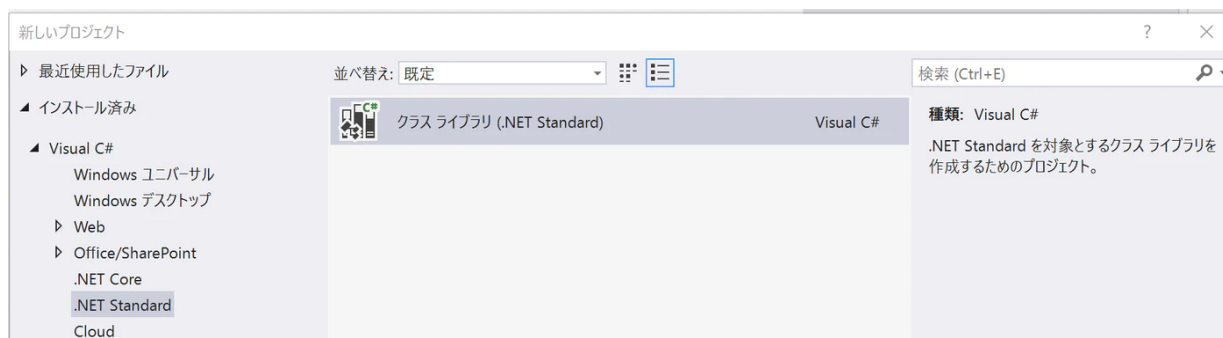
## .NET Core 版 ML.NET を Windows Forms から利用する

---

ML.NET はクロスプラットフォームを実現するため .NET Core で作成されています。.NET Core で作成した DLL は、.NET Framework で作成しているアプリケーションからは直接参照することはできませんが、.NET Standard ライブラリを作成し、その中で .NET Core を利用するコードを記述すれば、.NET Framework から参照できるようになります。今回は ~~ML.NET 0.7.0 (.NET Core 2.0) を利用しているため、.NET Standard 2.0 / .NET Framework 4.6.1 以降が必要です。また、依存しているネイティブライブラリが x64 を対象としているため 64bit環境でのみ動作します。~~

~~なお、2018年12月時点の最新版である ML.NET 0.8.0 は、.NET Core 2.1 ベースとなっており、.NET Framework から利用するためには、.NET Standard 2.1 が必要となりますが未リリース~~です。2019年5月時点の ML.NET 1.0 では .NET Core 2.1 ベースになっていますが、この記事の範囲内では .NET Core 2.0 でコンパイルしても動作するため、.NET Standard 2.0 でビルドしています。ビルドには .NET Framework 4.6.1 以降が必要です。また、依存しているネイティブライブラリが x64 を対象としているため 64bit環境でのみ動作します。

1. .NET Standard 2.0 のクラスライブラリのプロジェクトを作成し、nuget を利用して、必要なライブラリ(今回は、Microsoft.ML 1.0.0, Microsoft.ML.ImageAnalytics 1.0.0, Microsoft.ML.TensorFlow 0.12.0)を追加する



2. 1.のプロジェクトを .NET Framework のプロジェクトから参照する
3. .NET Framework のプロジェクトの .csproj を開き、  
    <RestoreProjectStyle>PackageReference</RestoreProjectStyle> を最初の  
    <PropertyGroup> 内の先頭に追加する

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="15.0" xmlns="http://schemas.microsoft.com/developer/msbuild
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Comm
  <PropertyGroup>
    <RestoreProjectStyle>PackageReference</RestoreProjectStyle>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
```

- #### 4. x64でビルドする

下記のようなメッセージが出てデバッグ実行に失敗する場合は、10秒くらい待ってもう一度実行してください。

The OutputPath property is not set for project 'CoreReferenceForm.csproj'. Please check to make sure that you have specified a valid combination of Configuration and Platform for this project. Configuration='Debug' Platform='x64'. This error may also appear if some other project is trying to follow a project-to-project reference to this project, this project has been unloaded or is not included in the solution, and the referencing project does not build using the same or an equivalent Configuration or Platform. CoreReferenceForm

## .NET Framework 版 ML.NET

Windows プラットフォーム限定となりますが、ML.NET の .NET Framework 版も提供されています。.NET Framework プロジェクトから、上記と同様に利用することが可能です。

## 参考

- [.NET Standardなライブラリプロジェクトを作成して参照する](#)
- [Referencing .NET Standard Assemblies from both .NET Core and .NET Framework](#)

## 実装について

---

ML.NETのサンプルにある [Image Classification](#) をベースにしています。以下は説明のため部分的に抜粋しています。詳しくは [ソースコード](#)を参照してください。

## 学習

ML.NET では下記のようにパイプラインを構成していきます。

```
var pipeline = mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: LabelToKey,
    .Append(mlContext.Transforms.LoadImages(outputColumnName: "input", imageFc
    .Append(mlContext.Transforms.ResizeImages(outputColumnName: "input", image
    .Append(mlContext.Transforms.ExtractPixels(outputColumnName: "input", inte
```



```
.Append(mlContext.Model.LoadTensorFlowModel(featurizerModelLocation).
    ScoreTensorFlowModel(outputColumnNames: new[] { "softmax2_pre_activation" })
.Append(mlContext.MulticlassClassification.Trainers.LbfgsMaximumEntropy(learningRate: 0.01f))
.Append(mlContext.Transforms.Conversion.MapKeyToValue(PredictedLabelValue, ActualLabelValue))
.AppendCacheCheckpoint(mlContext);
```

上記のコードでは

- 画像ファイルとその画像のラベル(クラス名; 日本語 (UTF-8) が利用可能)が記載された.tsvファイル読み込み
- 画像のリサイズ
- Tensorflowモデルの読み込みと特徴抽出に利用する層(softmax2\_pre\_activation)の指定
- **Stochastic Dual Coordinate Ascent(SDCA)** を利用した画像特徴の多クラス分類

といったような各処理をつないでいます。パイプラインの構成は、画像に限らず、自然言語処理や数値(CSVファイル)データの学習でも同様です。詳しくは、ML.NETの[チュートリアル](#)や[サンプルコード](#)を参照してください。

パイプラインができたなら、そこに学習データを渡して学習を行います。

```
var data = mlContext.Data.LoadFromTextFile<ImageNetData>(path: dataLocation, hasHeader: false);
var model = pipeline.Fit(data);
```

つづいて、できたモデルを評価します。

```
var classificationContext = mlContext.MulticlassClassification;
ConsoleWriteHeader("Classification metrics");
var metrics = classificationContext.Evaluate(trainData, labelColumnName: LabelToKey, predictedLabelColumnName: PredictedLabelToKey);
Console.WriteLine($"LogLoss is: {metrics.LogLoss}");
Console.WriteLine($"PerClassLogLoss is: {String.Join(" , ", metrics.PerClassLogLoss.Select(m => m.Key))}");
```

モデルの保存も簡単です。

```
mlContext.Model.Save(model, trainData.Schema, outputModelLocation);
```



## 推論

推論はモデルを読み込んで、画像ファイルを渡すだけです。作成したモデルにはラベル(クラス名)も含まれています。

```
// モデル読み込み
loadedModel = mlContext.Model.Load(modelLocation, out var modelInputSchema);
var predictor = loadedModel.MakePredictionFunction<ImageNetData, ImageNetPrediction>(env);

// 画像読み込み
var predictor = mlContext.Model.CreatePredictionEngine<ImageNetData, ImageNetPrediction>(1
var testData = ImageNetData.ReadImage(filename, "---");

ImageNetPrediction data = predictor.Predict(testData);
return data.PredictedLabelValue + " " + data.Score.Max();
```

[編集リクエスト](#)[📄 ストック](#)[LGTM](#)

52

**Kazuhiro Sasao** @ksasao[フォロー](#)**日鉄ソリューションズ株式会社**

お堅いと評判のユーザ系Slrです。※各記事の内容は個人の見解であり、所属する組織の公式見解ではありません。

<https://www.nssol.nipponsteel.com>

ユーザー登録して、Qiitaをもっと便利に使ってみませんか。

[登録する](#)


ログインする

---

---

---

---

 この記事は以下の記事からリンクされています



PowerPointを使って動画を作成してみる からリンク 1 year ago

 コメント



@hyougohr520

2019-07-05 13:56 ...

Ryzenでも使えますか？



@ksasao

2019-07-09 21:27 ...

ML.NET はこの例の使い方ではGPUを使用していません。Ryzenでも使えると思います。



@bettilovebsk

2019-07-24 10:31 ...

sipeedのMAiX DOCKで利用できるモデルを作ることはこの手順で可能でしょうか。



あなたもコメントしてみませんか :)

[ユーザ登録](#)すでにアカウントを持っている方は[ログイン](#)

How developers code is here.



Qiita

[About](#) [利用規約](#) [プライバシー](#) [ガイドライン](#) [API](#) [ご意見](#) [ヘルプ](#) [広告掲載](#)

## Increments

[About](#) [採用情報](#) [ブログ](#) [Qiita Team](#) [Qiita Jobs](#) [Qiita Zine](#)

© 2011-2020 Increments Inc.