Neural Network Console クラウド版
スターターガイド
-物体検出(CenterNet)編-

ソニーネットワークコミュニケーションズ株式会社

概要

本ドキュメントではNeural Network Console(NNC)で物体検出モデルを作成する一連の流れをまとめました。

サンプルプロジェクトを利用してモデル作成を行う流れになっていますので、Deep Learningモデルの設計ノウハウが無い方でも取り組み易い構成になっています。

また、ウェブにも物体検出の解説動画(実践Deep Learning：物体検出)を準備しておりますので、モデルのネットワーク設計の詳細を理解されたい場合には、こちらも併せてご覧ください。

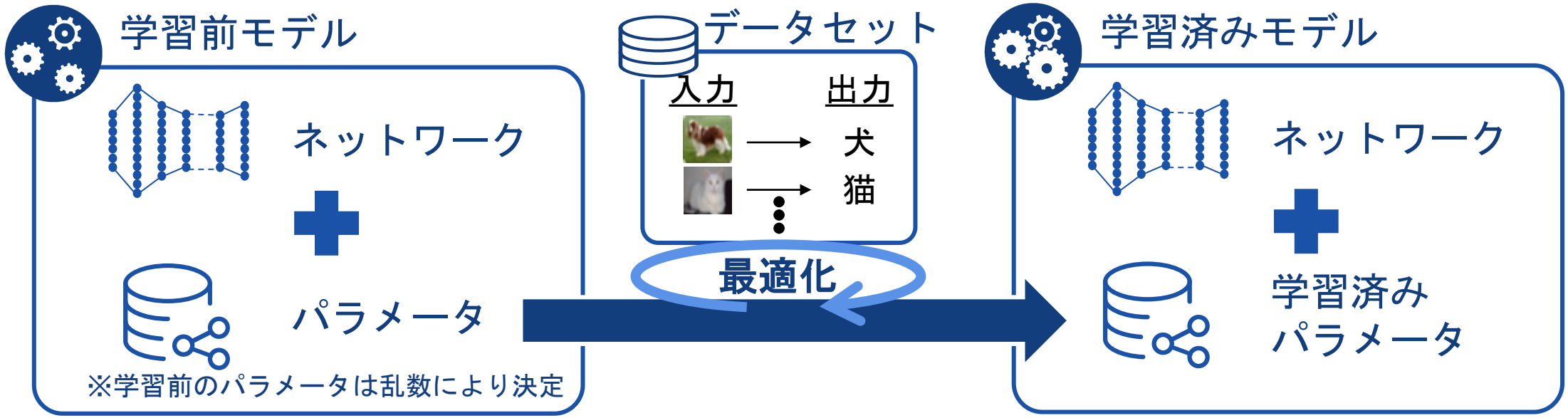
目次

Deep Learningモデルを作るとは

Deep Learningモデルとは分類や予測などを行うためのアルゴリズムです。モデルはネットワークとパラメータに分解できます。よくDeep Learningは脳の神経構造に例えられますが、ネットワークとは回路図で、パラメータとはその上の抵抗値のようなものです。

モデル作成とは、目的に合わせたネットワークを構築し、準備したデータセットを用いてパラメータを最適化する作業です。データセットによるパラメータの最適化を学習と呼び、学習をしてできたモデルを学習済みモデルと呼びます。

Deep Learningの学習



Deep Learningを用いた物体検出

Deep Learningを用いた物体検出では、画像の中にある複数の物体を認識し、おおよその位置がわかります。

Deep Learningを用いた画像認識技術には様々なものがあり、より正確な位置や形状を知る場合には領域抽出が適切ですし、画像そのものを分類する場合には画像分類で十分です。

様々な画像認識技術の中から解決したい問題やモデル作成の期間・工数に応じて適切な手法を選択する必要があります。

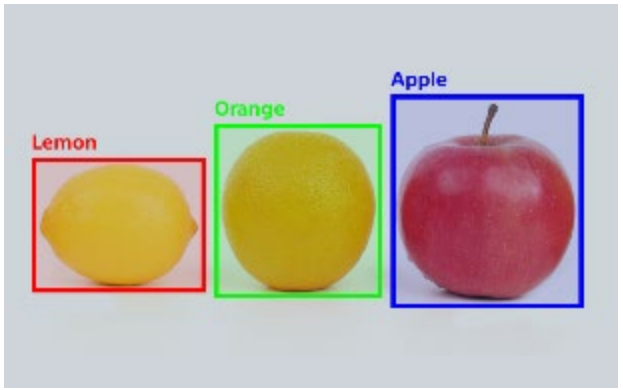
Deep Learningを用いた画像認識技術

画像分類

入力画像の種類を予測する

物体検出

入力画像の中にある物体の種類とその位置を予測する

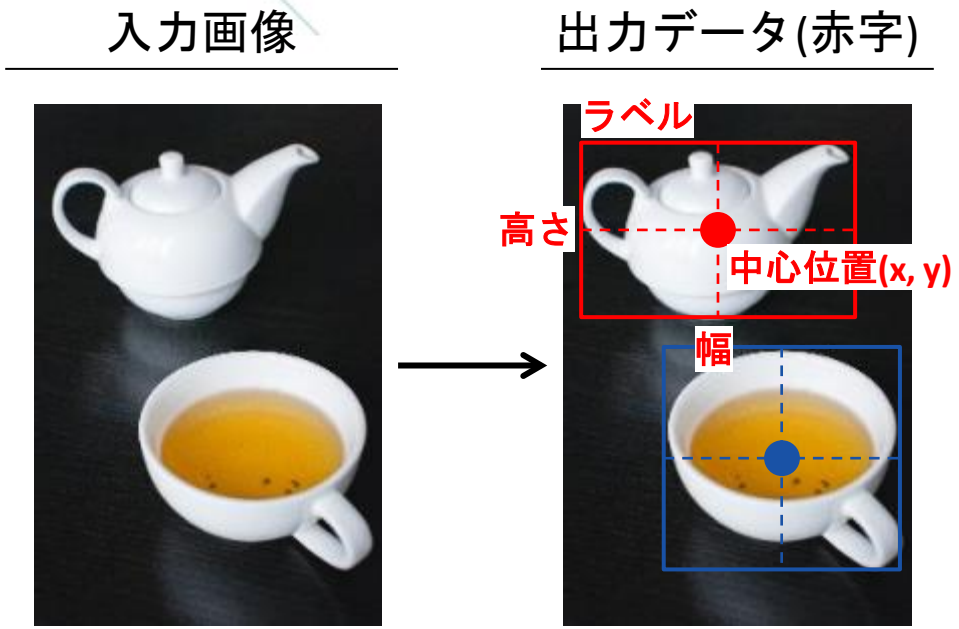


領域抽出

ピクセル単位で入力画像を物体の種類を予測する

物体検出の基本的な入出力

物体検出では画像データを入力として、検出対象である物体のラベルと物体を囲う長方形(バウンディングボックス)を出力します。バウンディングボックスを定義するために、その中心位置と高さ、幅を記載することが一般的です。



目次

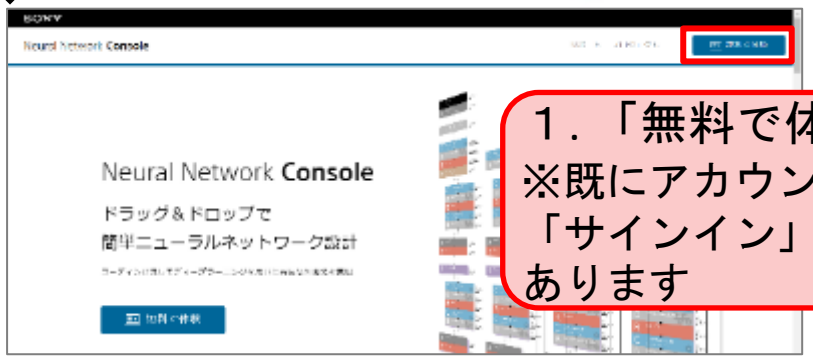
サインインページへの移動

Chromeを利用して、https://dl.sony.com/jaに移動します。

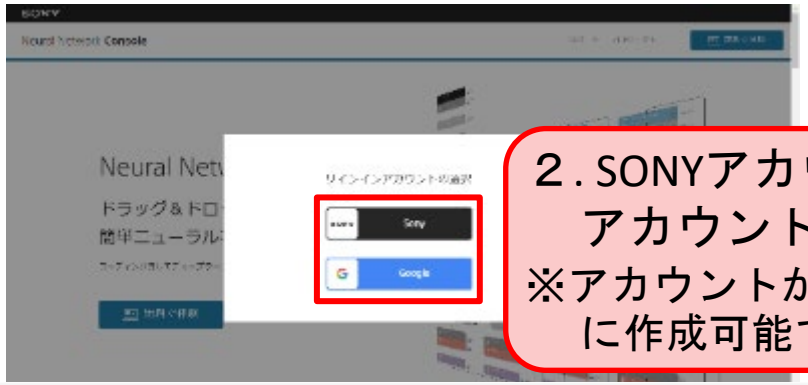
ページの右上にある「無料で体験」をクリックします。

サインインするためのアカウントをSONYアカウントまたはGoogleアカウントから選択します。どちらを選択してもこのドキュメントの内容は進めることが可能です。

サインインページへの移動方法



1．「無料で体験」をクリック※
※既にアカウント作成済みの場合は「サインイン」と表示される場合があります



2．SONYアカウントまたはGoogleアカウントのどちらかを選択
※アカウントがない場合には選択後に作成可能です

GoogleアカウントとSonyアカウントの違い

Sony アカウント	✓Sonyアカウントを利用しているその他のサービスとのアカウント連携が可能 ✓Sonyアカウントを既にお持ちの場合は、アカウント作成不要でNNCの利用が可能
Google アカウント	✓Googleアカウントを利用しているその他のサービスとのアカウント連携が可能 ✓Googleアカウントを既にお持ちの場合は、アカウント作成不要でNNCの利用が可能

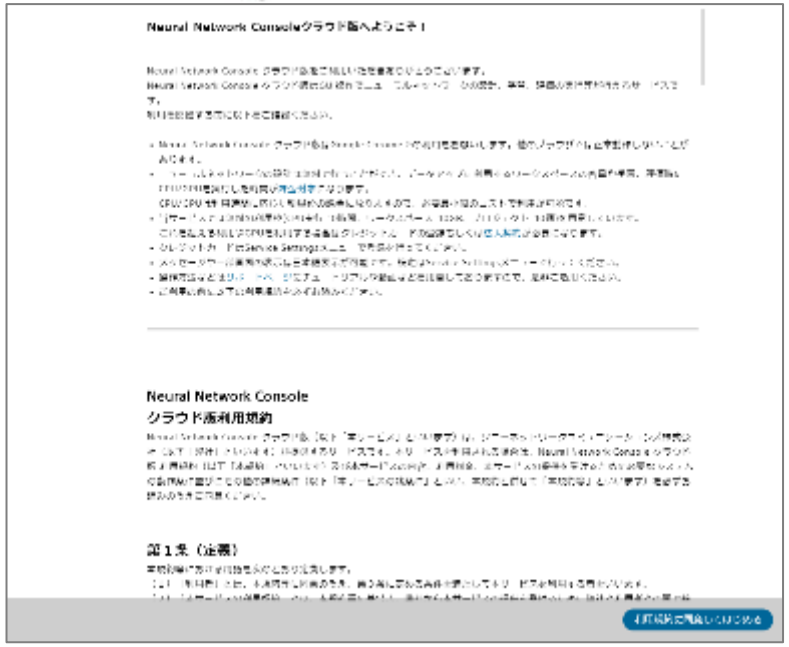
SONYアカウントでのサインイン

SONYアカウントに登録しているメールアドレス・パスワードを入力し、ログインを行います。

※アカウントをお持ちでない方は、「新しいアカウントの作成」から新規作成を行ってください。
詳細は、AppendixのSONYアカウントの取得方法に記載があります。

1．メールアドレスの入力

2．利用規約への同意



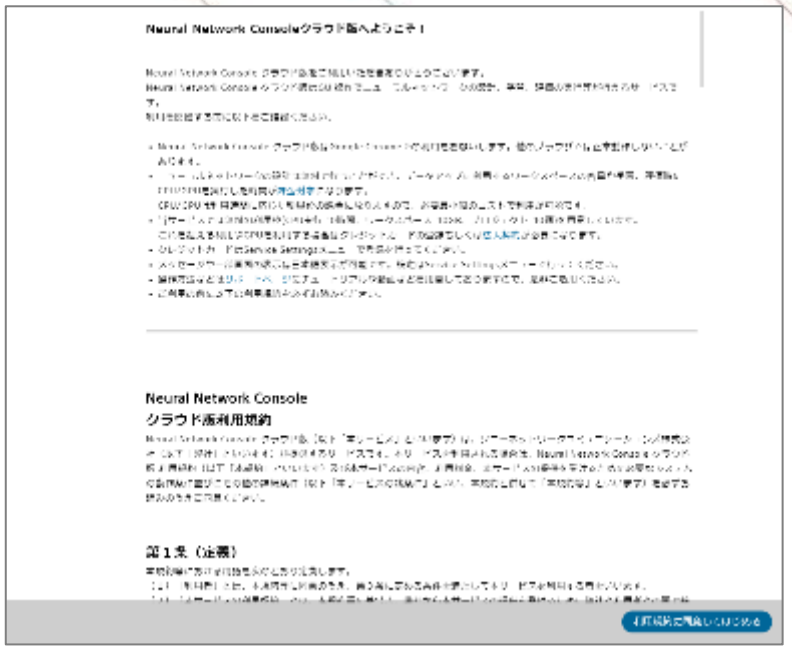
Googleアカウントでのサインイン

Googleのメールアドレス・パスワードを入力し、ログインを行います。

1．メールアドレスの入力

2．パスワードの入力

3．利用規約への同意



目次

データの有無と本パートの進め方について

本ドキュメントではデータをお持ちの方とそうでない方の両方を読者として想定しています。
データがない方はNNCのサンプルデータをご利用いただくか、オープンデータをご利用ください。
以下でそれぞれにあった進め方を確認してください。

１．データをお持ちの方

アノテーション(物体のラベルや位置の入力)やデータセット分割などの作業が必要ですので、次頁以降を読み進めてください。ご準備いただいたデータの画像サイズは事前に統一していただく必要はありません。

２．データがなく、サンプルデータを利用される方

データをお持ちでない場合でも、NNCのサンプルデータを用いて本ドキュメントを進めることが可能です。
NNCのサンプルデータの詳細はサンプルデータの説明に記載がありますので、ここから読み進めることも可能です。

３．データがなく、オープンデータを利用される方

データがお持ちでない場合でも、UCI Machine Learning RepositoryやKaggle等で公開されているウェブ上のオープンデータを利用することで、データセット作成から一連のモデル作成を体験できます。データをダウンロードのうえ、「１．データをすでにお持ちの方」と同様に、次頁以降を読み進めてください。ダウンロードしたデータの画像サイズは事前に統一していただく必要はありません。

データセット作成のステップ

画像データに物体のラベルとバウンディングボックスの対応付けをしたデータセットを作成し、本ドキュメントで用いるモデル構築手法を利用するためのフォーマット変換を行います。

データセットは学習用と検証用に分割したのちに、それぞれNNCにアップロードします。

ステップの2,3はNNCのPlugin機能を用いることで、一度に実施することができます。

データセット作成のステップ

	ステップ	概要	利用ツール
1	アノテーション作業	画像データと物体のラベル・バウンディングボックスの対応付け(アノテーション)作業を行います	アノテーションツール※
2	フォーマット変換 (グリッドの導入)	グリッドという学習効率化と速度向上のための手法を用いるため、フォーマット変換をします	NNCのPlugin機能
3	データセット分割	データセットを学習用と検証用の２つに分割します	
4	データセットアップロード	２つのデータセットをNNCにアップロードします	NNC専用のアップローダ

※ アノテーション作業は特化したツールをご利用ください。詳細はアノテーション作業に記載

アノテーション作業

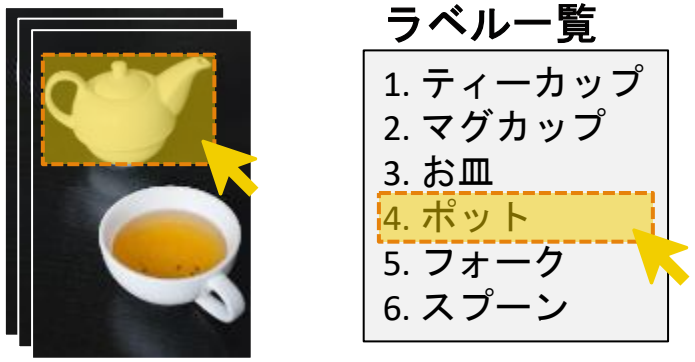
画像データと物体のラベル・バウンディングボックスの対応付けには、アノテーションに特化したツールをご利用ください。

代表的なアノテーションツールは下表になります。ここでは、次のステップで変換が必要なためYOLOフォーマットで出力可能なツールのみを記載しています。

データセットの保存はYOLOフォーマットをご利用ください※1。

アノテーションツールのイメージ

アノテーションツールではGUIを通して、バウンディングボックスとラベルを設定することができます



YOLOフォーマットで出力が可能な代表的なアノテーションツールの例

ツール名	URL
VoTT V1※2	https://github.com/Microsoft/VoTT
labelImg	https://github.com/tzutalin/labelImg

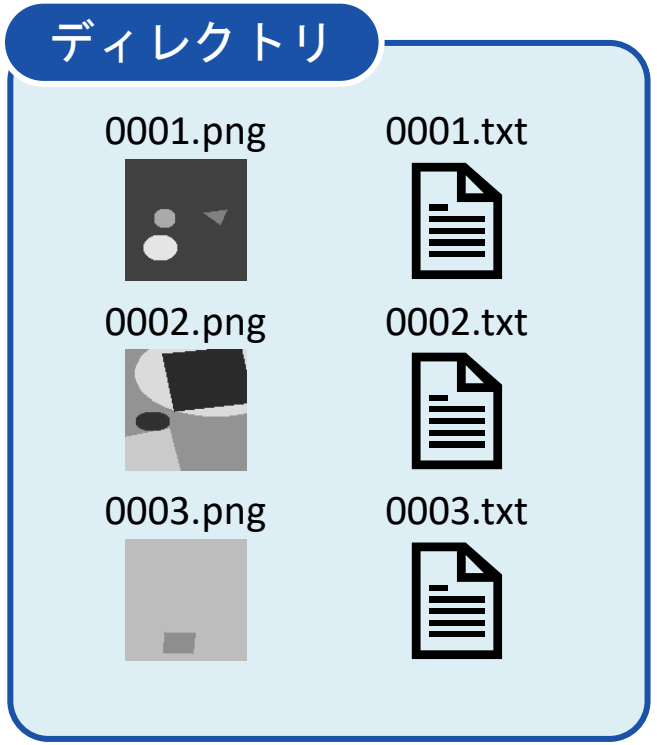
※1 YOLOフォーマット以外で出力されるツールをご利用の場合には、プログラミングなどでフォーマットを変換いただく必要があります。
※2 VoTT V2ではYOLOフォーマットでの出力機能がありませんのでご注意ください。

(補足) YOLOフォーマット

YOLOフォーマットでは画像と同じディレクトリに、画像ファイル名と同じ名称のテキストデータを作成し、そこに物体のラベルとバウンディングボックスの情報を保存します。

ディレクトリ構造の例

ディレクトリ



画像とアノテーションデータの対応

0001.txt

0	0.333	0.488	0.170	0.151
0	0.289	0.730	0.269	0.205
1	0.741	0.477	0.183	0.129

ラベル　中心位置の横、縦　バウンディングボックスの幅、高さ

※それぞれの大きさは画像サイズで規格化

※1 ファイル名は連番である必要はありませんが、NNCでは日本語などの2バイト文字に対応していませんので、ご注意ください。
※2 NNCではpng、jpg、jpeg、gif、bmp、tifの画像フォーマットに対応しています。

フォーマット変換: グリッド

画像を格子状に分割したグリッドを用いて、物体のおおよその位置を予測することで学習効率とモデル精度を向上できます。

グリッドを用いることでフォーマットが2次元の表データとなり、バウンディングボックスの中心が含まれるセルにグリッド内での相対位置(詳細はAppendix参照)とバウンディングボックスのサイズが記載されます。

ラベルはPluginによる変換でガウシアンフィルタによりその周囲にも重みが展開されます。また、 相対位置はグリッドサイズで規格化します。さらにバウンディングボックスのサイズはグリッドサイズで規格化した後に対数変換します。Pluginによる出力はAppendixを参照してください。

画像サンプル

YOLOフォーマット

0	0.44	0.30	0.68	0.76
1	0.61	0.66	0.70	0.50

ラベル　中心位置の横、縦　バウンディングボックスの幅、高さ

※YOLOフォーマットではそれぞれの大きさは画像サイズで規格化

グリッドによる変換後フォーマット

グリッドごとのラベルの重み

0	0.0001	0.01	0.0001	0
0	0.01	1	0.01	0
0	0.0001	0.01	0.0001	0
0	0	0	0	0
0	0	0	0	0

グリッド内での相対位置(横)

0	0	0	0	0
0	0	0.2	0	0
0	0	0	0	0
0	0	0	0.05	0
0	0	0	0	0

グリッド内での相対位置(縦)

0	0	0	0	0
0	0	0.1	0	0
0	0	0	0	0
0	0	0	0.3	0
0	0	0	0	0

バウンディングボックスの幅

0	0	0	0	0
0	0	0.53	0	0
0	0	0	0	0
0	0	0	0.54	0
0	0	0	0	0

バウンディングボックスの高さ

0	0	0	0	0
0	0	0.58	0	0
0	0	0	0	0
0	0	0	0.40	0
0	0	0	0	0

※Pluginによる変換では、ラベル以外は物体の中心が含むセルのみに、ラベルはそれに加えて周囲のピクセルに値が出力され、それ以外のセルは0で埋められます。

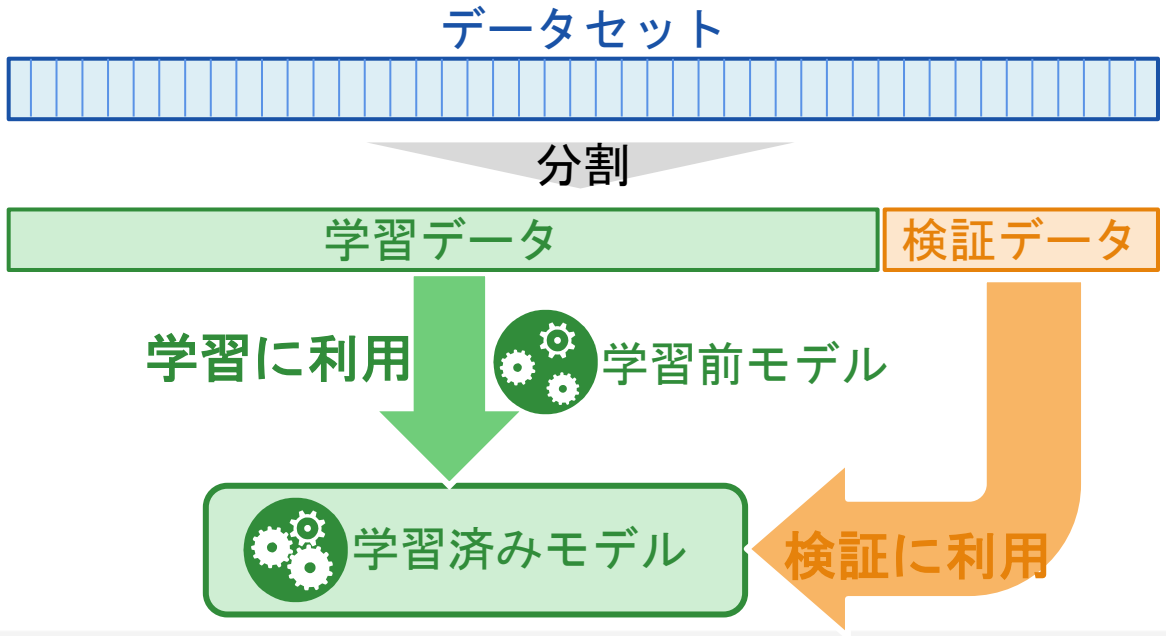
データセット分割

モデル作成には、学習に利用するデータセット（学習データ、Training Data）と、モデル精度を検証するデータセット（検証データ、Validation Data）の２つが必要になります。

作成したモデル精度を正しく検証するためには、学習に利用していないデータを準備する必要があるため、あらかじめデータセットを学習データと検証データに分割しておきます。

このとき、学習データと検証データの分割割合は7:3や8:2が一般的です。
Pluginによる変換では引数にそれぞれの割合を指定することができます。

データセットの件数については、Deep Learningモデルはデータ数が多ければ多いほど、精度が高くなる傾向があります。(参考: データの重要性)



Plugin機能の利用

NNC Windowsアプリ版に搭載されたPlugin機能から、pythonファイルを利用することで、グリッドを用いるためのフォーマット変換、入力画像サイズの統一とデータセット分割を一度に実行することができます。

Plugin機能でconvert_dataset_for_centernet.pyを利用するには、以下のディレクトリ構成でpythonファイルを保存してください※1。追加するPlugin関連のファイルはこちらからダウンロードできます。

その後、データセットタブからデータセット作成ボタンをクリックし、「Convert object detection dataset for centernet.」をクリックすることでconvert_dataset_for_centernet.pyを利用することができます。

```
neural_network_console/                                    ※2
  └ libs/
      └ plugins/
        └ _Post_Precoss/
          └ _Create_Dataset
            ├ convert_dataset_for_centernet.py
            ├ create_object_detection_dataset.py
            └ for_centernet/
              └ __init__.py
```

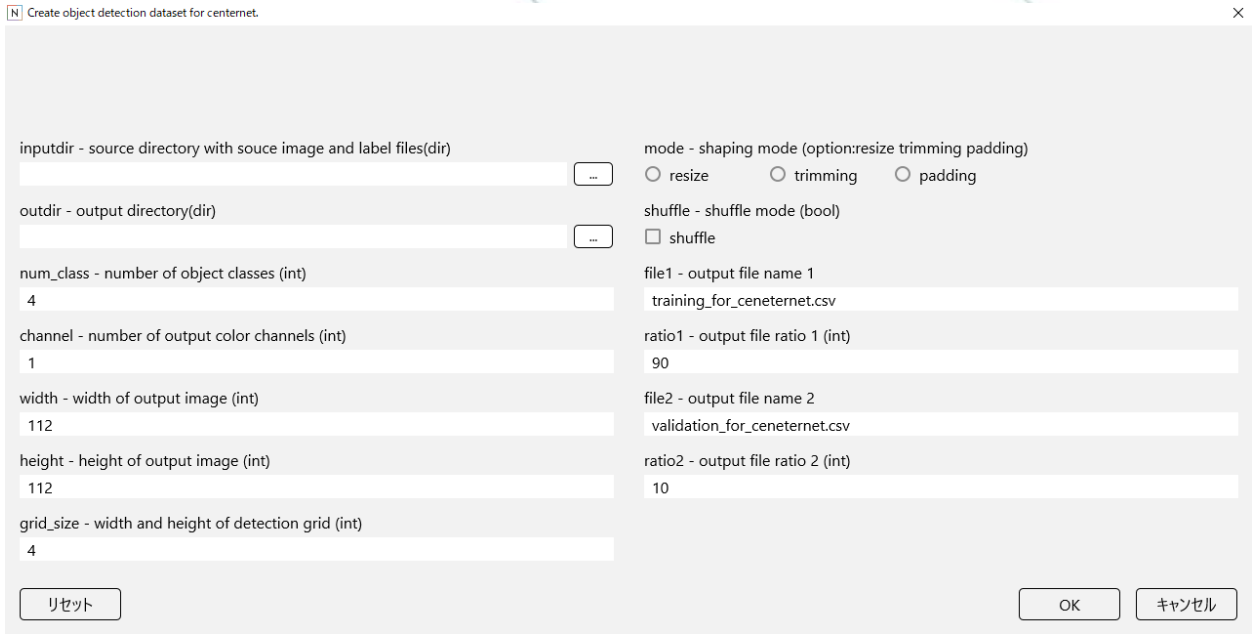
※1 Version2.1.0以前のバージョンを使用している場合はplugins直下に赤字のファイル・フォルダを配置し、評価タブから評価結果を右クリックし、「プラグイン」→「convert_dataset_for_centernet」をクリックすることで利用できます。
※2 要追加ファイル・フォルダは赤字で記載しています。

Pluginでのデータセット変換の実行

convert_dataset_for_centernet.pyを利用することで、グリッドを用いるためのフォーマット変換、入力画像サイズの統一とデータセット分割を一度に実行することができます。

Plugin実行後の出力の詳細は、次頁で実行例とともに記載をしております。

GUIの操作画面



修正が必要なパラメータの説明

パラメータ	説明	パラメータ	説明
inputdir	変換前データのディレクトリ 値を指定しない場合は「neural_network_console/samples/sample_dataset/synthetic_data/object_detection/original_for_centernet」が指定されます。	mode	画像修正方法(resize/trimming/padding) 画像サイズを変更しない場合は、resizeを指定します。
outdir	変換後の出力ディレクトリ 値を指定しない場合はneural_network_console/samples/sample_dataset/synthetic_data/object_detection/data_for_centernetに出力されます。	shuffle	ランダム分割(true/false)
num_class	判別クラス数	file1	学習データファイル名
channel	モノクロ(1)/カラー(3)	ratio1	学習データ割合
width	入力画像の幅	file2	出力検証データ名
height	入力画像の高さ	ratio2	検証データ割合
grid_size	グリッドサイズ		

コマンドラインでのデータセット変換の実行

convert_dataset_for_centernet.py はコマンドライン上でも実行することが出来ます。

《コマンドライン上での実行方法》

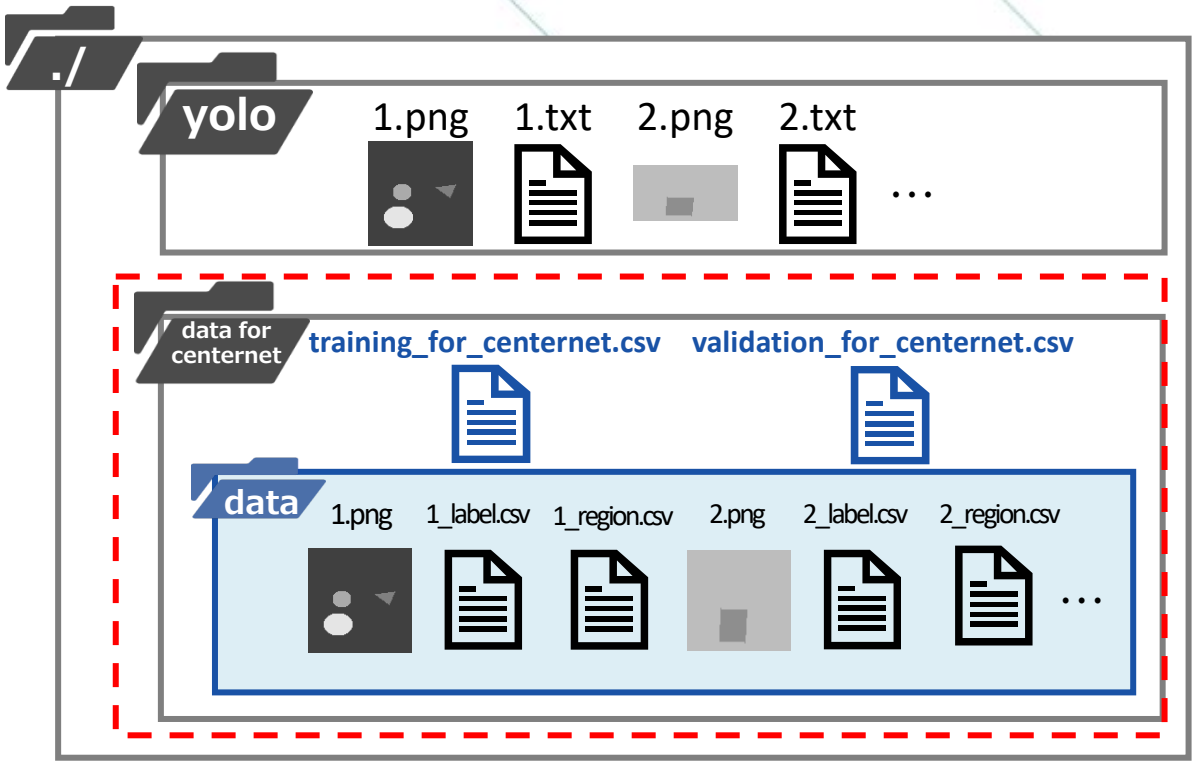
python convert_dataset_for_centernet.py -i データのディレクトリ -o 出力ディレクトリ -nc 分類クラス数 -ch モノクロ(1)/カラー(3) -w 入力画像の幅 –ht 入力画像の高さ -g グリッドサイズ –m 画像修正方法(resize/trimming/padding) –s ランダム分割(true/false) -f1 学習データファイル名 -r1 学習データ割合 -f2 出力検証データ名 -r2 検証データ割合 --shuffle

黄字が変更箇所になります。

データセット変換の実行例

Pluginの実行後、出力ディレクトリに学習用と検証用のデータセットを取りまとめたCSVファイルが出力されます。また、サイズ統一をした入力画像と出力データを記載したCSVファイルがdataディレクトリの中に出力されます。

《出力ディレクトリの例》



コマンド実行により生成

《出力ファイルの説明》

	出力ファイル	説明
1	training_for_centernet.csv	学習用データセットの入出力の対応関係を取りまとめたCSVファイル
2	validation_for_centernet.csv	検証用データセットの入出力の対応関係を取りまとめたCSVファイル
3	data/[画像ファイル名].png	元画像を入力画像サイズにサイズ統一した画像
4	data/[画像ファイル名]_label.csv	画像にある物体のラベル情報を示したcsvファイル
5	data/[画像ファイル名]_region.csv	画像にある物体のバウンディングボックスの情報を示したcsvファイル

データセットのアップロード

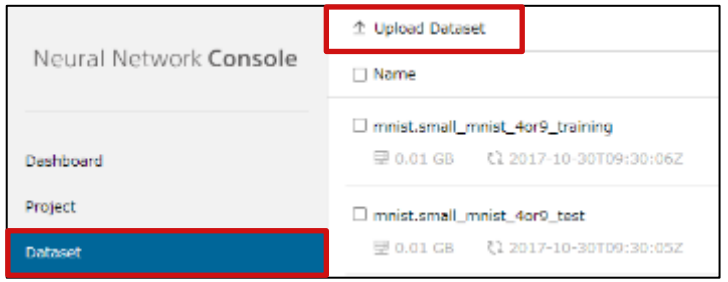
NNCはクラウドサービスのため、専用のアップローダを利用し、モデルを作成するために必要なデータセットをあらかじめクラウドにアップロードする必要があります。
アップローダ上で準備したCSVファイルを指定し、データをアップロードします。
アップロードは学習用と検証用の２回行う必要があります。

１．アップローダの取得

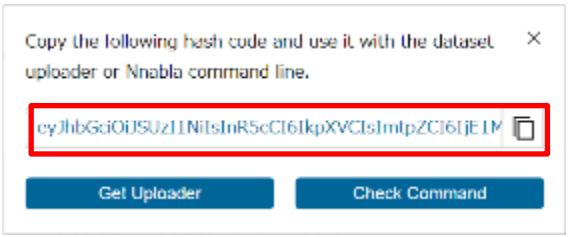
✓以下のリンクからアップローダをダウンロード
https://support.dl.sony.com/docs-ja/データセットアップロードツールのダウンロード/

２．アップロードキーの取得

✓NNCにログインし、Datasetタブの中のUpload Datasetをクリック

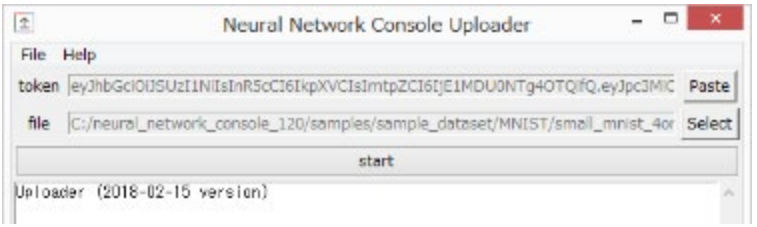


✓ポップアップ画面に表示されるアップロードキーをコピー



３．アップローダの実行

✓１で取得したアップローダを起動
✓tokenに２で取得したアップロードキーを貼り付け
✓fileに作成したCSVファイルを指定
※ ここでのファイル名がデータセット名になります
✓Startをクリックし、アップロードを実行



※アップロードには時間がかかる場合があります

アップロード先のデータセット確認

アップロード後はDatasetタブの一覧にデータセットが追加されます。

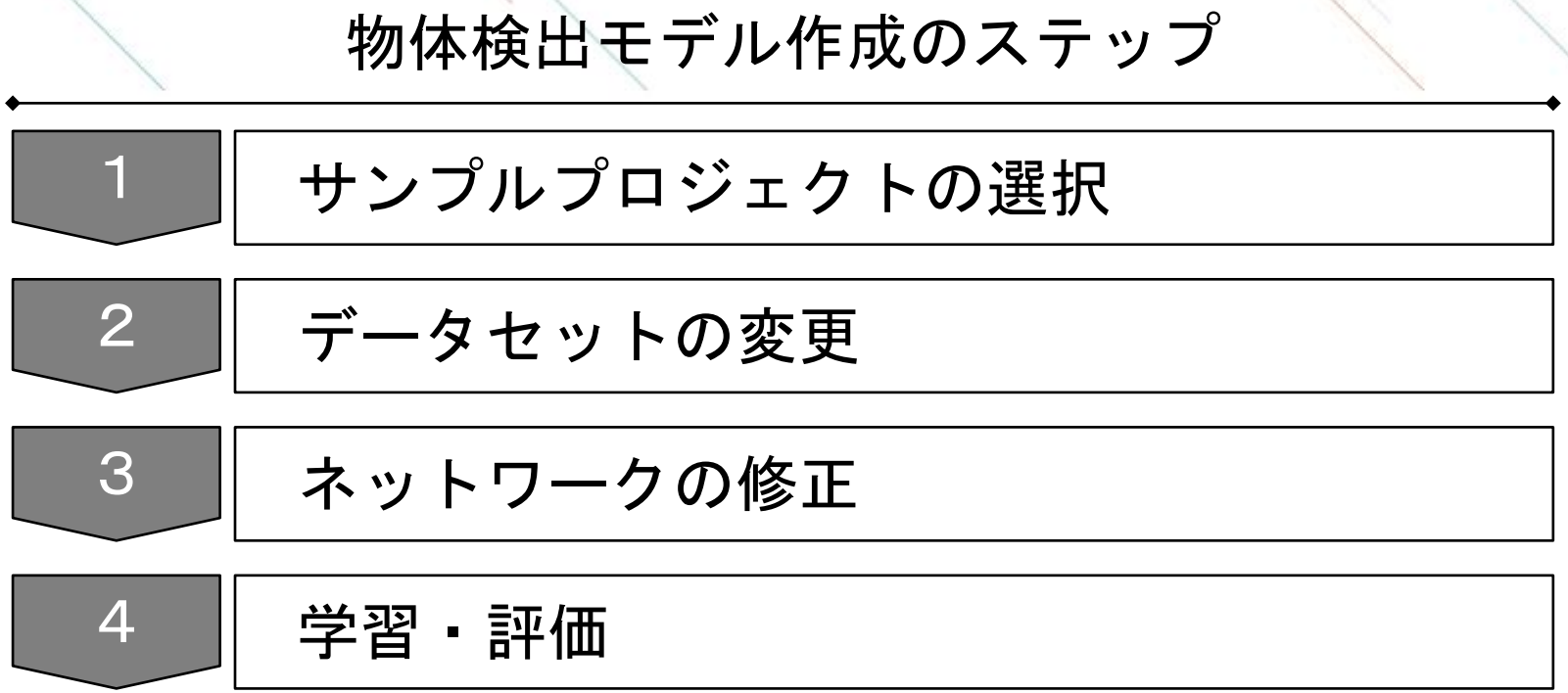
アップロード時のCSVファイルのファイル名がデータ名として一覧に表示され、選択することで中身を確認することができます。表示の際に、画像などはサムネイルの形で確認できます。

データセット一覧に追加されたデータセット

目次

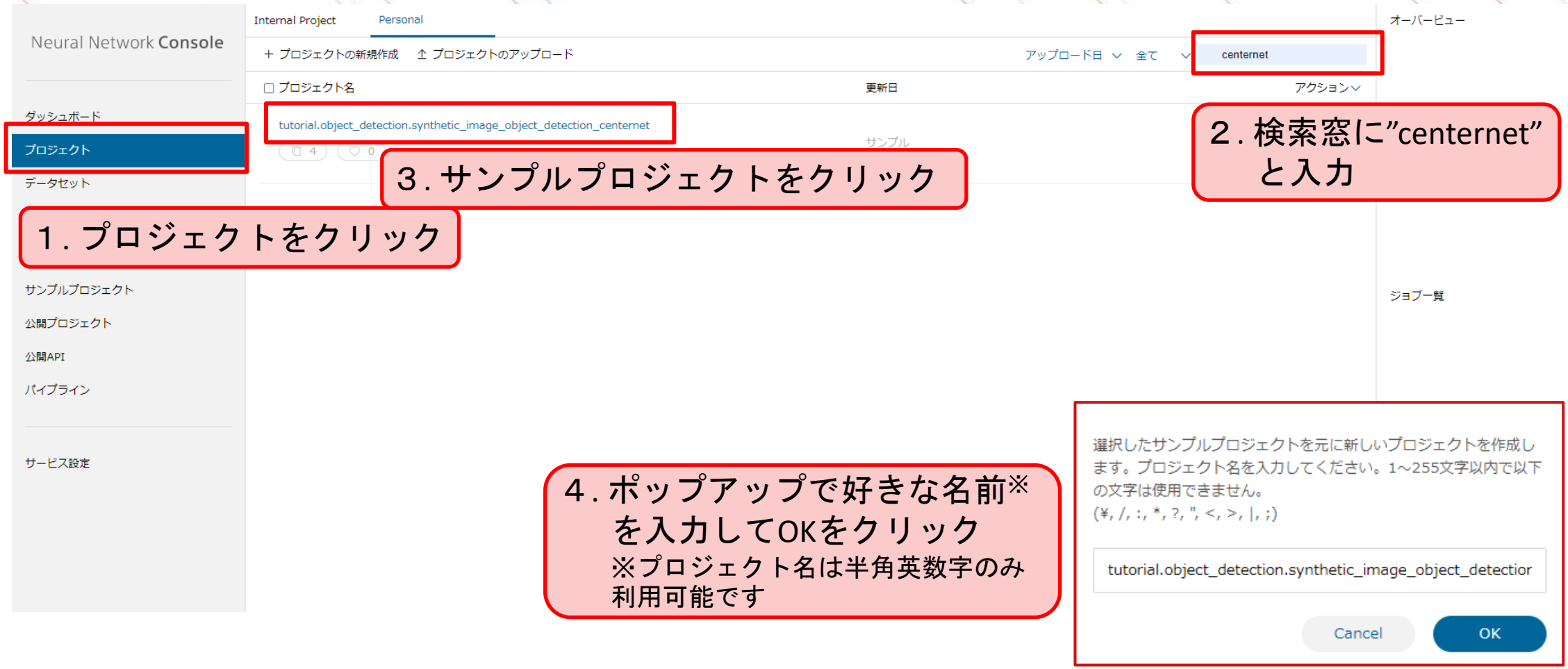
物体検出モデル作成のステップ

NNCでは新規にモデルを作成することも可能ですが、このドキュメントでは、物体検出モデルのサンプルプロジェクトをベースにすることで、比較的容易にモデル作成を可能にします。



サンプルプロジェクトのコピー

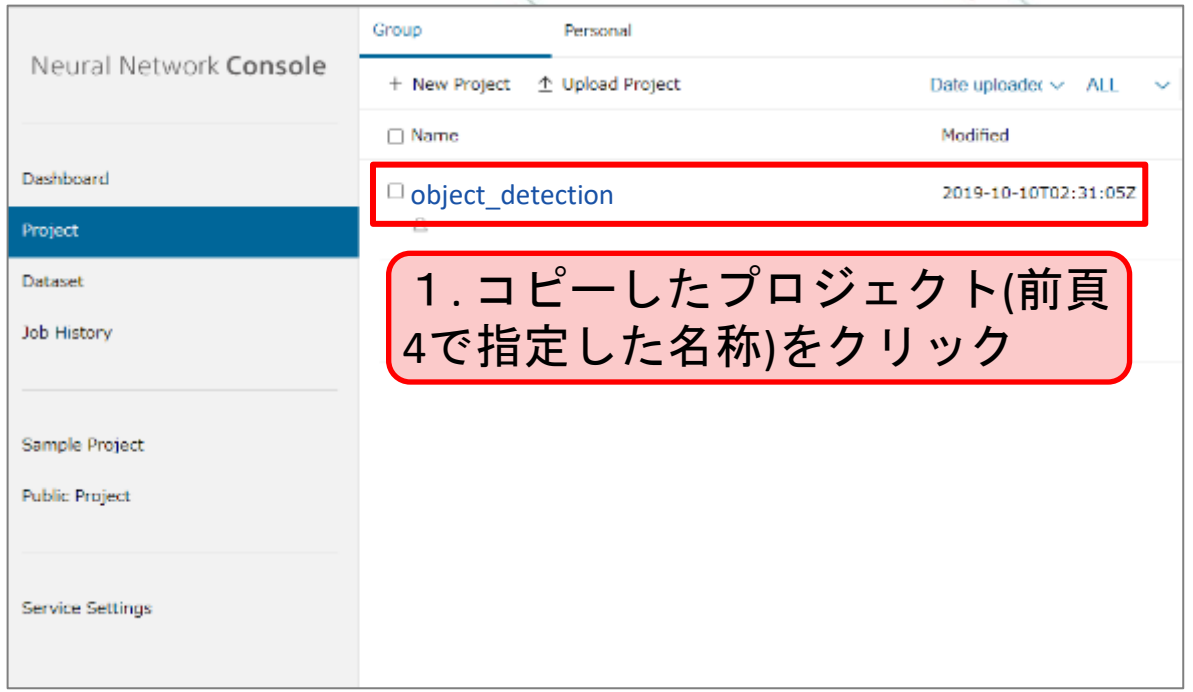
サンプルプロジェクト(CenterNet)を検索し、コピーします。



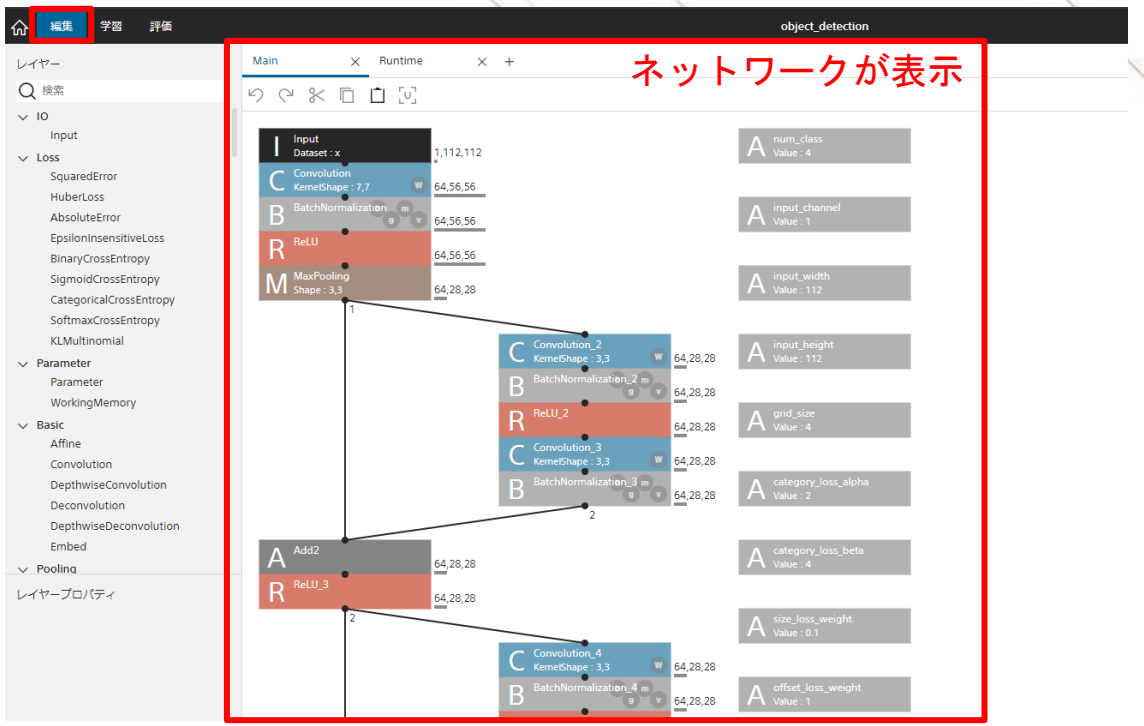
※学習結果やモデルも含めコピーを行うため、処理に時間がかかる場合があります。

プロジェクトの起動

コピーしたプロジェクトをクリックし、プロジェクトを起動します。
以下のようなネットワークが表示されていることを確認します。



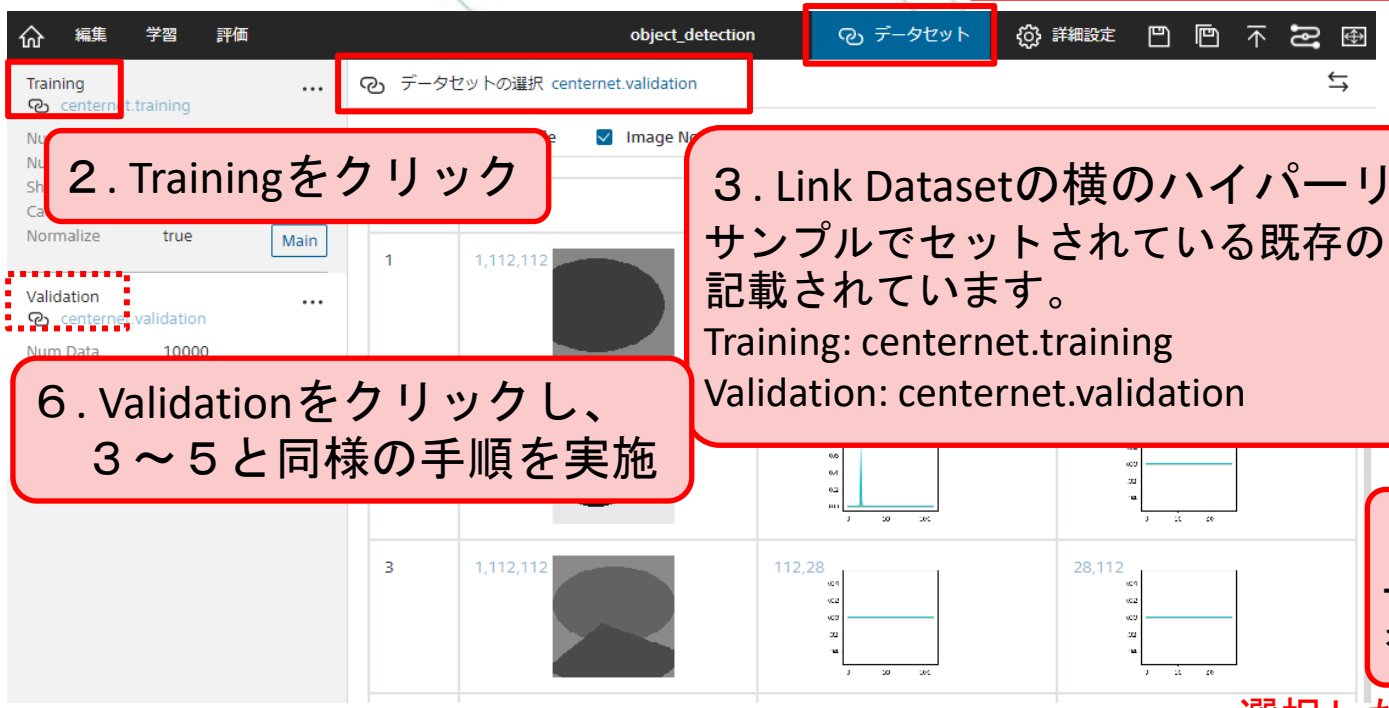
編集タブ: ネットワークを作成するページ



データセットの変更

データセットタブからTrainingとValidationのデータセットをそれぞれ変更します。
(サンプルデータを利用する場合には変更は不要です)

1. Datasetタブをクリック



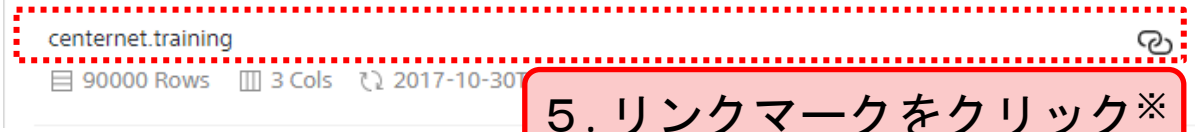
2. Trainingをクリック

3. Link Datasetの横のハイパーリンクをクリック
サンプルでセットされている既存のデータセット名が記載されています。
Training: centernet.training
Validation: centernet.validation

4. 一覧から学習に用いるデータセットを選択
データセットのアップロード時に指定したファイル名を選択してください。

6. Validationをクリックし、
3～5と同様の手順を実施

選択したデータセット名



5. リンクマークをクリック※

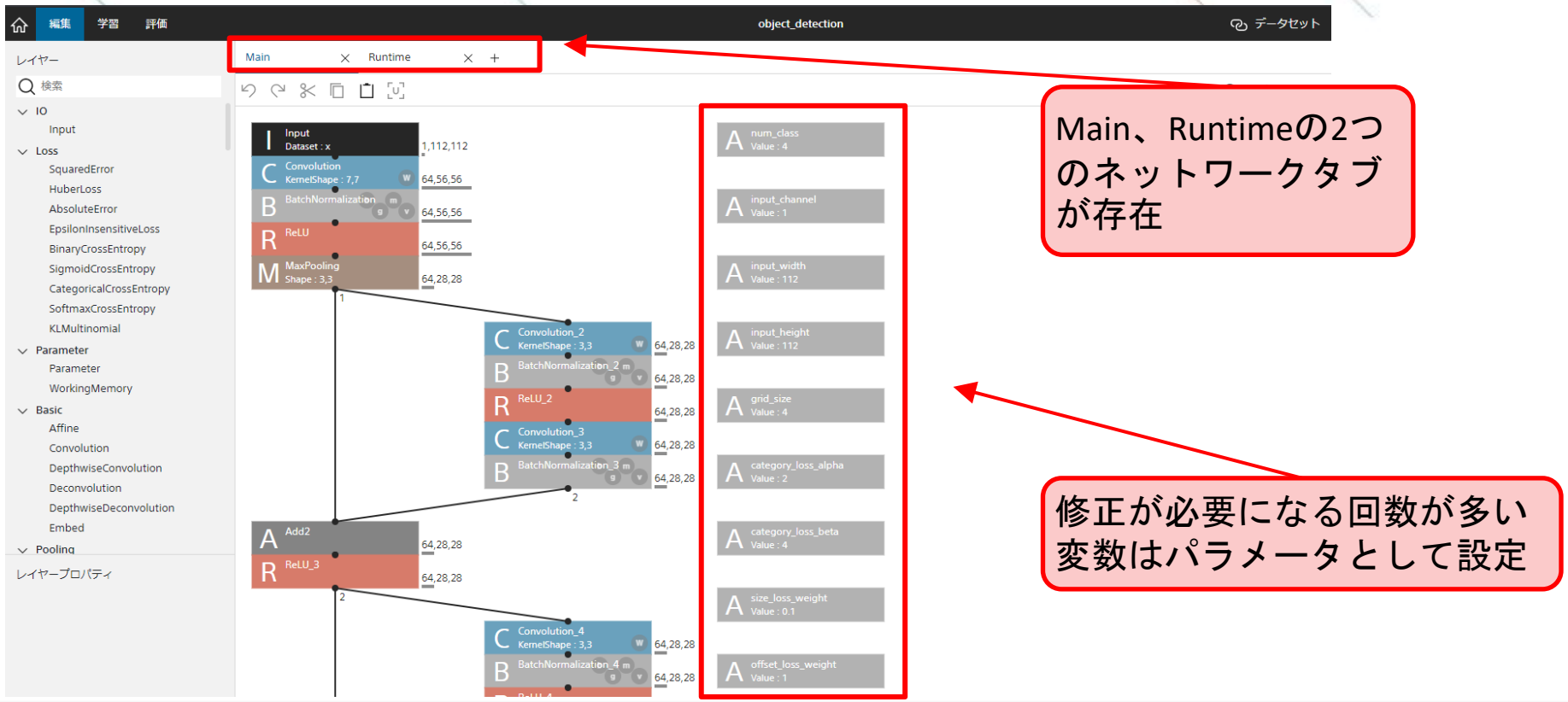
※ブラウザの拡大率によって表示されないことがあります。
表示されない場合は表示の縮小をお試しください。

ネットワークの構成

本サンプルでは学習・検証用、推論用のネットワークを個別に設定するため、MainタブとRuntimeタブを別々に記載しています。

学習・推論用のネットワークはMainから自動生成されたMainValidationを使用し、推論時はRuntimeを使用します。それぞれのネットワークは詳細設定タブの設定画面で学習用、検証用、推論用に対応づけられています。

ネットワーク画面の説明



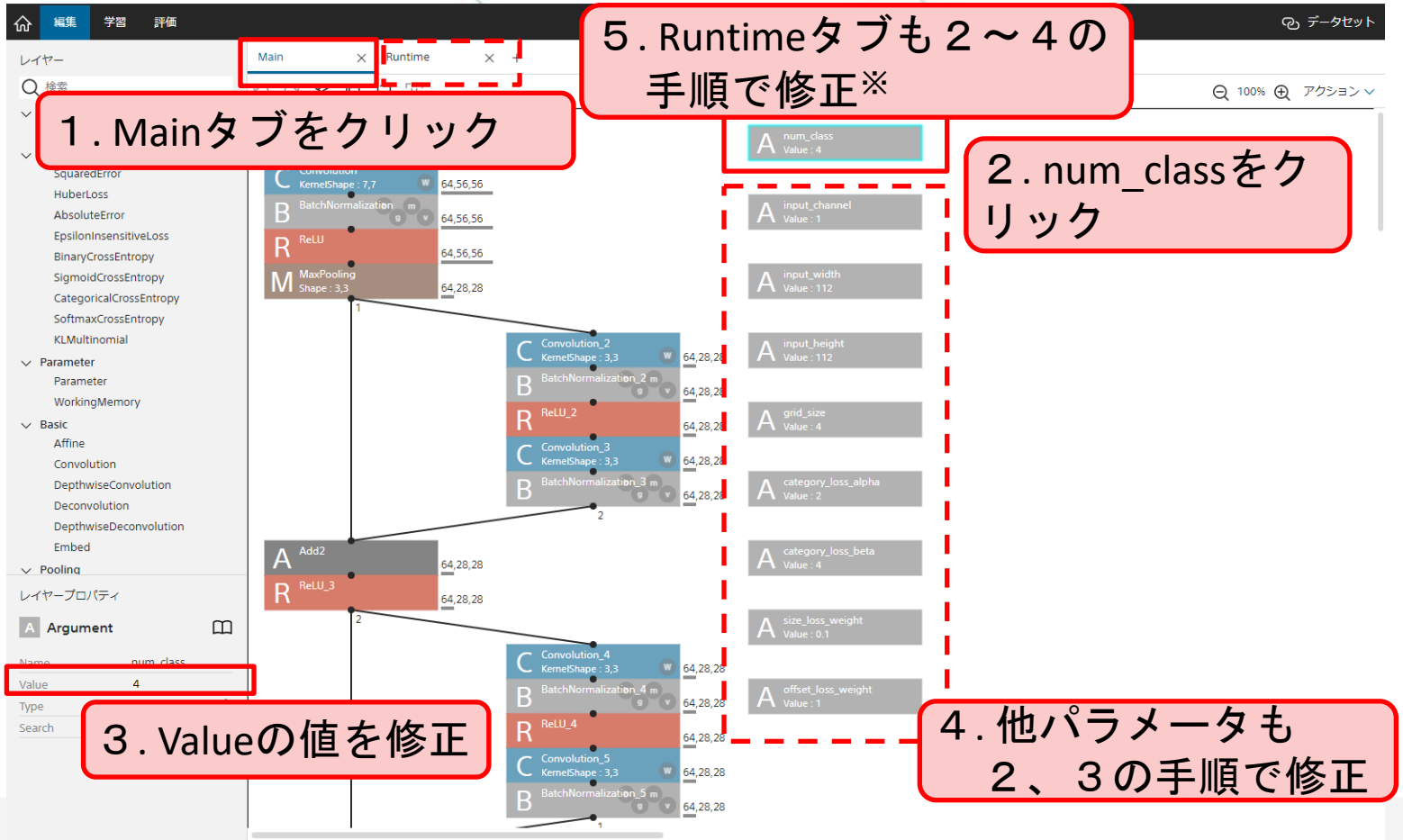
Main、Runtimeの2つのネットワークタブが存在

修正が必要になる回数が多い変数はパラメータとして設定

ネットワークの修正

本サンプルではデータセットの変更、精度向上の際に調整するパラメータを引数レイヤー(Argument)として抜粋しています。データセット作成時の入力画像・分類数をもとにパラメータを修正してください。
(サンプルデータを利用する場合には変更は不要です)

ネットワーク修正の操作説明



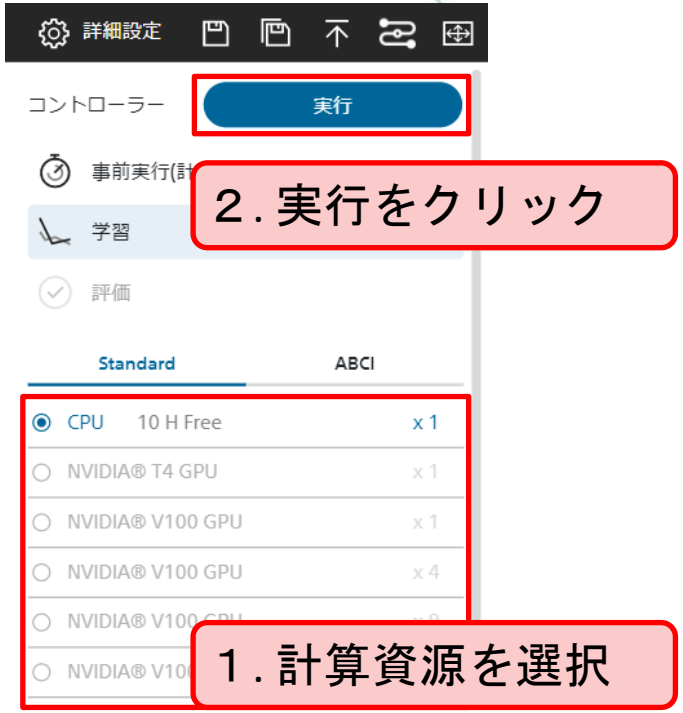
修正が必要なパラメータの説明

パラメータ	説明
num_class	分類数
input_channel	入力画像のモノクロ/カラーを設定 1:モノクロ、3:カラー
input_width	入力画像の幅
input_height	入力画像の高さ
category_loss_alpha	ラベルに関するlossのパラメータ
category_loss_beta	ラベルに関するlossのパラメータ
grid_size	グリッドサイズ
offset_loss_weight	グリッド内の相対位置(x,y)のLossの重み
size_loss_weight	幅、高さのLossの重み

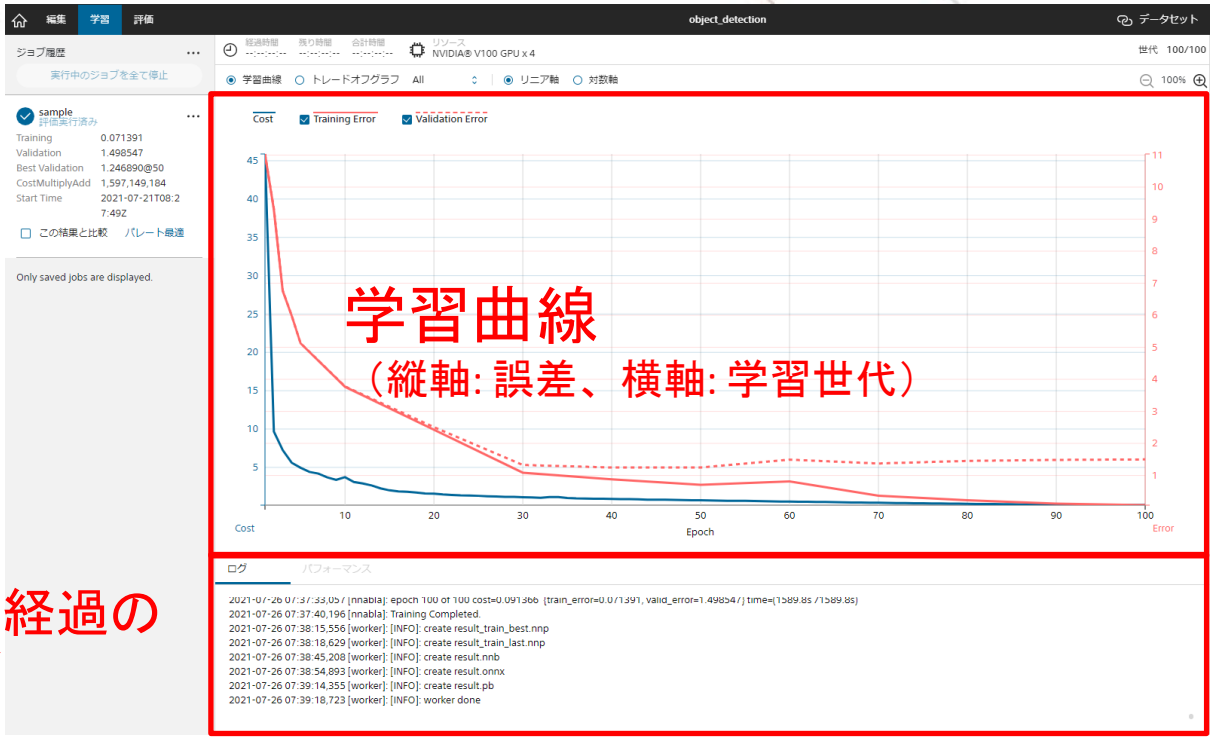
学習の実行

編集ページの実行ボタンをクリックすることで学習が実行されます。
GPUを選択すると、高速に学習を行うことができます※。（参考: 学習環境と処理時間）
GPU等有料のメニューを利用する場合は事前にクレジットカード登録もしくは法人契約が必要になります。
(法人契約：https://dl.sony.com/ja/business/)

学習実行の方法



学習ページの概要



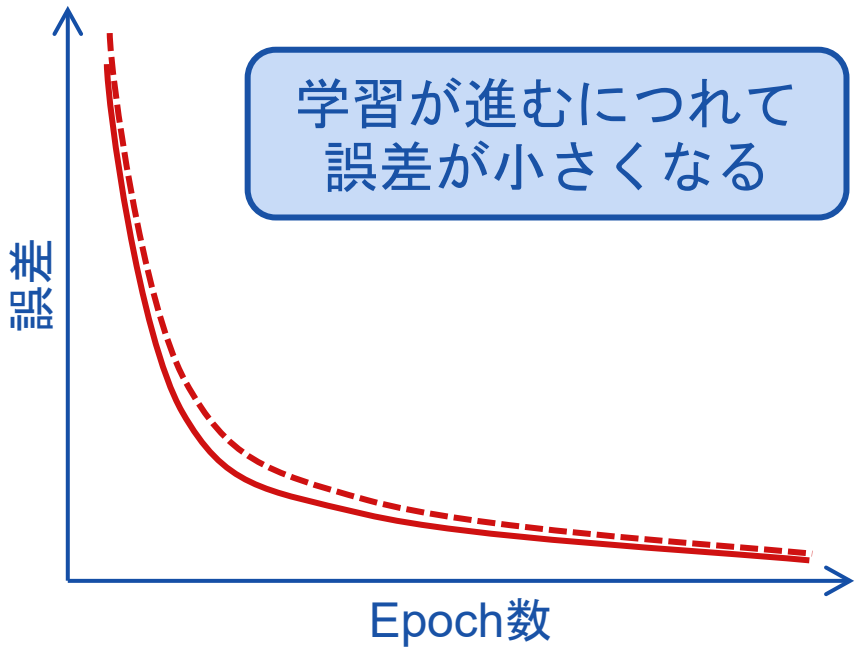
※ サンプルプロジェクトのネットワークは複雑なため、CPU実行の場合にはGPU実行と比較して学習時間が長時間になります。
計算途中にウェブブラウザを閉じても計算が止まることはありませんので、長時間に及ぶ場合にはあらためて結果をご確認ください。

学習曲線の読み取り方

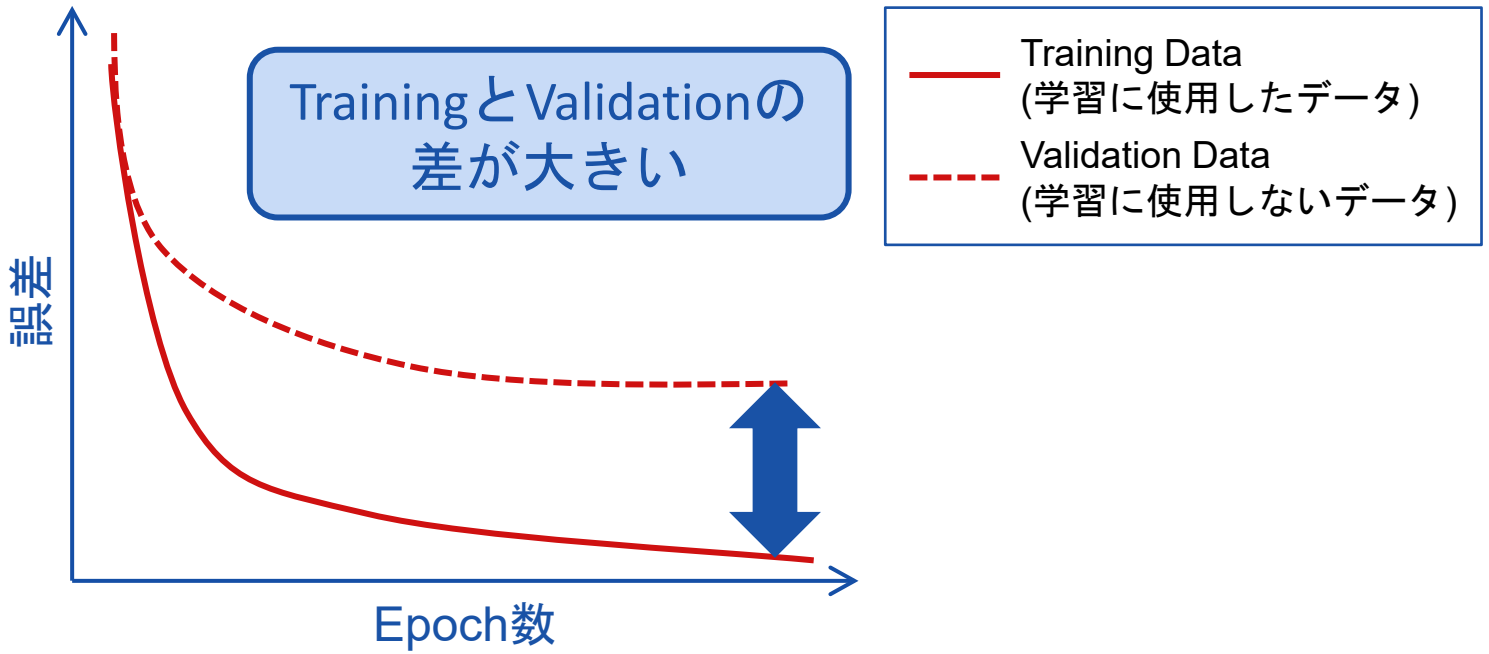
学習結果の良し悪しは、まずは学習曲線から判断をします。

TrainingとValidationの差が大きい場合(過学習)は、モデルがTraining Dataに特化し過ぎた状態(教科書を丸暗記した場合に応用問題が解けないのと似た状態)です。
未知のデータの予測精度が低いため、学習データ量を増やすなどの改善が必要です。

良いモデルの学習曲線



悪いモデルの学習曲線（過学習）



評価の実行

学習ページの実行をクリックすると評価ページに遷移し、詳細な判定結果を確認できます。
分類問題の場合には、各データに対するモデルの判定結果や統計的な精度や指数、混同行列などを確認できます。

評価実行の方法

表示可能なグラフの概要

評価グラフ	内容	問題
評価結果	各データの１つ１つの判定結果	すべて
混同行列	データセット全体の統計的な指標と混同行列(分類ラベルごとに結果を集計した表)	分類
分類結果	各データの判定確率上位３カテゴリの確率	分類
分類マトリックス	カテゴリごとのモデルの判定傾向	分類
尤度グラフ	判定確率と正答率の傾向	分類

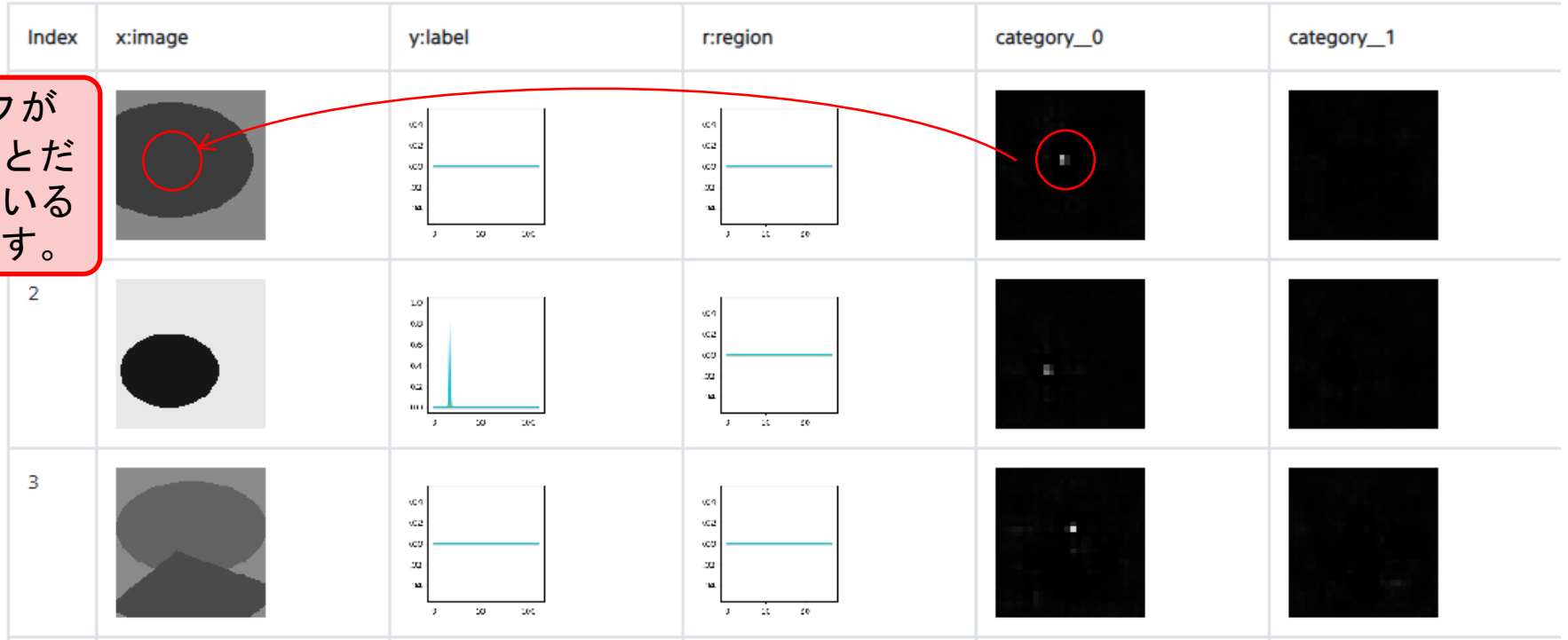
物体検出は対象外のため、表示されない

評価結果の見方

評価結果では検証データの推論結果を確認できます。
予測結果のScoreのピークが物体中心にあるかどうかで、定性的にモデル精度がよいかを確認できます。
その他の出力の説明は次頁をご覧ください。

評価結果の概要

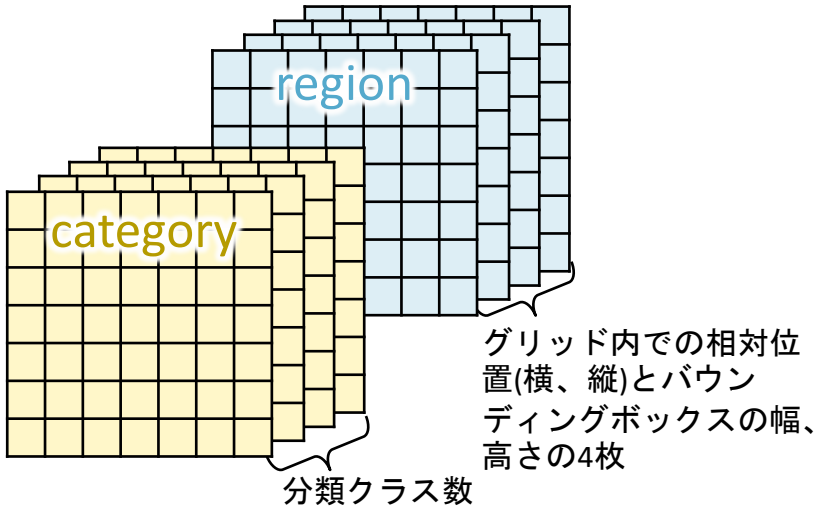
categoryのピークが物体の中心位置とだいたい一致していること確認できます。



サンプルプロジェクトの出力

サンプルプロジェクトの出力はとグリッド内に含まれる物体中心のラベルのスコア(category)とバウンディングボックスの情報(region)の2種類になります。

出力データのイメージ※1



出力ラベルの説明

ラベル	説明
category	分類クラスごとのグリッド内に物体中心が含まれるスコア
region	グリッド内に物体中心があるバウンディングボックスのグリッド内の相対位置(横、縦)※2とバウンディングボックスの幅と高さ

※1 実際の出力データはcategoryが分類クラス分の確率値を示したモノクロ画像、regionが4枚のモノクロ画像になります
詳細はサンプルプロジェクトの出力ファイルを参照
※2 グリッドサイズにより規格化されています。

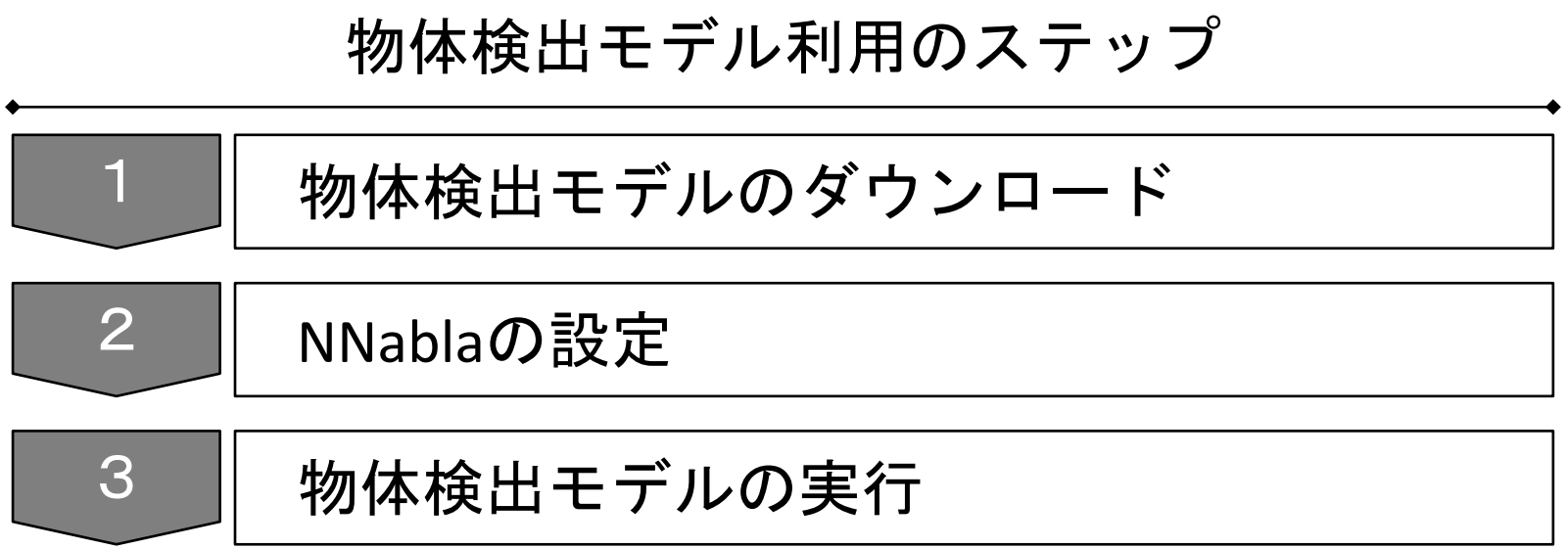
目次

物体検出モデル利用のステップ

NNCから分類モデルをダウンロードすることで、お客様の環境で自由にモデル利用ができます。

モデルを実行するためには、NNablaが必要になります。

NNablaを用いることでコマンドラインやPythonなど様々な方法で作成した物体検出モデルが実行可能となります。

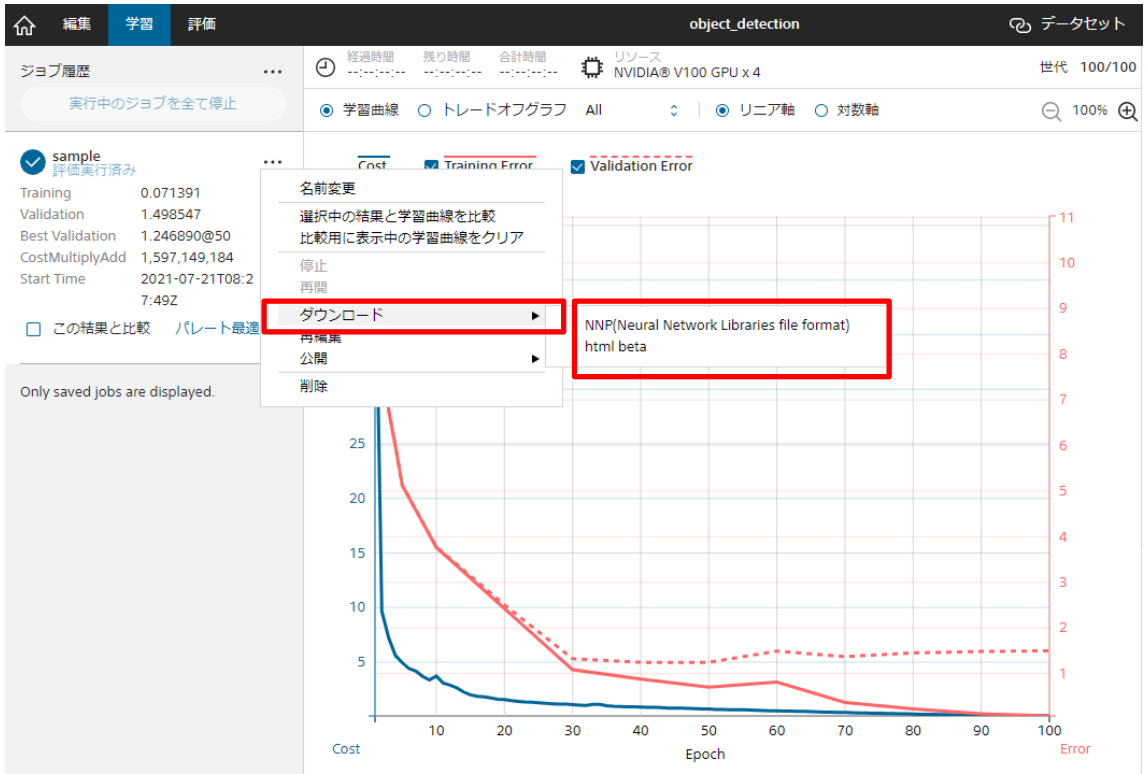


物体検出モデルのダウンロード

学習が完了し最適なモデルを作成した後は、「ジョブ履歴」の中から該当のモデルを右クリックし、選択肢の中の「ダウンロード」をクリックすることでモデルをダウンロードできます。

NNPは作成したネットワークと学習済みパラメータの値が含まれたファイルで、実行方法に応じて使い分けをします (詳細はモデルの実行方法) 。また、html betaは学習結果などの内容をhtml形式で出力したものです。

作成したモデルの権利は作成者に帰属し、自由にDeep Learningモデルを利用することができます。



NNablaの設定

PCにNNablaをインストールします。

NNablaのインストールについては、以下のドキュメントをご参照ください。
http://nnabla.readthedocs.io/en/latest/python/installation.html

モデルの実行方法

NNablaを用いてモデルを実行する方法は、使用する言語に応じて様々な方法があります。
また、ONNXを利用することで、他のDeep Learningのフレームワークを利用することも可能です。
次頁以降では、コマンドラインとPythonで実行する方法を解説いたします。

	実行方法	GPU利用	特徴	ダウンロードファイル	参考URL	
1	コマンドライン	可能	最も簡単に利用可能	NNPファイル	https://support.dl.sony.com/docs-ja/チュートリアル：neural-network-consoleによる学習済みニューラ/	次頁に解説あり
2	Python	可能	比較的容易に利用可能	NNPファイル	https://support.dl.sony.com/docs-ja/チュートリアル：neural-network-consoleによる学習済みニューラ/	
3	C++	可能	推論環境にPythonのインストールが不要	NNPファイル	https://github.com/sony/nnabla/tree/master/examples/cpp/mnist_runtime	
4	C	不可	非常にコンパクトであり、組み込み利用向き	NNBファイル	https://github.com/sony/nnabla-c-runtime	
5	他Deep Learningフレームワーク	環境依存	環境依存	ONNXファイル	https://nnabla.readthedocs.io/en/latest/python/file_format_converter/file_format_converter.html	
6	TensorFlow	環境依存	環境依存	TensorFlow frozen graphファイル	TensorFlowのウェブページをご覧ください	

コマンドラインでの推論実行

NNablaのインストールされたPython環境で、コマンドラインから以下を実行します。

出力はグリッドごとのcategory、regionになります(参照:サンプルプロジェクトの出力、サンプルプロジェクトの出力ファイル)、バウンディングボックスの描画には categoryの値の高いグリッドのregionのデータを取り出し、計算していただく必要があります。

```
nnabla_cli forward ¥
  -c [ダウンロードしたネットワークモデルファイル(*.nnp)] ¥
  -d [推論をするデータセットを取りまとめたCSVファイル] ¥
  -o [推論結果の出力ディレクトリ]
```

※ NNCのEVALUATIONタブでの推論実行時に同様のコマンドを使用しているため、
ログの出力ウインドウに同様のものが出力されています。

```
2017-10-24 05:54:28,942 [worker]: [INFO]: nnabla_cli forward -c ./result.nnp
-d ./index.csv -o ./results
```

Pythonでの実行方法

ダウンロードしたネットワークファイルをPythonで読み込んで利用します。
以下のサンプルでは入力画像を読み込み、モデルを実行し、バウンディングボックスを描画するまでの一連の流れを記載しています。

1/5

```
# 必要なmodulesのインストール
import math

import matplotlib.patches as patches
import matplotlib.pyplot as plt
import nnabla as nn
import numpy as np
from nnabla.utils import nnp_graph
from nnabla.utils.image_utils import imread

# ファイル名などの設定
nnp_file = "./result_object_detection.nnp"
input_figure = "./input.png"
output_figure = "./output.png"
colorlist = ["r", "g", "b", "c"]
classlist = ["Ellipse", "Triangle", "Rectangle", "Pentagon"]
threshhold = 0.6
num_class = len(classlist)
img_height = 112
img_width = 112
figscale = 3
grid_size = 4
```

この部分を問題に応じて変更する必要があります

Pythonでの実行方法


```
#モデルがカラーの場合には下記
# img = imread(input_figure, grayscale=False, size=(img_width, img_height))

# 画像の読み込みと縦横サイズの取得
img = imread(input_figure, grayscale=True, size=(img_width, img_height))

# nnpファイルの読み込み、ネットワークモデルの取り出し
nnpFile = nnp_graph.NnpLoader(nnp_file)
networkModel = nnpFile.get_network('Runtime', batch_size=1)

# 入出力レイヤーの設定
nnval_x = networkModel.inputs['Input’]
nnval_y_category = networkModel.outputs["category"]
nnval_y_offset = networkModel.outputs["offset"]
nnval_y_size = networkModel.outputs["size"]

# 入力レイヤーに画像をセットし、推論を実行
nnval_x.d = np.array(img) .reshape(nnval_x.d.shape) / 255
nn.forward_all([nnval_y_ category, nnval_y_ offset, nnval_y_ size], clear_buffer=True)
```

画像データを読み込み、
推論実行をしている箇所。
ここより後方の部分はバウンディングボックスの計算と描画部分である。

※入力画像がカラーの場合には、ソースコードの一部をコメントアウト部分と変更してください。

Pythonでの実行方法

3/5

```
# ヒートマップの8近傍に対するピークを抽出
class_max_pooling_map = nn.functions.max_pooling(nnval_y_category, kernel=(3, 3), stride=(1, 1), pad=(1, 1))
class_is_max_map = nn.functions.equal(nnval_y_category, class_max_pooling_map)
class_maps = nn.functions.mul2(nnval_y_category, class_is_max_map)
class_maps.forward()
class_maps_array = class_maps.d.copy()
```

Pythonでの実行方法

4/5

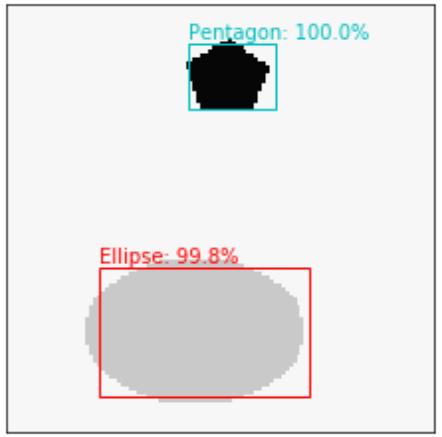
```
bboxes, texts = [], []
for class_num in range(num_class):
    for point_num in range(100):
        if class_maps_array[0, class_num, ...].max() < threshhold:
            break
        # category情報の取得
        x_idx = np.argmax(class_maps_array[0, class_num, ...]) // 28
        y_idx = np.argmax(class_maps_array[0, class_num, ...]) % 28
        # offset情報の取得
        x_offset = nnval_y_offset.d[0, 0, x_idx, y_idx]
        y_offset = nnval_y_offset.d[0, 1, x_idx, y_idx]
        # size情報の取得
        bbox_width = nnval_y_size.d[0, 0, x_idx, y_idx]
        bbox_height = nnval_y_size.d[0, 1, x_idx, y_idx]
        # バウンディングボックスのx, y座標の計算
        bbox_x = (x_offset + y_idx) * grid_size - bbox_width / 2
        bbox_y = (y_offset + x_idx) * grid_size - bbox_height / 2

        texts.append("{0}: {1:.1f}%".format(classlist[class_num], 100 * class_maps_array[0, class_num, ...].max()))
        bboxes.append(patches.Rectangle(xy=(bbox_x, bbox_y), width=bbox_width, height=bbox_height, ec=colorlist[class_num],
fill=False))

        class_maps_array[0, class_num, x_idx, y_idx] = 0
```

Pythonでの実行方法

5/5

```
# バウンディングボックスの描画
fig = plt.figure(figsize=(figscale*img_width/img_height, figscale))
ax = plt.axes()
ax = fig.add_axes([0, 0, 1, 1])
# モデルがカラーの場合には下記
# ax.imshow(img)
ax.imshow(img, cmap='gray', vmin=0, vmax=255)
for (r, text) in zip(bboxes, texts):
    ax.add_patch(r)
    ax.text(r.xy[0], r.xy[1], text, ha='left', va='bottom', transform=ax.transData, color=r.get_ec() )
ax.tick_params(labelbottom=False,labelleft=False, labelright=False, labeltop=False)
ax.tick_params(bottom=False, left=False, right=False, top=False)
plt.savefig(output_figure)
```

保存画像の例



※入力画像がカラーの場合には、ソースコードの一部をコメントアウト部分と変更してください。

Appendix

SONYアカウントの取得

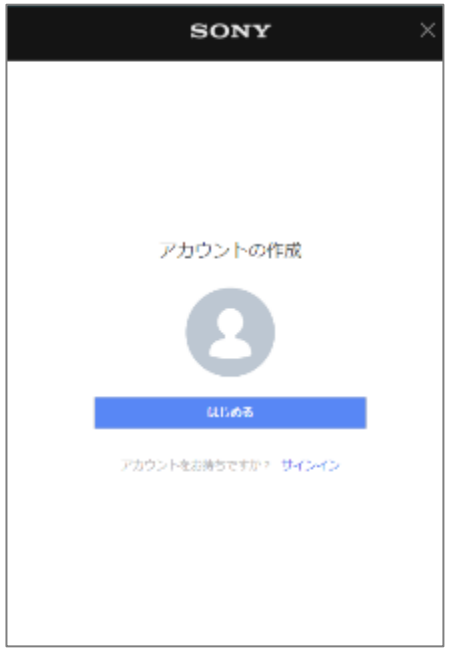
アカウント作成ページに移動し、メールアドレスやパスワードなどを設定します。

１．作成ページへの移動１

✓「新しいアカウントの作成」を押下

２．作成ページへの移動２

✓「はじめる」を押下



３．メールアドレス等の入力

✓登録するメールアドレスとパスワードを入力

SONYアカウントの取得

生年月日などを入力し、利用規約などの確認を行います。

４．生年月日の入力

✓国/地域、言語、生年月日を入力

５．利用規約への同意

✓メール配信の有無を選択
✓利用規約・アカウントポリシーの確認

６．セキュリティ認証

✓「私はロボットではありません」を押下
※画像選択が表示された場合には指示に従う

SONYアカウントの取得

確認メールを受信し、アカウントの有効化を行います。

7．確認メールの送付

✓登録したメールアドレス宛に確認メールが送付される

8．確認メールの確認

✓確認メールを開き、「確認する」を押下

学習環境と処理時間

一般的にDeep LearningはGPUを用いることにより、CPUと比べ高速に学習処理を行うことが可能です。

学習実行環境と処理時間・ご利用料金

	学習実行環境	学習処理時間	1時間当たりのご利用料金	ご利用料金目安
1	CPU	2,822,229秒 (784時間)	85円	約66,636円
2	NVIDIA® TESLA® T4 GPU	14,865秒 (4.13時間)	130円	約537円
3	NVIDIA® TESLA® V100 GPU	6,067秒 (1.69時間)	560円	約944円

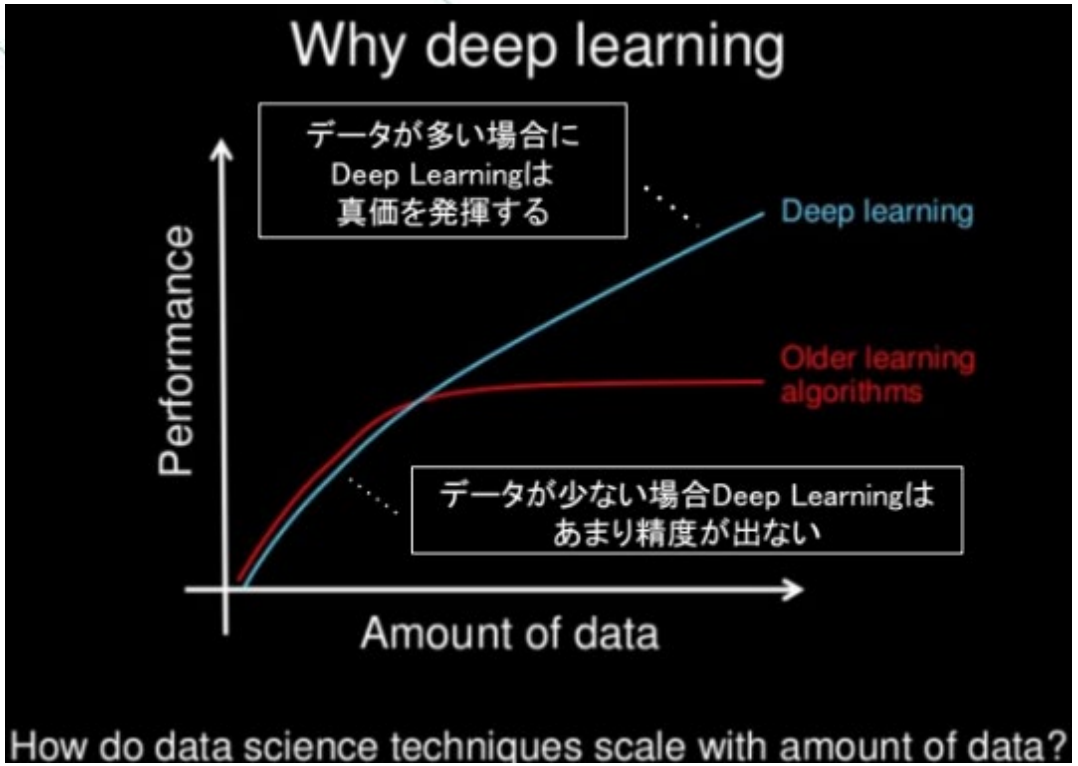
【検証環境】

- ✓ データセット：CIFAR 10
- ✓ ネットワーク：ResNet-110
- ✓ epoch：300

データ量の重要性

Deep Learningで高い精度を得るにはデータ量が重要になります。Deep Learningではデータを増やせば増やすだけ精度が向上する傾向にあります。

一方でデータ量が少ない場合には、Deep Learning以前の従来型の機械学習に比べても精度が劣ることもあります。



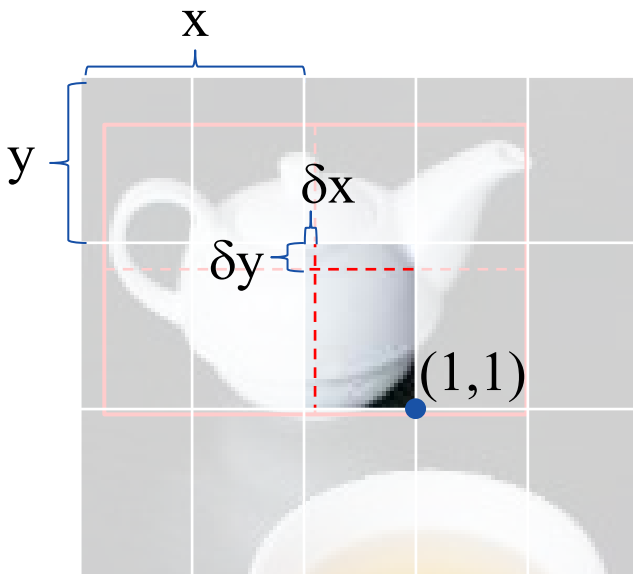
出典: https://www.slideshare.net/ExtractConf/andrew-ng-chief-scientist-at-baidu

グリッド内での相対位置

バウンディングボックスの中心が含まれるグリッドがわかれば、おおよその中心位置は決まります。
グリッド内での相対位置(δx, δy)をさらに予測することで、詳細にバウンディングボックスの中心位置を決定することができます。

δx, δyは学習効率化のため、グリッドサイズで規格化されており、0〜1の値をとります。

グリッド内での相対位置とバウンディングボックスの中心位置



実際のバウンディングボックスの中心位置(X, Y)
=(x+[グリッドの幅] x δx, y+[グリッドの高さ] x δy)

convert_dataset_for_centernet.pyで出力されるデータセットの説明

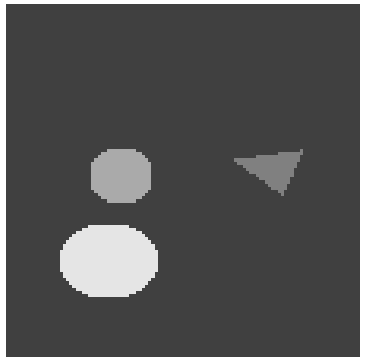
convert_dataset_for_centernet.pyによって出力されるcsvファイルは以下の通りです。

入力データが画像データ(image)とYOLOフォーマットのラベルデータ(text)となり、出力データがグリッドごとのラベル情報(label)とバウンディングボックスの情報(region)が含まれたcsvファイル、modeで指定されたリサイズの方法でリサイズされた画像になります。

ラベル情報は物体の中心が含まれる場所とその近傍に、バウンディングボックスの情報は物体の中心が含まれる場所にデータが記載されています。それ以外の箇所は0で穴埋めがされています。

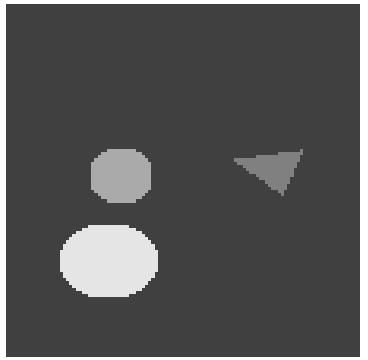
convert_dataset_for_centernet.pyの入出力(グリッド:5×5、クラス数:3の場合)

入力: 画像(image) ・ラベルデータ(text)



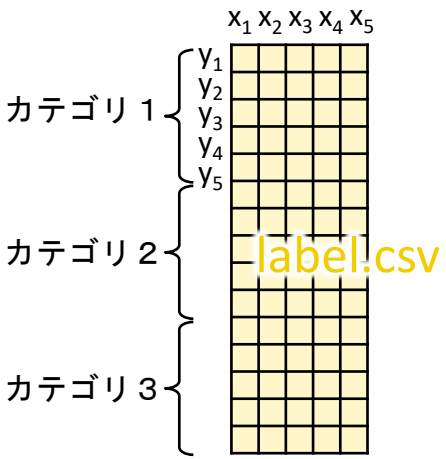
0	0.333	0.488	0.170	0.151
0	0.289	0.730	0.269	0.205
1	0.741	0.477	0.183	0.129

出力: 物体のラベル情報(label)とバウンディングボックス情報(region)

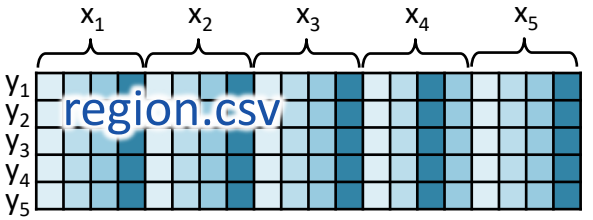


resize,trimming,paddingのいずれかの方法でリサイズされた画像

ラベルの重み



グリッド内での相対位置(横、縦)とバウンディングボックスの幅、高さ※



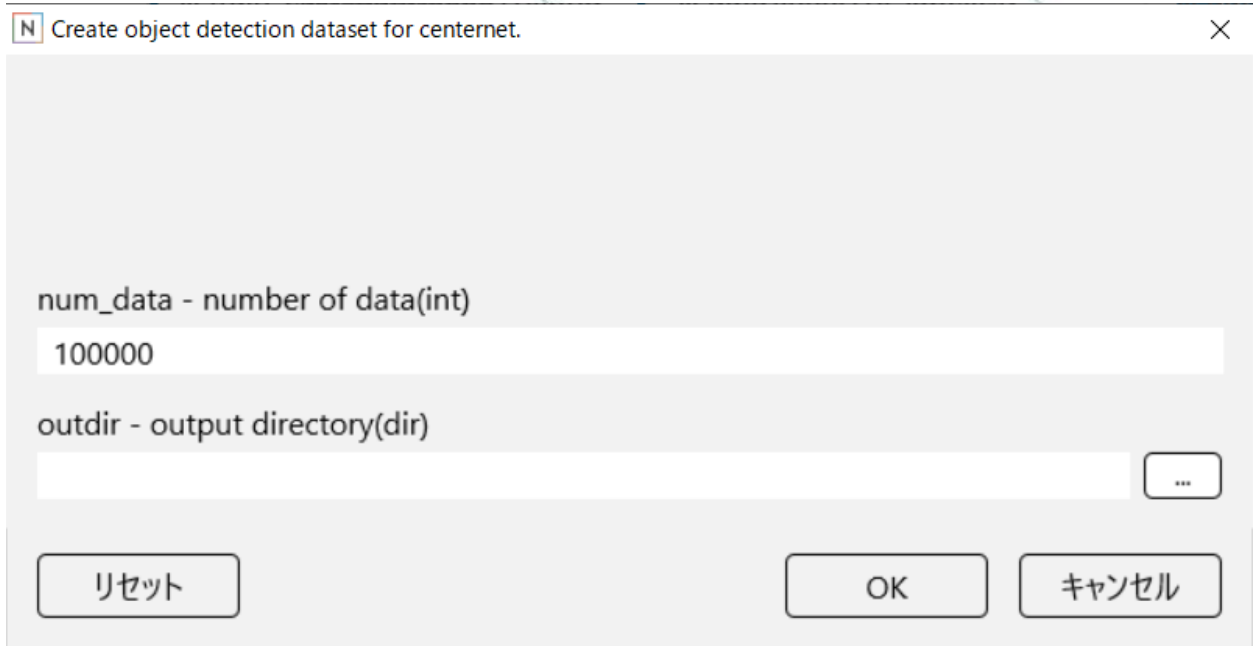
※ 相対位置はグリッドサイズで規格化し、バウンディングボックスのサイズはグリッドサイズで規格化をした後に対数に変換します。

Plugin(create_object_detection_dataset.py)の実行

Pluginを利用することで、データセットをお持ちでない方も図形のデータセットを作成することが出来ます。

Pluginによって出力されるデータの詳細はサンプルデータの説明に記載がありますので、こちらを確認してください。

GUIの操作画面



修正が必要なパラメータの説明

パラメータ	説明
num_data	作成するデータ数
outdir	出力ディレクトリ 値を指定しない場合は「neural_network_console/samples/sample_dataset/synthetic_data/object_detection/data_for_centernet」に出力される。

コマンドラインでのデータセット作成の実行

create_object_detection_dataset.py はコマンドライン上でも実行することが出来ます。
コマンドライン上では以下のコマンド例を基に入力してください。

《コマンドライン上での実行方法》

```
python create_object_detection_dataset.py –n 出力するデータ数 -o 出力ディレクトリ
```

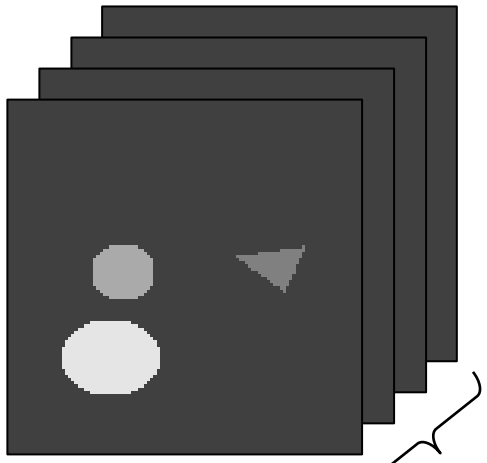
黄字が変更箇所になります

create_object_detection_dataset.pyで出力されるデータセットの説明

楕円、三角形、四角形、五角形をランダムに配置したデータセットです。この際、色、個数、形状もランダムに選ばれます。

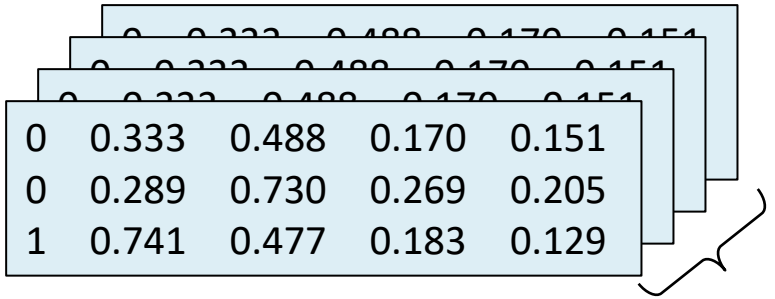
入力がデータ数と出力ディレクトリとなっており、入力したデータ数分のデータ(画像・YOLO Formatのラベルが記載されたtextファイル)を作成します。

Pluginの出力



画像(image)

入力したデータ数分



入力したデータ数分

YOLO Formatで記載されたラベル(text)

カテゴリマスタ

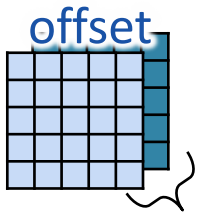
index	カテゴリ
0	楕円
1	三角形
2	四角形
3	五角形

サンプルプロジェクトの出力ファイル

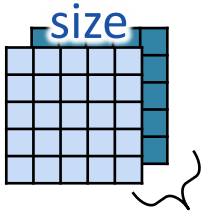
categoryは2次元データが分類クラス分あるため、確率値を表すモノクロ画像を分類クラス数分出力します。
offset、sizeは2次元データが2枚分あるため、それぞれの値を表すモノクロ画像を2枚ずつ出力します。

出力データの変換(グリッド:5x5、クラス数:4の場合)

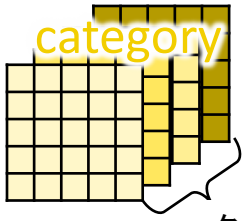
出力配列



グリッド内での相対
位置(横、縦)の2枚



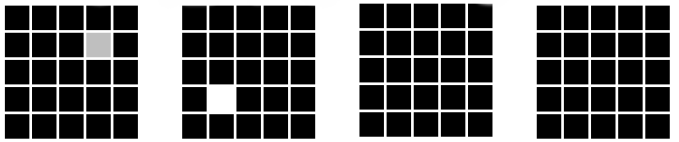
バウンディングボッ
クスの幅、高さの2枚



クラス数の4枚

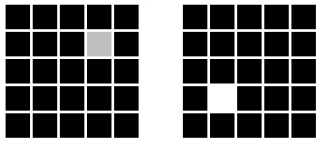
出力ファイル

category_0.png～category_3.png



各categoryの値を
0(黒)～1(白)の画像データで保存

offset_0.png, offset_1.png



各offsetの値を
画像データで保存

size_0.png, size_1.png

各sizeの値を
画像データで保存