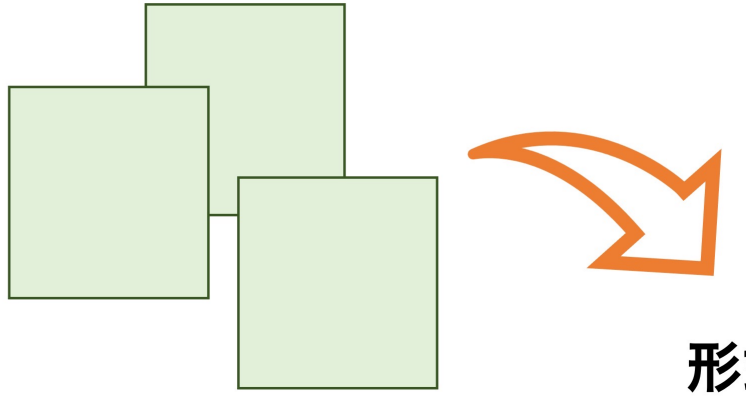
自作のデータセットでDCNNを
ファインチューニングする

TFDSについて

データセットの作成

さまざまな種類・形式のデータセット
さまざまな場所に分散されている

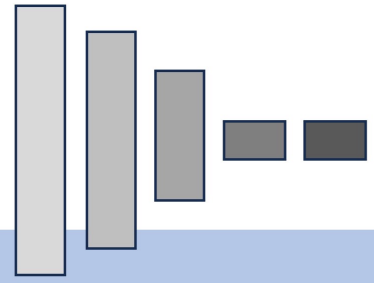


パイプラインに
乗せられない

形式を標準化

TFDS

機械学習モデルに投入

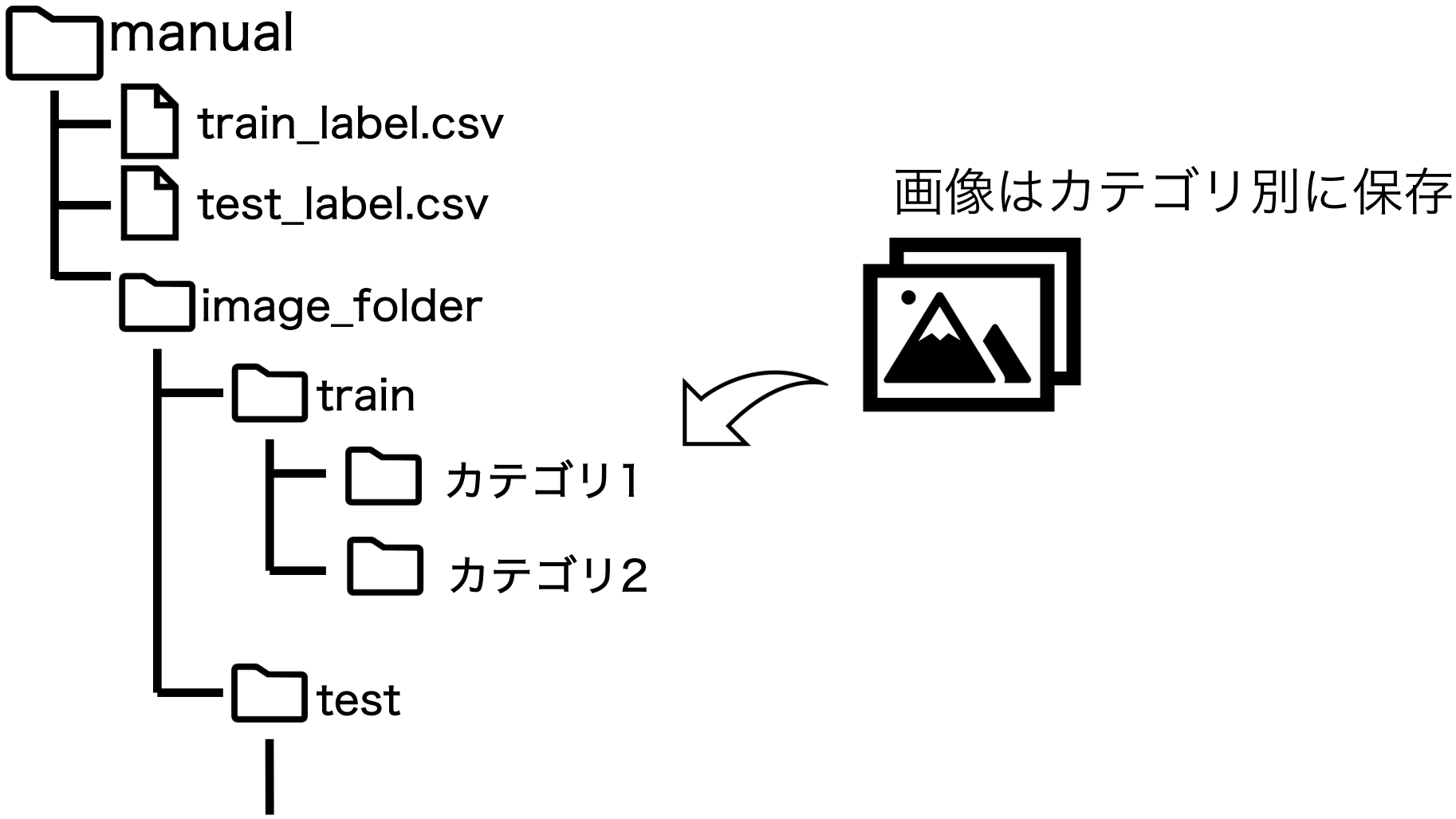


機械学習
パイプライン

データの送信元は？（URL? ローカルファイル？）
データセットはどのように見えるか（特徴量）
データはどのように分割されるか（TRAIN と TEST など）
データセット内の個別の Example

ディレクトリ構成について

/home/<user-name>/database/<dataset-name>/downloads/manual/

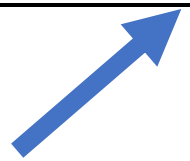


データセット作成

step1. tfds newコマンドでデータセットを標準化するための**”箱”**を作る

terminal上で以下を実行

```
$ cd ~/
$ mkdir database/
$ cd database
$ tfds new sndl_size_discrim
```



データセット名を入れる

※命名規則

- 頭にsndl (SaNaDaLab) をつける
- 後ろはタスク名がわかる名前
  (Size Discrimination : 大きさ弁別)
- **全て小文字にする ← 大文字だとデータセットが認識されない**

データセット作成

step2. create_dataset_tutorial.ipynbをGithubからダウンロードする

→ ダウンロードしたファイルはsndl_size_discrim以下に保存

step3. ファイル保存先を設定

```
username = ''                                                    # ユーザ名
datasetname = 'sndl_size_discrim'                                # データセット名
train_csv = 'train_label.csv'                                    # trainラベルを示すcsvファイル名
test_csv = 'test_label.csv'                                      # testラベルを示すcsvファイル名

path_to_manual = f'/home/{username}/database/{datasetname}/downloads/manual/' # 現在のディレクトリまでのパス

train_data_path = path_to_manual + 'image_folder/train/'     # 作成したtrainデータの保存先
test_data_path = path_to_manual + 'image_folder/test/'       # 作成したtestデータの保存先
print(train_data_path)
print(test_data_path)
```

✓ 0.0s

```
/home//database/sndl_size_discrim/downloads/manual/image_folder/train/
/home//database/sndl_size_discrim/downloads/manual/image_folder/test/
```

データセット作成

step4. 生成する画像の変数を設定

```
categories = ['small', 'large']                                                  # ラベルのカテゴリ
N_data = 10                                                                      # 1カテゴリあたりの画像枚数
N_category = len(categories)                                                     # カテゴリ数

image_shape = (224, 224, 3)                                                      # DCNNに入力する画像のサイズ

left_cntr = (int(image_shape[1] * 1/4)+5, int(image_shape[0]/2))                 # 左側の中心座標(x,y)
right_cntr = (int(image_shape[1] * 3/4)-5, int(image_shape[0]/2))                # 右側の中心座標(x,y)
cntr = [left_cntr, right_cntr]                                                   # 中心座標

freq=0.2                                                                         # 生成する画像のテクスチャの空間周波数
vNum=[3,4,5,6,7,9]                                                               # 生成する多角形の頂点の数

RefDisk_radius_pix = 20                                                          # Reference diskの半径のピクセル値
TestDisk_radius_arr = np.array([0.461, 0.569, 0.727, 0.825, 1.169, 1.340, 1.530, 1.741])   # Test diskの半径(Reference diskに対する相対値)
TestDisk_radius_pix = TestDisk_radius_arr * RefDisk_radius_pix                   # Test diskの半径のピクセル値

print('基準刺激の半径 : ', RefDisk_radius_pix)
print('テスト刺激の半径 : ', TestDisk_radius_pix)
print('条件組み合わせ数', N_data*len(TestDisk_radius_arr)*2*len(vNum))
```

```
基準刺激の半径 :  20
テスト刺激の半径 :  [ 9.22 11.38 14.54 16.5  23.38 26.8  30.6  34.82]
条件組み合わせ数 960
```

データセット作成

step5. 生成した画像を格納するディレクトリを作成

```
    '''
    ./image_folder/
        train/
            category_A/
                    |-- category_A_000.jpg
                    |-- category_A_001.jpg
            categoty_B/
                    |-- category_B_000.jpg
                    |-- category_B_001.jpg
        validation/
    '''
    for ii in range(N_category):
        try:
            os.makedirs(train_data_path+categories[ii])   # trainデータ用ディレクトリ
        except:
            print('ディレクトリの生成に失敗しました. ')
            pass
        try:
            os.makedirs(test_data_path+categories[ii])   # testデータ用ディレクトリ
        except:
            print('ディレクトリの生成に失敗しました. ')
            pass
```

データセット作成

step5. コードを実行して画像を生成

step6. ラベルをcsvファイルで保存

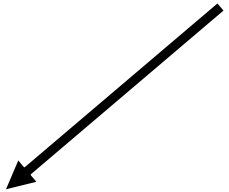
```
train_image_list = glob.glob(train_data_path+'*/*')                                  # trainデータのパスをリストで取得
train_image_list = natsort.natsorted(train_image_list)                               # 画像を連番になるようにソート

train_label_list = [os.path.split(os.path.split(fname)[0])[1] for fname in train_image_list]    # ラベルを画像数分用意

file_label_list = zip(train_image_list, train_label_list)                            # trainデータにラベルを対応付け

df = pd.DataFrame(file_label_list, columns=['file_path', 'label'])                   # データフレーム化
df.to_csv(path_to_manual + train_csv)                                                # csvファイルに保存
df.head()
```

TFDSでデータセット形式を標準化

step1. <dataset-name>_dataset_builder.pyの内容を自分のタスクに合うように変更

```
$ cd sndl_size_discrim
$ ls
```

この中身を変更



```
__init__.py
sndl_size_discrim_dataset_builder.py      # Dataset definition
sndl_size_discrim_dataset_builder_test.py      # (optional) Test
dummy_data/                    # (optional) Fake data (used for testing)
checksum.tsv                   # (optional) URL checksums (see `checksums` section).
```

TFDSでデータセット形式を標準化

sndl_size_discrim.pyの中身を変更する
赤色の部分を追加・変更

```
import csv

class Builder(tfds.core.GeneratorBasedBuilder):
    """DatasetBuilder for sndl_size_discrim dataset."""
    # manualのダウンロード方法
    MANUAL_DOWNLOAD_INSTRUCTIONS = " The destination of the manual :
    /home/<user-name>/database/sndl_size_discrim/downloads/manual/"

    VERSION = tfds.core.Version('1.0.0')
    RELEASE_NOTES = {
    '1.0.0': 'Initial release.',
    }
```

TFDSでデータセット形式を標準化

```
def _info(self) -> tfds.core.DatasetInfo:
    """Returns the dataset metadata."""
    # TODO(sndl_size_discrim): Specifies the tfds.core.DatasetInfo object
    return tfds.core.DatasetInfo(
        builder = self,
        # データセットの説明
        description = "Custom dataset with 'large' and 'small' labels.",
        features=tfds.features.FeaturesDict({
            # These are the features of your dataset like images, labels # 画像の形
            状，ラベルの種類を指定.
            'image': tfds.features.Image(shape=(224, 224, 3)),
            'label': tfds.features.ClassLabel(names=['large', 'small']),
        }),
        # If there's a common (input, target) tuple from the
        # features, specify them here. They'll be used if
        # as_supervised=True in builder.as_dataset

        # データの対応関係を定義
        supervised_keys=('image', 'label'), # Set to `None` to disable
    )
```

TFDSでデータセット形式を標準化

```
def _split_generators(self, dl_manager: tfds.download.DownloadManager):
    """Returns SplitGenerators."""
    # TODO(sndl_size_discrim): Downloads the data and defines the splits
    # データセットの保存場所.
    # 訓練データ用のラベルファイルの絶対パス
    train_label_path = dl_manager.manual_dir / 'train_label.csv'
    # テストデータ用のラベルファイルの絶対パス
    test_label_path = dl_manager.manual_dir / 'test_label.csv'

    # TODO(sndl_size_discrim): Returns the Dict[split names, Iterator[Key, Example]]
    return [
        # trainデータの分割方法を記述
        tfds.core.SplitGenerator(
            name=tfds.Split.TRAIN,
            # gen_kwargsのkey名は，_generate_examples()に渡す引数名と同じにする.
            gen_kwargs={'label_path': train_label_path},
        ),
        # testデータの分割方法を記述
        tfds.core.SplitGenerator(
            name=tfds.Split.TEST,
            gen_kwargs={'label_path': test_label_path},
        ),
    ]
```

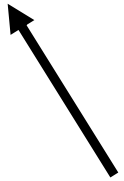
TFDSでデータセット形式を標準化

```
def _generate_examples(self, label_path): # 引数：gen_kwargsのkey名
    """Yields examples."""
    # TODO(sndl_size_discrim): Yields (key, example) tuples from the dataset
    # [ファイルパス, ラベル]のペアになったcsvを読み込んで画像の例とラベルを出力.
    with open(label_path) as f:
        for row in csv.DictReader(f):
            image_id = row['file_path']
            # And yield (key, feature_dict)
            yield image_id, {
                'image': image_id,
                'label': row['label’],
            }
```

TFDSでデータセット形式を標準化

step2. tfds buildコマンドで作成したデータセットをビルドする

```
$ cd ~/database/sndl_size_discrim
$ tfds build ./ --manual_dir=~/database/my_dataset/downloads/manual
```



ビルドの引数に ”--manual_dir=“ を設定することで,
_split_generators()のdl_manager.manual_dirのパスを指定できる.

※データセットの中身を変更したら，tfds newからやり直す

TFDSでデータセット形式を標準化

データセットの中身を変更した場合

1. /home/<ユーザ名>/tensorflow-datasets/ 以下に自分のデータセットが保存されているので，それを削除する.

2. データセットの中身を変更する.

3. tfds build コマンドを再度実行する.

機械学習モデルに投入

step1. 作成したデータセットを読み込む

```
learning_dataset = ‘sndl_size_discrim'

# 作成したデータセットの読み込み
dataset, dataset_metadata = tfds.load(
    learning_dataset,
    with_info = True,
    shuffle_files = True,
    as_supervised = True,
    batch_size = -1)

print(dataset_metadata)
```