

# Azure MLモデル (転移学習) デプロイ編

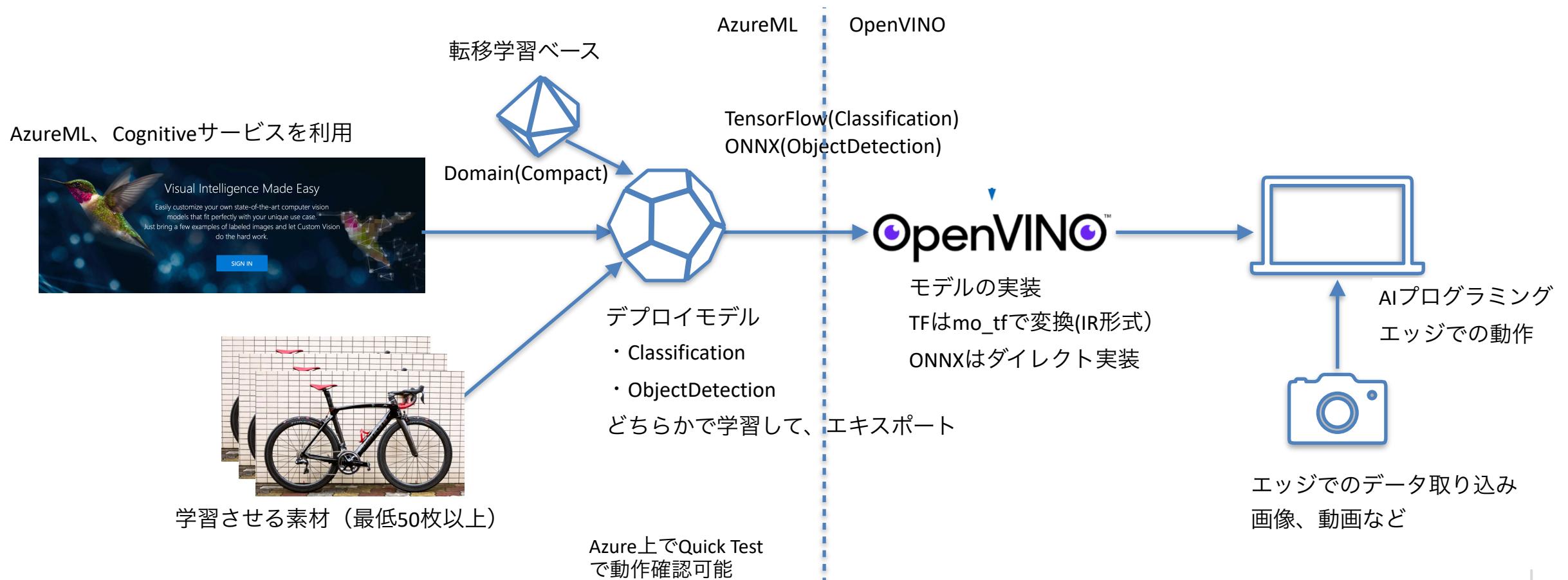


2022/2/23 Ver1.0 Inoishi

# 概略

Azure上で推論モデルの学習から、デプロイ、動作確認、OpenVINOで実装、動作確認までのフローについて解説

- Classification（画像ファイルの分類）、ObjectDetection（画像の中の物体を認識）モデルの作り方
- 学習させたい画像ファイルは、Userが用意する必要あり（最低でも50枚以上が推奨）
- AzureMLは転移学習なので、少ない画像ファイルでもある程度の認識率を出すことが可能**



# 転移学習(Transfer Learning)とは

AzureML のCognitive serviceは転移学習のサービスです。

転移学習は中間層の重みづけはそのまま（ドメインを適用）

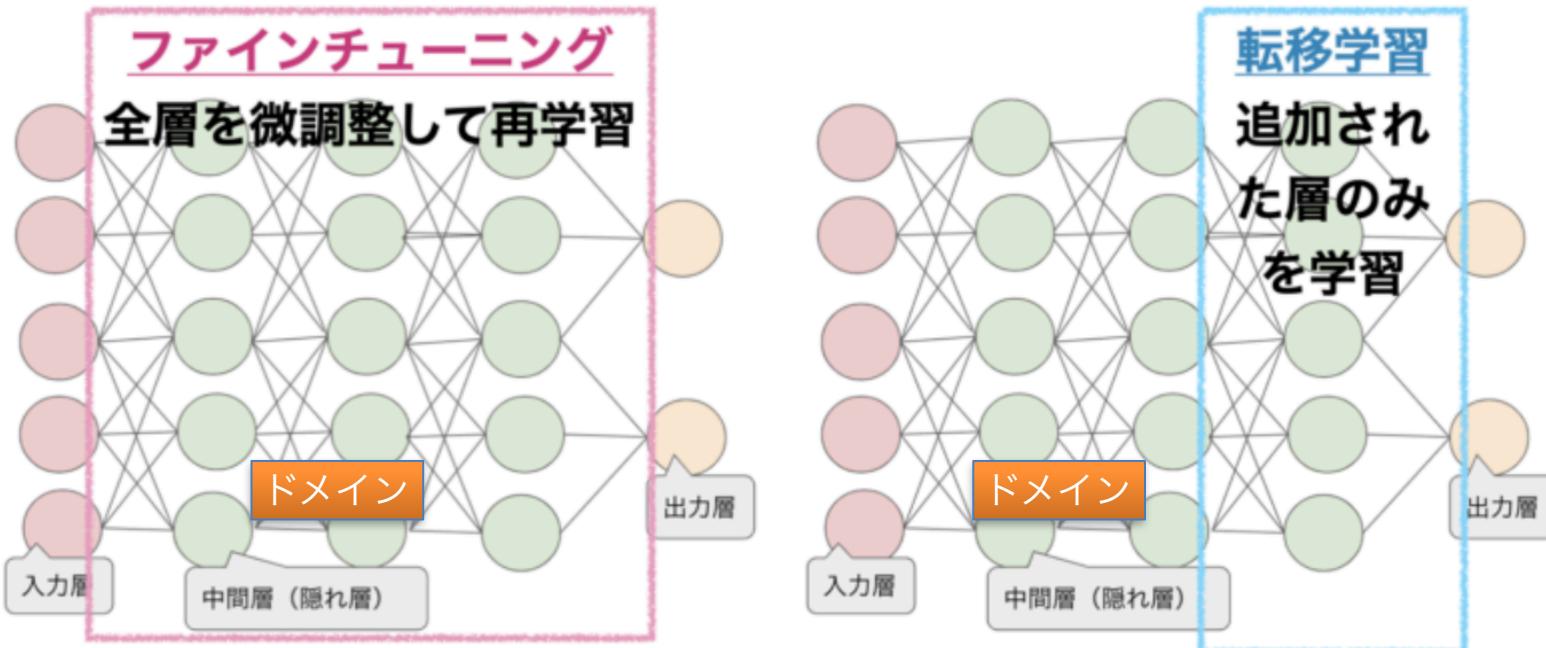
基本学習部分（ドメイン）をそのままに、追加で層を学習することで、最小のデータセットで転移学習できます。

\*例えば 犬を認識するモデル<→猫を認識するモデル を学ぶプロセスは同じモデル（ドメイン）をえる

メリット：短時間、最小のデータで学習可能。広い応用範囲（ドメイン）

デメリット：明らかに違うある基礎学習部分（ドメイン）が異なると、精度が上がらない

未知の領域などには適用不可。逆にいうと単純な領域であれば、転移学習は最適（0から作る必要なし）



# 作業フロー（概略） AzureMLで学習モデル作成編

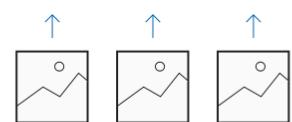
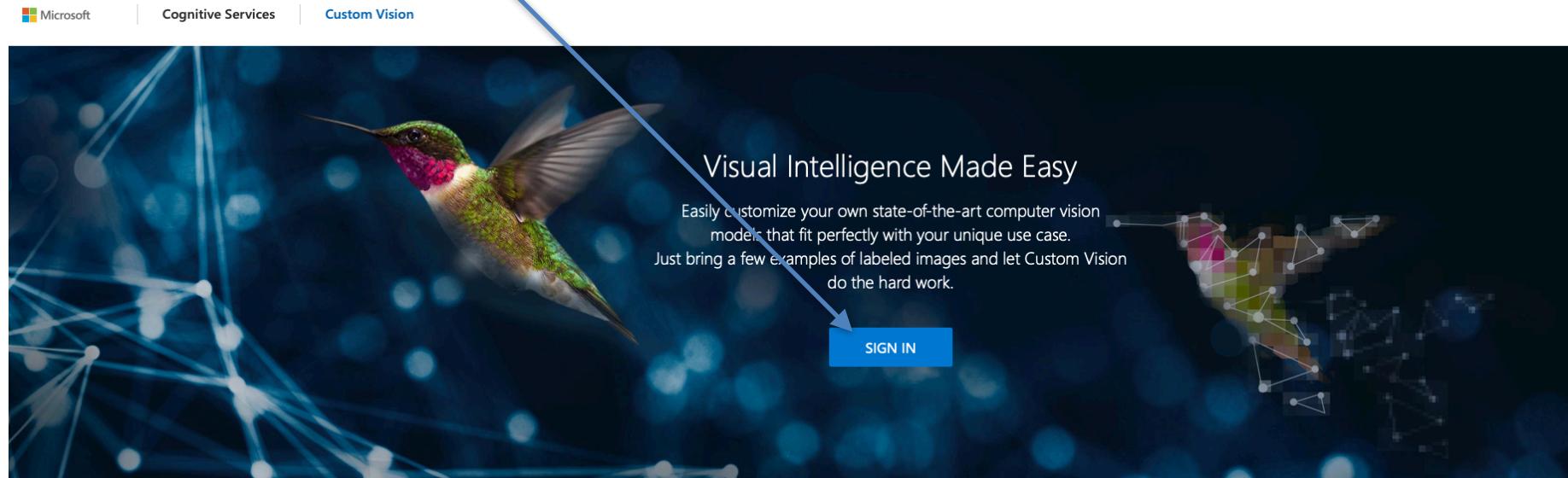
- Step1 AzureMLにアカウント作成（クレジットカード必要）＊初期登録時は2万円相当のクーポンあり
- Step2 AzureMLのCognitive Serviceに登録しログイン
- Step3 AzureML内のCustomVisionサービスにログイン
- Step4 CustomVisionポータルでNewProjectで新規作成
- Step5 新規Projectの設定（Classification またはObjectDetectionを選択）
- Step6 学習させたい画像ファイルを準備して、Add Imagesでファイルをアップロード
- Step7 アップロードした画像ファイルを開き、  
Classificationの場合は画像一枚につき一つのタグを付与  
ObjectDetectionの場合は画像の中の物体を囲ってタグを付与。複数のタグが付与可能
- Step8 全ての画像のタグが、間違いなく付与されていること。要確認
- Step9 Trainボタンで学習モデルのデプロイ開始（サーバーのパワーを使用するため課金発生）  
Quick Training お試しでの学習。学習傾向を見るためのお試し程度。時間としては5分程度  
Advanced Training しっかりと学習。本番運用を見据えたモデル。時間としては1時間以上かかる
- Step10 学習が終わるPerformanceボタンでモデルの精度を確認
- Step11 QuickTestボタンで別の画像（教師データ以外）を用いてモデルのシミュレートして動作確認（Azure上）
- Step12 ExportボタンでモデルをローカルPCにダウンロード（吐き出す際のモデルはONNXモデル、またはTensorFlow）  
＊OpenVINOモデルはバグあり

次ページよりStep3からの手順について説明

# AzureML-Step3 CustomVisionサイト

Custom Visionサイトにログイン

リンク <https://www.customvision.ai/#>



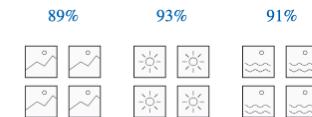
## Upload Images

Bring your own labeled images, or use Custom Vision to quickly add tags to any unlabeled images.



## Train

Use your labeled images to teach Custom Vision the concepts you care about.

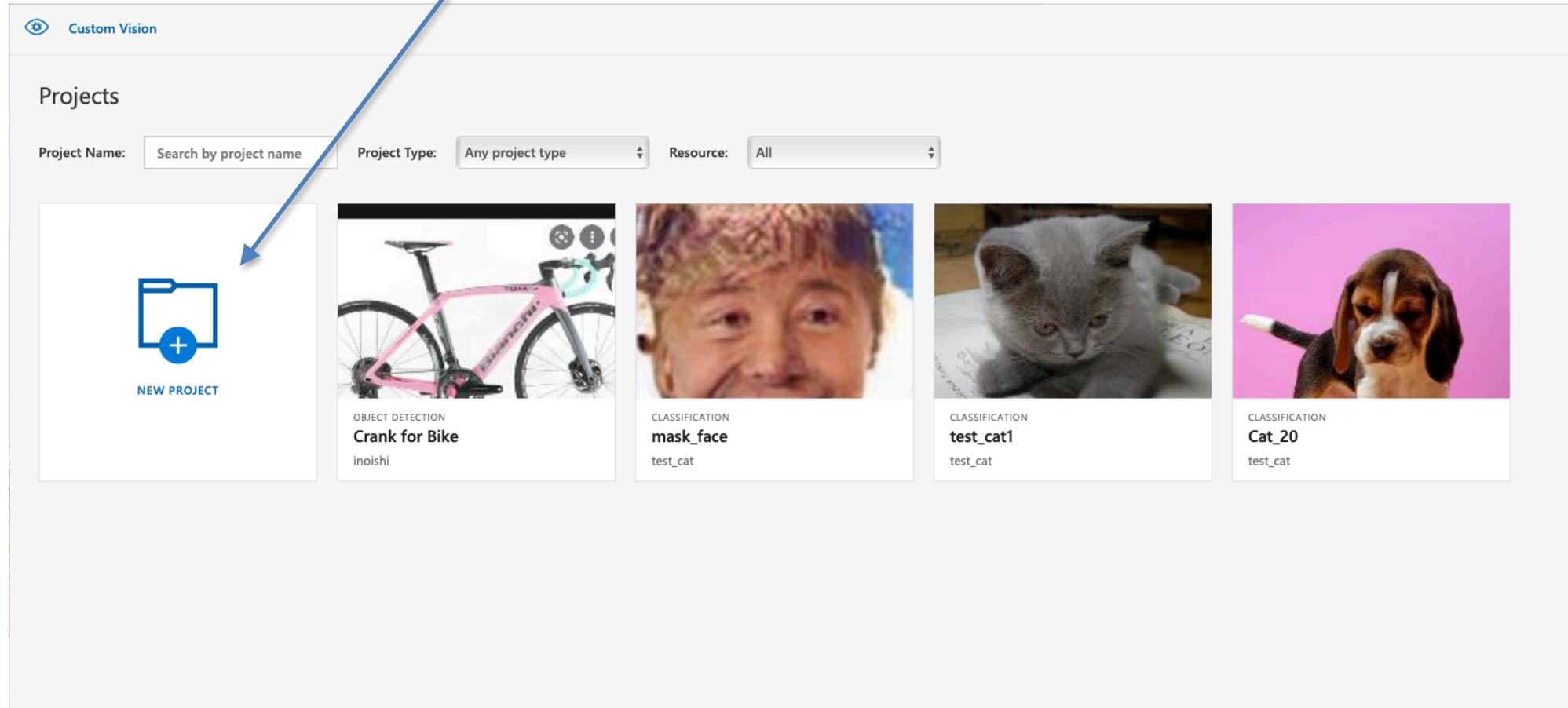


## Evaluate

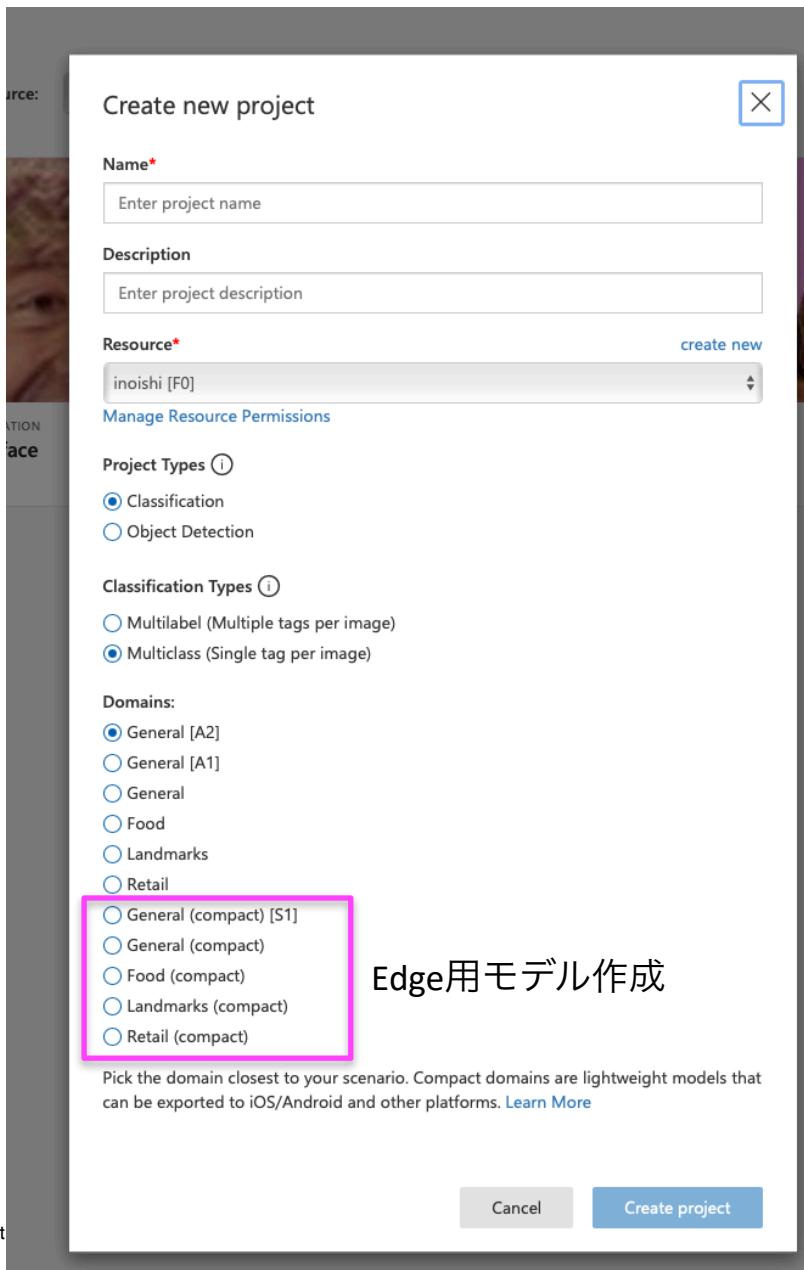
Use simple REST API calls to quickly tag images with your new custom computer vision model.

# AzureML-Step4 Custom Visionポータル

New Projectで新規モデルを作成開始



# AzureML-Step5 新規プロジェクトの設定



Name : プロジェクトの名前 (必須)

Description : プロジェクトの説明 (任意)

Resource : 使用するサーバーを指定 (必須) サーバーにより料金は変わる (必須)

Project Types : (必須)

Multilabel : ObjectDetectionで使用。一枚の画像に複数のタグを付与可能

Multi class : Classificationで使用。一枚の画像に一つのタグを付与可能

Domains : 学習モデルのベースを指定 (転移学習なので分野を指定) (必須)

ただしエッジ (Open VINO) などで動かす場合はCompactモデルを選択する

Classification : General(Compact)を指定

ObjectDetection : General (Compact )[S1]を指定

最後にCreate Projectで新規プロジェクトのベースが作成される

# AzureML-Step5-1 新規プロジェクトの設定(ObjectDetection)

## Project Settings

### General

#### Project Name\*

Road bike for object detection

#### Project Id

cc8f0b41-4c03-4ba3-8973-7707814c7724

#### Description

Shimano Crank

#### Usage: ①

26 training images uploaded; 4974 remain

3 tags created; 47 remain

9 iterations saved; 11 remain

#### Domains:

General [A1]

General

Logo

Products on Shelves

General (compact) [S1]

General (compact)

## 重要設定

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

#### Smart Labeler Preference:

##### Billing Options

Choose the max number of images you would like to run through the Smart Labeler.

More images you analyze will incur higher Azure prediction costs.

All untagged images

Set max limit

1000

**Save Changes**

ObjectDetectionの設定は以下を参照してください

Name : プロジェクトの名前 (必須)

Description : プロジェクトの説明 (任意)

Resource : 使用するサーバーを指定 (必須) サーバーにより料金は変わる (必須)

Domains : 学習モデルのベースを指定 (転移学習なので分野を指定) (必須)

ただしエッジ (Open VINO) などで動かす場合はCompactモデルを選択する

ObjectDetection : **General (Compact )[S1]**を指定

Smart Labeler Preference:

All untagged images

最後にCreate Projectで新規プロジェクトのベースが作成される

General

Project Name\*  
mask\_face for classification

Project Id  
2831ec55-1522-4fae-b57a-a0d24a77bac0

Description  
detect only mask face

Usage: ①  
206 training images uploaded; 99794 remain

2 tags created; 498 remain

1 iterations saved; 19 remain

Domains:

- General [A2]
- General [A1]
- General
- Food
- Landmarks
- Retail
- General (compact) [S1]
- General (compact)
- Food (compact)
- Landmarks (compact)
- Retail (compact)

Pick the domain closest to your scenario. Compact domains are lightweight models that can be exported to iOS/Android and other platforms. [Learn More](#)

Classification Types: ①

- Multilabel (Multiple tags per image)
- Multiclass (Single tag per image)

Export Capabilities: ①

- Basic platforms (Tensorflow, CoreML, ONNX, ...)
- Vision AI Dev Kit

Smart Labeler Preference:

Billing Options

Choose the max number of images you would like to run through the Smart Labeler.

More images you analyze will incur higher Azure prediction costs.

All untagged images

Set max limit

1000

**Save Changes**

## 重要設定

# 新規プロジェクトの設定(Classification)

ObjectDetectionの設定は以下を参照してください

Name : プロジェクトの名前 (必須)

Description : プロジェクトの説明 (任意)

Resource : 使用するサーバーを指定 (必須) サーバーにより料金は変わる (必須)

Domains : 学習モデルのベースを指定 (転移学習なので分野を指定) (必須)

ただしエッジ (Open VINO) などで動かす場合はCompactモデルを選択する

ObjectDetection : **General (Compact)** を指定

Classification Type:

Multi class (Single tag per image) 一枚の画像に対して一つのTag割り当て

Export Capabilities:

Basic platforms(TensorFlow, ONNX, OpenVINO)

Smart Labeler Preference:

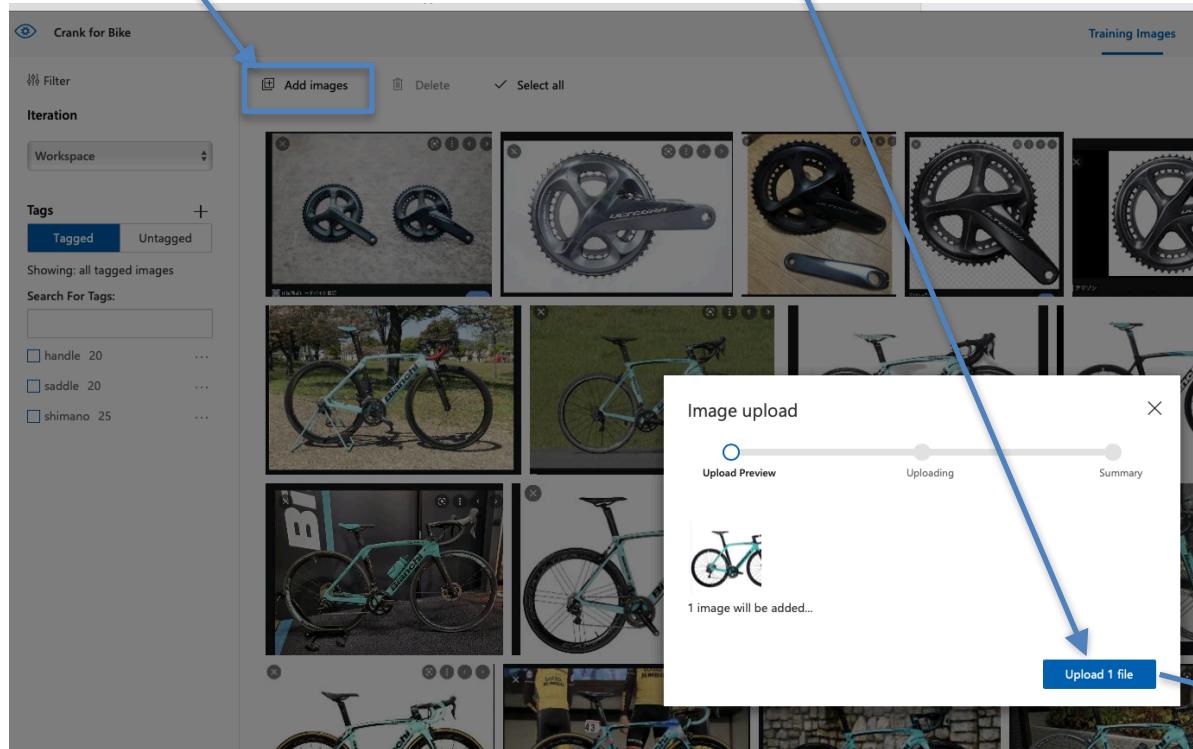
All Untagged images

最後にCreate Projectで新規プロジェクトのベースが作成される

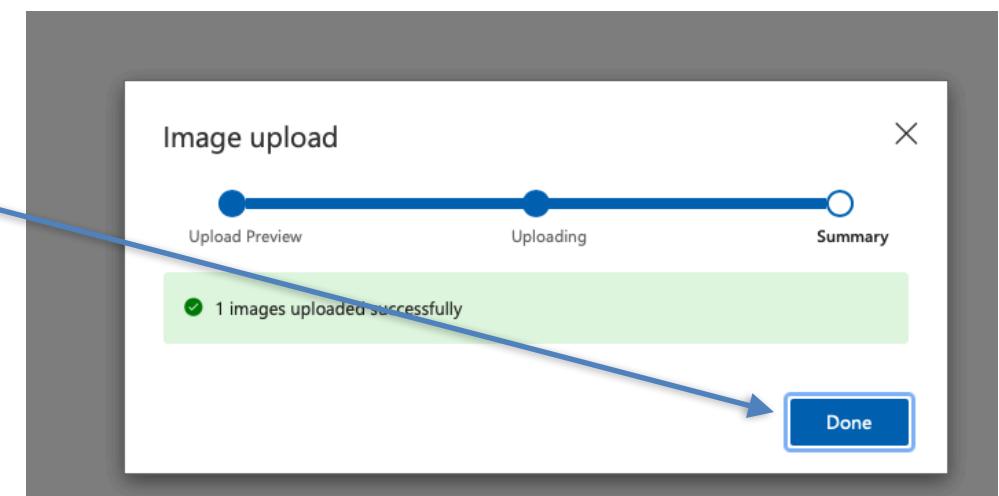
# AzureML-Step6 画像ファイルのアップロード

Add Imagesで準備した画像ファイルをアップロード。精度向上のためには最低でも50枚以上が推奨

\*一度に複数選択可能

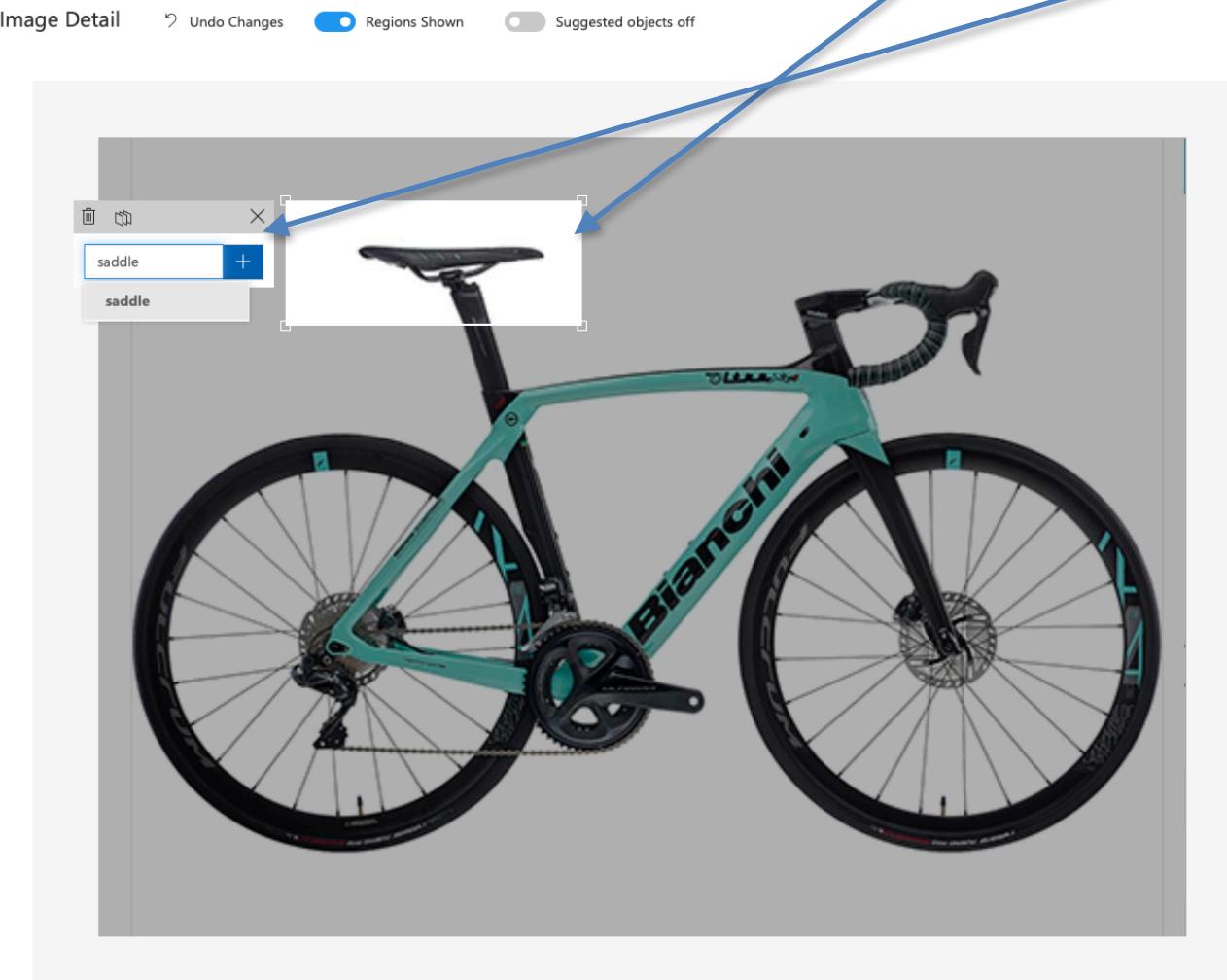


アップロード完了画面

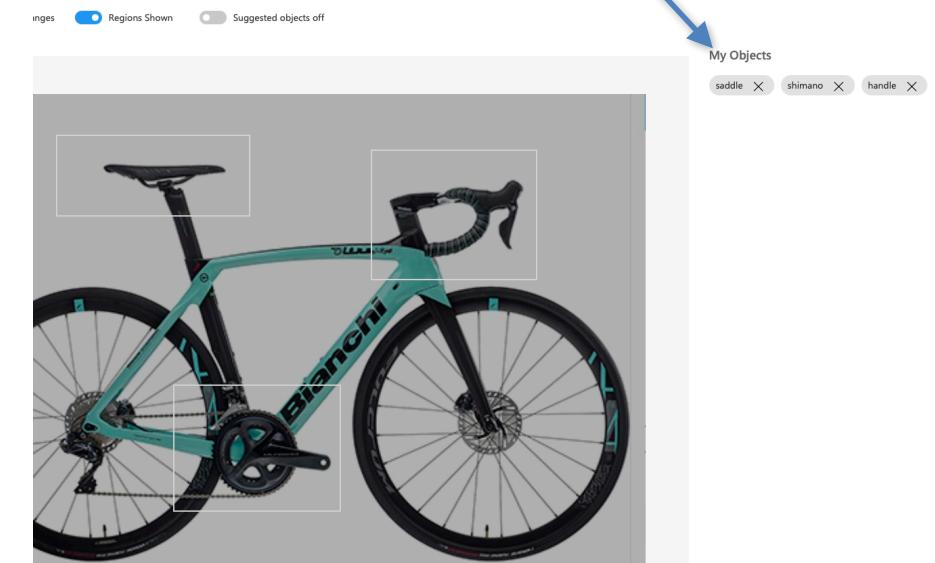


# AzureML-Step7-1 アップロードした画像にタグ付与 (ObjectDetection)

Object Detectionの場合：画像の中のオブジェクトをマウスで囲む。Boxがポップアップるのでタグ名を付与  
一枚の画像に複数のタグを追加付与可能。指定方法は上記と同じ。例saddle,Handle,Crankなど



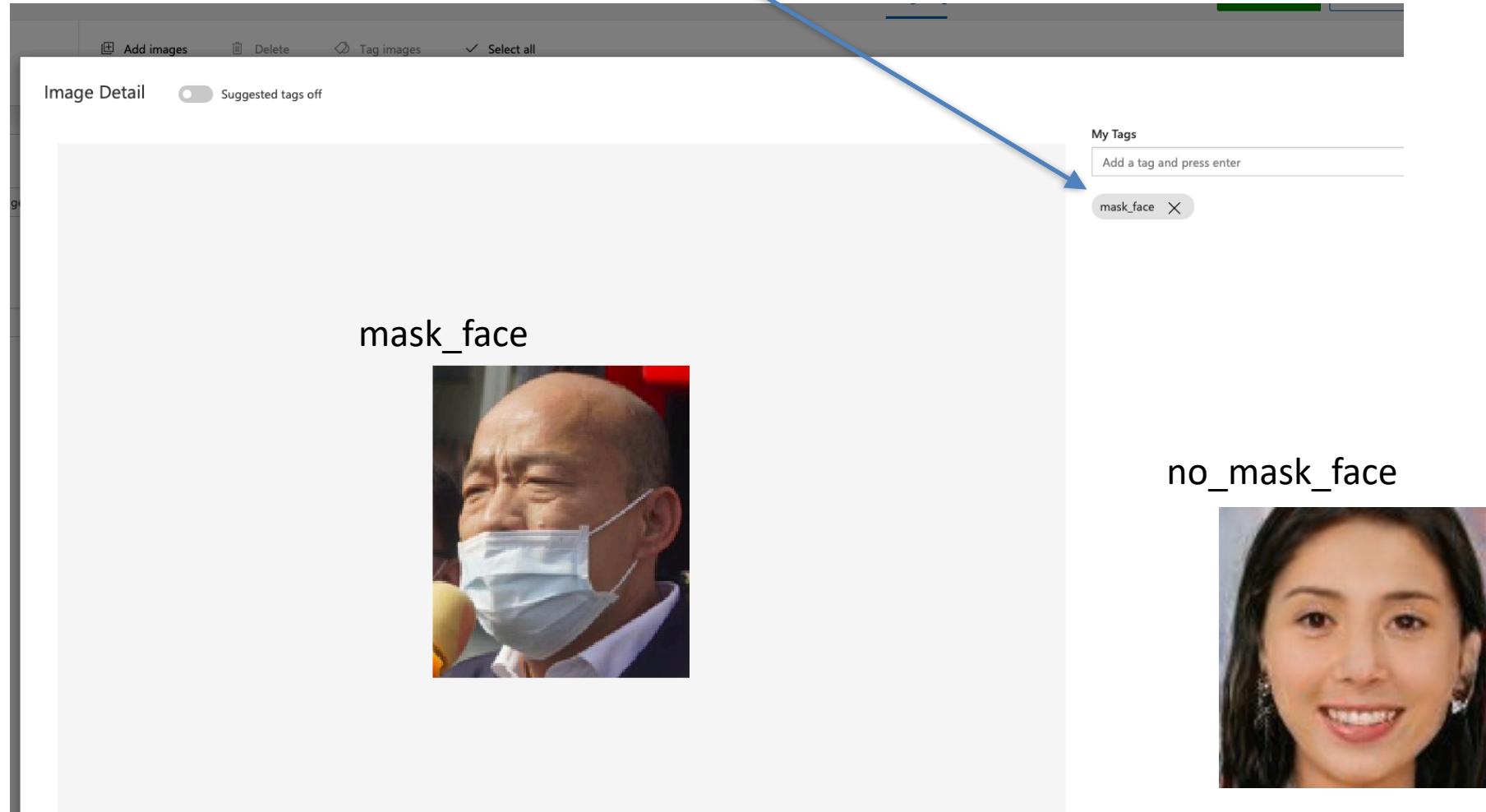
下記は3つのタグを付与した例



## AzureML-Step7-2 アップロードした画像にタグ付与 (Classification)

Classificationの場合：一枚の画像に対して、一つのタグを付与する

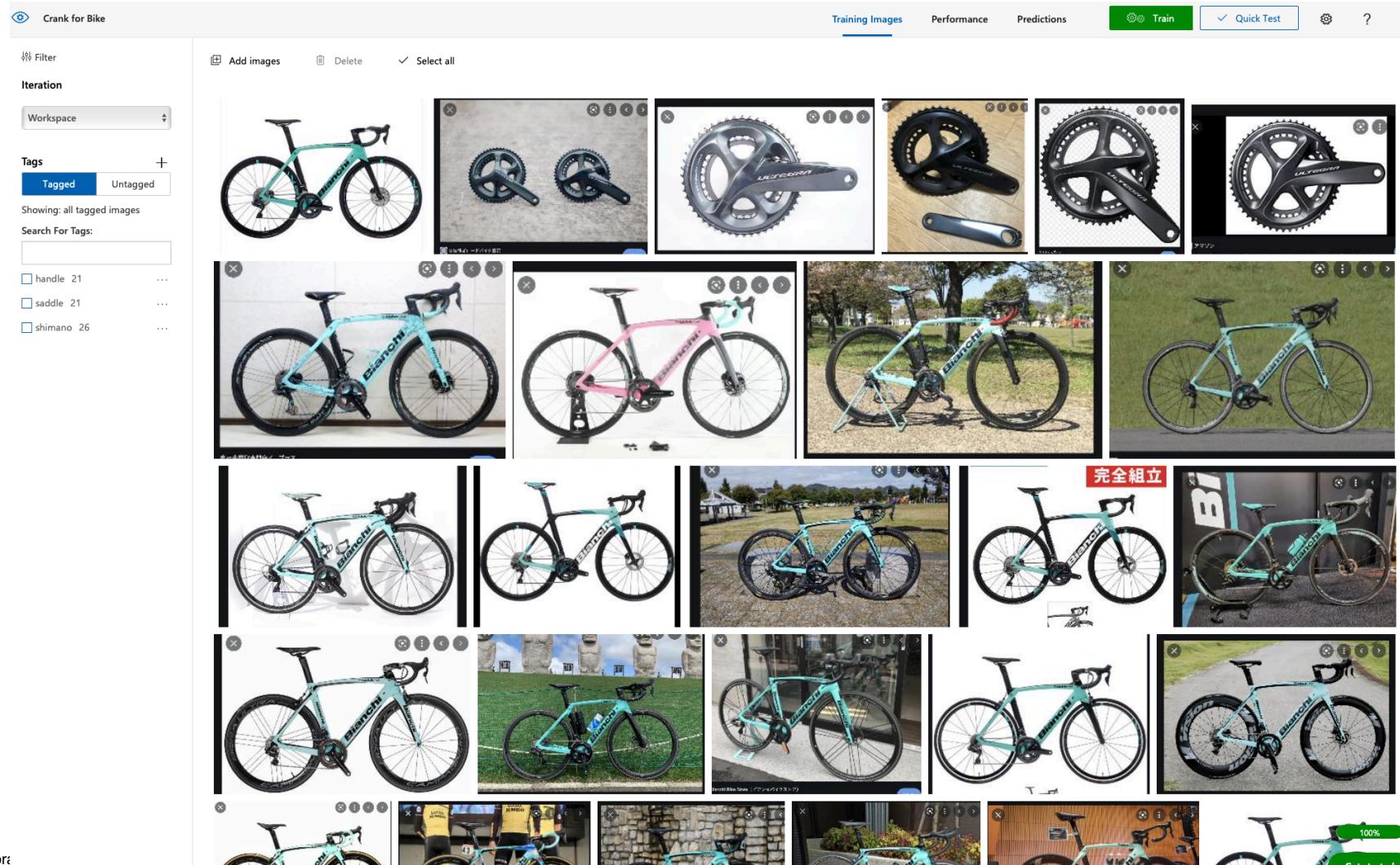
下記例：Mask\_face , no\_mask\_face の2種に分類する



# AzureML-Step8-1 アップロード画像とタグを要確認 (ObjectDetection)

下記例では3つのタグをそれぞれの画像に付与(handle , saddle , shimano)

アップロードした画像をクリックしてタグの位置、名前が正しいこと(再確認すること)



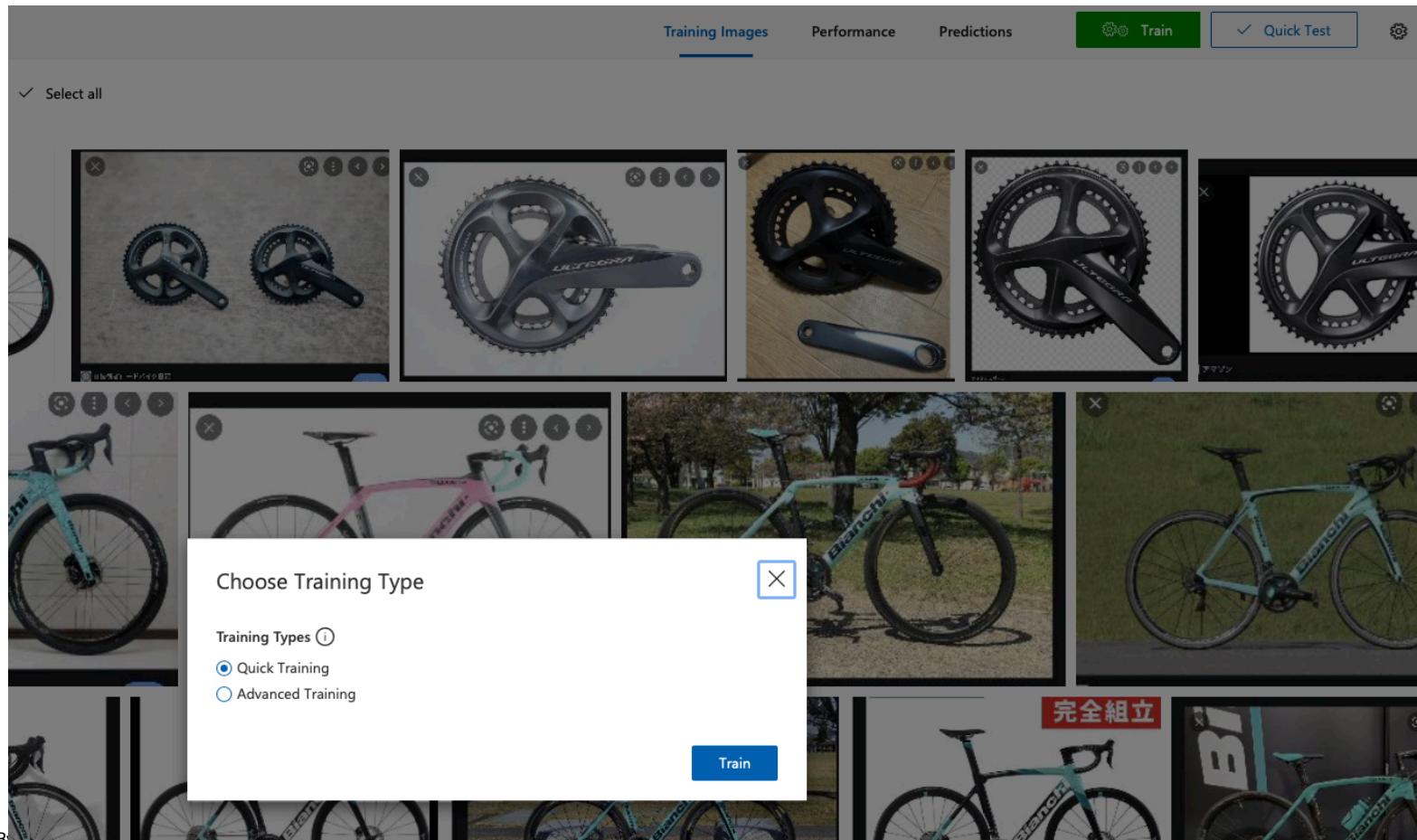
# AzureML-Step9 デプロイ開始

Trainボタンを押すとトレーニングタイプがポップアップ

Quick Training : 5-10分程度でデプロイ完了（傾向などを確認する）想定通り、狙い通りか？ これでもエッジで動作可能

Advanced Training : 1時間程度でデプロイ完了（Quick）で確認後、本格運用として使用する時など

\*吐き出されるトレーニング済みモデルがQuickとAdvancedで推論結果の出力の仕方、タグ名などが変わる場合あり



Azureサーバーを使用する為、  
課金発生するので要注意

最初に登録したクレジットカードから  
引き落としされる。

ただし2万円分のクーポンが最初に付与  
されているので、当分はフリーで使える

# AzureML-Step10 デプロイ完了、Performance確認

デプロイが完了すると、自動的に下記の画面が現れる。現れない場合は以下のボタンを押す

条件を変えてデプロイした履歴一覧

もっともPerformanceが良いものを選択

Iteration	Trained:	Domain
Iteration 9	2 days ago	General (compact) [S1]
Iteration 8	2 days ago	General (compact) [S1]
Iteration 7	2 days ago	General (compact) [S1]
Iteration 6	Advanced Trained: 6 days ago	General (compact) domain, Training Budget: 1 hour
Iteration 5	Trained on: 2022/1/31	General (compact)
Iteration 4	Trained on: 2022/1/31	General (compact)

Crank for Bike

Iterations

Probability Threshold: 50% (50%)

Overlap Threshold: 30% (30%)

Precision (100.0%)

Recall (100.0%)

mAP (100.0%)

Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
shimano	100.0%	100.0%	100.0%	25
handle	100.0%	100.0%	100.0%	20
saddle	100.0%	100.0%	100.0%	20

付与したタグ

使用したイメージファイル数

アップロードした画像ファイル（教師データ）をもとにデプロイした結果が出力される。

Precision：タグと学習後のデータの精度

Recall：学習後、自分で動作確認後の精度

mAP：全体的なパフォーマンス精度

全ての値が100%に近いほど、タグと学習結果に差異がない=精度が高い

# AzureML-Step11 Quickテストで簡易動作確認

QuickTestを押下して、画像を選択（教師データはダメ） 画像をアップロードすると、自動的に推論実施 Predictionsで動作後の精度を確認。意図した結果となっているか？確認する

The screenshot shows the Azure ML studio interface with the 'Predictions' tab selected. A blue box highlights the 'Quick Test' button in the top navigation bar. A blue arrow points from the text 'Quick Test' to this button. Below the navigation bar, a modal window titled 'Quick Test' is open. It contains a grayscale image of a teal and black road bicycle. Three specific parts of the bike are highlighted with red boxes: the handlebar, the saddle, and the rear wheel area. To the right of the image, there is a configuration section with three main sections: 'Image URL', 'Browse local files', and 'Iteration'. A blue arrow points from the text '試験用ファイル選択' (Select test file) to the 'Browse local files' button. Another blue arrow points from the text 'デプロイしたモデルを指定' (Specify deployed model) to the 'Iteration' dropdown menu, which is set to 'Iteration 9'. At the bottom of the modal, there is a 'Predictions' section with a table showing three categories: 'handle', 'shimano', and 'saddle', each with a probability of '99.9%'. A blue box highlights this table. A blue arrow points from the text '試験後の結果' (Test results) to this table. The text below it explains: 'それぞれのタグをマウスで指定すると左図の中の部位がポップアップする' (When you click on each tag with the mouse, the corresponding part in the left image will pop up).

Quick Test

試験用ファイル選択

デプロイしたモデルを指定

試験後の結果

それぞれのタグをマウスで指定すると左図の中の部位がポップアップする

Tag	Probability
handle	99.9%
shimano	99.9%
saddle	99.9%

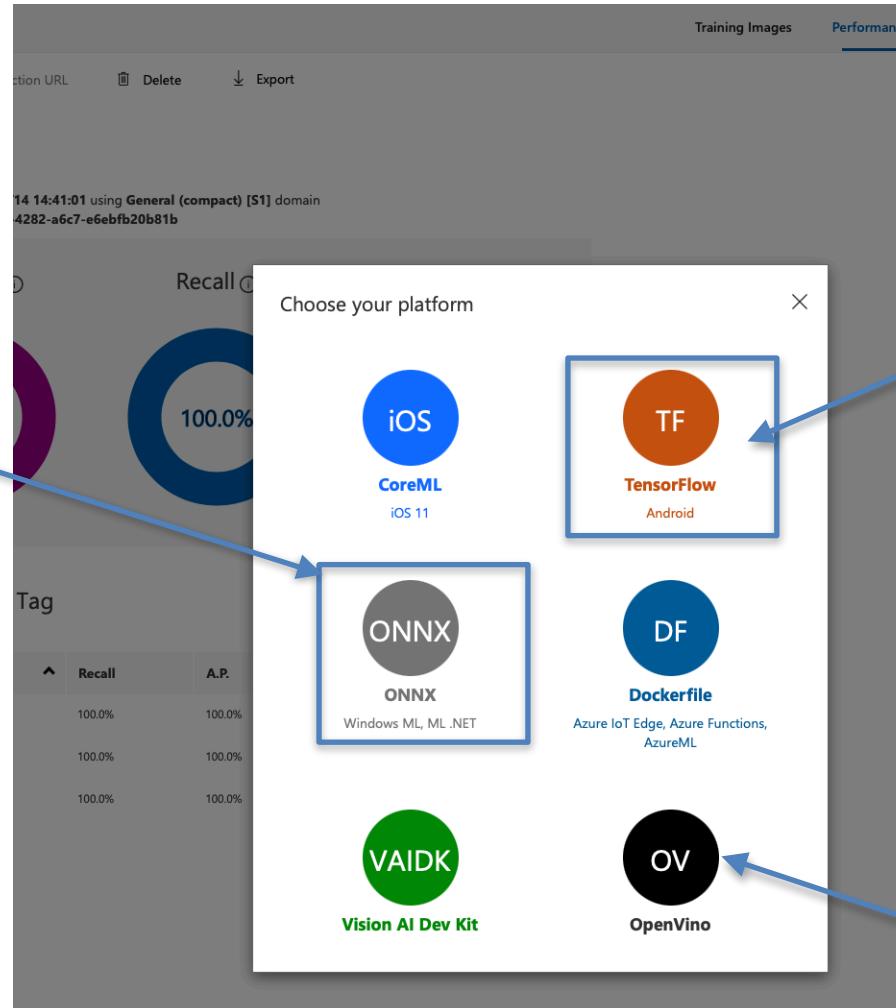
# AzureML-Step12 学習済みモデルをエクスポート

OpenVINOで動作させるために学習したモデルをエクスポート（ダウンロードする）

要注意！！

ObjectDetectionの場合：ONNXモデルを指定する（他のモデルはバグで動作しなかった）

Classificationの場合：TensorFlowモデルを指定する（同上）



ObjectDetectionの場合

Classificationの場合

ONNXモデルはOpenVINO  
で変換せずに使用する（ダイレクトサポート機能）

TensorFlowモデルはOpenVINO  
でMO\_TF.pyで変換して使用する

\*色々試したのだが、上記方法以外は  
全て誤動作となつた。

今のところ、上記以外の方法なし

OpenVINOモデル（IR）形式で吐き出されるが  
推論結果がバグだらけで使い物にならない



AzureMLで作成した学習済みモデルを使用して、OpenVINOに実装する方法について。

- ClassificationはTensorFlowモデルを使用
- ObjectDetectionはONNXモデルを使用

それぞれの方法について次ページにて説明

# OpenVINO Classification動作編

Classificationとは一枚の画像を推論して、何の画像分類に区分けされるか？のモデルである  
本例で分類は0：マスク有り、1：マスク無し を区別する

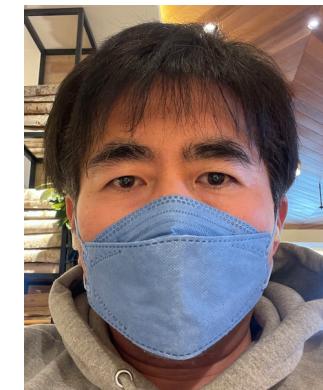
```
cup.png labels.txt may.jpg mitsuruz.png squeezenet1.1.labels test2.jpg top_squeeze4.  
parallels@parallels-Parallels-Virtual-Platform:~/Documents/mask_face$ python3 top_mask.py  
Get_batch_size= 1  
Input Spec = 1 3 224 224  
libpng warning: iCCP: extra compressed data  
top_mask.py:74: DeprecationWarning: 'outputs' property of InferRequest is deprecated. Please  
    res = self.exec_net.requests[0].outputs[self.output_blobs]  
all_list_____ [0, '1.00000', 'mask_face']  
all_list_____ [1, '0.00000', 'no mask face']  
  
Get_batch_size= 1  
Input Spec = 1 3 224 224  
all_list_____ [0, '0.15879', 'mask_face']  
all_list_____ [1, '0.84121', 'no mask face']  
  
parallels@parallels-Parallels-Virtual-Platform:~/Documents/mask_face$
```

## 結果

1枚目：100% mask\_face

2枚目：84% no mask face

2枚の画像を推論  
1枚目



2枚目



# OpenVINO実装編 (Classification)

1. TensorFlowモデルをOpenVINO形式 (IR) に変換する。

```
python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo_tf.py --input_model 'model.pb' -b 1 --data_type FP32
```

変換後はmodel.xml / model.bin が生成される。そのファイルを任意の場所にコピーする

2. OpenVINOのプログラムの中で、変換したモデルのパスを指定 (通常のやり方)

例 : model = './ir\_eng/model' <—モデルのパス指定



```
#Step2__ SqueezeNet1.1を使用するためのクラス
class squeesenet:
    #本クラスの初期化メソッド
    #推論モデルの構築、使用するプラグインを指定して、プラグインヘロードし
    #推論モデルのInput条件も取得
    def __init__(self,__image):
        #推論モデルのパスを指定
        model = './ir_eng/model'
        #model = './ir_eng/squeezeNet1.1'
        self.image = __image

    #read_network(OpenVINO API)で実行可能なモデルを構築
    net = ie.read_network(model=model+'.xml', weights=model+'.bin')
    #使用するプラグインを指定
    __plug='CPU'
    #__plug='GPU'
    #__plug='MYRIAD'
    #__plug='HETERO:FPGA,CPU'

    #使用する推論モデルのInput条件を取得
    print("Get_batch_size1=",net.batch_size)
    self.input_layer = next(iter(net.input_info))
    self.output_blobs = next(iter(net.outputs))
    self.model_n, self.model_c, self.model_h, self.model_w = net.input_info[self.input_layer].input_data.shape
        #net.input_info[self.input_layer].input_data.shape
    print("Input Spec = ",self.model_n,self.model_c,self.model_h,self.model_w)

    #load_network(openvino API)にて推論モデルをプラグインにロード
    self.exec_net = ie.load_network(network=net, device_name=__plug)
    return
```

# OpenVINO ObjectDetection動作編

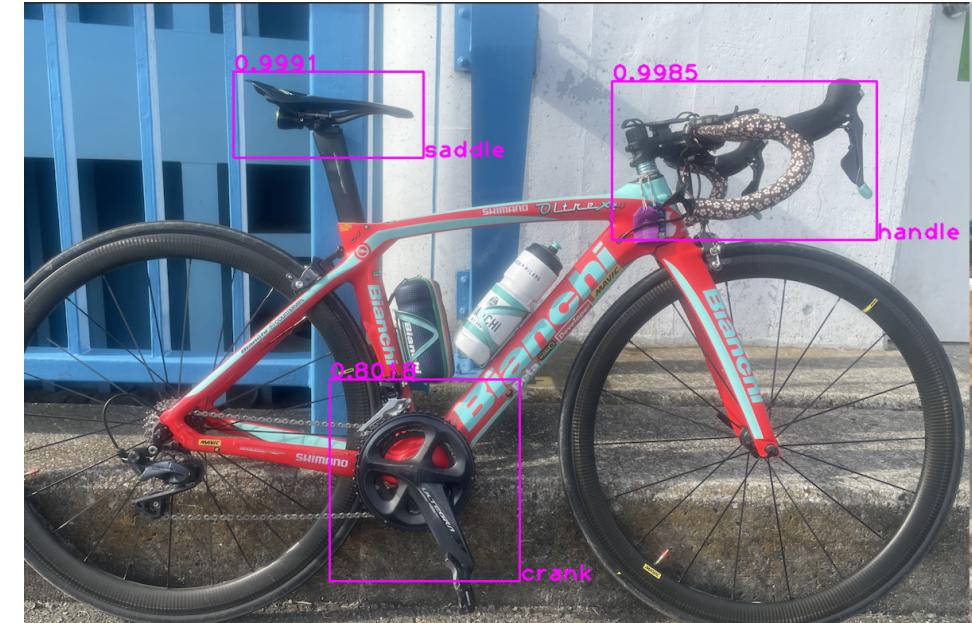
ObjectDetectionとは一枚の画像の中に何のオブジェクトがあるのか？を推論するモデルである。  
本例で分類はsaddle , handle , crankを画像の中から検出、その場所と精度を出力する

```
parallels@parallels-Parallels-Virtual-Platform:~/Documents/shimano$ python3 top_shimano.py
Get_batch_size1= 1
Input Spec = 1 3 320 320
top_shimano.py:103: DeprecationWarning: 'outputs' property of InferRequest is deprecated. Please in
    res = self.exec_net.requests[0].outputs
image_size_ 816 1247
len_ 3
labels2_ ['handle', 'saddle', 'crank']
list1_ [[0.9991, 'saddle']]
list2_ [[0.9991, 'saddle']]
__list_1_ 0.9991 0
__list_2_ saddle 0
xy_&item_ 321 85 560 193 0
labels2_ ['handle', 'saddle', 'crank']
list1_ [[0.9991, 'saddle'], [0.9985, 'handle']]
list2_ [[0.9985, 'handle'], [0.9991, 'saddle']]
__list_1_ 0.9985 1
__list_2_ handle 1
xy_&item_ 797 97 1130 296 1
labels2_ ['handle', 'saddle', 'crank']
list1_ [[0.9985, 'handle'], [0.9991, 'saddle'], [0.8018, 'crank']]
list2_ [[0.8018, 'crank'], [0.9985, 'handle'], [0.9991, 'saddle']]
__list_1_ 0.8018 2
__list_2_ crank 2
xy_&item_ 442 471 681 724 2
```

## 結果

saddle : 99%  
handle : 99%  
crank : 80%

以下の画像を推論



# OpenVINO実装編（ObjectDetection）

1. ONNXモデルをそのまま使用する。変換はしない。任意のフォルダにコピーする。以下に説明があるが、2,4を適用すれば良い

[https://docs.openvino.ai/latest/openvino\\_docs\\_IE\\_DG\\_ONNX\\_Support.html?sw\\_type=switcher-python#](https://docs.openvino.ai/latest/openvino_docs_IE_DG_ONNX_Support.html?sw_type=switcher-python#)

2. OpenVINOのプログラムの中で、変換したモデルのパスを指定（通常のやり方）

例 : model = './ir\_eng/\*\*\*\*\*.onnx' <—モデルのパス指定

3. 出力データは6次元

物体の場所(xmin,ymin,xmax,ymax)

物体のタグ(0:handle 1:saddle 2:rank)

物体の精度(\*\*%)

ObjectDetectionの出力データは、  
他のモデルでも同じ次元、形式となる傾向である

ONNXモデルのダイレクトサポートは  
OpenVINO2020.4～なので要注意

4.net = model

reshapeは手入力

値は次ページで解説

```
#Step2__ SqueezeNet1.1を使用するためのクラス
class squeezenet:
    #本クラスの初期化メソッド
    #推論モデルの構築、使用するプラグインを指定して、プラグインヘロードし
    #推論モデルのInput条件も取得
    def __init__(self, __image):
        #推論モデルのパスを指定
        #model = './ir_eng/model'
        #model = './ir_eng/ef2e6ac2817541eeb778c550b16d47d7-3/model.onnx'
        #model = './ir_eng/ef2e6ac2817541eeb778c550b16d47d7/model'
        #model = './ir_eng/squeezeNet1.1'
        #model = './ir_eng/ef2e6ac2817541eeb778c550b16d47d7-3/model.onnx'
        #model = './ir_eng/ccd135ef34d74c680895ae5432f639e6/python/model'
        #model = './ir_eng/446957126f274980afe567994e76d84c/model'
        #model = './ir_eng/446957126f274980afe567994e76d84c-2/model.onnx'
        #model = './ir_eng/1778e97e8baf4d39a7b161c979a234c7-2/model.onnx'
        #model = './ir_eng/8b90631c280c4282a6c7e6ebfb20b81b-3/model'
        #model = './ir_eng/8b90631c280c4282a6c7e6ebfb20b81b-2/model'
        #openvino mode error happen , unkown reason...
        model = './ir_eng/8b90631c280c4282a6c7e6ebfb20b81b/model.onnx' # sa
        self.image = __image

        #read_network(OpenVino API)で実行可能なモデルを構築
        #net = ie.read_network(model=model+'.xml', weights=model+'.bin')
        #net = ie.read_network(model='./ir_eng/tf_model/model.pb')
        net = ie.read_network(model)

        self.input_layer = next(iter(net.input_info))
        #net.reshape({self.input_layer: (1, 3, 416, 416)})
        net.reshape({self.input_layer: (1, 3, 320, 320)})
```

## reshapeの値

reshape=画像ファイルのサイズを指定するための値は、ONNXモデルの場合手入力となる。

値を確認するには、metadata\_properties.json ファイルを開くと値が記載されている。

\*このファイルはAzureMLからONNXモデルをエクスポートしたときに含まれているファイルである。

その値をOpenVINO側に設定する必要があるので要注意

\*1枚の画像、3原色（RGB）、画像サイズ → 1, 3, 320, 320 である

```
export.MANIFEST_TABLES.TXT LICENSE Metadata_properties.json MODEL.ONNX README.MD
parallels@parallels-Parallels-Virtual-Platform:~/Documents/shimano/ir_eng/8b90631c280c4282a6c7e6ebfb20b81b$ cat metadata_properties.json
{
    "CustomVision.Metadata.AdditionalModelInfo": "",
    "CustomVision.Metadata.Version": "1.2",
    "CustomVision.Postprocess.Method": "SSD",
    "CustomVision.Postprocess.Yolo.Biases": "[ ]",
    "CustomVision.Postprocess.Yolo.NmsThreshold": "0.0",
    "CustomVision.Preprocess.CropHeight": "0",
    "CustomVision.Preprocess.CropMethod": "NoCrop",
    "CustomVision.Preprocess.CropWidth": "0",
    "CustomVision.Preprocess.MaxDimension": "0",
    "CustomVision.Preprocess.MaxScale": "0.0",
    "CustomVision.Preprocess.MinDimension": "0",
    "CustomVision.Preprocess.MinScale": "0.0",
    "CustomVision.Preprocess.NormalizeMean": "[0.0, 0.0, 0.0]",
    "CustomVision.Preprocess.NormalizeStd": "[1.0, 1.0, 1.0]",
    "CustomVision.Preprocess.ResizeMethod": "Stretch",
    "CustomVision.Preprocess.TargetHeight": "320",
    "CustomVision.Preprocess.TargetWidth": "320",
    "Image.BitmapPixelFormat": "Rgb8",
    "Image.ColorSpaceGamma": "SRGB",
    "Image.NominalPixelRange": "Normalized_0_1"
```