

CA Tech Lounge 課題3

Computer Vision2: 画像分類と物体検知

澤木陽人

April 19, 2023

Contents

1 概要	1
2 データセットについて	2
3 問1：画像の多値分類	3
3.1 学習データの下準備	3
3.2 モデルの転移学習	5
3.3 出力結果の比較	7
4 問2：画像の物体検知	10
4.1 YOLOによる推論のための下準備	10
4.2 YOLOの学習	13
4.3 テストデータの推論結果	16
4.4 バウンディングボックス情報の取得	18
5 終わりに	21
6 参考	22

1 概要

この課題では、まずデータセットに含まれる80種類のクラスのうち、指定のあった19クラスの分類を行う画像分類タスクに取り組んだ。分類では、Resnet50とVGG16の転移学習およびファインチューニングを行った。また、問2ではYOLOを用いた物体検知タスクとして、同じデータセットをもとに、ファインチューニングを行った。検出した物体のバウンディングボックス情報を得るための手続きについても検討した。このレポートでは、行った手順およびその結果と、考察について簡単にまとめる。

(註) 本問題の出題ミスについてご連絡した件 (4/17) : 課題内容には21クラスとありました。しかし一方で、その追加されたと書かれた21クラスはデータのことだと思われます。しかし一方で、その追加されたと書かれた21クラスについても、dogやcatなどはデータセットに含まれていませんでした。そのため、この課題ではテストデータセットに含まれる80クラスのうち、課題で提示されていた19クラスを分類クラス数と捉えてモデル作成していきます。宝田さんより確認および了承をいただいております。

2 データセットについて

今回扱ったAnimals Detection Images Datasetは、80クラスの生物の画像データと、各画像に対して対象の生物クラスおよび写る領域の座標がラベルとして含まれている。今回は先述の通り、以下の19クラスについて分類モデルを作成し、動作を確認した。

扱ったクラス：Zebra・Lion・Leopard・Cheetah・Tiger・Bear・Brown Bear・Butterfly・Canary・Crow・Bear・Bull・Camel・Crab・Chicken・Centipede・Cattle・Caterpillar・Duck

dataset以下のディレクトリ構成は以下のようになっている。

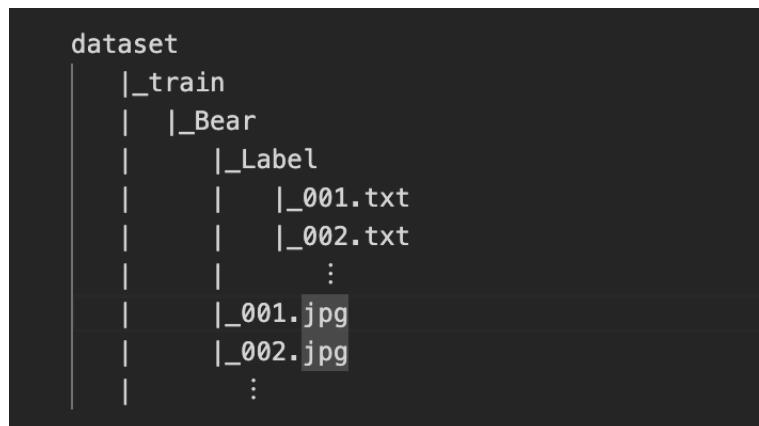


Figure 2.1: データセットのディレクトリ構成

3 問1：画像の多値分類

3.1 学習データの下準備

このデータセットに含まれる画像サイズや形式は一様ではない。画像は大小様々で、画像は多くはカラー画像だが白黒のものがある。後述するが、今回はResnet50を用いた転移学習によって分類モデルを作成したため、モデルのデフォルト入力値である以下の形式に学習データおよび正解ラベルを整形する。

	学習データ数	テストデータ数	データサイズ
画像データ X	2781	691	(3, 224, 224)
正解ラベル y	2781	691	(1)

Table 3.1: 用いるデータサイズ

残念ながら、白黒画像を入力に用いることができないため、今回は画像データからそれらは除くことにした。以下のコードは各々のディレクトリから画像パスを取得しそれに対応するラベルデータを作成する`make_pathlist`関数である。これによって作成した訓練データセットと、テストデータセットについて、画像データサイズが[3,x,y]の形でないもの（白黒画像）を画像データとラベルから消去する処理を行う（その処理については割愛）。なお、今回の学習データは全体の半分を用い、そのうち白黒画像として除かれたものは合計33枚（train:27,test:6）であった。

3 問1：画像の多値分類

Listing 3.1: make-pathlist

```
1 def make_pathlist(train_ratio, test_ratio):
2
3     X_train_path = []
4     X_test_path = []
5
6     y_train = []
7     y_test = []
8
9     #train data
10    classes = os.listdir('dataset/train')
11    for label in classes:
12        num = 0
13        files_path = os.listdir('dataset/train/{}'.format(label))
14        files_path = [s for s in files_path if s != 'Label']
15
16        for path in files_path:
17            X_train_path.append('dataset/train/{}/{}'.format(label, path))
18            y_train.append(label)
19            num += 1
20            if num > len(files_path)*train_ratio:
21                break
22
23    #test data
24    classes = os.listdir('dataset/test')
25    for label in classes:
26        num = 0
27        files_path = os.listdir('dataset/test/{}'.format(label))
28        files_path = [s for s in files_path if s != 'Label']
29
30        for path in files_path:
31            X_test_path.append('dataset/test/{}/{}'.format(label, path))
32            y_test.append(label)
33            num += 1
34            if num > len(files_path)*test_ratio:
35                break
36
37
38    return X_train_path, y_train, X_test_path, y_test
```

また、画像はtorchvision.transformsを用いて、リサイズおよびデータ拡張を行う。データ拡張は今回は左右反転のみを用いた。画像分類のためにしっかり学習するなら回転拡大やmixupなどを用いるのが良いが、今回はのちに物体検知タスクまで取り扱うことを見据えて、矩形領域の逆変換が簡単な反転のみにした。リサイズについては、クリッピングではなく、伸縮によって224x224に統一している。

データセットおよびデータローダはtorch.utils.dataのDatasetを継承/Dataloaderを使用して作成し、バッチサイズ48とした（学習時間の関係であまり小さいと時間が足りなかつたため）。すなわち学習データセットのバッチはtorch型[48, 3, 224, 224]である。

3.2 モデルの転移学習

今回画像分類に用いたモデルはpytorchがImagenetによる事前学習済みモデルを提供しているResnet50およびVGG16である。Resnet50は、ILSVRC2015の優勝モデルであり、残差ネットワークが用いられる深層CNN学習モデルである。VGG16はILSVRC2014の2位モデルであり、精度はResnet50に比べると劣るもの、こちらも同様のCNNベースモデルである。全体のパラメータ数はResnet50が25M、VGG16が135M程度なので、アーキテクチャとしてはResnetの方が軽量である。いずれも最終層はフルコネクションでImagenetの1000クラスの確率値を出力しているが、今回は19クラスに限定するよう最終層のノード数を書き換えている。Imagenetによる事前学習で入力層側の特徴抽出の機能は十分備わっていると考えられるため、今回は学習時間の兼ね合いもあって、最終出力層となるフルコネクション部分のみをファインチューニングすることにする。今回は学習データ数もあまり多くはないため、重み全体による学習よりも、一部のパラメータのみの学習の方がモデルの表現力を大きく変更することなくタスクにあったチューニングできると思われる。なお、最終層の入力特徴量は2048であるため、出力特徴量19と少なくとも、調整パラメータはバイアスパラメータを含め38913となる。

今回はモデルの重さとマシンスペックの兼ね合いで、5Epochというやや不十分気味な学習になってしまったが、そのそれぞれのEpochにおける経過が以下である。

3 問1：画像の多値分類

model		Resnet50		
Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
Epoch1	1.9636	0.7184	1.1775	0.7627
Epoch2	0.4245	0.8972	1.3188	0.8191
Epoch3	0.3092	0.9252	1.0443	0.8321
Epoch4	0.2922	0.9220	1.1413	0.8350
Epoch5	0.2739	0.9266	0.8242	0.8567

Table 3.2: Resnet50の学習過程

model		VGG16		
Epoch	Train Loss	Train Accuracy	Test Loss	Test Accuracy
Epoch1	2.2022	0.7005	1.6420	0.7496
Epoch2	1.0761	0.8231	1.9151	0.7681
Epoch3	1.1315	0.8263	2.9531	0.7641
Epoch4	1.0503	0.8493	2.2668	0.7699
Epoch5	0.9542	0.8587	2.5595	0.7916

Table 3.3: VGG16の学習過程

やはりモデルとしての表現力の違いもあってか、全くの同じ条件であっても、Resnetの方が精度がよく、収束も早いことが伺える。しかし、どちらのモデルもやや過学習気味であり、汎化性能の向上には工夫が必要である。今回はデータ拡張もシンプルな反転しか行っていないため、実際はこれらによってもう少し過学習を抑えることが可能であるだろう。また、VGG16を見ると、学習が進まず上がったり下がったりしてしまっている。同じラベルでもかなり画像ごとに異なる雰囲気のものも多いため、分類クラスの特徴量をうまく捉えられていないと、バッチごとに大きなパラメータの修正が起きてしまうためであろう。Resnet50では損失が遞減している様子が見られることか

ら、VGG16に比べ安定して転移学習が進んでいることが見受けられる。

3.3 出力結果の比較

いくつかのテストデータからの抜粋で、同じ画像をそれぞれがどのように解釈しているのか調べよう。以下は画像とクラスごとの推論確率値である。ただし、pytorchはlogitで予測値を出力するため、表の値は出力された値そのものではなくsoftmaxによって確率値に変換したものである。

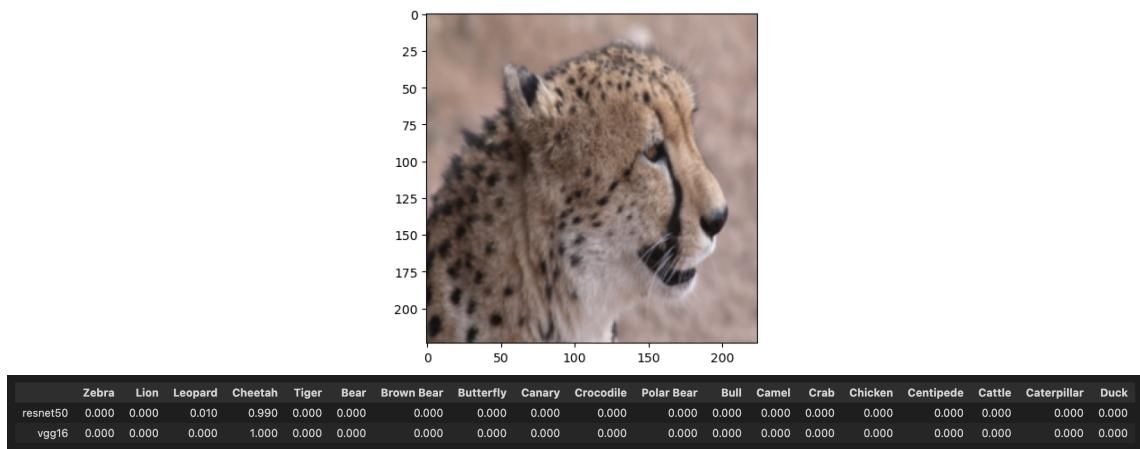


Figure 3.1: テストデータに含まれるチーターの例

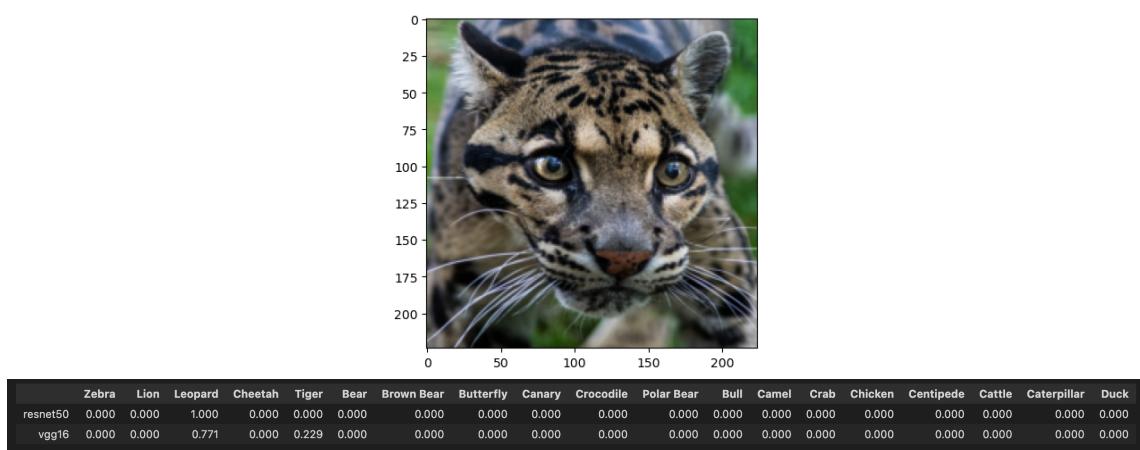


Figure 3.2: テストデータに含まれるレオパードの例

3 問1：画像の多値分類

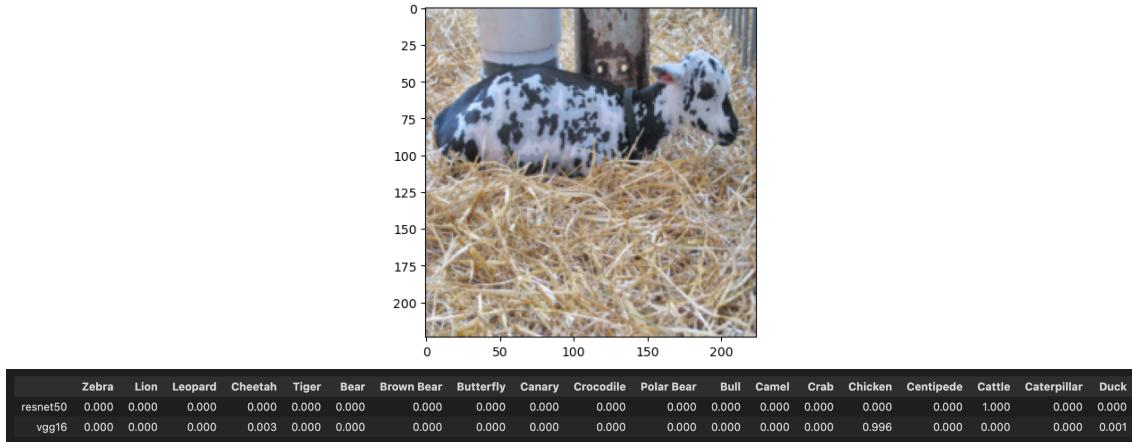


Figure 3.3: テストデータに含まれる牛の例

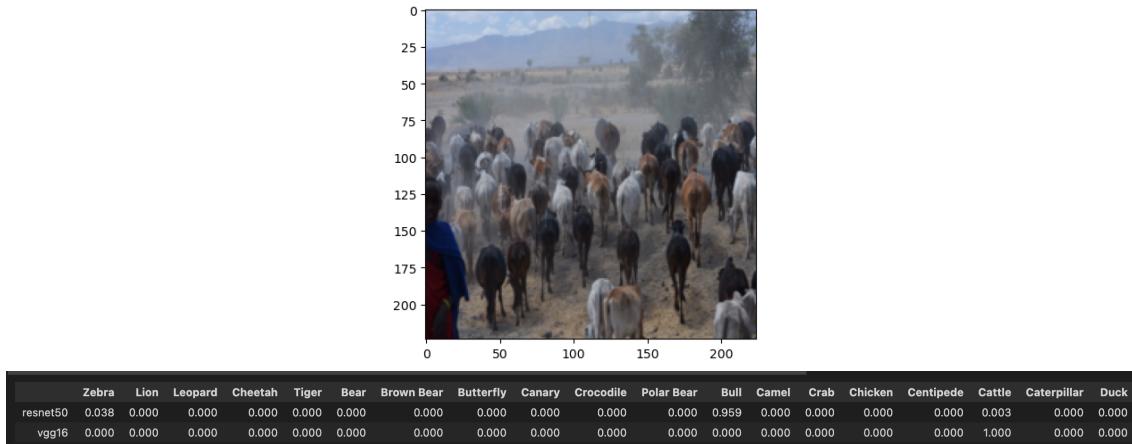


Figure 3.4: テストデータに含まれる牛の例2

一枚目のライオンの例を見ると、これはどちらのモデルも正しくライオンを出力しており、何か他の選択肢と迷っている様子は見受けられない。しかし二枚目のレオパードの画像では、VGGがタイガーと迷っている様子が見受けられる。今回の分類クラスの中には猫科・熊科・牛科の似ている動物が多く含まれるので、分類確率を見ると、その動物の特徴表現を何と近しいものとして捉えているかを考えることができる。実際にVGGはタイガーとレオパードを紛らわしいものとして捉えているようだが、Resnetにとってはこの二枚目の画像はまごうことなきレオパードだと考えているようである。さて、三枚目と四枚目は実はどちらもCattleが正解ラベルに当たる。面白いことにその選択肢を、三枚目ではVGGが0%として分類しており、四枚目ではResnetがほぼ0%として分類している。どちらももう一方のモデルは100%の確信度を持って分類し

ていることを考えると、分類で注目している特徴量がこのクラスでは根本的に違う可能性が示唆されて興味深い。もし時間ががあればGrad-Camなどでどの部分に着目しているか両者可視化を行いたかったが、今回は時間が足りないため割愛する。どちらのモデルももう少し学習を長く回せばよりはっきりとその違いが現れそうであるから、ぜひ比較検討してみたい。

4 問2：画像の物体検知

4.1 YOLOによる推論のための下準備

問2では、同じデータセットを用いて物体検知を行なった。物体検知の詳細は後ほど述べるが、YOLOを用いたためそれに見合った形に学習データやラベルのディレクトリ構成の変更が必要であった。YOLOの学習やテストでは、以下のようなディレクトリ構成が求められるが、元のデータセットのディレクトリ構成は2章に示した通りである。

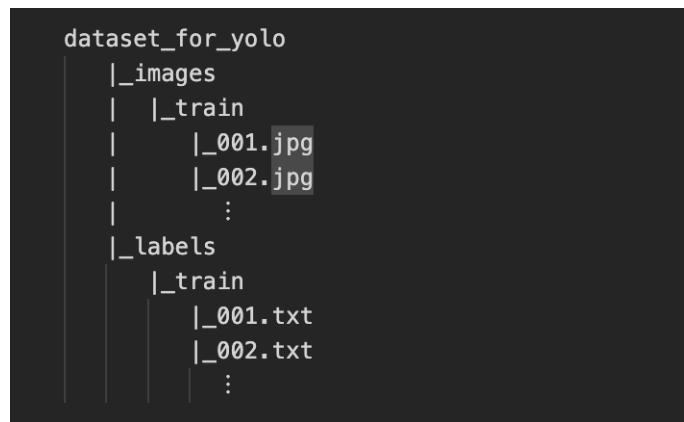


Figure 4.1: YOLOのためのデータセットのディレクトリ構成

Labelに含まれるクラスラベルおよびバウンディングボックスの正解情報は「クラス名 左上X 左上Y 右下X 右下Y」のように記述されているが、YOLOでは座標ではなく画像サイズに対して正規化された値で記述されていなくてはならない。そのため、ラベルファイルを読み出し空白ごとに値を取り出して正しいフォーマットに変換する関数convert_annotationの実装を示す。ただし、データセットにおける「Brown bear」「Polar bear」はラベル名に空白が入っており、convert_annotationの適用前にスペースの穴埋めをする必要があるため、それを行う関数fill_space_in_labelsとそれをtrainデータに適用するコードを先に示す。

Listing 4.1: fill-space-in-labels

```

1label_list = ['Zebra', 'Lion', 'Leopard', 'Cheetah', 'Tiger', 'Bear', 'Brown bear', 'Butterfly', 'Canary', 'Crocodile', 'Polar bear', 'Bull', 'Camel', 'Crab', 'Chicken', 'Centipede', 'Cattle', 'Caterpillar', 'Duck']
2
3def fill_space_in_labels(input_labels_dir, output_labels_dir,
4    class_names_with_space):
5    os.makedirs(output_labels_dir, exist_ok=True)
6
7    for label_file_name in os.listdir(input_labels_dir):
8        if not label_file_name.endswith(".txt"):
9            continue
10
11        input_label_file_path = os.path.join(input_labels_dir,
12            label_file_name)
13        output_label_file_path = os.path.join(output_labels_dir,
14            label_file_name)
15
16        print(input_label_file_path)
17        with open(input_label_file_path, "r") as input_file, open(
18            output_label_file_path, "w") as output_file:
19            print(input_file)
20            for line in input_file:
21                print(line)
22                for class_name in class_names_with_space:
23                    filled_class_name = class_name.replace(" ", "")
24                    line = line.replace(class_name, filled_class_name)
25                    output_file.write(line)
26
27# apply to traindata
28class_names_with_space = ["Polar bear", "Brown bear"]
29input_base_dir = "dataset/train/"
30output_base_dir_fixed = "dataset/train/"
31
32for class_name in class_names_with_space:
33    class_name_fixed = class_name.replace(" ", "")
34
35    input_labels_dir = os.path.join(input_base_dir, class_name, "Label")
36    if class_name else os.path.join(input_base_dir, class_name, "Label")
37
38    print(input_labels_dir)
39    output_labels_dir_fixed = os.path.join(output_base_dir_fixed,
40        class_name_fixed, "Label")
41
42    fill_space_in_labels(input_labels_dir, output_labels_dir_fixed,
43        class_names_with_space)

```

```
1def convert_annotation(label_file, img_file, output_file, label_list):
2    with open(label_file, 'r') as f:
3        lines = f.readlines()
4        print(lines)
5
6    img = Image.open(img_file)
7    img_width, img_height = img.size
8
9    converted_lines = []
10
11    for line in lines:
12
13        parts = line.strip().split()
14        if len(parts) != 5: print(line, parts)
15        label, x_min, y_min, x_max, y_max = parts
16        class_id = label_list.index(label)
17
18        #convert into correct format
19
20        x_center = (float(x_min) + float(x_max)) / 2
21        y_center = (float(y_min) + float(y_max)) / 2
22        width = float(x_max) - float(x_min)
23        height = float(y_max) - float(y_min)
24
25        x_center_normalized = x_center / img_width
26        y_center_normalized = y_center / img_height
27        width_normalized = width / img_width
28        height_normalized = height / img_height
29
30        converted_line = f"{class_id} {x_center_normalized} {y_center_normalized} {width_normalized} {height_normalized}"
31        converted_lines.append(converted_line)
32
33    with open(output_file, 'w') as f:
34        f.write("\n".join(converted_lines))
35
36
37 input_base_dir = "dataset/train/"
38 output_base_dir = "dataset_yolo/labels/train/"
39
40 for class_name in label_list:
41     input_labels_dir = os.path.join(input_base_dir, class_name, "Label")
42     input_images_dir = os.path.join(input_base_dir, class_name)
43     output_labels_dir = os.path.join(output_base_dir)
44
45     os.makedirs(output_labels_dir, exist_ok=True)
46
47     for label_file_name in os.listdir(input_labels_dir):
48         if not label_file_name.endswith(".txt"):
49             continue
50
51         label_file_path = os.path.join(input_labels_dir, label_file_name)
52         img_file_name = f"[label_file_name.replace('.txt', '.jpg')]"
53         img_file_path = os.path.join(input_images_dir, img_file_name)
```

```

54     output_file_name = f"{label_file_name}"
55     output_file_path = os.path.join(output_labels_dir,
56                                     output_file_name)
57
58     print(label_file_path, img_file_path, output_file_path)
      convert_annotation(label_file_path, img_file_path, output_file_path
      , label_list)

```

(正直、この辺りの実装がなんだかんだ一番骨が折れた...)

4.2 YOLOの学習

データセットの準備が終ったところで、実際に学習に移る。今回はモデルとしては一番軽量なYOLOv5Sの学習済みモデルを利用し転移学習を行うことにした。すべての層を学習させるのではなく、バックボーンとなる入力側10層はパラメータの更新を行わずに学習させた。畳み込み層は入力層側にあるため、すでに学習されている特徴抽出の特性をある程度保存しながら学習ができる。これも計算時間の都合であるが、今回はバッチサイズ64、エポック10とした。学習による各種メトリクスおよび評価指標は以下のようになった。

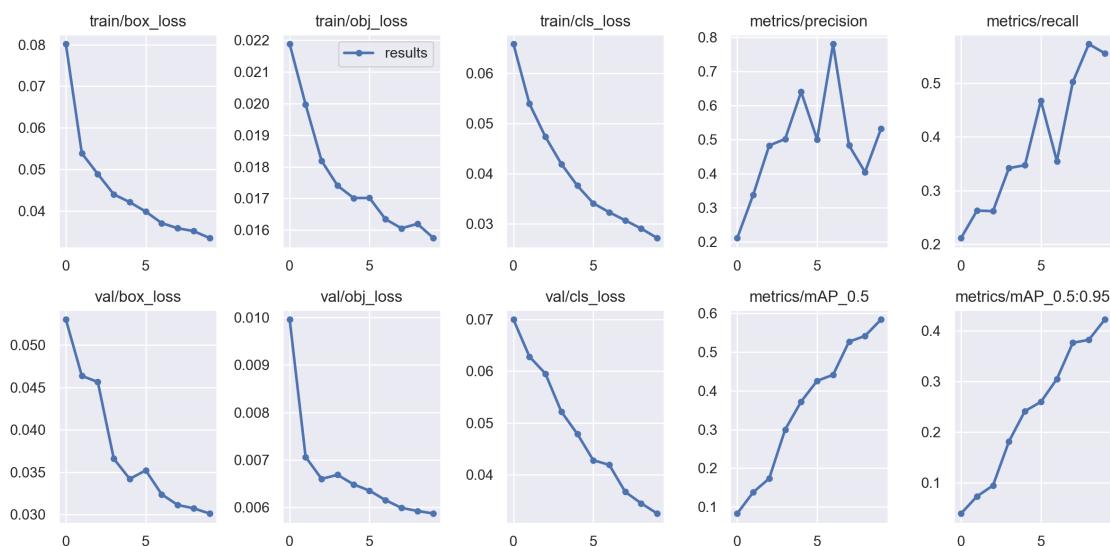


Figure 4.2: 学習における各種評価指標の変化

4 問2：画像の物体検知

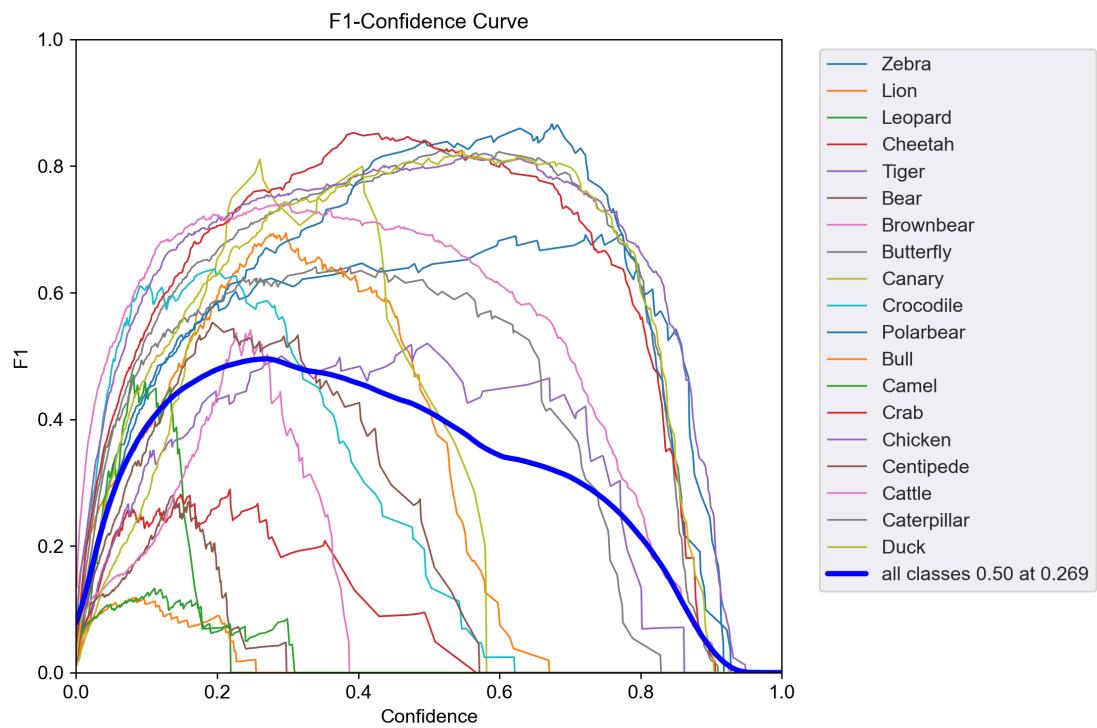


Figure 4.3: 各分類クラスにおけるF1曲線

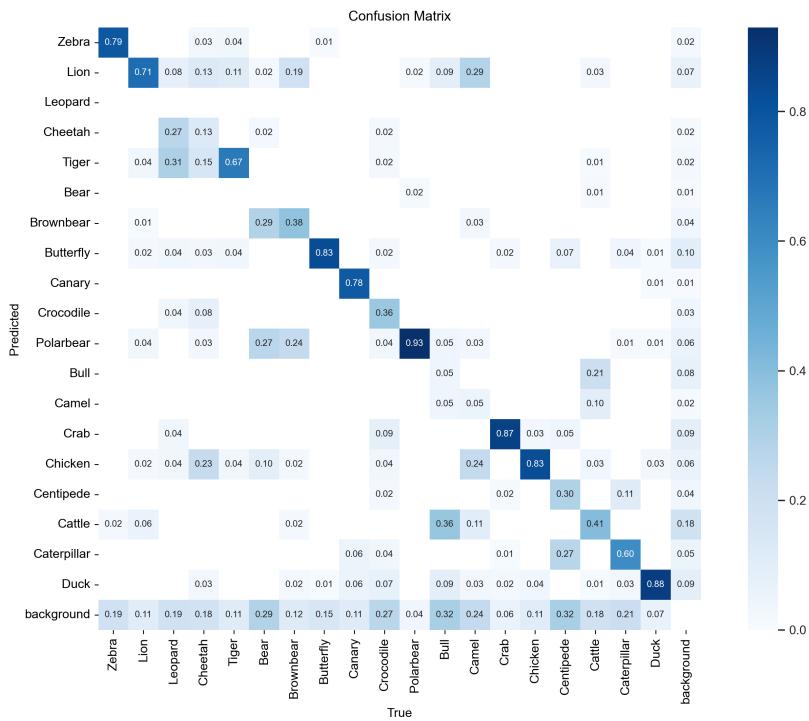


Figure 4.4: 各分類クラスにおける混同行列

まず、いずれもエポックを経るごとに損失は小さくなり、クラス分類における損失も小さくなりつつあるため、今回は10epochで止めてしまっているものの、もう少し学習は回せば精度は向上しそうである。とはいえ、学習データですらmAPは0.5程度とあまり領域の推定がうまくいっていない様子である。学習データで体全体が写っていない場合を強く学習してしまっている場合や、学習データと全く異なるアングルや態勢のものは、バウンディングボックスの位置が正しく推定できないのだと思われる。混同行列を見ると、BrownBearとBearの分類が難しかったり、BullやCamelがほぼ分類に失敗していたりと、似ているから間違いややすいものと、未学習の状態に近いと思われるものに分かれていることがわかる。学習データ数に偏りがあるため、その影響もあると思うが、未学習防止や汎化性能の向上のためにも、やはりデータ拡張など、打てる手は打つべきであった。さて、F1と確信度を軸にとったグラフを見ると、革新を持つて答えるとF1が下がるという様子が見受けられる。これは、確信度が高い場合におけるPrecisionかRecallがどちらかが極端に低いことに由来すると考察できる。仮にどちらも0.5ずつあれば、F1の値は0.25となるはずだから、確信度が8割を超えていたりの分類では、自信満々に間違えているか、見逃しがあるということである。実際に、ど真ん中に明らかにクマがいても、端っこによくわからない草を芋虫と間違えるといったミスが目立つし、これはライオンだと言いながらタイガーを選ぶような場合も多い。モデル全体として、よく学習が進んでいないことがこのプロットを見るとよくわかる。そもそも、確信度を0.7以上で出力していないクラスも多く、全体的に自信がなさげである（つまり学習データから学び取ったことが少ないorテストデータとの乖離が大きく生かすことが難しいということである）。

4.3 テストデータの推論結果

いくつかテストデータがどのように物体検知されたか確認しよう。

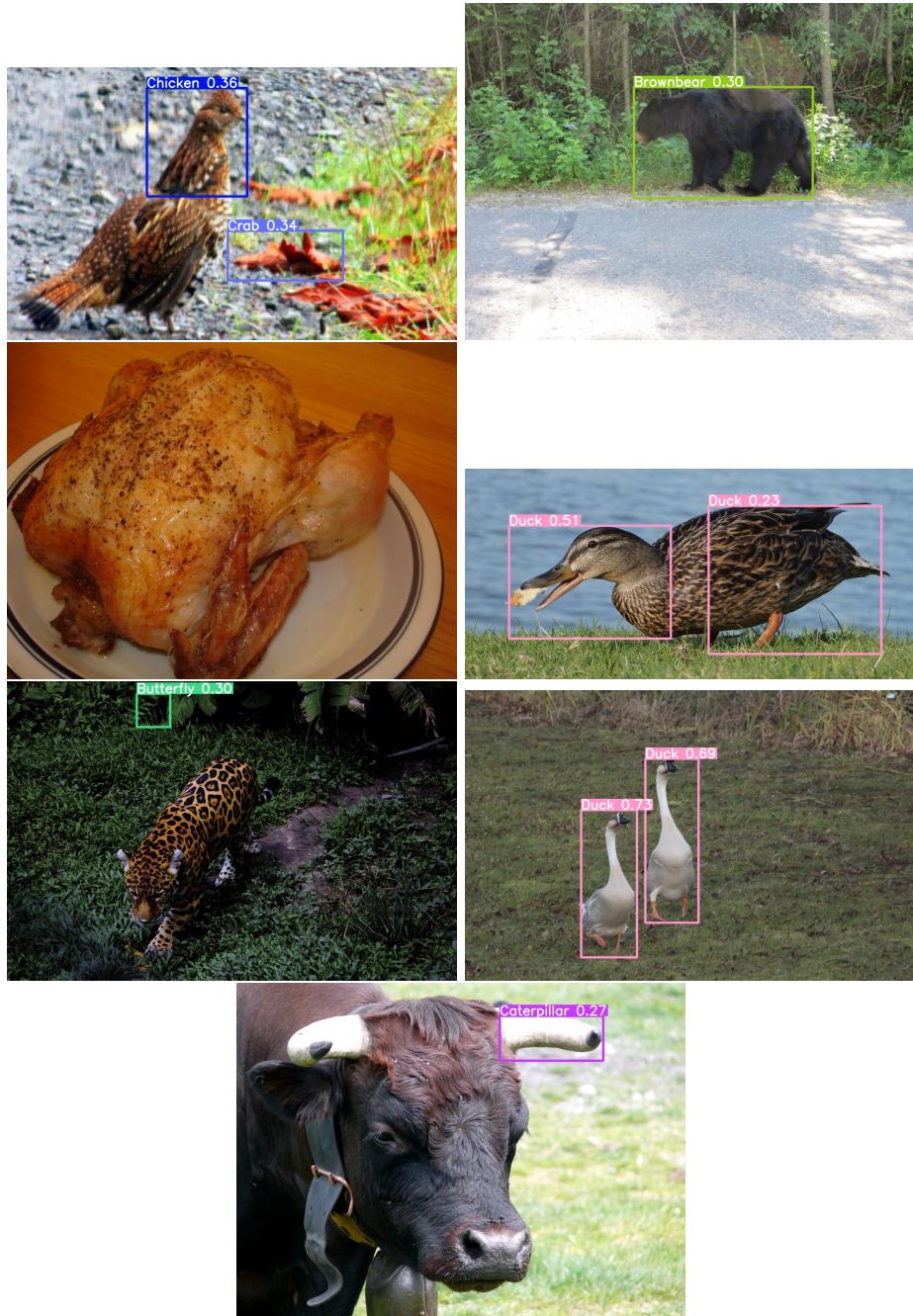


Figure 4.5: 物体検知結果の例

うまく推定できているものもあれば、全然うまくいかないものもある。duckの例を見ると、頭とお尻で二匹分カウントされている。これはおそらく学習データに頭だけのものやお尻だけのものが含まれており、それぞれが別のduckなる存在だと学習してしまっているのであろう。がっつり写っているのに、華麗に検知をスルーしている画像もとても多く、学習データがある程度大きなもので、かつ十分に学習をしないと、角度の変化や写っている箇所が部分的な場合には推論が厳しそうである。牛の角を芋虫だと勘違いしているのは、なんとも惜しいような気さえする。まず、データセットに含まれるものが当てさせる氣があまりないものもある。データに、chickenとして調理済みのものが入っているのはどうなのか!?(たくさん同じようなものがあれば良いが、そういうわけでもなさそうだった)

4.4 バウンディングボックス情報の取得

YOLOでは前節のような画像が出力として得られるため、座標情報やラベル情報を取得することができない。そのため、推論を行なっているdetect.py中にある、これらの情報が格納されている変数predからjsonファイルを書き出すことで、推論結果の情報にアクセスできるようにする。detect.py中に挿入するコードを以下に示す。

Listing 4.2: save-BoundingBoxInfo-json

```
1# json output directory
2output_detection_dir = 'output_dir'
3
4
5json_data = []
6
7# pred has boundingbox's data
8for pr in pred[0]:
9    x1, y1, x2, y2, confidence, class_id = pr.tolist()
10   json_data.append({
11       'filename': p.name, #p_name is image'name
12       'coordinates': [x1, y1, x2, y2],
13       'confidence': confidence,
14       'class_id': int(class_id)
15   })
16
17
18json_filename = os.path.join(output_detection_dir, os.path.basename(p.name
19                           ).replace('.jpg', '.json'))
20# save jsonfile
21with open(json_filename, 'w') as f:
22    json.dump(json_data, f, indent=4)
```

これによって、情報を取り出すことが可能になった。実際にテストデータのインデックスを与えると、出力画像とjsonに書き出された情報がoutputされる関数get_resultおよび、その呼び出し結果を次に示す。

Listing 4.3: get-result

```
1 def get_result(idx, detected_folder="dataset_yolo/images/test/", output='
2     yolov5/runs/detect/exp3'):
3
4     files_path = os.listdir(detected_folder)
5
6     filename = files_path[idx]
7     result_image = '{}/{}'.format(output, filename)
8
9
10    img = Image.open(result_image)
11    img = np.asarray(img)
12    plt.imshow(img)
13    plt.show()
14
15    json_path = 'yolov5/output_dir/{}'.format(filename.replace('.jpg', '.'
16                                              json'))
17    result_json = json.load(open(json_path, 'r'))
18
19    for i in range(len(result_json)):
20        print('Class:', result_json[i]['class_id'],
21              'Bounding Box: (x1,y1,x2,y2)=', result_json[i]['coordinates
22              '],
23              '(confidence:', result_json[i]['confidence'], ')')
24
25
26    return
```

4 問2：画像の物体検知

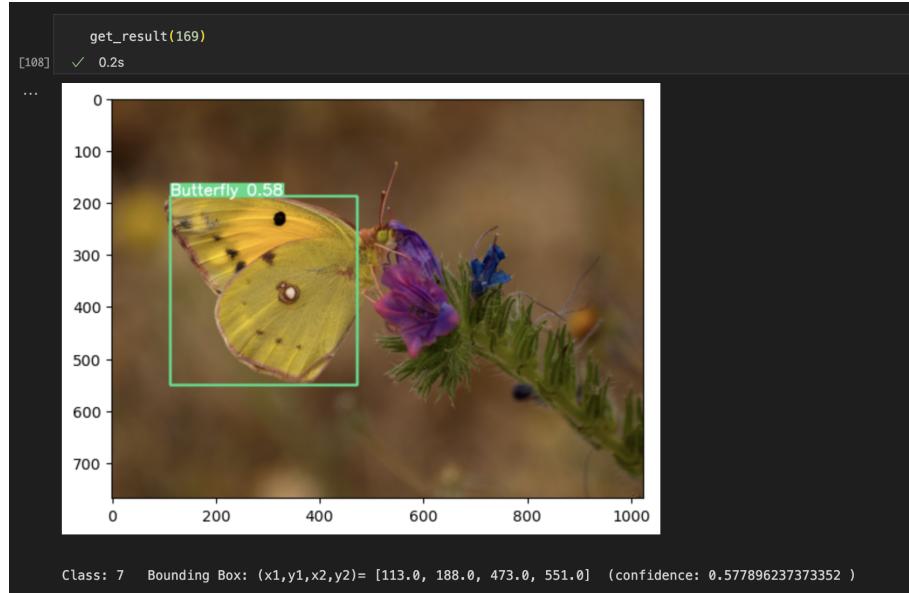


Figure 4.6: 物体検知結果の例

確かに、座標などの情報を取り出すことができた。

5 終わりに

CA Tech Loungeの入会を目指し、今回与えられた課題に全力で取り組みました。課題1～3に挑戦し、課題1と3だけがなんとか形になりましたが、課題3ではもう少し独創性を取り入れた実装や工夫ができるとよかったです。ありきたりな実装になってしまったと反省しています。また、せっかく関連のある問1と問2をドッキングさせることでやり切ることができませんでした。期限が迫っているため、現状で提出しますが、個人的には引き続き改良を試みたいと思っています。今回の課題提出期間中、資格試験や大学の新学期が重なり、取り組める時間は限られていきましたが、毎日できるだけ入会課題に集中しました。CA Tech Loungeの入会募集を見てからおよそ1ヶ月経ちますが、参加したいという強い気持ちは変わっていません。もし他の方がもっと多くの課題に取り組んでいたら、質の高い成果を出していたら、選考通過できるかとても不安ですが、自分の時間と技術的リソースを活用して全力を尽くしたことは確かです。どうか好意的な評価をいただけることを願っています。実務に役立つ、楽しい課題でした。今回取り組めなかった課題4や5、経験が足りず完成できなかった課題2もチャレンジしたいと考えています。

6 参考

コーディングで参考にしたページやリファレンスなど。部分的に使用したコードはありますですが、丸ごとコピペしたりはしていません。

- MODELS AND PRE-TRAINED WEIGHTS
 - <https://pytorch.org/vision/stable/models.html>
- YOLOv5 in PyTorch
 - <https://github.com/ultralytics/yolov5>
- PyTorchによるファインチューニングの実装
 - <https://venoda.hatenablog.com/entry/2020/10/18/014516>
- Google ColabでYOLOv5を使って物体検出してみた
 - <https://qiita.com/shoku-pan/items/31bf3c975b73db153121>
- YOLOv5の学習時に指定可能なオプションについての解説
 - https://qiita.com/shinya_sun_sun/items/8c368f3024bf5b0d14aa