令和２年度　重点研究課題報告書

人工知能技術活用のための人材・データプラットフォーム開発に関する研究

報告書

令和３年３月

土木学会　構造工学委員会

人工知能技術活用のための人材・データプラットフォーム開発に関する研究

概要 4
1. 研究の背景と目的 5
2. 教育システム・研究成果共有のためのシンポジウム／論文集 9
2.1. 教材 10
2.1.1. AI と機械学習について 11
2.1.2. 橋梁形式分類 AI を実装してみよう　その１　犬猫分類問題で AI を理解する.16
2.1.3. 犬猫分類問題を応用して橋梁形式を分類してみよう 22
2.1.4. 事例で AI のできることを知ろう　その１　画像物体検知とアノテーション...29
2.1.5. 事例で AI のできることを知ろう　その２　時系列データに対する機械学習例 31
2.1.6. 事例で AI のできることを知ろう　その３　自然言語処理学習例....34
2.1.7. AI を利用したメンテナンスユースケースを考えよう 37
2.1.8. 対話システムと文章生成....43
2.1.9. Google Colab の使い方 46
2.1.10. ロジスティック回帰によるデータ分類....51
2.1.11. 決定木とランダムフォレストによるデータ分類 57
2.1.12. ニューラルネットワークによるデータ分類....63
2.1.13. Keras で MNIST の判定 68
2.1.14. Keras で CNN....76
2.1.15. ResNet を Fine Tuning して自分が用意した画像を学習させる 83
2.1.16. Keras で回帰....96
2.1.17. 画像キャプションの自動生成....99
2.1.18 ひび割れ検出チュートリアル....119
2.2 AI 習得セミナー....131
2.3 研究成果共有のためのシンポジウム／論文集の立ち上げ 132
2.3.1. AI・データサイエンスシンポジウム 132
2.3.2. AI・データサイエンス論文集の立ち上げ 134
3. データプラットフォームについての検討....135
3.1. 複数分野の研究者へのインタビュー調査....136
3.2. 目指すべきデータプラットフォームの構造と開発すべき機能....139
3.2.1. AI にマッチしたデータプラットフォームの構造 139

3.2.2. ETL 処理の例：画像データからのメタデータ抽出.....141
3.3. J-STAGE Data と AI・データサイエンス論文集との連携.....152
4. まとめ.....154
付録.....155
付.1. 西村文武准教授へのインタビュー.....155
付.2. 岡村未対教授へのインタビュー.....168
付.3. 竹山智英准教授へのインタビュー.....178
付.4. 一言正之様へのインタビュー.....183
付.5. 森脇亮教授へのインタビュー.....189
付.6. 吉井稔雄教授へのインタビュー.....194
付.7. 秋山充良教授へのインタビュー.....205
付.8. 北原武嗣教授へのインタビュー.....217

概要

近年，人工知能技術が急速に発展してきており，様々な分野において活用がなされはじめている．しかし土木工学分野においてはなかなか人工知能技術が浸透していない．そこで本研究では，土木工学における人工知能技術の活用のために必要な基礎的知識や技術を習得できる教材の開発と，人工知能技術の活用を見据えたデータプラットフォームのあるべき姿の提案・プロトタイプ構築を行い，今後の本分野における進むべき方向性を示すことを目的とした．特に，①教育システムが未整備でありプレイヤーが増えていない，②人工知能の学習の根幹であるデータの活用の仕組みが整っていない，という 2 点の課題の解決を目指した．

教材については構造工学委員会の「構造工学での AI 活用に関する研究小委員会」と積極的に連携し，開発を行った．特に人工知能の各技術について，土木工学での実例と関連付けながら解説やソースコード紹介を行っており，またこれらは小委員会の HP などで公開している．また，関連してセミナー教材も開発しており，それらについても本報告書で述べた．データプラットフォームについては，データの構造化が強く求められること，そして土木工学分野ではレガシーデータの構造化が遅れているので ETL 処理（Extract; Transform; Load の頭文字）の積極的な実装が求められることを述べ，またプロトタイプとして画像データの構造化手法を示した．これらは学術研究として進めていく必要があり，そのための発表先として AI・データサイエンスシンポジウム，AI・データサイエンス論文集を立ち上げた．また，J-STAGE Data とも連携することで，積極的なデータ共有をすすめる枠組みについても整備した．

今後，人工知能技術を含む情報工学分野の知見を土木工学のニーズに合致するように取り入れ，また独自技術を開発していくことで，土木工学の新たな展開が誘起されると期待できる．例えば i-Construction などもその文脈にあるが，何を学び，何を研究する必要があるかを明確にするためには体系化，成書発刊などが重要であると研究代表者は考え，最後にそのような提言を行った．

1. 研究の背景と目的

近年，データ取得・記録コストの劇的な低下や計算能力の大幅な向上に伴い，ディープラーニングに代表される人工知能（以下，ＡＩ）技術が急速に発展してきており，様々な分野において活用がなされはじめ，またその性能・威力について知られてきている．特に 2015 年に，ディープラーニングを活用した囲碁ソフトである AlphaGo がトッププロに勝利した際は，世界に大きなインパクトを与えた．また，その後も様々な新技術の開発がなされており，たとえば OpenAI により作成された GPT-3 (Generative Pre-Training3)は，人間と見分けが付かないほど高精度な文章を生成する AI として知られている．また，Google translate (https://translate.google.co.jp)や DeepL (https://www.deepl.com/ja/translator)といった翻訳 AI も，精度が日々向上しており，活用が進んでいる．

囲碁でトッププロに勝つ，人間にも見分けのつかない文章を生成する，文意を理解しながら適切な翻訳を行う，というのは，タスクによっては人間の知能・知性の代替となり得る可能性を示しているものと言える．その能力をもって，労働者の減少が課題となっている土木工学分野においても，人間の業務の代替，あるいは人間にはできない高度な判断をできるようにして，人材不足の解消や生産性向上を目指したいと考えるのは当然の発想である．

土木工学分野でも例外ではなく，例えばコンクリートのひび割れ検出[1)-9)など]，変状検出[10)-12)]，劣化要因把握[13)]，アスファルト舗装の損傷検出[14),15)]，鋼材の腐食度評価[16)]，橋梁部材の損傷評価[17),18)]，振動計測[19)-21)]，河川水位予測[22),23)]，衛星画像や航空写真からの地すべり評価[24),25)]，風速予測[26)]，降水量予測[27)]，建設現場分析[28)]，など筆者らの研究を含む様々な研究がなされている．この中でも，コンクリートやアスファルトのひび割れ検出などは商用化もなされる程度には成熟してきた．

ただ，これらの研究はそれぞれ有用で，意義もあると考えているものの，AI の活用対象がそれだけに留まるのは期待値を下回っていると言わざるを得ない．囲碁や将棋での実績，翻訳の性能などと比較すると，もちろん異なる問題のため一概に比較することはできないものの，もう少し性能が高く，そして実務での活用が可能な程度にロバスト性が高まってもよさそうなものである．

その原因としてまず，人工知能技術に接している土木技術者が非常に少ないことが挙げられる．現状，なんらかの土木工学における問題を人工知能技術で解決しようとする場合，IT ベンダーに発注してプロダクトを作り上げるというケースが多い．しかし一般論として，

土木工学のドメイン知識を持たないIT技術者が土木工学の問題を解決するためのプロダクトを開発するのは極めて難しく，逆にAI技術に関する知見がない土木技術者が適切に要求仕様を定めることも無理難題であり，この両者が交わると大抵の場合は使えないものが出来上がる．そもそも，AI 技術に関する知見がないと，どういうものを作ればよいかというアイデアも出てこない．

また，現在主流となっている教師あり学習，特にディープラーニングは大量のデータを用いた学習により，帰納的にモデルを構築するものである．特に，データが考えられる全ての境界条件のもとで網羅的にあり，かつ入力の形式と出力の形式が固まっていれば，的確な内挿の実現により高い精度の結果が得られるところにディープラーニングの有効性がある．しかし，土木工学分野においては，活用可能なデータは，一般的にディープラーニングの入力として求められる量よりはるかに（おそらく数オーダーは）少なく，しかも偏ったものしか得られないことが多い．インフラ構造物を例にとると，例えば以下のような原因が考えられる．

- インフラ構造物は，建設される環境が異なるために一品生産物であり，工場製品と比べて個体差が大きいこと
- 管理者間や，業務担当者間でデータの共有がなされないこと

これらは，データの網羅性を高める上での障害となる．この状況では，ディープラーニングの「学習範囲外の状況に弱く，実世界状況への臨機応変な対応ができない」という欠点の影響を避けることが出来ない．そこで，ディープラーニングに代表されるAI技術を土木工学分野で活用するためには，データの共有，管理，活用の仕組みを整えることが必須となる．

ここまで述べた背景を踏まえ，上記の2課題を解決するため，本研究では以下の項目の実現を目的とした．

1. 土木工学における人工知能技術の活用のために必要な基礎的知識・技術を習得できるような，人材プラットフォームである教育システムを構築する．教育システムには，ソースコードやサンプルデータ，問題解決事例を含むものとする．
2. 人工知能技術の活用を見据えたデータ収集・記録の基点となる，データプラットフォームが満たすべき機能，要件について複数分野のユースケースの整理を通して提案する．また，プロトタイプを構築し機能することを確認する．

それぞれについて，本報告書で述べるものとする．

参考文献

1) 全邦釘，井後敦史：Random Forest によるコンクリート表面ひび割れの検出，土木学会論文集 F3，Vol.71，No.2，pp.I_1-I_8，2015.
2) 野間康隆，早川健太郎，黒台昌弘，西村毅：UAV マルチコプタ撮影画像を用いたコンクリート構造物のひび割れ画像処理，日本リモートセンシング学会誌, Vol.38，No.3，pp.234-240，2018.
3) 横山傑，松本高志：SNS を用いた Deep Learning によるコンクリートの変状自動検出器の開発，土木情報学シンポジウム講演集，Vol. 42, pp.235-236, 2017.
4) 全邦釘, 嶋本ゆり, 大窪和明, 三輪知寛, 大賀水田生：ディープラーニングおよび Random Forest によるコンクリートのひび割れ自動検出手法, 土木学会論文集 F3, Vol.73, No.2, pp.I_297-I_307, 2017.
5) Chun, P., Izumi, S., and Yamane, T.: Automatic detection method of cracks from concrete surface imagery using two-step light gradient boosting machine, *Computer-Aided Civil and Infrastructure Engineering*, pp.1-12, 2020.
6) Dung, C. V.: Autonomous concrete crack detection using deep fully convolutional neural network, *Automation in Construction*, Vol. 99, pp.52-58, 2019.
7) Zhang, J., Lu, C., Wang, J., Wang, L., & Yue, X. G.: Concrete Cracks Detection Based on FCN with Dilated Convolution. Applied Sciences, Vol.9, No.13, pp.2686, 2019.
8) 藤田悠介，中村秀明，浜本義彦：画像処理によるコンクリート構造物の高精度なひび割れ自動抽出，土木学会論文集 F，Vol.66，No.3，pp.459-470，2010.
9) 河村圭，宮本文穂，中村秀明，佐藤亮：対話型遺伝的アルゴリズムによるデジタル画像からのひび割れ抽出，土木学会論文集，No.742/VI-60，pp.115-131，2003.
10) 青島亘佐，河村伸哉，中野聡，中村秀明：深層学習による画像認識を用いたコンクリート構造物の変状検出に関する研究, 土木学会論文集 E2, Vol.74, No.4, pp.293-305, 2018.
11) 天方匡純，吉田武司，藤井純一郎：深層学習方式を活用した河川のコンクリート護岸の劣化領域抽出，第 73 回年次学術講演会，2018.
12) 藤井純一郎，嶋本ゆり，天方匡純，松川正彦，秋山和也，小原洋：fM と Semantic Segmentation による河川護岸の劣化診断支援手法，人工知能学会全国大会論文集，pp. 4K3J1301-4K3J1301，2019.
13) Okazaki, Y., Okazaki, S., Asamoto, S., and Chun, P.: Applicability of machine learning to a crack model in concrete bridges, *Computer-Aided Civil and Infrastructure Engineering*, pp.1-18, 2020.
14) Maeda, H., Kashiyama, T., Sekimoto, Y., Seto T., and Omata H.: Generative adversarial network for road damage detection, *Computer-Aided Civil and Infrastructure Engineering*, 2020.
15) 全邦釘， 井後敦史， 南免羅裕治， 黒木航汰， 大窪和明：車載カメラにより撮影され

た舗装画像からのディープラーニングによるひび割れ率評価，土木学会論文集 E1, Vol.73, No.3, pp.I_97-I_105, 2017.
16) Chun, P., Yamane, T., Izumi, S., and Kameda, T.: Evaluation of tensile performance of steel members by analysis of corroded steel surface using deep learning, *Metals*, Vol.9, No.12, pp.1259, 2019.
17) Chun, P., Tsukada, K., Kusumoto, M., and Okubo, K.: Investigation and repair plan for abraded steel bridge piers: case study from Japan, *Proceedings of the Institution of Civil Engineers-Forensic Engineering*, 172(1), pp.11-18, 2019.
18) 鈴木達也, 西尾真由子：橋梁定期点検における部材損傷度判定への深層学習の適用に関する検討，土木学会論文集 F3, Vol.75, No.1, pp.48-59, 2019.
19) Shrestha, A. and Dang J.: Deep Learning-Based Real-Time Auto Classification of Smartphone Measured Bridge Vibration Data, *Sensors*, Vol.20, No.9, pp.2710.1-21, 2020.
20) Chun, P., Yamane, T., Izumi, S., and Kuramoto, N.: Development of a Machine Learning-Based Damage Identification Method Using Multi-Point Simultaneous Acceleration Measurement Results. *Sensors*, 20(10), pp.2780, 2020.
21) Chun, P., Yamashita, H., Furukawa, S.: Bridge damage severity quantification using multipoint acceleration measurement and artificial neural networks. *Shock Vib*, Vol.2015, pp.1–11, 2015.
22) 一言正之，櫻庭雅明，清雄一：深層学習を用いた河川水位予測手法の開発，土木学会論文集 B1, Vol.72, No.4, pp.I_187-I_192, 2016.
23) 徳田大輔, 金炯俊：深層学習を用いた河川水位予測モデルにおけるネットワーク構造と入力データ量の依存性，土木学会論文集 B1，Vol.74，No.5，pp.I_169-I_174，2018.
24) 河村圭，中村優志，若月強，佐村俊和：深層学習を用いた空中写真からの土砂移動範囲自動検出に関する研究，土木学会論文集 F3，Vol.74，No.2，pp.I_132-I_143，2018.
25) 植田大介，間普真吾，呉本尭：災害前後 SAR 画像と DEM データを用いた CNN による土砂災害検出，人工知能学会全国大会，1D1-022，2018.
26) 森脇亮，今村実，全邦釘，藤森祥文：深層学習を用いた風速の短時間予測の試み，土木学会論文集 B1，Vol.74，No.4，pp.I_229-I_234，2018.
27) 藤森祥文，今村実，全邦釘，西村文武，森脇亮：AMeDAS 観測データを用いた深層学習による降水量予測の試み，水工学論文集，Vol.64，pp.I_1189-1194，2019.
28) Fang, Q., Li, H., Luo, X., Ding, L., Luo, H., Rose, T. M., and An, W.: Detecting non-hardhat-use by a deep learning method from far-field surveillance videos. Automation in Construction, Vol.85, pp.1-9, 2018.

2. 教育システム・研究成果共有のためのシンポジウム／論文集

1章で述べたように， AI技術を一定程度理解し，活用できる土木技術者を増やす必要があると考えている．世の中には人工知能技術の教育システムは多くあり，また良書も最近1～2年で多く出版されるようになったが，土木技術者や土木工学科の学生がはじめやすく，手を動かしやすいように，土木分野の事例を解析例とした教材の作成を行った．（また，継続的に「構造工学でのAI活用に関する研究小委員会」では行っている）．それらの成果は，土木学会誌２０２１年1月号の特集である「さあ、AIを始めよう―土木工学へのAI導入のススメ―」でその一部が小委員会メンバーによって発表されるとともに，小委員会 HP（https://committees.jsce.or.jp/struct10/node/13）にソースコード付きで公開されている．まず2.1節ではそれらから抜粋し示すこととする．次いで，本教材や追加教材を活用したセミナーを企画した．これについては2.2節で述べる．

さらに，萌芽的な分野であることを踏まえると，研究成果を共有することのできる仕組みが必要であると考えた．そこで，シンポジウムを立ち上げるとともに，査読付き論文集についても立ち上げ，それをJ-STAGEに登録した．これらについて2.3節で述べる．

2.1. 教材

本節では，作成した教材について示す．それぞれ，①概要を理解したい人向け，②実際にプログラムを組みたい初心者向け，③ある程度プログラムが組める中級者向け，というような分類をしている．教材一覧と分類を以下の表－１に示す．また，それぞれの内容について，「土木工学における AI 応用チュートリアル」および「P-Chun Labo」に記載した内容については以下に示す．（土木学会誌掲載の内容については土木学会誌を参考にされたい）

表－１　教材一覧と分類

媒体	教材名	①	②	③
土木工学における AI 応用チュートリアル	（1）AI と機械学習について	○	○	○
	（2）橋梁形式分類 AI を実装してみよう　その１　犬猫分類問題で AI を理解する		○	
	（3）橋梁形式分類 AI を実装してみよう　その２　犬猫分類問題を応用して橋梁形式を分類してみよう		○	○
	（4）事例で AI のできることを知ろう　その１　画像物体検知とアノテーション	○		
	（5）事例で AI のできることを知ろう　その２　時系列データに対する機械学習例	○	○	
	（6）事例で AI のできることを知ろう　その３　自然言語処理学習例	○		○
	（7）AI を利用したメンテナンスユースケースを考えよう	○		
	（8）対話システムと文章生成	○		○
	ひび割れ検出チュートリアル			○
土木学会誌 2021 年 1 月号	構造工学での AI 活用に関する研究小委員会の取り組み	○	○	○
	AI 入門講座（1）　―AI に必要となる学習データを集めよう―	○		
	AI 入門講座（2）　―AI プログラムを構築してみよう―		○	○
	AI 入門講座（3）　―AI の出力した結果を評価・活用しよう―		○	○
	土木技術者も AI に取り組もう	○	○	
	深層学習に基づく配管バルブの状態認識・異常検知システムの検討	○	○	○
	AI と土木工学の融合による水害対策技術の発展	○	○	○
	座談会　AI とアフターコロナ時代の国土利用	○	○	○
	付属プログラムコード：AI 入門講座		○	○
P-Chun Labo http;//pchun.work	Google Colaboratory の使い方		○	
	ロジスティック回帰によるデータ分類		○	
	決定木とランダムフォレストによるデータ分類		○	
	ニューラルネットワークによるデータ分類		○	
	Keras で MNIST の判定		○	
	Keras で CNN		○	○
	ResNet を Fine Tuning して自分が用意した画像を学習させる			○
	Keras で回帰			○
	画像キャプションの自動生成			○

2.1.1. AI と機械学習について

a) AI でできることを調べてみよう

人間の判断を代替したり支援したりする AI システムや，肉体労働の補助をするようなロボットにも AI 技術が活用されています. AI 技術は機械学習を利用したタスクを組み合わせて利用することが多いと考えられています．また近年の AI ブームにおいてはデータを活用する観点，いわゆるデータドリブンな AI 手法の開発が期待されています．ここでは，機械学習におけるタスクとデータの種類の観点から AI でできることを整理してみましょう.

○AI のタスク

AI はデータを学習することで下記のようなタスクを行うことができます．人間の仕事と比較してみましょう.

・クラス分類（classification）

あらかじめ用意されているカテゴリーに対象となるデータを分類します.

・クラスタリング（clustering）

元々分類されていないデータを，特徴を利用して分類するタスクです. 近いデータを同じグループとして分類します．教師なし学習などでも利用できます.

・次元圧縮（dimensionality reduction）

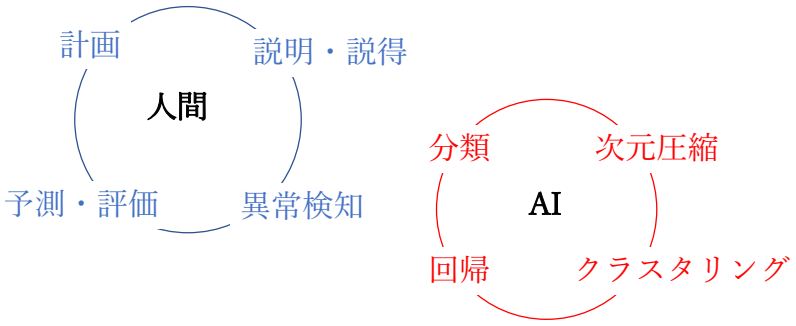
多くの変数をもつ多次元のデータなどに対して，変数を削減するタスクです．2 次元や 3 次元などの変数で説明できればグラフによる可視化などに役立ちます．統計でいうところの因子分解や主成分分析などの手法です.

・回帰（Regression）

モデルを利用した予測などをするために，モデルを決めるためのタスクが回帰です.

人間の行っているタスクとの比較をしてみましょう．図には人間と AI のタスク例を図化してみました．マネジメントの手法である PDCA などでは，人間は PLAN（計画）をします. PLAN のためには AI でいうところの予測が大事です. 予測結果を利用して Do（実行）するためには，予測結果の Check（評価）が必要です．次元圧縮などの手法は人間が結果を評価するための情報と関連しているでしょう．また，どの選択肢が良いかを分類するなども評価のためのタスクと言えるでしょう．AI の予測，評価，分類などを自動で行うことで実行を代替することもできるかもしれません．ここでは，クラスタリングなど，人間が理解で

きないような情報も分類することができるかもしれません．異常時などの危機対応も人間にとっての重要なタスクですが，疲れをしらない AI による異常検知は，この人間のタスクを助けるためのツールとなるでしょう．自動運転などの技術は AI による判断でハンドルを操作するということですが，車の運転のように目的がはっきりとしている判断には AI の活用がすぐにでも期待されています．また，株の売買のような利益を上げるというはっきりとした目的があれば，AI の活用は正当化されるでしょう．将来的には，マネジメント判断のように，組織を説明したり説得したりする際に，AI による判断が優先される時代がくるかもしれません．ただし，企業，まして人間の生きる目的のように，目的関数が決まっていない問題に AI の判断をどのように利用しいていくのか，AI 活用にビジョンが必要と言われる理由かと思われます．



図－１　人間と AI のタスク例

b)　データと AI

深層学習など近年の AI ではデータが肝だと言われています．ここではデータの種類でユースケースを分類してみましょう．

○時系列

時系列データとは，時間に応じて変化するデータです．時間に応じて変化するので順番が重要となります．人間の言葉などは順序が重要なデータですので，このような順序を評価するための深層学習モデルとして RNN や LSTM といったモデルが開発されています．また，センサなどの観測値も順序が重要なデータですので同様のモデルを利用することができます．センサなどの観測データには誤差が含まれることが知られていますが，フィルタリングや平滑化，さらに予測などを行う AI が開発されており，異常検知などのタスクで利用されています．

○画像

深層学習の開発により機械が目を持ったと言われています．カメラから得られる画像を

AI が解析することで機械が目を持つわけですが，画像をデータしたタスクとしては，以下のものがあります.

・画像認識：画像に何が写っているかを認識する　→犬なのか猫なのかなど

・物体検知（画像のどこに何が写っているかを検知する）　→医療などの画像診断（異常正常まで評価）

・セグメンテーション(画像に写っているものをピクセル単位で分類する）　→自動運転時の周辺評価など，より厳密な位置判定を求められる場合

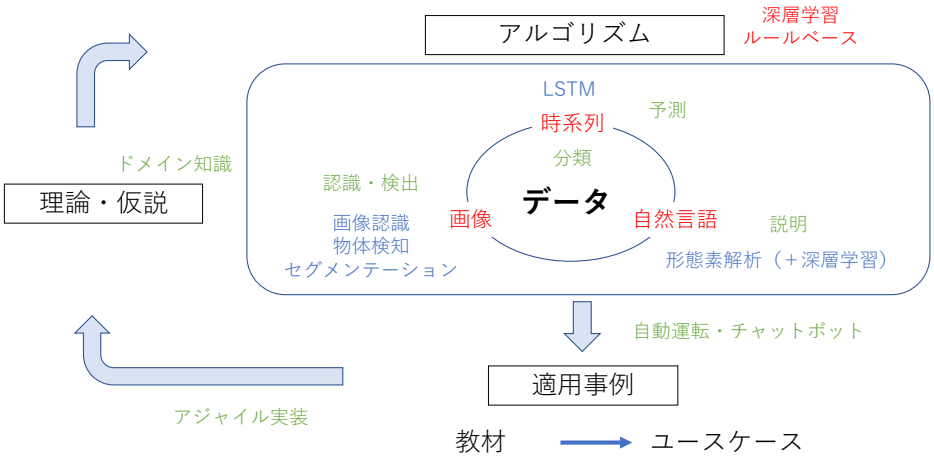
画像データはピクセルごとに輝度値などの値が入った二次元配列データ，あるいは RGB 値が入った三次元配列データとして扱われますので，これらを効率よく解析するための深層学習モデルとして CNN などが開発されています．インフラメンテナンスにおいても，検査画像からコンクリートのひび割れを自動認識するなどの技術で，物体検知やセグメンテーションなどが検討されています.

○自然言語

時系列データのでも記載しましたが，人間の利用している言語データを解析する AI があります. チャットボットと言われる AI は, テキストや音声といった言語データを分析して，Q &A を作成したりすることができます．自然言語は記号の集まりですが，英語や日本語など言語の種類に応じてルールが相違します. 単語なども相違しますが，これらはそれぞれの辞書などを利用した形態素解析と言われる分析により機械が理解しやすい頻度データなどが作成されます. それらのデータを利用して，ある単語の後にはある単語が来る確率が高いなどのアルゴリズを利用して AI システムが実装されます．自然言語処理を利用したテキストの認識の精度は，特定の対象に対する固有表現を学習できているかが影響します．そのため，固有辞書と言われる，いわゆる土木用語などの辞書の作成が重要となります．このような辞書をあらかじめ広く共有することで，手元の少ないデータからでも，モデルの学習や適用がしやすくなると考えられます.

○マルチモーダル

上記で整理した，時系列，画像，自然言語などを複数のデータを利用する AI がマルチモーダルな AI と言われています．Google でおなじみのナレッジグラフなどはインターネットに存在するテキストや画像データを統合的に可視化することで，知りたいことについて人間の理解を深める方法です.



図－２　様々な種類のデータとその活用ユースケース

※構造化データと非構造化データ

データには構造化データと非構造化データという考え方があります．分析によく利用される Excel などで扱われる表形式でデータを扱う際には，列ごとに項目を分けて，行方向に値を入れていくことが多いと思いますが，データの区切りにカンマを使ったものを CSV データと呼びます．また，データを複数の表として管理し，表と表との関係性を定義したものを RDB（リレーショナルデータベース）と呼びます．このように決まった枠の中に文字や数値が格納されているものが構造化データです．

非構造化データは，表形式のような構造を持たないものですが，JSON，XML，HTML のようにキーとバリューの組み合わせといった規則性を持つものがあります．さらに，ベタ打ちのテキストデータや，振動，音声のような数字配列のみのデータも非構造化データです．非構造化データは，サンプリングレートや単位などのメタデータを与えることで，データの意味を理解したり，分析をすることができます．

構造化データの分析では，エクセルなどを利用すれば集計や回帰などのタスクを行うことが容易にできます．非構造化データでも規則があるものであれば，データの入出力を方法を含めたアプリケーションソフト（API）などを利用して，検索や集計，データ連携や圧縮などのタスクを行うことができますので，インターネットなど WEB 上での様々な活用がなされています．

規則の無い非構造化データをさらに利活用していく上でも人工知能は期待されています．例えば画像データは，解像度（ピクセル数）などが様々なモノがありますが，画像サイズを合わせるなどの前処理を適切に行うことで，どのような画像に対しても自動的にタスクが実行できるようにする技術を AI 化することが考えられます．データの構造を自動で認識して利活用ができれば，既存のデータベースや過去に EXCEL で整理したデータなどを再度人間が前処理する労力を大幅に減らすことで，AI による利活用の実現性が高まると考えられます．

c) AI と土木

人工知能の活用分野は，移動，製造，個人向けサービス（医療、金融を含む），対話・交流（コミュニケーション）などが挙げられ，土木はすべて関係があると言えるでしょう．AI を活用する際の論点として表－２のようなものが挙げられており，これらを土木で考えてみることも大切です．その際には，土木の特徴として，公共財，他産業への経済波及効果（スピルオーバー），統計との相性，業務が空間的に分散かつ時間的に長期（メンテナンス）などを考慮してユースケースを検討することが重要です．AI を活用することで，人間にできない作業を代替や支援して，危険な作業，長期労働や夜間作業を減らすなど働き方改革が期待されます．また，デジタルトランスフォーメーション（DX）が進行し，デジタルで情報を共有することで，仮想空間で解析した情報を実空間で活用するデジタルツインといった手法が進みでしょう．それにより，様々なプロセスの効率があがることによる生産性向上だけでなく，不要な業務が無くなるなどの，業務革新が進むことが期待されています．

表－２　AI 活用と土木

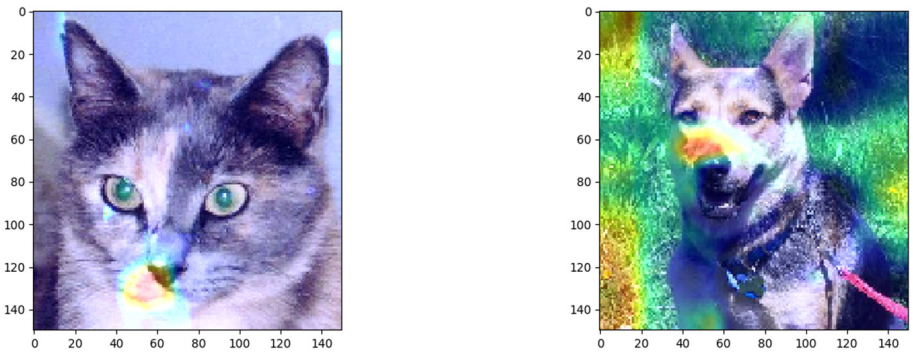
AI 活用の論点	土木との関連
倫理的論点	技術者倫理，設計と施工の在り方
法的論点	責任（事業者と請負・委託者の関係），基準
経済的論点	→労働内容の変化，新しい雇用は？データサイエンティスト？ →業務再構成（機械による代替），創造的な業務？ →請負契約，包括契約，報酬の変化，積算， →系列化，エコシステム，中小企業
社会的論点	労働力不足，高齢化，ミスマッチ 賃金，待遇，新３Ｋ コンパクトシティ，地方創生，環境問題 インクルージョン，インバウンド 持続性（選択と集中） 防災，レジリエンス
教育的論点	技術継承 大学教育（ミスマッチ）　統計教育，インターン
研究開発的論点	オープンイノベーション（透明性） 異業種連携 デュアルユース 社会実装

2.1.2. 橋梁形式分類 AI を実装してみよう　その１　犬猫分類問題で AI を理解する

a)　AI による画像認識で犬猫を分類する

AI における深層学習を利用した画像認識として，以下に説明する犬と猫を分類する事例があり，これらを理解することで，AI 手法の中で特に深層学習の理解を深めることができます. 犬猫の分類は，インターネットなどにデータがあり，実装できるプログラムを含めて，既存の手法が豊富にネット上にあります．また，学習プログラムだけでなく、AI が何を着目しているか分類するプログラムなどもネット上のものを活用できます. 例えば, Grad-cam という手法を利用すれば，以下の図－３のように犬と猫を鼻の特徴に着目して分類していることがわかります.

猫　　犬



図－３　犬と猫の分類のための着目位置の導出

b)　AI を学習してモデルを評価するとは

○AI の学習の仕方

AI がデータから学習する方法は，一般的に以下の 3 種類があると言われています．犬猫分類の問題は一番実装が容易な教師あり学習を利用します. 教師ありですので，画像には犬なのか猫なのかあらかじめ答えを用意する（アノテーション）必要があります.

A. 教師あり学習

犬猫の画像があり，犬と猫を分けるラベルが付いていて，それを分類する問題です.

⇒分類されたデータに境界線を引き，未知のデータを分類

B. 教師なし学習

犬猫の画像をもっているが，犬猫のラベルはついておらず，クラスタリングなどで分類す

る問題です.
⇒特徴量をデータから抽出して，クラスタリングにより分類（異常データを分類できるかなどでも活用）

C. 半教師あり学習
犬の画像しかなく，犬を表すモデルを学習して，猫画像が新規に来た時に犬と違うと分類する問題です.
⇒正常データの特徴量を抽出して，未知の異常を検出問題など

○AI を評価しよう
学習した AI は評価しなくてはいけません．その際に学習したデータ（train データ）で評価するのではなく，検証用（validation データ）のデータを用意しましょう．検証用のデータをモデルに入力して正解である率が高ければ（正解率）よい学習が行えたモデルと言えます．これらの作業は，交差検証とよばれます．教師データでは正解率が高いが，検証用データで正解率が低くなる問題を過学習と呼びます．検証用データで正解率が高いモデルは汎化性能が高いモデルと呼ばれます.

○AI と特徴量
AI は画像の何を根拠に犬と猫を分類するのでしょうか．よく言われる問題として，AI，特に深層学習はブラックボックスであると言われます．画像はピクセルごとに輝度や RGB 値のデータが入ったものですので，それらのデータの特徴を利用して分類しているに違いありません．どのような輝度の特徴を利用して分類しているかがわかる，つまり，犬の特徴や猫の特徴などをちゃんととらえていることがわかれば，AI を利用する我々も安心して AI の答えを信頼することができます．これらの手法は，説明可能 AI（XAI：Explainable AI）と呼ばれて，様々な方法が研究開発されています.

○学習環境の構築と実装環境
AI はニューラルネットワークというアルゴリズムを多く利用しています．この計算は複数の入力をニューロン（ノード）が受け取り，それぞれに重みを付けて統合して，活性化関数を通してある値より大きなものを次のノードに送っていく，それを繰り返して最終的な出力を出す計算です. 計算された出力と正解を比較してその誤差を小さくする（バックプロパゲーション）ことで重みを最適にする計算を繰り返します.
近年流行している深層学習はこのノードの数や層が深くなったものですので，一つ一つは単純な計算ですが，多数の計算を行います．そのため CPU を利用した計算よりも GPU を利用した計算の方が早く実行できることが知られています．GPU は一般的な PC には搭載されていないことが多く（あるいは搭載されていても貧弱なものが多い），深層学習を行

うためにスタンドアロン PC で GPU パソコンを用意するか，もしくは，クラウド環境で GPU を利用することが行われています．

スタンドアロンで環境を用意する場合は，Python などのコードを実行するためのソフト的な環境（Anaconda などが有名）を構築する必要があることにも注意が必要です．クラウドを利用する場合はソフト的な環境も用意されていることが一般的ですので，ちょっと試してみたいなどではよいかもしれません．たとえば Google Colab などは Google アカウントがあれば無料で GPU が使え，しかもわかりやすいので頻繁に用いられます．

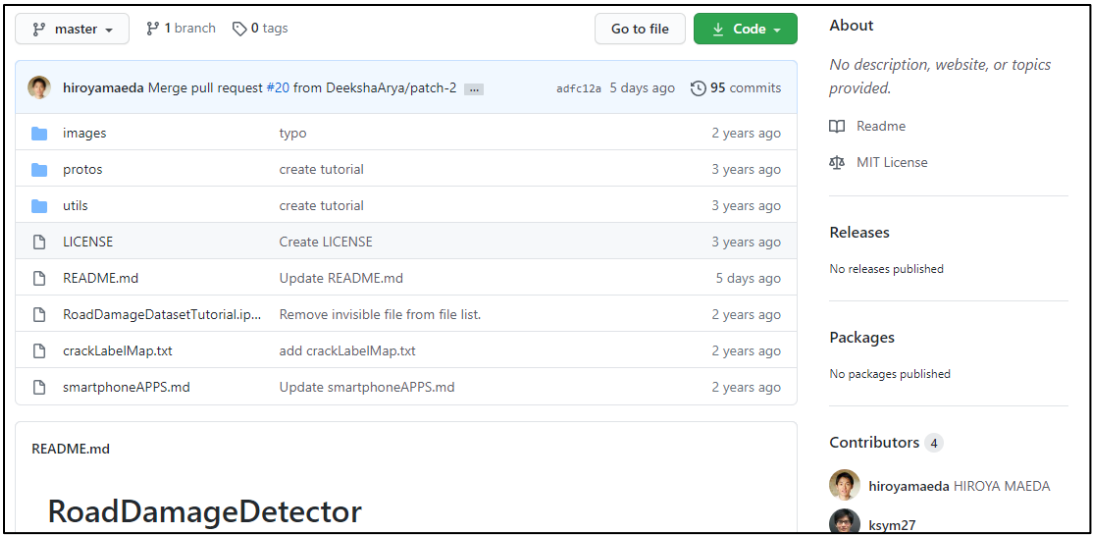
python などで AI プログラムを構築する際には，機械学習を効率よく実装できるフレームワークが開発されています．深層学習などを利用する代表的なものとしては，TensorFlow や PyTorch などがあります．さらに，Tensorflow をベースにしてさらに利用しやすくした Keras などのラッパーと呼ばれるものが用意されています．プログラムの柔軟性と利用しやすさはトレードオフですので，問題の難易度に応じて選択してください．

○既存のデータとモデルを活用しよう

AI は一から自分ですべて作るのは大変です．特に深層学習のデータやモデルはインターネットにある既存のものを活用することが近道です．よく利用されているものとして，GitHub という開発プラットフォームがあり，オープンソースとしてデータがリポジトリにアップロードされています．例えば下記のようなものが参考になるでしょう．利用されているプログラムやデータがダウンロード（クローンやフォーク）でき，それらの利用方法も記載されています．また，SNS 機能があり，質問や回答をやりとりすることができ，まさに開発技術者のプラットフォームと言えるでしょう．

https://github.com/crotsu/Bousai_AI

https://github.com/sekilab/RoadDamageDetector



図－４　GitHub にアップロードされているコード類の例

c)　犬猫分類問題を実装する

○画像データ収集

インターネットなどを利用してデータを収集します．　ネットから画像収集するクローリング技術と呼ばれる方法を利用することもできます．ただし画像を利用する際には著作権などに気を付けてください．点検データなどでは，橋梁名を抜くなどのデータの匿名化などによって対応することも考えられます．

○画像の前処理

教師データは多ければ多いほど良いですが，画像を収集するのには労力が必要です．このため教師データの画像枚数を増やす方法として，画像を引き延ばしたり，回転させたりするデータ拡張技術があります（Data augmentation）．加工した画像に対しても答えは同じですが，加工されることで人間の目でも画像が変化したことがわかりますので，そのような画像を正確に分類できればAIはより賢くなっていると言えるでしょう．

※画像データは2次元や3次元などの配列です．モデルに入力する際には配列の構造が重要です．pythonで利用されているnumpyのarrayとpytorchで利用されているtensorなどではこの配列構造が相違します．お互いに変換するプログラムなどが利用できます．AIモデルを作成するパッケージに応じて入力データを変換してください．

○アノテーション

画像に対して正解を付ける作業です．例えば，犬の画像はdogフォルダに，猫の画像はcatフォルダに分類したり，犬の画像のファイル名にはdogを，猫の画像にはcatなどを付与しておけば，プログラムの中で正解となるベクトル（label）を作ることができます．

○学習データの作成（既存プログラムの修正）

分類を実装するプログラムは，Kaggleと呼ばれるコンテストなどでも世界的に行われており，そのため様々な開発者が挑戦し，インターネットにファイルがアップロードされています．また，深層学習の教科書にも取り上げられているためGitHubと呼ばれるソフトウェア開発のプラットフォームに実装したプログラムや学習用の画像データがアップロードされています．それらをダウンロードして実装する際にはいくつか修正が必要となります．特に利用するデータの保存先や保存方法によってプログラムでデータを読み込む際のパスなどの設定があります．プログラムでデータを学習する際には，データを教師データと検証用データに分ける必要があります．

○学習方法

学習は，入力データをモデルに入力して出力を算出し，出力と正解との差が小さくなるようにパラメータを変化させる最適化計算を行います．これには勾配法と言われる方法が通常利用されます．勾配法を適用する際には，パラメータを更新するたびにすべてデータで誤差を計算するのは効率が悪いために，バッチサイズと呼ばれる誤算を計算する際の画像の枚数を決め，ランダムに選んで画像の組を作成して計算を行います．そのために，データローダーというものを利用します．バッチサイズは2のn乗の値が使われることが多いですが，画像の枚数に応じて選択してください．データセットの件数が数百件程度であれば32，64で試してみればよいでしょう．学習回数はイテレーションやエポックと呼ばれますが，

学習する画像枚数とバッチサイズ数で回数が決まります．例えば 1000 枚の画像があり，バッチサイズが 200 であれば，すべての学習データを利用するためには 5 回学習する必要があるのでイテレーション数は 5 になります. この学習を何回繰り返すかがエポック数です.

○学習

モデルを一から作るのではなく，既存の学習済みモデルを利用することでよい AI モデルを作ることができます．その場合は，転移学習やファインチューニングと呼ばれる手法を利用します．エポック数が小さすぎて学習が完了しない場合や，ハイパーパラメーターと呼ばれる学習率などにより答えが変わることがありますので注意が必要です.

○検証

学習したモデルを評価します．正解率，適合率など複数の指標がありますので注意しましょう．学習結果の可視化はグラフで行う場合などがありますが，どの評価関数を選ぶかなどを決めます．パッケージには可視化するかしないかを変数で決めれば勝手に可視化してくれるものが多いです.

○モデルを利用する

学習したモデルのパラメータを保存しておくことが可能です．保存したパラメータを再度読み込むこともできます．検証用データは学習ができているかを確認するためのデータですが，それとは別にテスト用のデータを利用する場合もあります．テストデータを実際にモデルに入力して分類結果を確認するなど，モデルを利用してみることでよりプログラムの理解を高めることができます.

※参考　医療における下記のサイト

https://japan-medical-ai.github.io/medical-ai-course-materials/

Google Colaboratory を利用して，実際にプログラムの実装までできるようになっています.

2.1.3. 犬猫分類問題を応用して橋梁形式を分類してみよう

データ，手法と合わせてユースケースの設定が重要となりますが，犬猫分類の応用で，例えば，土木で考えた事例としては，桁橋とその他の橋梁形式（トラス、アーチ、吊橋）を分類するような問題を考えてみましょう．これは，腐食などの損傷を評価するような AI 技術において，そもそも対象物（鋼材，部材等）の位置を認識するなどの基礎的な技術と考えることができます．なおサンプルプログラムは以下のサイトからダウンロードできます．
https://committees.jsce.or.jp/struct10/node/16

○データの準備とアノテーション

犬猫分類では，犬猫画像を収集して，画像に対して犬なのか猫なのかを目視でタグ付けを行うアノテーションをしました．同様に橋の画像をネット上で収集する必要があります．橋の全景画像データはインターネットに多くあるため収集が可能です．アノテーションをどのようにするかを考える必要がありますが，より簡単な問題として，桁橋とそれ以外でフォルダ分けしてみましょう．ここでは，学習データ１５０枚　検証データ５０枚　テストデータ５０枚で整理することとします．

桁橋	その他形式（トラス，吊橋，アーチ）

図－５　桁橋と桁橋以外の橋梁の例

○モデルの構築

深層学習モデルを構築します．層の比較的浅いニューラルネットワークと，層の深い転移学習を行ったモデルを比較することで深層学習の特徴を考察しましょう．転移学習とは様々な画像で学習したモデルを利用して，自分で集めた画像で再度学習をする方法です．ただし，モデルの答えが犬や猫だけでは困りますので，橋の分類に合った答えを返してくれるようにモデルを改良する必要があります．具体的には，プログラムの最終層を変更します．また，ファインチューニングという方法があり，こちらは集中した画像で最終層だけでなく，途中のモデルのパラメータも学習しなおす方法です．

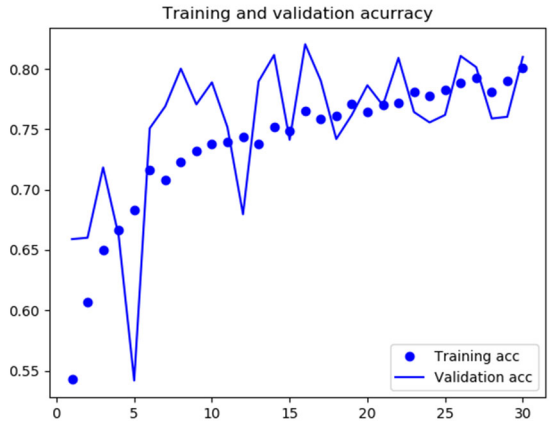
○モデルの結果を評価する

適用結果を以下に示しました．層の比較的浅い深層学習では，正答率が低く，また、学習が進むことで正答率は上がっていきますが，検証データに対しての正答率は増えていきません．これは過学習という問題です．それに対して，既存の様々な画像データの特徴を学習したモデルによって転移学習を行うことで，正答率が高くなっています．学習に利用していないテストデータの分類もうまくいくので，過学習の問題も起こらないことがわかります．

・適用結果１　層が浅い場合

学習モデルは以下の４層の CNN

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 398, 398, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 199, 199, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 197, 197, 64)      18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 98, 98, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 96, 96, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 48, 48, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 46, 46, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 23, 23, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 67712)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 67712)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               34669056
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 34,910,401
Trainable params: 34,910,401
Non-trainable params: 0
```

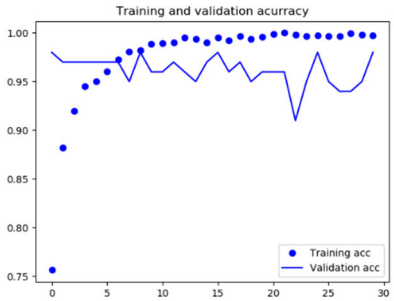


・転移学習を利用した場合

ImageNet で学習済みの VGG１６モデルを転移学習

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 400, 400, 3)	0
block1_conv1 (Conv2D)	(None, 400, 400, 64)	1792
block1_conv2 (Conv2D)	(None, 400, 400, 64)	36928
block1_pool (MaxPooling2D)	(None, 200, 200, 64)	0
block2_conv1 (Conv2D)	(None, 200, 200, 128)	73856
block2_conv2 (Conv2D)	(None, 200, 200, 128)	147584
block2_pool (MaxPooling2D)	(None, 100, 100, 128)	0
block3_conv1 (Conv2D)	(None, 100, 100, 256)	295168
block3_conv2 (Conv2D)	(None, 100, 100, 256)	590080
block3_conv3 (Conv2D)	(None, 100, 100, 256)	590080
block3_pool (MaxPooling2D)	(None, 50, 50, 256)	0
block4_conv1 (Conv2D)	(None, 50, 50, 512)	1180160
block4_conv2 (Conv2D)	(None, 50, 50, 512)	2359808
block4_conv3 (Conv2D)	(None, 50, 50, 512)	2359808
block4_pool (MaxPooling2D)	(None, 25, 25, 512)	0
block5_conv1 (Conv2D)	(None, 25, 25, 512)	2359808
block5_conv2 (Conv2D)	(None, 25, 25, 512)	2359808
block5_conv3 (Conv2D)	(None, 25, 25, 512)	2359808
block5_pool (MaxPooling2D)	(None, 12, 12, 512)	0
sequential_1 (Sequential)	(None, 1)	18874881

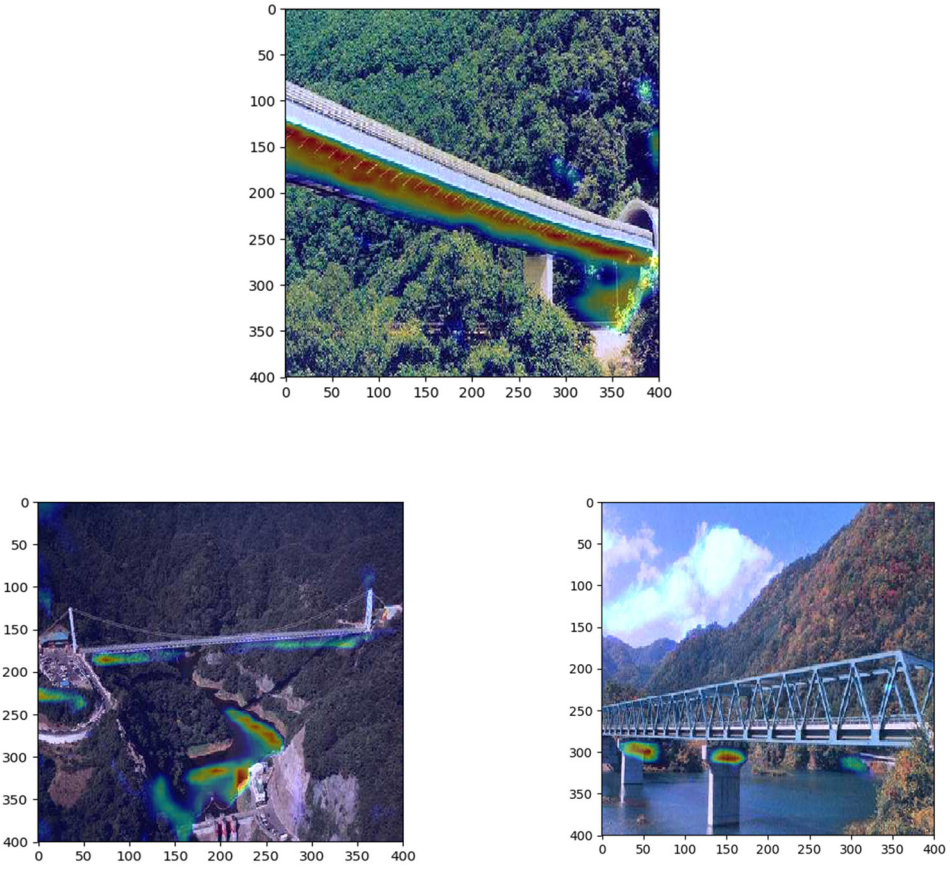
Total params: 33,589,569
Trainable params: 25,954,305
Non-trainable params: 7,635,264



○モデルの評価

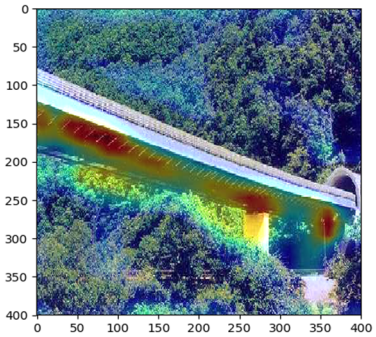
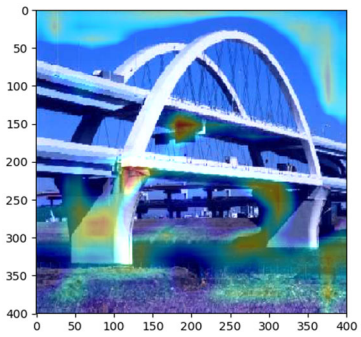
以上でモデルが学習できましたが AI は画像のどこをみて分類しているのでしょうか．これを可視化する方法として，Grad-CAM という方法があります．浅いモデルのヒートマップを表示してみると橋でない場所を見ているデータがよく出てきます．それに対して転移学習の結果では橋の特徴を利用して分類しているように見えますね．また正解できなかった画像のヒートマップを見てみることでどうして間違ったかがわかるかもしれません．それにより，AI モデルの適性や，どのような学習データを追加して改良する必要があるかなどわかるかもしれません．

・浅いニューラルネットワークの結果



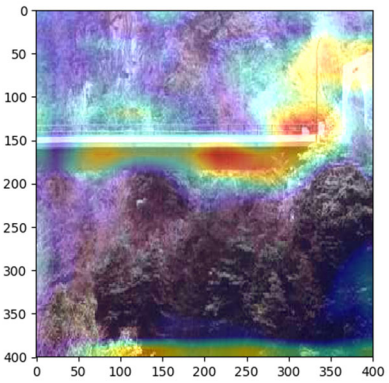
図－6　GradCAM による浅い位置での可視化結果

・深いニューラルネットワークの結果（block5_conv3 層の可視化）

桁橋	その他
	

図－７　深い位置での重み可視化結果

また，以下のように，桁以外に間違えそうになった画像についても，どこからその判断をしたかが確認できる．



図－８　判断が難しい画像の重み可視化結果

2.1.4. 事例で AI のできることを知ろう　その１　画像物体検知とアノテーション

〇物体検出とアノテーション

画像データを利用した AI のキラーコンテンツと言えば自動運転かと思います．自動運転で利用される AI 画像タスクとして物体検出があり，YOLO（You only look once）や SSD（Single Shot Multibox Detection）などの技術があり研究も盛んです（図－９）．

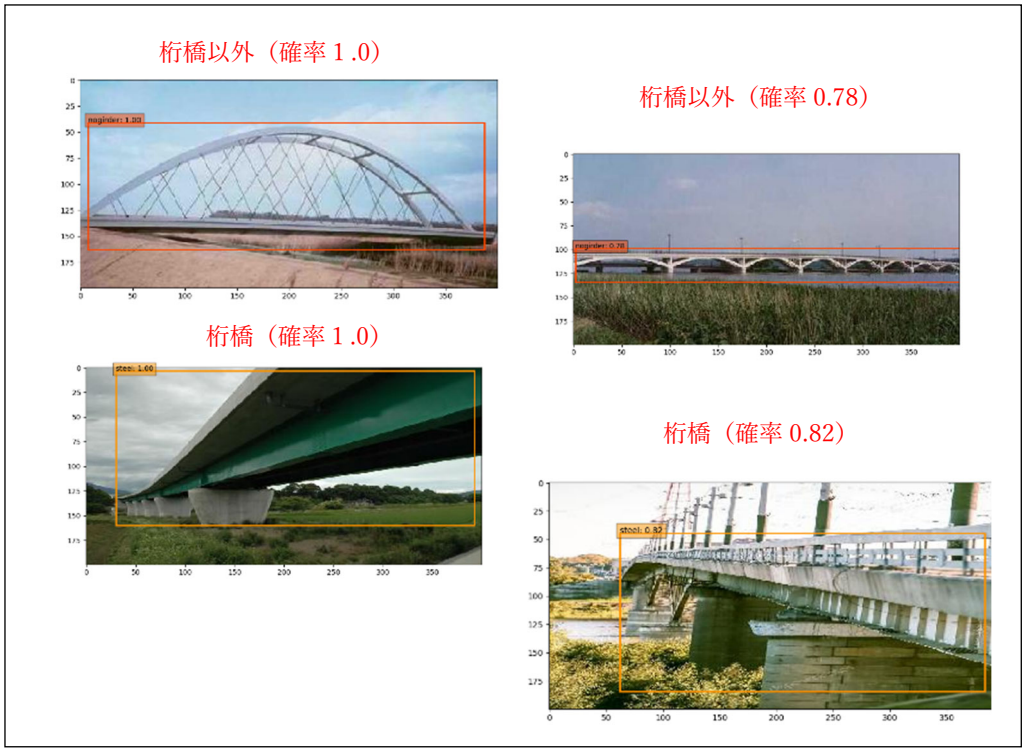
橋梁を撮影した画像データにも，橋以外が写り込んでいる場合や，橋が複数写り込んでいるものなどがあるかと思います．これらの画像では，画像の中のどこに橋があるのかを検知する物体検出手法を適用することが有効でしょう．橋がどこにあるかを検出することは，例えば災害時に落橋が起きていないか，桁の移動はどの程度か，といった画像監視を利用した異常検知手法のためのファーストステップともなりますし，また，点検支援などにおいて，支承部など特定の部材を認識して評価したい場合などでも，まずは点検画像のどこに橋があるかを検出することが重要と思われます．

図－９　対向車でもっとも至近距離の自動車を検出した事例

〇橋梁の検出実装

SSD を利用した橋の検出を実装してみましょう．物体検出では，バウンディングボックスを利用することで，画像に対して，位置や大きさを含めて物体を認識して検出することができます．ここでも VGG16 で学習したモデルを，物体認識で利用した橋の画像に対し，桁橋と桁橋以外に分けたバウンディングボックスを利用してアノテーションした画像で転移学習を行いました．物体検出した結果を以下に示しました．

このような物体検知手法などは，API 的に利用するためのライブラリが整理されており，実装が容易となっていますので試してみることができます．ただし，実際の問題に適用するためには，アノテーションツールを利用して教師データを作成するなどの作業が必要となります．



図－１０　桁橋と桁橋以外を区別して検出する SSD による結果の例

2.1.5. 事例で AI のできることを知ろう　その２　時系列データに対する機械学習例

〇時系列データに対する予測や異常検知

多変量解析や時系列データなどにも機械学習は適用できます．実際のデータには異常の有無は既知でないことが多いですが，あらかじめ異常がわかっているものや，異常を模擬した試験体等でデータを得ることができれば，教師あり学習を行うことができます．教師データが得られない場合には，教師なし学習が開発されており，データに応じて適切な手法を選ぶことが課題です．表－３にそれぞれの学習方法の分類について示します．

なお本節のサンプルプログラムは以下よりダウンロードできます．

https://committees.jsce.or.jp/struct10/node/18

表－３　機械学習の分類

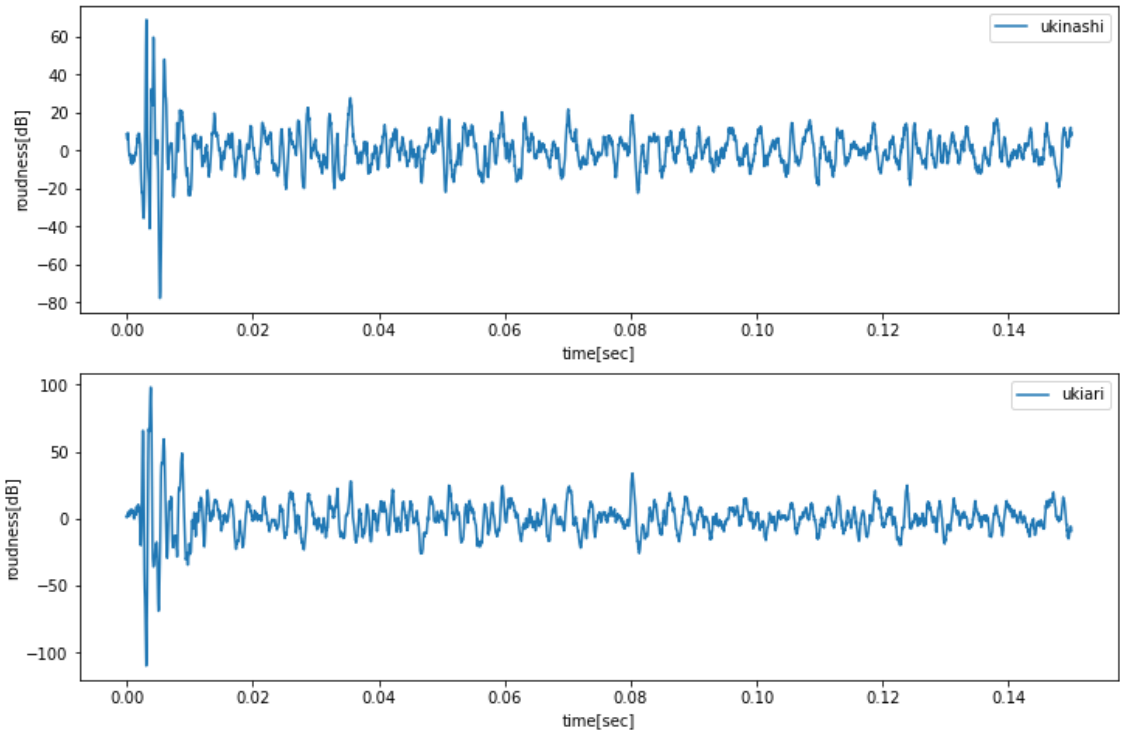
分類	適用条件	適用の際の課題	メリット
教師あり学習	教師付けされたデータが多く手に入る場合	・未知の異常を検知できない	・手法が理解しやすい ・異常の説明がしやすい（特徴量の変化に対する評価） ・学習したデータ以外にも適用できる
教師なし学習	異常を含むデータが多く手に入る場合	・異常データが観測されていない場合がある ・モデルの説明性	・異常の種類や頻度を決められる
半教師あり学習	異常データがあまり手に入らないがデータ自体が多く手に入る場合	・そもそも正常データの特徴を学習できるか（特徴量がみつからない，ばらつくなど） ・モデルの説明性 ・正常ではないが異常でもない事象を抽出した時の解釈性	・対象データごとの正常モデルを作れる

○実装事例

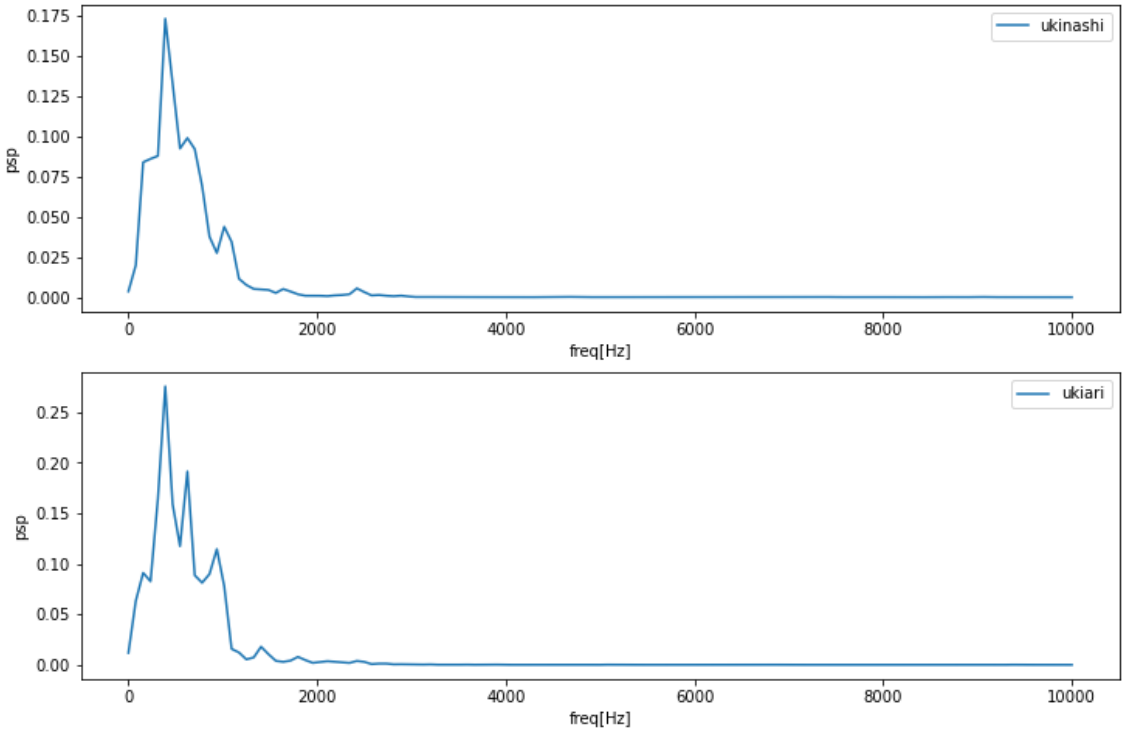
ここでは振動や音の波形データを利用して，特徴を学習させて予測や異常検知をする方法について記載します．具体的にはコンクリートを打音した際の音圧データを利用した浮きの検知について実装します．

データを作成するためには，計測や試験の条件等様々な検討が必要です．ここでは，叩いた位置の浮きの有無がわかっているとして，打撃した際の音をマイクで収録した音圧データが csv ファイルで得られているとします（なお，本データは東工大竹谷研究室で行った試験のデータの一部を利用させて戴いております）．音圧データに対して浮きの有無を分類する 2 値の分類問題を実装したプログラムを紹介します．

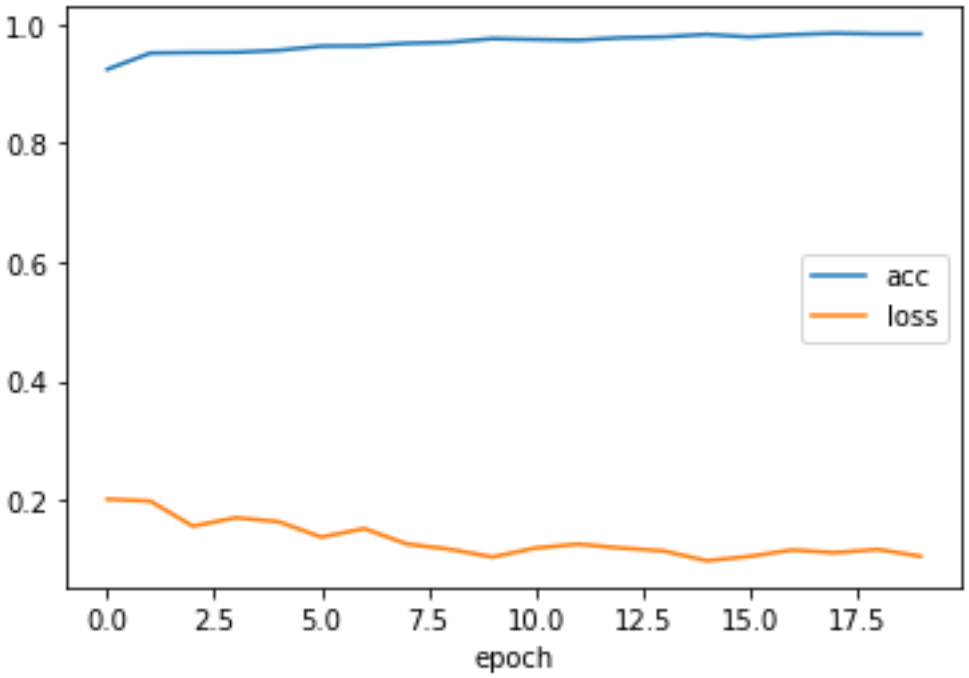
分類では，利用するモデルはニューラルネットワークを使い，入力を波形データ（要素数 3000 の 1 次元ベクトル），出力を 2 要素の 1 次元ベクトル（浮きがある確率，浮きが無い確率）としております．近年ではニューラルネットワークの層を深くする深層学習により精度が向上するような問題が多く取り上げられていますが，深層学習はデータを表現するための特徴量が多くなる問題で有効です．本問題では，分類のための特徴量はそれほど多く必要が無いと考えられ，それほど層を深くしなくても分類ができる例となっています．なお，層を深くするなどの改良は簡単に行うことができます．



図－１１　浮き有無と音圧波形の比較



図－１２　浮き有無とパワースペトルの比較



図－１３　学習結果

2.1.6. 事例で AI のできることを知ろう　その３　自然言語処理学習例

○自然言語処理

自然言語処理 AI のキラーコンテンツとしては自動翻訳がありますが，問い合わせ対応を支援するチャットボットや，事務作業支援のための RPA（ロボティック・プロセス・オートメーション）などすでに実用化されているものが多くあります．具体的な手法としては，twitter など SNS のデータを利用して文章中で出現頻度が高い単語を複数選び，その頻度に応じた大きさで図示する Word Cloud と言われる手法や，言葉のつながりや意味を解釈するための word2vec などの手法などがあり，広告業界などでマーケティング手法として活用されております．インフラメンテナンスにおいても，過去の点検記録などを自然言語処理して，対策方法などがレコメンドされる機能などが期待されます．

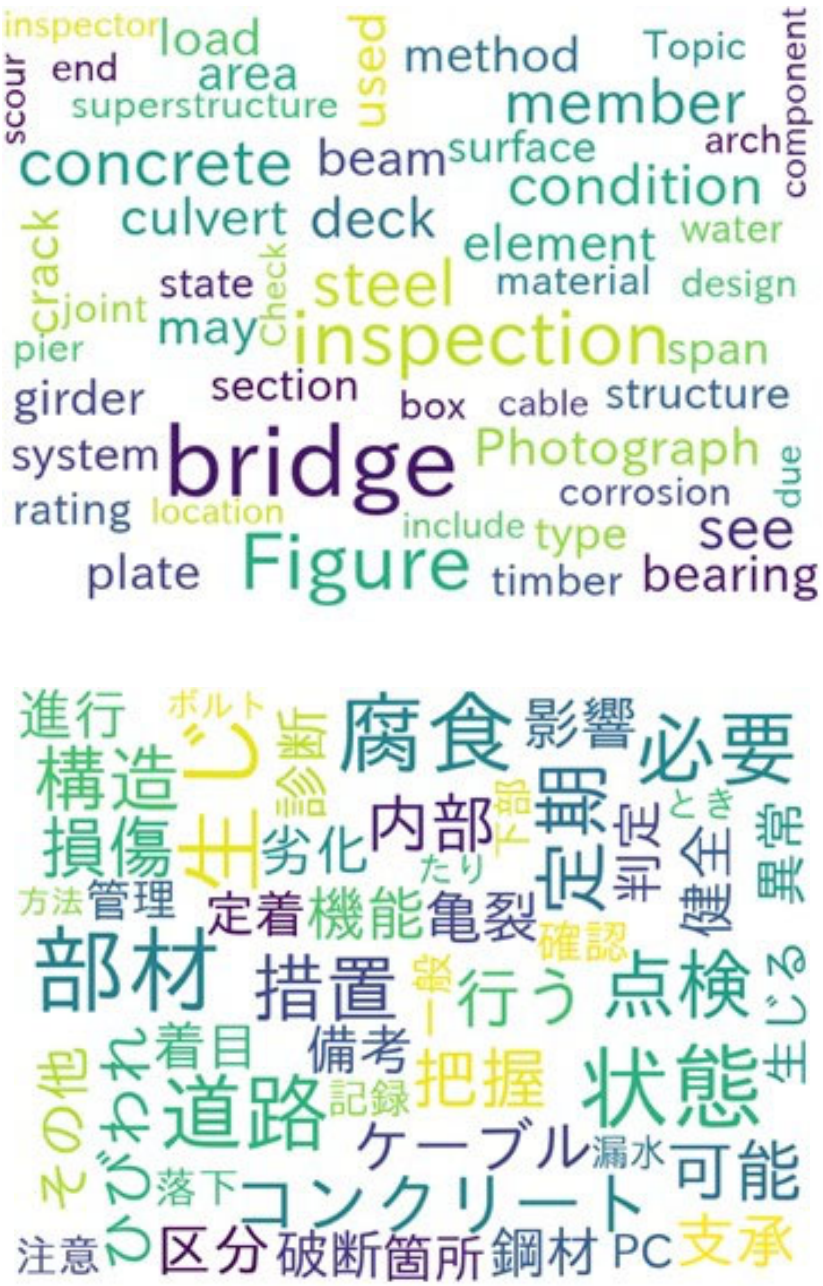
なお本節のサンプルプログラムは以下よりダウンロードできます．

https://committees.jsce.or.jp/struct10/node/19

○実装事例　示方書や点検マニュアルの解析

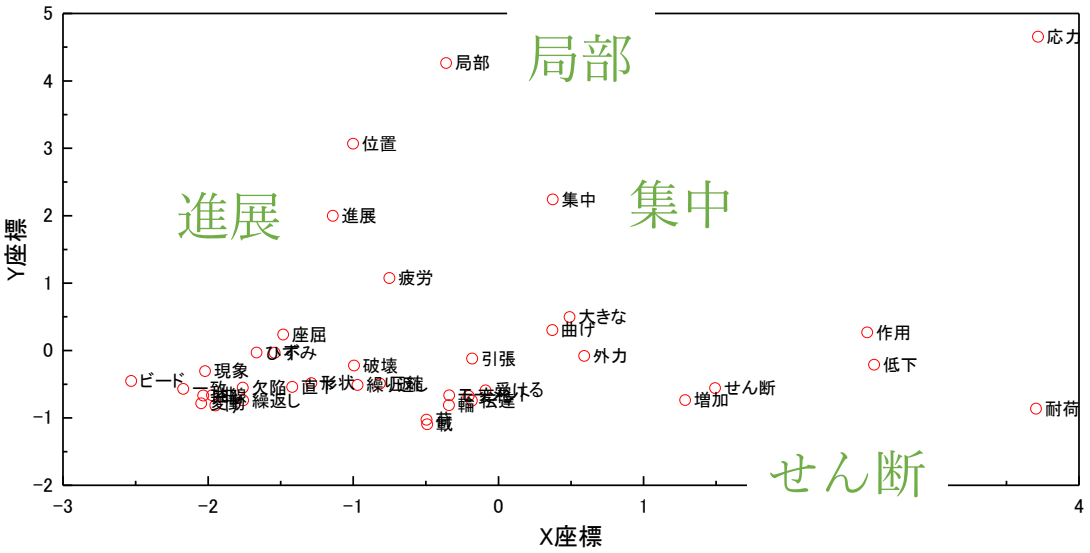
橋梁点検マニュアルに記載されているテキスト情報について，Word Cloud を実装して，米国の点検マニュアルと日本の定期点検要領を比較してみましょう．図は適用した結果です．米国のマニュアルのほうが，橋自体に対する単語(bridge，structure，deck，culvert など)が多いように思われます．それに対して，日本の点検要領は，橋自体に対する説明は多くなく，ひび割れ，腐食といった損傷に関する単語や，診断，措置，記録などのプロセスに関する言葉が相対的に多いことがわかりますね．この要因として推察されることですが，日本の点検要領は基準であるのに対し，米国のマニュアルでは教材的な要素が強いなどが挙げられるでしょう．

次に日本の点検基準や補修補強の手引きなどのテキストを，言葉のつながりや意味を解釈するための word2vec を利用して分析してみましょう．word2vec は仮説として，単語の意味は周辺に現れる単語の頻度で表現できるというものです．例えば，「応力」に関連する単語について関連する単語を抽出して関連性が高いものをマッピングして可視化した結果ですが，応力という言葉には「局部」,「集中」などの言葉が周辺に多くあらわれるため，近くに表示されていることがわかります．また「進展」,「疲労」などの言葉もあります．これらの単語は足し合わせなどをすることもでき，「応力」＋「溶接」などを足し合わせると「集中」などの言葉が大きく関連していることが確率値として出てきます．この結果は，補修補強などの文章において，これらの言葉の頻出頻度が類似しており，また時系列的に順番で出てくるなどがわかりますので，原因や処方箋などを書く際の参考になるかもしれません．



図－１４　点検要領の頻出 WORD の可視化例

（上：米国　下：日本）



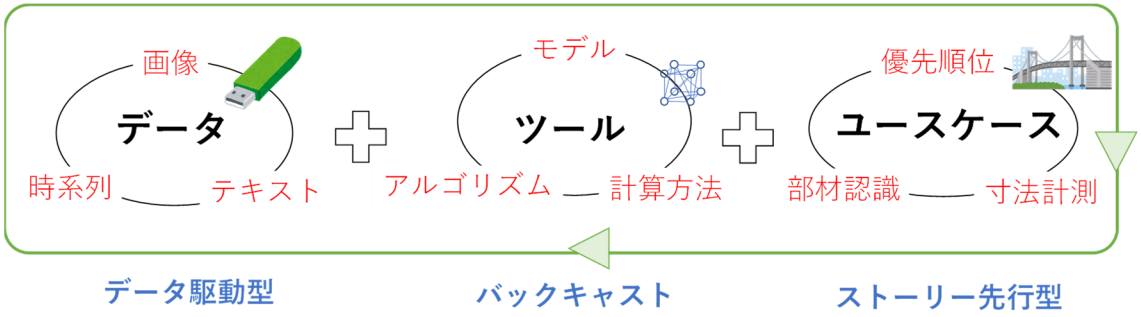
図－１５　「応力」に関連する単語の可視化

単語	関連度
集中	0.87
リブ	0.79
ビード	0.78
デッキプレート	0.77
疲労	0.75
ひずみ	0.74
進展	0.74
亀裂	0.72
伝達	0.71
近傍	0.7
曲げ	0.69
・・・	・・・

図－１６　「応力」＋「溶接」に関連する単語

2.1.7. AI を利用したメンテナンスユースケースを考えよう

AI 技術を実装する際には図－１７に示すように，データ，手法，ユースケースを考えることが大事です．ここでは手法ごとにユースケースを整理して，実装するためのデータを検討してみましょう．



図－１７　データとツールとユースケースの関係性

○教師あり学習

教師あり学習を行うためには大量の教師データが必要となります．インフラデータでは実際の構造物で異常が発生することは稀であり，教師データを得ることが難しい問題があります．例えば，コンクリートのひび割れや鋼材の腐食などの比較的軽微な損傷についても，正常なデータに比べれば圧倒的に異常データの比率は小さく，これはデータの不均衡と呼ばれています．

損傷データが写った画像データはその特性上公開されることが少なく，また，アノテーション作業をするのも大変です．学習するためのデータを共有する仕組みが重要です．また，時系列データなどで教師データを作成する方法としては，損傷を模擬した実験を行うこと，もしくは臨床試験を行うなどの方法があります．

表には教師あり学習のユースケースとして，目で見える異常（材料表面の異常，構造物の大きな傾斜や変位）に対するユースケースを整理しています．目で見ればわかるという現象が大半ですが，人間による見落としを防ぐこと，また長期間監視するための自動化をする上で AI は役に立つと思われます．表には目で見えない異常（材料内部の欠陥，微少な傾斜，変位，剛性低下）の評価についても整理しました．非破壊検査については既存の技術に対して AI が得意となるものを見つけ出すことが肝要です．

表－4　教師あり学習のユースケースとデータ（見えるものを指標化して監視する）

ユースケース	データ例	実装できそうな事例
画像から変状程度や原因推定および対策施工時期を診断したい	コンクリート表面画像のアノテーション結果	セグメンテーションを利用してひび割れなどの損傷を抽出する
・交通荷重と構造物応答の関係把握と，応答による異常検知をしたい ・過積載車両の自動抽出をしたい	車両通行時の画像とひずみ応答が対応した時系列データ	物体検出による車両種別の特定および時系列解析による固有振動数の推移による異常検知
塗膜劣化や鋼材腐食の進行性把握および塗替え時期の判断	塗膜やさびの状態，腐食量などの判定のついた鋼材画像データ	・セグメンテーションを利用した腐食面積の把握 ・画像認識を利用した耐候性鋼材の腐食度の評価
・雨量と斜面崩壊時期の関係．斜面画像データからの斜面崩壊検知	近傍で雨量計測をしている斜面画像データ（静止画，動画）	気象データと画像解析を併用したハイブリッドな手法
・共振やレインバイブレーションなどによる変位量（振動）評価，強風の影響評価と対策検討（共振振動数） ・構造物の大きな変位や変形量の把握	近傍で風速計測しているケーブルや電柱などの画像データ	物体検出とセグメンテーションなどを併用した粒子解析
舗装のひび割れ，平坦性などの評価，舗装補修時期や箇所の選定	ひび割れ，ポットホール判定のある舗装画像（非走行，走行時），もしくは走行振動データ	画像を利用した深層学習結果と車載加速度データによるハイブリッド手法

表－５　教師あり学習のユースケースとデータ（見えないものを検知しスクリーニング）

ユースケース	データ	実装できそうな事例
打音点検時の音声や画像による浮きの検知手法の検討	浮き判定のある打音時系列，もしくは赤外線画像データ	音声認識による分類
・掘削時の構造物への影響や異常検知．橋脚の応答を利用した洗堀検知	災害前後の実測データ，模型や模擬データなど，掘削量などが既知の構造物応答の時系列データ（振動，変位，傾斜など）	時系列を扱える機械学習手法による異常検知（SVM，LSTMなど）
外力変化による構造物の応答特性の把握と異常予知および検知手法の検討	・水位データが対応できる橋脚振動時系列データ（洗堀検知） ・雨量データが対応できる擁壁や斜面の傾斜データ	外力と応答の時系列予測手法を利用した異常検知
可動不良や剛性変化による構造物の異常検知方法の検討	温度計測を行っている構造物の応答データ（歪，変位，傾斜，加速度）	温度と歪など，波形パターンの評価や異常検知

○教師なし学習

教師なし学習は，既存の大量のデータに対してアノテーションをするコストを要さずに活用できる可能性があります．正常データを学習して異常データを検知する方法など，データの不均衡性にも対応できる手法も開発されています．ここでは，教師なし学習の適用についてユースケースを検討してみましょう．

表－6　教師なし学習のユースケースとデータ（見えるものを指標化して監視する）

ユースケース	データ	実装できそうな事例
・画像データを利用したオペレーション障害検知（車両，列車，河川） ・巡回点検としてのスクリーニング手法の検討	走行時の画像やレーザー＆レーダーのデータ（障害物，建築限界，竜巻など異常気象）	画像認識を利用した画像内の異物や異常変位の特定
河川の氾濫時期と位置を検知する	雨量計データと水位計データ	最適化手法
・災害時の落橋や斜面崩壊，災害前後の変化などの異常検知方法の検討 ・画像データを利用した，サービス阻害要因の検知（降雪検知，路面凍結検知，河川氾濫），第三者被害検知	・道路わきのカメラ画像データ ・護岸，堤防，溜池，砂防ダムなどの監視画像データ ・衛星画像	画像認識を利用した画像内の異物や異常変位の特定
・画像データを利用した点検者の診断支援 ・走行車両など移動体による画像を利用した損傷検知（腐食，漏水，破断）	・ボルトの画像（腐食，弛緩） ・雨樋などの付帯物の画像（晴天および降雨後） ・壁面タイルの画像データ ・鉄塔，電柱などの碍子，電線などの画像データ	・ステレオ計測などを利用した非接触寸法計測 ・画像認識や物体検出による異常箇所の自動抽出

表－7　教師なし学習のユースケースとデータ
（見えないものを指標化して監視する）

ユースケース	データ	実装できそうな事例
構造物異常の予兆検知	構造物応答の時系列データ（変位，変形，加速度） デジタルツインなどを利用したシミュレーションデータ	時系列波形を利用したオートエンコーダーなどの異常検知
移動体を利用した間接的な構造物の異常検知	・列車走行時の車両の応答データ ・路面性状車や車両に搭載したスマホの振動データ	レーダー画像を利用した異常検知
・路面や壁面などの非破壊検査の簡易化	・コンクリートや舗装上から撮影したレーダー画像（空洞） ・太陽光パネルなどの赤外線画像	レーダー画像を利用した異常検知

○マルチモーダルな AI

画像認識により損傷を検知するのみでは診断などの高度な評価はできません．様々データを利用したマルチモーダルな手法を実装することで，より人間が行っているような作業を代替することが可能となります．ニーズの観点からマルチモーダルな手法のユースケースを考えてみましょう．

表－8　マルチモーダルな手法

ユースケース	データ	実装できそうな事例
損傷画像と所見を利用した維持管理ストーリーの作成	画像と点検記録	画像キャプショニング Graph Convolution
避難誘導方法の検討	災害時の車両，列車，人の GPS データ ツイッターデータ	チャットボット
点検結果の入力支援	点検画像データ 点群データ	画像からの点検対象部材の抽出，要素の特定
競合作業の把握，現場監理の効率化，作業員の安全監理（高所作業，毒物環境），作業における阻害要因の把握，交通整理や列車見張りなど保安業務の自動化	現場における作業員の画像やバイタルデータ 現場における保安員や工事車両（重機など）の移動データ（GPS） 迂回車両の GPS データ	RPA
現場準備，作業の事前調整	台帳データと地図情報（図面，点群，GIS） 保安記録データ 議事録データ 資材や資格者の情報	データプラットフォーム
積算基準の見直し，自動積算方法の検討	維持管理予算の推移データ（直接費，経費，機械損料，リース代） 入札情報	チャットボット
作業の平準化，費用対効果の高い工法の選定 材料のイノベーション	材料の品質管理，試験データ 対策工の工法や費用データ 適用事例集	知識モデリング
メンテナンス作業（清掃，簡易補修）への参加度評価	ボランティアの活動内容および作業時間データ メンテナンスへの理解度アンケートデータ	ブロックチェーン

2.1.8. 対話システムと文章生成

○自然言語処理の目的

自然言語処理の目的としては，機械翻訳，文書要約，文書分類，対話（質問応答）などがあります．近年ブームになっている深層学習では，入力と出力のデータがたくさん必要となりますが，機械翻訳を考えれば，ある言語を別の言語で言い替えた入力と出力データの対が多くできることがわかると思います．それに対して，文書要約などでは，ある文書を要約したデータというのはそれほど多くないと思われますので，深層学習を利用するためには一工夫必要となるでしょう．次に，対話システムについて考えると，近年は SNS やツイッターなど，誰かのつぶやきに対する応答データが多くあることがわかります．このようなつぶやきと回答のペアデータを利用した対話モデルなどが深層学習を利用して開発され，それらはアップルの Siri やグーグルやアマゾンなどのスピーカー機能などとして実用化されています．

○対話システムの目的は

それでは，グーグルやアマゾンなどの対話システムをそのまま専門的な知識を得るための対話システムとして利用することができるでしょうか？

まず，みなさんはどのような質問をスピーカーにしているでしょうか？例えば，天気を聞くというようなことがあるかと思います．このような天気を知りたいなどの明確な目的がある対話システムはタスク指向対話システムと呼ばれています．天気を知りたいだけであれば，深層学習などを利用しなくても，天気を知りたい場所や時間などの特定の条件を入力して，天気予報などの情報を検索して回答することができると考えられますが，このような特定のルールに対して回答を作るルールベース方式の対話システムが検討されてきました．しかし，対話は天気の質問だけとは限りません，色々な質問に対して，何か答えを作る対話システムは，非タスク指向対話システムと言われますが，これはいわゆるチャットボットと呼ばれているものです．このような特定なタスクを伴わない対話にも，人間にとっての癒しなどの効果があると考えられ多く検討されています．

また，対話を実現する際に，なんでもかんでも答えるような対話システムを作るためには，特定のドメインに特化しないデータ（open domain）を学習する必要がありますが，特定の対話にだけ答えるような FAQ システムなどでは，適用分野と用途を絞り込んだ対話システ

ムとして，絞り込んだ専門分野や，ある特定の領域のデータ（Closed domain）の学習を行うような対話システムが考えられます．

土木の分野で期待されている対話システムというものはどのようなものでしょうか．

○土木に必要な対話システムとは

我々は専門的な情報を得るためには，各種の基準を参照することをします．専門的な知識を持っているということは，必要な情報がどこに書いてあるかを探せる能力であるともいえます．しかし，近年では，必要な情報は多く散在し，また個人や組織のやるべきタスクは特定されず，様々なものをこなす必要があります．このため，いわゆるエキスパートシステムのような対話システムが期待されています．用途を絞ることで，より専門的な対話を実現しようとするものがエキスパートシステムと言えますが，従来のエキスパートシステムはルールベースのものであり，基本的に予め用意した対話のシナリオ（スクリプト）等を利用して動作するものです．近年の深層学習の進展を考えれば，用例ベースや生成ベースの対話システムを考えてみることがよいと思われます．また土木の業務の遂行においては，技術的な知識だけではなく，受発注を効率化するための情報や，部外協議などの業務を円滑に進めるような知識も重要となります．また，施工や品質の管理などでは，必要な情報をあるべきところに入れておき，必要になったら取り出すような仕組みが重要となると考えられます．これらのエビデンスベースの情報管理においても，チャットボットのような仕組みは重要となると考えられます．

○対話システムにおけるモデルと学習方法

言葉を理解するとはどのようなことでしょうか？深層学習を利用しない手法では，例えば，品詞，単語分割，語義曖昧性解消，固有表現抽出，構文解析，述語項構造認識などの手順をモデル化する手法が検討されてきました．このような自然言語処理の手法はかなり敷居の高い分析だったと言えるでしょう．それに対して，深層学習では，入出力データをニューラルネットワークでつなげればよいと考えることができます．

さてニューラルネットワークはどのように賢くなっていくのでしょうか．ニューラルモデルを利用した自然言語処理手法をして，BERT（Bidirectional Encoder Representations from Transformers)というのがあります．BERT でできることは，文章の感情分類，質問応

答，文章から人名，組織名，地名などを抽出する固有表現抽出など様々ですが，教師なしデータで事前学習を行うのが特徴となっています．事前学習では，単語穴埋め問題と文の連続判定問題というのを行っています．文章に対して，ランダムに穴埋め問題を作って，入力と出力（答え）を学習していくことでニューラルネットワークを賢くしていきます．また，実際に連続した文，実際には連続していない二つの文を用意して，それらを学習して正答率を上げていくことで，あたかも AI が文脈を理解しているかのように賢くなっていきます．このような事前学習をしたモデルを目的に応じてファインチューニングすることで対話を実現します．

次に，ニューラルネットワークの構造について考えてみましょう．文章は単語のつながりが重要になることは明白です．また，単語の共起というものが大事になります．同じ文脈で出てくる単語は関連があると言え，これらは関連がある単語としてベクトル化されます．入力と出力を繋ぐ際に単語の順番を考慮するためのモデルは RNN と言われ，時系列的な依存関係を隠れ層に持ったニューラルネットワークですが，このようなニューラルネットワークを上手に学習することができるようになり，機械翻訳の精度が上がったことが知られています．さらに長期的な記憶のような機構を入れるために LSTM といったモデルが提案されています．「Attention Is All You Need(Transformer)」という論文以降は，attention 機構「入力されたベクトルによって重みを計算し，その重みに基づいて索引ベクトルから任意のベクトルを出力するモデル」を利用した手法が構築され，Transformer モデルとして開発され，さらに効率よく学習する Reformer モデルなどが開発され精度が向上しています．

2.1.9. Google Colab の使い方

2.1.9 からは pchun.work で構築しているチュートリアルを示します．ただ，Google Colab は仕様が頻繁に微修正されるため，そのままでは使えない可能性もある点にはご留意ください．ただ，一般的にはかんたんな修正で動くようになります．

a) Google Colab とは？

機械学習，Deep learning などの AI 関係のプログラムは基本的に Python をベースに書かれることが非常に多いです．ただ，Python のインストールは結構面倒くさいです．そして，Deep learning は GPU がないと計算時間が非常に長くなりますが，GPU が機能するようにインストールするのは更に面倒くさい．本気の解析をするならともかく，練習用途にはこれらのインストールは少々ハードルが高い．

そこで，Google Colab です．Google Colab は，Google アカウントさえあれば計算環境が手軽に試せるサービスです．また，GPU 環境も利用可能で，Keras や Chainer といった Deep learning のフレームワークも簡単に使えます．（Keras や Chainer も，インストール結構たいへんなんですよ…）

このように素晴らしいサービスですが，とりあえず使用制限もあるので，以下に注意点をまとめておきました．気をつけておいてください．

注意点：

・90 分間何もしないと，接続が切れてしまう．

・GPU は 12 時間以上は連続で使えない．

b) Google Colab で検索

図－１８　Google Colab の検索結果

Google Chrome を立ち上げ，”Google Colab”で検索します．そうすると，上図のような検索結果になるので，Google Colab というリンクをクリックします．（Google アカウントにログインしていることが必要です）

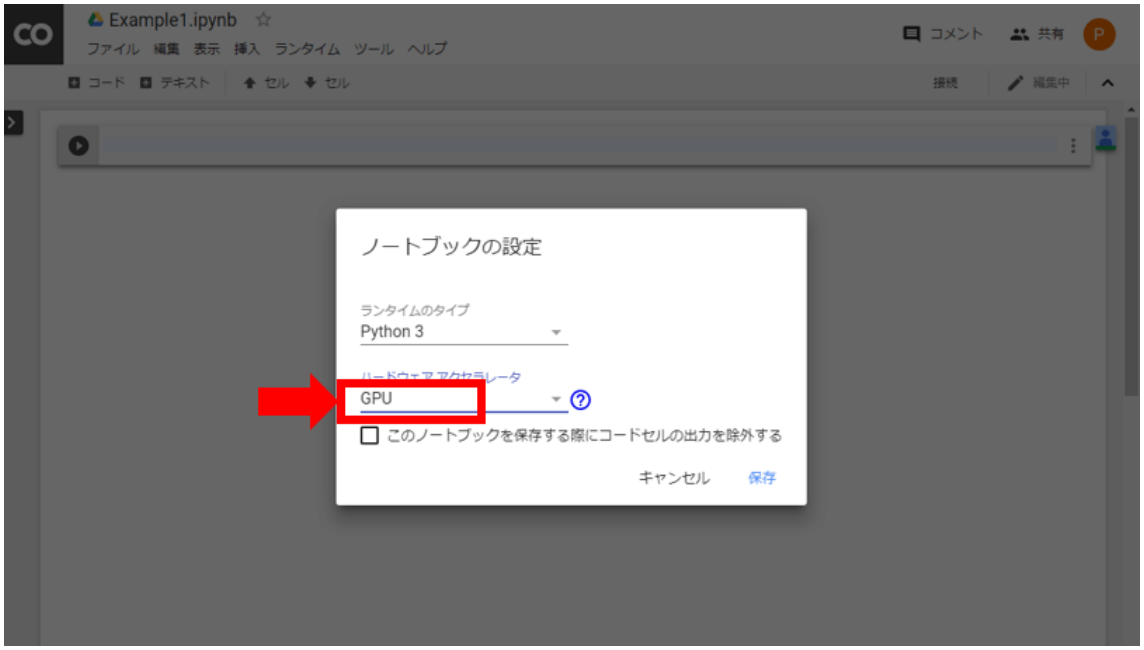
c）ノートブックの作成

検索結果をクリックすると，図－１９のような画面が出てきます．そこで，”ノートブックを新規作成”というリンクをクリックします．そうすると，図－２０のようなエディタが立ち上がります．最初は例えば Untitled0.ipynb というファイル名になっているので，赤い四角で囲った部分をクリックして，ファイル名を変更しましょう．ここでは，Example1.ipynb と仮にしてみます．

図－１９　Google Colab の新しいノートブック作成

図－２０　新しいエディタ

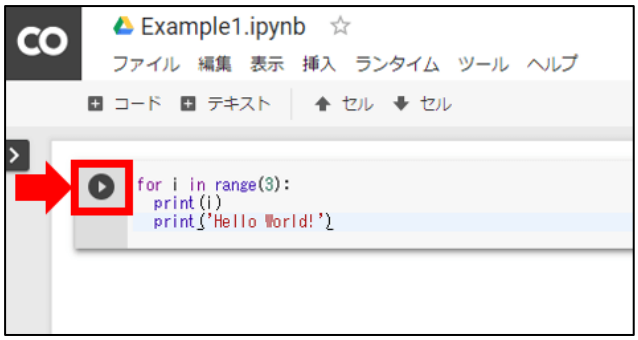
そして，ランタイムというタブをクリックして，「ランタイムのタイプを変更」というのを見てみます．そうすると，図－２１のような画面が出てきます．ここで，ハードウェアアクセラレータのところをクリックしてみると，GPU を使うかどうかの選択ができます．特に Deep learning のプログラムを動かすときは，ここを GPU に設定します．このように作成したノートブックを使って，AI プログラミングの練習をしていきます．



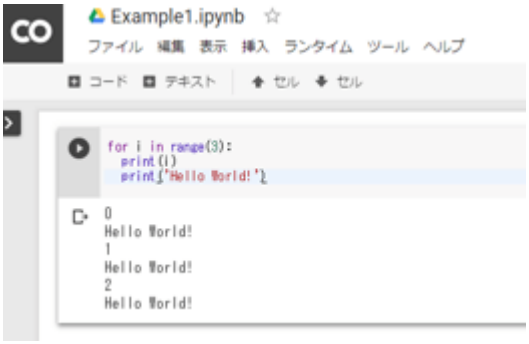
図－２１　ランタイムの設定

d）サンプルプログラムの作成→実行→保存

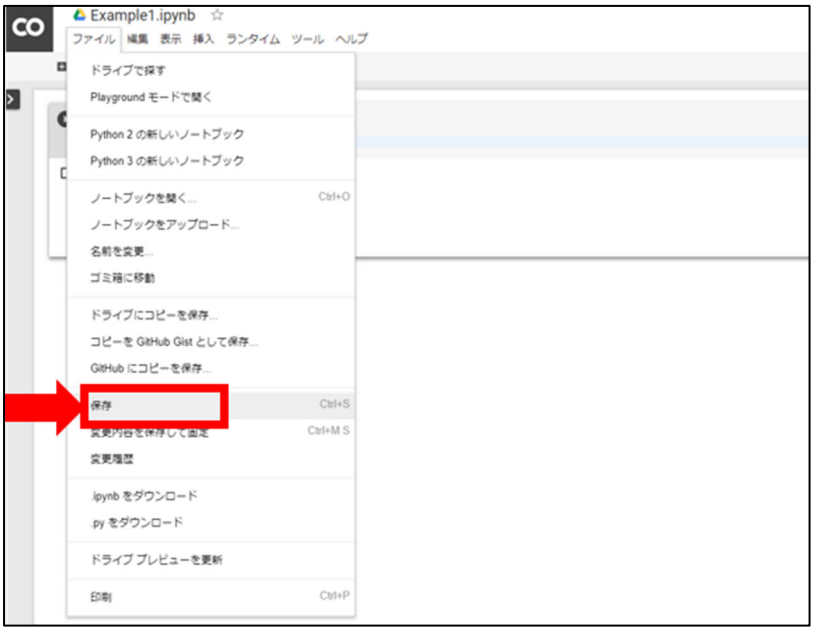
ここではまず図－２２のような非常に単純なプログラムを作成してみます．for 文を使った繰り返し構文です．これを実行するには，赤い四角で囲んだ再生ボタンを押すか，Shift＋Enter キーです．そうすると，図－２３のように結果が出力されます．これを保存するには，左上のファイルというタブをクリックします．



図－２２　サンプルプログラムの例



図－２３　サンプルプログラムの出力結果



図－２４　プログラムの保存

これを保存するには，左上のファイルというタブをクリックします（図－２４）．そうすると，保存ができ，次回以降も継続して作業することが可能になります．

2.1.10. ロジスティック回帰によるデータ分類

機械学習に取り組む前段階として，まずはロジスティック回帰を行ってみます．

a）データの説明

ここでの練習用に，データファイルを作りました．

→http://pchun.work/wp-content/uploads/2018/07/binarize.csv

また，以下のソースコードの全体像(jupyter notebook 形式）と，データをまとめたファイルを圧縮して以下に置いておきます．

→http://pchun.work/wp-content/uploads/2018/07/Example2.zip

特になんということもないデータです．完全に架空のデータです．図－２５に先頭５つの例を示します．左から２つの列は説明変数，一番右の列は被説明変数というイメージで，0 か 1 かの 2 クラスに分かれています．

5.1	1.5	0
4.2	3.2	1
0.7	0.9	0
1.2	0.7	0
8.8	1.7	1
以下，合計200行		

図－２５　サンプルデータの例

b）データを活用した解析

ここでやりたいことは，説明変数２つから，クラスが 0 か 1 かを当てたい，というものです．やり方は色々とあり，機械学習手法で次回以降解いていきたいと思いますが，まずは歴史のあるロジスティック回帰を用いて行う，ということです．

以下，サンプルコードです．

```
# -*- coding: utf-8 -*-
#上のコメントは，普通pythonで日本語を使うために必須．ただ，Google Colaboratoryでは無くてもOK?
import pandas as pd #pandasはデータを扱うためのライブラリ
import numpy as np #numpyは数値計算を効率的に行うための拡張
import matplotlib.pyplot as plt #matplotlibはグラフ描画用ライブラリ
```

基本的にコメントをつけた通りですが、改めて。

最初の文章はおまじないのようなもので、これが無いと一般的には python を動かす際に、日本語コメントが入っているとエラーを吐き出します。でも、Google Colaboratory では内部で処理してくれているようで無くても大丈夫なのですが、一応つけています。3 行目からは、ライブラリを使うための宣言です。Pandas はデータを扱うための、numpy は数値計算を効率的に行うための拡張です。両方とも（特に numpy は)、今後多くのプログラムで利用します。matplotlib も同様に頻出のライブラリです。これらについてもおまじないのように書くのがいいです。

```
#Google Colaboratoryのみに必要．既存ファイルを扱うのだけは，Google Colaboratoryは不便です．
#ファイルをアップロードするためのダイアログを出します．
from google.colab import files
uploaded = files.upload()
```

上のコードは、Google Colaboratory の場合にのみ必要なワンステップです。サーバー上で解析を行うため、ローカルのファイルを読み込むことができません。対策は色々とあり、Google Drive にアップする方法などもあって場合によってはそちらが便利なのですが（というか、そっちにしないとできない場合もある）、今回の binarize.csv のような小さなファイルを扱う場合にはこのくらいでも十分です。

上のコードを実行するとファイルを読み込むダイアログが出てきます。それに従いファイルを読み込むと、以下のような感じになります。

ダイアログを使って読み込んだのは、あくまでもシステムでファイルを扱えるように読み込んだものです。このプログラムの中では読み込めていません。このプログラムで使えるように、さっき最上部で pandas as pd と書いて pd として import した pandas を使って、データを読み込みます。pandas には read_csv という関数があり、それを使って binarize.csv を読み込みます。

そして、左から 2 列を説明変数として、X に代入します。python は数え方が 0 からスタートするため、0:2 列目を読み込め、と書いています。python では、0:2 というのは、0 からスタートして 2 になるまで、なので、0 列目と 1 列目という意味です。そして、1 番右の列を被説明変数として Y に読み込みます。以下にソースコードを示します．

```
df = pd.read_csv('binarize.csv', header=None)#Google Colaboratoryにアップしたファイルを，このコ
data1 = df.values #pandasの形式からnumpyの形式への変換
X = data1[:,0:2] #説明変数として，0列目と1列めを使う．0:2というのは，0からスタートして2になるまで，なので
y = data1[:,2] #被説明変数として，2列目を使う．
```

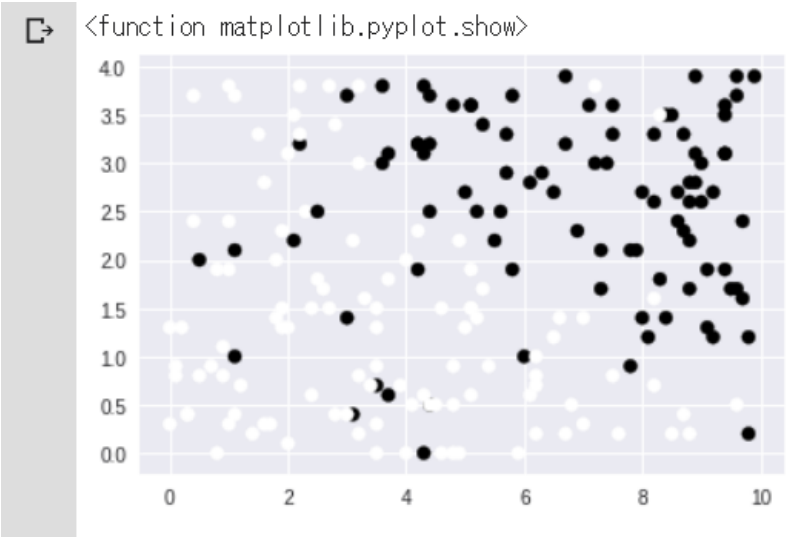
ここは無くてもいいのですが、ちゃんと作れているか表示しています。最初の 5 行くらい。また，最後の print 文は，行列の形を調べています．実行すると，以下のような結果が得られます．

```
#しっかりと思ったとおりのデータ形式になっているかチェック．最初の5行くらいのみを，表示の都合上チェックするよう
print(data1[0:5,:])
print(X[0:5,:])
print(y[0:5])
print(data1.shape)
```

```
[[5.1 1.5 0. ]
[4.2 3.2 1. ]
[0.7 0.9 0. ]
[1.2 0.7 0. ]
[8.8 1.7 1. ]]
[[5.1 1.5]
[4.2 3.2]
[0.7 0.9]
[1.2 0.7]
[8.8 1.7]]
[0. 1. 0. 0. 1.]
(200, 3)
```

最初に示したテーブル通りのデータになっていることがわかると思います。

また，説明変数が 2 つなので、それぞれを横軸、縦軸として、それらとクラスの分布についてプロットしてみました。こんな感じの図になっています．意図的に作ったデータだけあって、まあまあ分かりやすく分布しているものの、ちらちらと例外データもあったりします。



図－２５　クラス分布の matplotlib による表示

また，以下のソースコードにより，sklearn.model_selection から，train_test_split という関数を読み込んでいます．sklearn は scikit-learn といって，Python の機械学習用ライブラリです．こういう from … import …みたいな形は，普通はプログラムの最初に書くんですが，今回は説明用に，関数を使う場所の近所に書きました．X, y を，それぞれ，学習用 8 割，テスト用 2 割，といった分け方をしています．そして，下 2 行の print 文で，本当に 8 割，2 割になっているか確認しています．

```
from sklearn.model_selection import train_test_split #係数決定のためのデータと，テストのためのデー

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2) #20%をテスト用に.
print('train size:', X_train.shape[0])
print('test size:', X_test.shape[0])
```

次は，sklearn から，ロジスティック回帰を行うための関数 LogisticRegression のインポートを以下のソースコードにより行っています．次の行とその次の行で，X_train と y_train という入力と出力の組み合わせを与えることで，それにもっともマッチする係数を出すことができます．lr.fit というのがポイントです．下 2 行は，それぞれ切片と係数を確かめています．（なお，train_test_split はランダムで分割するので，以下の結果はランダムで変わります）

```
from sklearn.linear_model import LogisticRegression #ロジスティック回帰を行うための関数


lr = LogisticRegression()
lr.fit(X_train, y_train) #ここで，計算を行っている.
print (lr.intercept_) #計算して得られた切片
print (lr.coef_) #計算して得られた係数
```

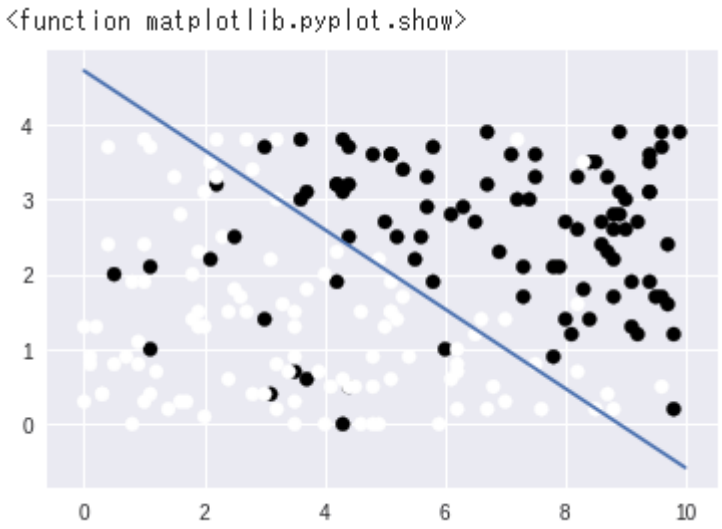
結果：

```
[-3.25466321]
[[0.36590136 0.68987314]]
```

以下のソースコードにより，求まった係数を w_0, w_1, w_2 に代入します．ちなみに，w_0+w_1*X1+w_2*X2 = 0 という関係があります．その線をもとにクラスが分かれます．

そして，下半分はプロットのための部分です．np.arange(0,10,0.01)で，0 から 10 まで 0.01 刻みで xx1 を設定します．そして，xx2 = (-w_1 * xx1 – w_0)/w_2 という関係のプロットを描いて，上の分布図と合わせます．それが以下の結果ですが，それなりにまっとうなところで分割できている様子がわかります．

```
w_0 = lr.intercept_[0]
w_1 = lr.coef_[0,0]
w_2 = lr.coef_[0,1]
print(w_0)
print(w_1)
print(w_2)

xx1 = np.arange(0,10,0.01)
xx2 = (-w_1 * xx1 - w_0)/w_2
plt.plot(xx1,xx2,'-')
plt.scatter(X1,X2,c=y)
plt.show
```



図－２６　分布図と関係式を合わせた図

以下のソースコードでは，テスト用に確保してあった X_test を，先程作ったモデルに入れたらどうなるかということを確認しています．lr.predict とすれば，予測をしてくれます．そして，テスト用データの正解と比較した正解率を，lr.score で出してくれます．

```
print(lr.predict(X_test)) #テスト用データを，上で得られた式に入れた結果
print(y_test)#テスト用データの正解
print(lr.score(X_test,y_test))#何割正解したか？
```

とはいえ，後ろ 2 割を引っ張ってきても全体が合っているかどうかこころもとないので，以下のソースコードでは交差検証を行っています．交差検証とは，統計学において標本データを分割し、その一部をまず解析して、残る部分でその解析のテストを行い、解析自身の妥当性の検証・確認に当てる手法を指します．特に，K 分割交差検証というのをここではやりました．StratifiedKFold というのはその関数です．ここでは K=8 としています．つまり，標本群を 8 個に分割し，そのうち 7 つを用いて学習を行い，そして残り 1 つをテストデータとして精度を評価する，というのをテストデータを変えながら 8 回繰り返しています．ソースコードの下に表示された結果を示しています．

```
#最後の2割だけでは物足りないので，交差検証をして，全体的にどうなっているか調べてみる．

from sklearn.model_selection import StratifiedKFold, cross_val_score #StratifiedKFoldはk
stratifiedkfold = StratifiedKFold(n_splits=8) #K=8分割
scores = cross_val_score(lr, X, y, cv=stratifiedkfold)

# 各分割におけるスコア
print('Cross-Validation scores: {}'.format(scores))
# スコアの平均値
print('Average score: {}'.format(np.mean(scores)))
```

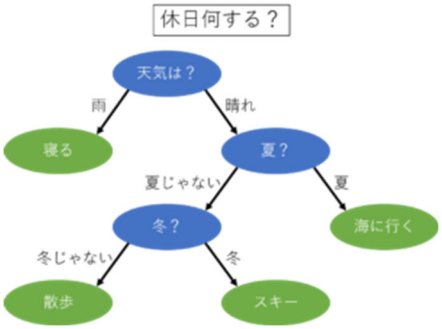
```
Cross-Validation scores: [0.84615385 0.80769231 0.76923077 0.8 0.84 0.875
0.95833333 0.79166667]
Average score: 0.8360096153846154
```

最初の行列は 8 回それぞれの正解率，最後はその平均です．だいたい 83.6%の正解率であることがわかります．

2.1.11. 決定木とランダムフォレストによるデータ分類

a）決定木とは

決定木とは，その名の通り木のような形をしているアルゴリズムです．例として，下の図－２７を見るとイメージが湧きやすいかと思います．



図－２７　休日に何をするかを決める決定木

上に示した決定木は，休日に何をするかを決める決定木の例です．決定木では，いろいろな選択肢（基準）のもとに，最終的に答えが決まります．

選択肢（基準）を決める方法としては，CART 法というアルゴリズムが多く使われています．CART 法では，最もよくデータを分類する基準を探して分岐を設定します．数学的にデータを最もよく分類する基準としてはジニ係数が多く使われます．ジニ係数の詳細はここでは省略しますが（各種教科書には記載されています），この値がより低くなるように設定することで，最もよくデータが分類できます．

それでは，実際に決定木を用いた分類を，コードを見ながら確認していきます．

b）データの説明

今回は機械学習用データセットとしてよく使われるアヤメのデータセットを用います．以下の表にその一部を示します．150 件のデータが Setosa, Versicolor, Virginica の 3 品種に分類されており，それぞれ，Sepal Length（がく片の長さ）, Sepal Width（がく片の幅）, Petal Length（花びらの長さ）, Petal Width（花びらの幅）の 4 つの特徴量を有しています．今回は，これらの特徴量から品種を推定・分類することを目指します．

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
		・		
		・		
		・		

c）サンプルコード（決定木）

本節では，決定木を用いたサンプルコードを説明します．Python の機械学習用ライブラリのひとつである scikit-learn を用いて，決定木による分類を行います．scikit-learn にはすでにアヤメのデータセットが用意されています．1 行目のコードでアヤメのデータセットをインポートしています．データセットのうち，いくつかは学習に用いて，残りは精度の評価に用います．2 行目のコードでデータセットを分けるための関数をインポートしています．3 行目のコードは，scikit-learn で決定木を扱うための関数，tree をインポートしています．

```
1 #-*- coding: utf-8 -*-
2 from sklearn.datasets import load_iris
3 from sklearn.model_selection import train_test_split
4 from sklearn import tree
```

まず，変数 iris に，インポートしたアヤメのデータセットを代入しています．さらに，代入したデータの中から，特徴量を示すデータ iris.data を変数 X に代入します．また，品種を示すデータ iris.target を変数 y に代入します．

```
iris = load_iris()
X = iris.data
y = iris.target
```

ここで，変数 X，y の中身を以下のソースコードで確認してみましょう．

```
print (X[0:5,:])
print(y)
```

X は中身が多いので最初の 5 行だけ表示します．このプログラムを実行すると，以下のような結果が得られます．

```
[[5.1 3.5 1.4 0.2]
[4.9 3. 1.4 0.2]
[4.7 3.2 1.3 0.2]
[4.6 3.1 1.5 0.2]
[5. 3.6 1.4 0.2]]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

X の中身は，最初に示した特徴量の表と同じ値になっていることがわかります．y の中身は 0，1，2 が表示されています．それぞれの数値が Setosa, Versicolor, Virginica の品種を表しています．

test_size の指定により，X と y の中身を，学習用 7 割，テスト用 3 割に分けます．

random_state はデータを分割する際の乱数のシード値で，こうすることで誰がやっても毎回同じように分類されます．

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

clf に，今回学習に用いる決定木を定義します．ここで，決定木の深さを max_depth で定義しています．この値が小さすぎるとうまく判別できなかったりしますが，深さが大きすぎても，今度はトレーニングデータに過剰に適合した過学習に陥る場合があるので注意が必要です．つづいて，clf.fit で学習を開始することで，トレーニングデータに適した判別条件が設定されます．

```
clf = tree.DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)
```

学習した決定木を用いて，テストデータの判定を行います．判定した結果の正解率を表示すると以下のようになります．

```
print (clf.score(X_test, y_test))
```

```
0.9777777777777777
```

だいたい 98%の正解率であることがわかります．せっかくなので，ほんとにそうなっているかの確認も兼ねて判定結果の中身も見てみましょう．まずは，判定に用いるテストデータ（今回は 150 個うちの 3 割なので 45 個）の中身を確認します．

```
print(y_test)
```

上のコードを実行すると，以下のように表示されました．

```
2102020111211110110021002001102102210111202
00
```

つぎに，学習させた決定木の判定結果を見てみましょう．

```
print(clf.predict(X_test))
```

判定を行う場合には predict を用います．さて結果は…

```
2102020111211110110021002001102102
21021120200
```

ほとんどすべて正解していますが，38 個目の 1（Versicolor）だけ 2（Virginica）だと誤判

定しています．きっと，どちらともつかない微妙な特徴量だったんでしょう．45 個中 44 個正解しているので，確かに正解率は約 98%ですね．
ここで，アヤメのデータセットに対して全体的にどの程度の正解率を出すことができるのか確認するため，K 分割交差検証を行います．今回は K=10 として，データを 10 分割してそれぞれの結果を見てみます．

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np
stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)
print('Cross-Validation scores: {}'.format(scores))   # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

StratifiedKFold は K 分割するための，cross_val_score は分割したデータで精度を検証するための関数です．さて，結果は以下のようになりました．

```
Cross-Validation scores: [1.        0.93333333 1.        0.93333333 0.93333333
0.93333333 0.93333333 0.93333333 1.        1.        ]

Average score: 0.96
```

交差検証の結果によると，平均して 96%の正解率を出すことができています．

d）ランダムフォレストとは

ランダムフォレストでは複数の決定木を用いて学習を行います．そのとき，元の学習データからランダムにデータを選択し，個々の決定木を生成します．そして，実際に未知のデータを評価する際には，個々の決定木が下した結論を多数決することで全体の結論とします．決定木はとても分かりやすいアルゴリズムですが，データによっては望ましい木構造が生成できなかったり，過学習を起こしやすかったりすることで知られています．一方，ランダムフォレストはその特性上個々の決定木間の相関が低くなるので，決定木に比べて過学習の影響を非常に小さくできるメリットがあります．以下に，上記で用いたアヤメのデータセットをランダムフォレストで解析した場合のサンプルコードを示します．

e)サンプルコード（ランダムフォレスト）

基本的な流れは決定木と全く同じなので，異なる箇所のみ解説します．

```
#-*- coding: utf-8 -*-
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

今回はランダムフォレストを用いるので関数 RandomForestClassifier をインポートしています．

```
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf = RandomForestClassifier(max_depth=3, random_state=0)
clf.fit(X_train, y_train)
```

ここではもちろんランダムフォレストの関数を用いています．ランダムフォレストで学習に使うデータは学習データ全体の中からランダムに選定されますが，random_state=0 によって，誰がやっても毎回同じように選定されます．このとき，n_estimators を指定すれば木の本数も変えられますが，今回はデフォルトのまま行います．ちなみにデフォルトの本数は 10 本です．

```
print (clf.score(X_test, y_test))
```

ランダムフォレストでの学習結果を用いて，テストデータの判定を行います．判定結果の正解率は以下のようになります．

```
0.9333333333333333
```

だいたい 93%の正解率であることがわかります．ここで，決定木の場合と同様に，K=10 として，K 分割交差検証を行います．

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np
stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)
print('Cross-Validation scores: {}'.format(scores))    # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

結果は以下のようになりました．

```
Cross-Validation scores: [1.        0.93333333
1.        0.93333333 0.93333333
0.93333333 0.8       0.93333333 1.        1.        ]

Average score: 0.9466666666666667
```

交差検証の結果によると，平均しておよそ 95%の正解率を出すことができています．

g）コード全文（決定木）

このページで紹介した決定木のコード全文を載せておきます．なお，関数などの呼び出しはコードの先頭にまとめ，途中で行った確認用の print 文などは省いています．

```python
#-*- coding: utf-8 -*-
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf = tree.DecisionTreeClassifier(max_depth=3)
clf.fit(X_train, y_train)

print (clf.score(X_test, y_test)) #正解率の表示

#K分割交差検証
stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)
print('Cross-Validation scores: {}'.format(scores))   # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

g) コード全文（ランダムフォレスト）

このページで紹介したランダムフォレストのコード全文を載せておきます．こちらも諸々の呼び出しはコードの先頭にまとめています．

```python
#-*- coding: utf-8 -*-
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf = RandomForestClassifier(max_depth=3, random_state=0)
clf.fit(X_train, y_train)

print (clf.score(X_test, y_test))#正解率の表示

#K分割交差検証
stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)
print('Cross-Validation scores: {}'.format(scores))   # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

2.1.12. ニューラルネットワークによるデータ分類

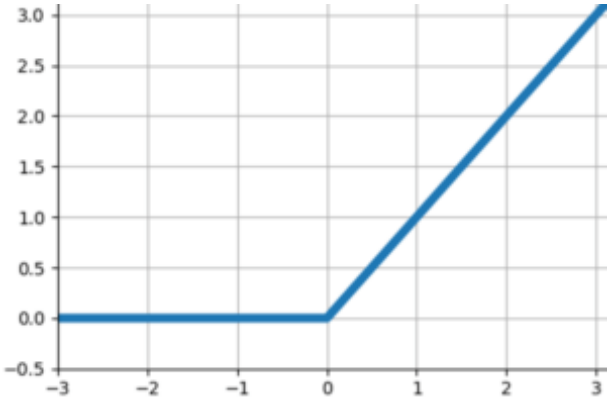
a) ニューラルネットワークとは

ニューラルネットワークは，近年流行りのディープラーニングの基礎となる手法です．下図のようにいくつかのニューロンというものを結合してネットワークを構成したものをニューラルネットワークといい，入力した値に応じて何らかの出力が得られます．

図－２８　ニューラルネットワークの概念図

各ニューロンに何らかの数値が入力されると，それに応じて新たな数値が次の層の各ニューロンに入力されます．最終的に出力層のニューロンが出力した数値が，そのニューラルネットワークが導いた答えとなります．

もう少し詳しく説明すると，各ニューロンに入力された値は，その時点でまず活性化関数とよばれる関数に代入され，解を出力します．活性化関数にも色々あるのですが，例えばReLU 関数（ランプ関数）であれば，0 未満の数値が入力されると 0 を，0 以上の数値が入力されるとその値を出力します．



図－２９　ReLU 関数

また，活性化関数を通して出力された値を次のニューロンに渡すときに，何らかの定数を乗じてから渡します．これは重みなどと言われます．そして，活性化関数と重みがきちんと設

定されていれば，最初の入力に応じて，それに応じた出力が得られるという仕組みです．重みの設定においては，予め正解が分かっているデータを用意して，出力値と正解値を比較し，その誤差に応じて各重みが調整されます（バックプロパゲーション）．

以上がニューラルネットワークの簡単な説明です．それでは実際にニューラルネットワークを用いた分類を，コードを見ながら確認していきます．

b）データの説明

前回に引き続き，機械学習用データセットとしてよく使われるアヤメのデータセットを用います．再掲になりますが，以下の表にその一部を示します．150 件のデータが Setosa, Versicolor, Virginica の 3 品種に分類されており，それぞれ，Sepal Length（がく片の長さ），Sepal Width（がく片の幅），Petal Length（花びらの長さ），Petal Width（花びらの幅）の 4 つの特徴量を有しています．これらの特徴量から，品種を推定・分類することを目指します．

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
		・		
		・		
		・		

c）サンプルコード

以下，サンプルコードです．基本的な内容のみ触れるため，全体の流れは前回示した決定木による分類とほとんど同じです．単に使う道具が変わった結果としてコードがほんの少し変わるだけなので，困ることはないと思います．また，ページ下部にコード全文を再度載せておきます．

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from sklearn.neural_network import MLPClassifier
```

Python の機械学習用ライブラリのひとつである scikit-learn を用いて，ニューラルネットワークによる分類を行います．scikit-learn にはすでにアヤメのデータセットが用意されています．1 行目のコードでアヤメのデータセットをインポートしています．データセットのうち，いくつかは学習に用いて，残りは精度の評価に用います．2 行目のコードでデータセットを分けるための関数をインポートしています．3 行目のコードは，scikit-learn でニューラルネットワークを扱うための関数をインポートしています．

```
1 iris = load_iris()
2 X = iris.data
3 y = iris.target
```

まず，変数 iris に，インポートしたアヤメのデータセットを代入します．さらに，代入したデータの中から，特徴量を示すデータ iris.data を変数 X に代入します．また，品種を示すデータ iris.target を変数 y に代入します．ここで，変数 X，y の中身を確認してみます．

```
print (X[0:5,:])
print(y)
```

X は中身が多いので最初の 5 行だけ表示します．実行すると，以下のような結果が得られます．

```
[[5.1 3.5 1.4 0.2]

[4.9 3.  1.4 0.2] [4.7 3.2 1.3 0.2] [4.6 3.1 1.5 0.2]
[5.  3.6 1.4 0.2]]

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

X の中身は，最初に示した特徴量の表と同じ値になっていることがわかります．y の中身は 0，1，2 が表示されています．それぞれの数値が Setosa, Versicolor, Virginica の品種を表しています．

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

test_size の指定により，X と y の中身を，学習用 7 割，テスト用 3 割に分けます．random_state はデータを分割する際の乱数のシード値です．

```
clf = MLPClassifier(max_iter=10000)
clf.fit(X_train, y_train)
```

clf に，今回学習に用いるニューラルネットワークを定義します．ここで，一回の学習を行う際に，一気にすべてのデータを使わず，一部をランダムに使って学習を行い，これが何回も繰り返されます．これを何回繰り返すのかを max_iter で定義しています．この値が小さすぎると解が収束する前に学習が終わってしまいますし，大きすぎるとトレーニングデータに過剰に適合した過学習に陥る場合があるので注意が必要です．また，このときに引数として activation を設定することで活性化関数を指定できます．デフォルトでは ReLU 関数（ランプ関数）になっています．

つづいて，clf.fit で学習を開始することで，トレーニングデータに適した重みになるよう

に調整が行われます.

```
print (clf.score(X_test, y_test))
```

学習したニューラルネットワークを用いて，テストデータの判定を行います．判定した結果を print 関数で表示した結果が以下のようになります.

```
0.9777777777777777
```

だいたい 98%の正解率であることがわかります．ここで，アヤメのデータセットに対して全体的にどの程度の正解率を出すことができるのか確認するため，K 分割交差検証を行います．今回は K=10 として，データを 10 分割してそれぞれの結果を見てみます.

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np

stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)

print('Cross-Validation scores: {}'.format(scores))    # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

結果は以下のようになりました.

```
Cross-Validation scores: [1.        1.        1.        1.
0.86666667 1. 0.93333333 1.        1.        1.        ]

Average score: 0.9800000000000001
```

交差検証の結果からしても，平均して 98%の正解率を出すことができています.

d）コード全体（ニューラルネットワーク）

このページで紹介したニューラルネットワークのコード全文を載せておきます．関数などの呼び出しはコードの先頭にまとめ, 途中で行った確認用の print 文などは省いています.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedKFold, cross_val_score
import numpy as np

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf = MLPClassifier(max_iter=10000)
clf.fit(X_train, y_train)

print (clf.score(X_test, y_test)) #正解率の表示

#K分割交差検証
stratifiedkfold = StratifiedKFold(n_splits=10)  #K=10分割
scores = cross_val_score(clf, X, y, cv=stratifiedkfold)
print('Cross-Validation scores: {}'.format(scores))   # 各分割におけるスコア
print('Average score: {}'.format(np.mean(scores)))  # スコアの平均値
```

2.1.13. Keras で MNIST の判定

a) Keras とは

Keras はディープラーニングのプログラムを比較的簡単に実装するためのフレームワークです．Keras の他にも， PyTorch などの似たフレームワークもいくつかあります．

そもそも，機械学習を行うためのライブラリとして Google が提供している TensorFlow というライブラリがあるのですが，機能が豊富すぎてこれからディープラーニングを始める人にとっては少し複雑すぎるという欠点があります．ですが，Keras を使うと TensorFlow を簡単に動かすことができます．実際にどのように使うのかは，この記事で追って説明します．Keras や TensorFrow のインストールはちょっと面倒くさいので Google Colab を使うとすぐに Keras が使えます．

今回は，Keras を用いてニューラルネットワークを構築して学習を行います．

b) MNIST とは

今回は，MNIST（エムニスト）というデータセットを使います．MNIST は手書きの 0~9 の数字が書かれた 28×28 画素の白黒画像データで，60000 枚のトレーニングデータと 10000 枚のテストデータが用意されています．画像それぞれに，手書きの数字に対応した正解ラベルが割り振られています．以下の画像は MNIST の中身の一例です．

図－３０　MNIST の画像例

c) サンプルコード

以下，サンプルコードです．前回のニューラルネットワークは非常に簡素なものでしたが，今回はもう少ししっかりしたものを構築するので，ちょっと長い解説になると思います．また，このページの下部にコード全文を掲載します．

```
1 import keras
2 from keras.datasets import mnist
```

まずは Keras のインポートです．そして，MNIST のデータセットもインポートします．Keras には元から MNIST のデータセットが用意されています．

```
1 (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

そしてこれまでと同様，MNIST のデータをトレーニングデータとテストデータに分割します．元から 6:1 に分けられているので，ここで数値を指定しなくても勝手に分けられま

す．x_で始まるのが画像データで，y_で始まるのがそれに対応する 0~9 のラベルです．

次に画像データをニューラルネットワークに入力出来る形にしていきます．トレーニングデータとテストデータそれぞれに同様の処理を施しています．

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
```

上のコードでは，28×28 の 2 次元配列を，784×1 の 1 次元配列に変換しています．

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

また，配列内の数字が整数型では扱えないので， int 型を float32 型に変換しています．

```
x_train /= 255
x_test /= 255
```

白黒画像なので各画素には 0~255 の値が入っていますが，これを全て 0.0~1.0 に変換しています．

次にラベルデータも変換します．Keras の to_categorical 関数を使って，整数値を 2 値クラスの配列に変換します．例えば 2 という値は[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]という配列に変換されます．

```
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

また，今回は 0~9 までの 10 種類のクラス分類を行うのでクラス数として 10 を指定します．さて，データの準備が一通り終わったのでニューラルネットワークモデルを構築します．

```
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

まずは Keras の Sequential モデルというものをインポートします．これはニューラルネットワークの各層を格納する箱のようなものです．この箱の中で層を積み重ねていくイメージです．

その次にインポートしている Dense と Dropout は層の種類です．前回用いたニューラルネットワークでは，入力層を除いてどの層のニューロンもすべて，手前の層の全てのニューロンから値を受け取っていました．このような層を全結合層といいます．Dense レイヤは，この全結合層のことを指します．

しかし，全結合層だけで構築されたニューラルネットワークは過学習に陥りやすくなってしまいます．そこで，学習の際に層の中の一部のニューロンの入出力を行わない（行わないと言っても実際には存在するので入出力を 0 にする）ことで汎化性能を高め過学習を抑制します．学習のたびに，対象とするニューロンはランダムに選択されます．これをドロップアウトといい，Keras では Dropout レイヤを設けることで直前の Dense 層において Dropout を行うことができます．

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

今回作ったニューラルネットワークモデルは上記のような形をしています．ひとつずつ確認しましょう．

（1 行目）Sequential モデルを定義します．この中で層を積み木のように重ねていきます．

（2 行目）今回は 784 ピクセル分のデータを入力とするので，入力層のニューロンは 784 個です．これは，input_shape=(784,)として定義しています．この入力層の次の層が，ニューロン数 512 の全結合層になっています．全結合層の活性化関数は ReLU 関数を指定しています．

（3 行目）続いて，Dropout レイヤを設けることで，直前の全結合層においてドロップアウトを行います．今回は 512 個のニューロンのうち，常に 2 割をランダムにドロップアウトさせます．

（4 行目）そしてもう一層，ニューロン数 512 個の全結合層を設けます．

（5 行目）その全結合層においても，常に 2 割をドロップアウトさせます．

（6 行目）最後の全結合層（出力層）のニューロン数は 10 個です．これはクラス分類数が 10 個だからです．また，活性化関数として softmax 関数を用いています．関数の詳細な説明は省きますが，分類問題の出力層で softmax 関数を用いることで，出力結果を各クラスの確率として表すことができます．たとえば，入力画像として 2 の画像を入力したとき，出力結果が

```
[0.01, 0.01, 0.73, 0.08, 0.01, 0.02, 0.03, 0.01, 0.09, 0.01]
```

だったとします．正解値は

```
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
```

なので，ここから誤差が小さくなるように重みが修正されます．なお，この出力結果の例では 73%の確率で 2 であると判定しているので，誤差はあれどもとりあえず正解できているといえます．

以上でモデルの形は設定できました．コードを見るとどんな形のモデルかひと目で分かるのが Keras の良いところです．

ここで，学習を始める前に，どのような学習処理を行なうかを設定する必要があります．この設定をすることを Keras ではコンパイルといっています．

```
from keras.optimizers import RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])
```

学習用データと出力データの誤差を関数の形に表し（損失関数）、これを最小化させることを最適化といいます．RMSprop はそのための最適化アルゴリズムの一つです．他にも選択肢はありますが，今回はこれを使いましょう．ということでこれをインポートしています．最小化させる損失関数として，categorical_crossentropy を設定しています．これはクロスエントロピーで，今回のような多クラス分類ではよく用いられます．
そして metrics は評価関数で，これはモデルの学習には直接寄与しませんが，モデルの性能を表す指標です．今回はモデルの性能として判定精度を出すことで性能の良し悪しを示そうというわけです．

それでは，これまでに構築したモデルを用いて学習を開始します．

```
history = model.fit(x_train, y_train,   #トレーニングデータ
                    batch_size=100,     #バッチサイズの指定
                    epochs=20,          #エポック数の指定
                    verbose=1,          #ログ出力の指定．
                    validation_data=(x_test, y_test))  #テストデータ
```

学習は筋トレのように一度ではなく何度も繰り返すことで汎化性能が高まります．（やりすぎると過学習になりますが…）この繰り返す回数をエポックといい，epochs として定義しています．

1 回の学習で計算するデータの数のことをバッチサイズといい，トレーニングデータの中からランダムにデータが選択されて学習に使われます． batch_size として定義しています．1 エポックの中では，指定したバッチサイズの学習をバッチ回数分行います．バッチ回数は通常，トレーニングデータをバッチサイズで割った数の切り上げになります．今回の場合はトレーニングデータが 60000 あるので，バッチ回数は 600 です．これをエポックの数だけ繰り返して学習させるわけです．

筋トレでいうと，腕立て 100 回の 600 セットを 20 日間行うみたいなものです．また，verbose=1 としておくことで，学習過程のログが出力されます．最後に学習とその結果の精度を表示するために，以下のコードを入力します．

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

結果は以下のようになりました．

```
Train on 60000 samples, validate on 10000 samples

Epoch 1/20

60000/60000 [==============================] -
12s 194us/step - loss: 0.2374 - acc: 0.9271 - val_loss:
0.1055 - val_acc: 0.9687

~（省略）~

Epoch 20/20

60000/60000 [==============================] -
12s 205us/step - loss: 0.0256 - acc: 0.9942 - val_loss:
0.1299 - val_acc: 0.9814

Test loss: 0.12989266594767915

Test accuracy: 0.9814
```

エポックが進む毎に少しずつ誤差が低下して精度は上がり，最終的には精度 98%まで上がっています. また, 上の出力結果は GPU を使わずノート PC で学習を行った結果ですが, 1 エポックあたり 12 秒ほどで処理できています．これくらいならまだ許せる範疇ですが, 層構造が深くなったり，入力データが大きくなったりすると GPU を使わないと正直厳しいです.

c）自分が用意した画像で簡単なテスト

せっかくなので，自分の用意したデータできちんと判定してくれるかテストしましょう. 以下のような 28×28 ピクセルの画像を作ってみました．なお，ファイル名は test3.png としました.

図－３１　用いたテスト画像

明らかに我々の目には【3】に見えますが，正しく判定してくれるか試してみます.

今回は画像の読み込みを行うので，Google Colab で行う場合と自前の環境で行う場合はコードが若干異なります．まずは Google Colab で行う場合のコードから説明します．

Google Colab で行う場合

Google Colab で行う場合，ローカルのファイルに直接アクセスすることは出来ません．ファイルを Google Drive にアップすれば読み込むこともできるのですが，今回はファイルを Google Colab に直接アップロードするやり方でやってみましょう．

```
1 from google.colab import files
2 uploaded = files.upload()
```

上のコードを実行すると，ファイルの選択を行うことができます．ここで，アップロードしたい画像を選択してください．選択すると，以下のような，アップロードが完了した旨を示す文が出てくると思います．

```
test3.png(image/png) - 311 bytes, last modified:
2018/9/22 - 100% done
Saving test3.png to test3.png
```

これで Google Colab へのファイルアップロードは完了したので，判定用のコードを入力します．

```
1 from keras.preprocessing.image import img_to_array, load_img
```

load_img で画像の読み込み，img_to_array で読み込んだ画像を配列に変換できます．

```
1 test_img = load_img('./test3.png', grayscale=True, target_size=(28,28))
2 test_img_array  = img_to_array(test_img)
```

今回用いる画像はグレースケールとして読み込まなければならないので，grayscale=True を指定しておきます．

```
1 test_data = test_img_array.reshape(1, 784)
2 test_data = test_data.astype('float32')
3 test_data /= 255
```

学習の際に行ったのと同様，配列を 28×28 から 784×1 に変換し，すべて 0~1 の数字に変換します．

```
1 print(model.predict_classes(test_data))
```

predict_classes で結果を出力することができます．結果は以下のようになりました．

```
3
```

きちんと 3 であると認識してくれているようです.

自前の環境で行う場合

Google Colab を使わず, 自前の環境で行う場合はもう少し簡単になります. Google Colab にファイルをアップロードするために入力した, 以下のコードは不要になります.

```
from google.colab import files
uploaded = files.upload()
```

上のコードは入力せず, あとは Google Colab で行ったのと全く同じコードを入力します. このとき, 読み込むファイルは実行するコードと同じディレクトリに置いておいてください. 別のところにファイルを置いておく場合は, './test3.png'の部分を適宜ファイルが置いてある場所に書き換えてください.

d) コード全文 (Keras を用いたニューラルネットワーク)

今回紹介したコード全文を以下に示します. 関数などの呼び出し文はコードの先頭にまとめています.

学習・評価コード

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model.fit(x_train, y_train,  #トレーニングデータ
                    batch_size=100,    #バッチサイズの指定
                    epochs=20,         #エポック数の指定
                    verbose=1,         #ログ出力の指定.
                    validation_data=(x_test, y_test))  #テストデー

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

自分の画像でテスト

```
from google.colab import files
uploaded = files.upload()
```

※Google Colab を用いる場合のみ上記のコードを追加してファイルのアップロードが必要

```
from keras.preprocessing.image import img_to_array, load_img

test_img = load_img('./test3.png', grayscale=True, target_size=(
test_img_array = img_to_array(test_img)

test_data = test_img_array.reshape(1, 784)
test_data = test_data.astype('float32')
test_data /= 255

print(model.predict_classes(test_data))
```

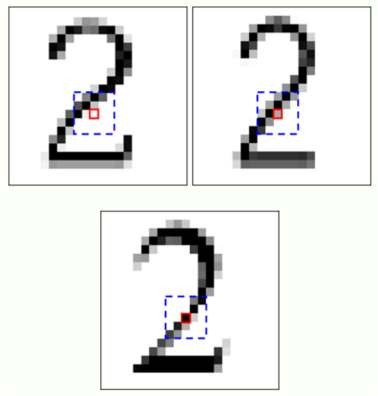
2.1.14. Keras で CNN

a) CNN とは

CNN とは

画像認識などの複雑な問題を簡単なニューラルネットワークで解くのは限界があります．そこで，その発展形である CNN（Convolution Neural Network）を使います．また，ディープラーニング（深層学習）という場合，レイヤの数が多いニューラルネットワークによる学習のことを指す場合もありますが，一般的には CNN を使った学習のことを指す場合が多いです．（他にも色々ありますが…）

ニューラルネットワークで画像認識を行おうとすると，画像の各画素値が各入力値として用いられることになりますが，対象物の位置が少しずれたり，形が変化したりするだけで入力値が大きく変わってしまいます．例えば以下の図のように，【2】という文字が書かれた画像を認識しようとしたとします．



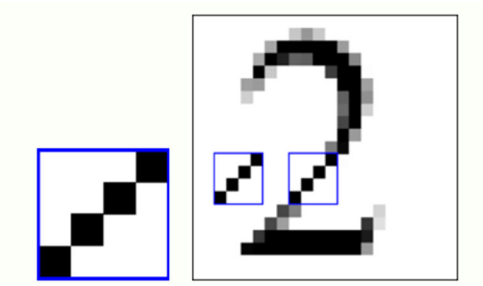
図－３３　【２】という文字が書かれた画像群

いずれも私達の目では 2 であることがわかりますが，各画素の赤枠部分は，どれも同じ位置にもかかわらず，色が異なっています．ですが，青い点線で囲まれた部分を見てみると，どれもなんとなく右上から左下に向かって黒い画素が並んでいるように見えます．つまり，個別の画素ではなく，ある程度広い領域をまとめて入力として使えば，より精度の高い判定ができそうだ，というのが CNN の考え方です．

畳み込み層

ある程度広い領域を入力するために用いるのが，フィルタと呼ばれる小領域（上の図の青枠のようなもの）です．例えば，【2】という画像に対して，「右上から左下に向かう黒い画素の集まり」のフィルタを用意して，これを画像の端から端まで走らせ，場所ごとの一致度を記録します．おそらく画像の真ん中や左上あたりでは，用意したフィルタとの一致度が高

くなるでしょう．



図－３４　フィルタおよびフィルタの重ね合わせのイメージ．畳み込み層では，この一致度（特徴マップ）を出力する

この，場所ごとの一致度の情報を出力する層が畳み込み層で，上記の処理を畳み込みといいます．なお，今回は例として「右上から左下に向かう黒い画素の集まり」のフィルタの話をしましたが，実際には様々な種類のフィルタを用意して，フィルタごとに畳み込みを行うので，大量の畳み込み結果（特徴マップ）が出力されます．また，各フィルタは判定したい対象に適したものが学習の結果から生成されます．

プーリング層

畳み込み層から出力された特徴マップは，それでも入力画像によってばらつきがあります．そこで，特徴マップを大まかなウィンドウに区切り，各ウィンドウの中での最大値を代表値として画像を縮小します（Max-pooling の場合）．これにより，ばらつきの影響を低減させます．

図－３５　Max pooling の例図

この，特徴マップを縮小する工程をプーリングといい，プーリングが行われる層をプーリング層といいます．また，プーリングを行うとデータの個数が区切ったウィンドウの個数に縮小されるのが分かると思います．よって以降の計算量が一気に少なくなるといった，もう一つの大きな利点があります．

CNN とは（まとめ）

これまでに説明した畳み込み層とプーリング層，そして活性化関数の層を幾重にも重ねることで構築されているのが CNN です．なお，最終的には特徴マップではなく判定結果を出力したいので，CNN では終盤に全結合層を設け，特徴マップから判定結果を導きます．以上が CNN の基礎事項になります．では，実際にデータを使ってその性能を確かめてみましょう．

b）データセット

今回は，CIFAR-10（サイファーテン）というデータセットを使います．CIFAR-10 は airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck の 10 クラスにラベル分けされた 32×32 画素のカラー画像データで，50000 枚のトレーニングデータと 10000 枚のテストデータが用意されています．以下の画像は CIFAR-10 の中身の一例です．

図－３６　CIFAR-10 の中身の例

c）サンプルコード

以下，サンプルコードです．Keras を使って CNN を構築し，CIFAR-10 のデータを判定します．

```
1 import keras
2 from keras.datasets import cifar10
```

MNIST と同様に，CIFAR-10 も Keras に最初から用意されているので，インポートするだけで読み込めます．自分のデータで学習したかったら，前回，MNIST の結果確認で行ったような手順でフォルダから読み込むことになります．

```
1 (X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

そして MNIST と同様に CIFAR-10 もトレーニングデータとテストデータに分割します．前回のニューラルネットワーク構築の際には，ここで画像データを 1 次元配列に変換していましたが，畳み込みとプーリングを行わなければならないので，この変換はここでは行いません．これは畳み込みとプーリングを終えて全結合層に入力する段階で行います．

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255.0
X_test /= 255.0
```

各画素値を 0~1 の間に収まるように変換します．今回はカラー画像なので，RGB の値それぞれにこの処理が適用されています．

```
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

そしてラベルデータを 2 値クラスの配列に変換しています．これも 10 クラスなので前回と同様．では CNN を構築します．ちょっと長いです．

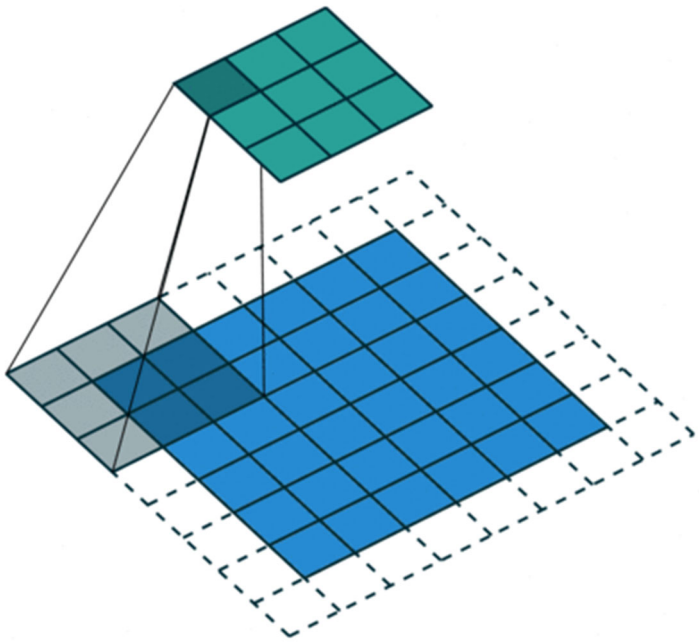
```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Convolution2D, MaxPooling2D
```

前回と違い， Flatten，Convolution2D，MaxPooling2D をインポートします．Flatten は平滑化層で，畳み込みとプーリングが終わったあと全結合層に入力する際に特徴マップを 1 次元配列に変換します．Convolution2D と MaxPooling2D はそれぞれ畳み込み層とプーリング層です．

```
model = Sequential()

model.add(Convolution2D(32, 3, 3, border_mode='same', activation=
                        input_shape=X_train.shape[1:]))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Convolution2D(64, 3, 3, activation='relu', border_mode=
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation= 'softmax'))
```

今回構築したモデルは，入力→畳み込み→畳み込み→プーリング→畳み込み→畳み込み→プーリング→全結合層→全結合層（出力）といった流れになっています．プーリング層と全結合層でドロップアウトを行っています．

最初の畳み込み層における，32, 3, 3 はそれぞれフィルタの数，フィルタサイズ，フィルタを一度に動かす幅（ストライド）です．border_mode='same'を指定することによって，画像の周囲にパディングを施して畳み込み後も画像の大きさが変わらないようにします．以下の図を見るとイメージしやすいかと思います．



図－３７　下の画像が入力，上の画像が出力（vdumoulin/conv_arithmetic – GitHub）

そして input_shape=X_train.shape[1:]は入力画像の形状です．今回は(32,32,3)になります．プーリング層における pool_size=(2, 2)は，プーリングしてダウンスケールする大きさです．今回の場合，一度のプーリングで画像の大きさが縦横半分になります．また，プーリング結果にドロップアウトを適用しています．

以上が今回構築した CNN の構造です．続いて，学習処理を決めるコンパイルです．

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

今回は最適化アルゴリズムに Adam を用いました．それでは，これまでに構築したモデルを用いて学習を開始します．

```
from keras.callbacks import EarlyStopping

history = model.fit(X_train, y_train,
                    batch_size=100,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[EarlyStopping()])
```

エポック数が多すぎると過学習する可能性があるので，今回はいい感じのところで学習を打ち切れるようにします（Early stopping）．

そのために，callbacks = [EarlyStopping()]と設定しておきます．こうすることで，エポッ

ク数増加による過学習を起こす前に，学習をストップさせることが出来ます．
Early stopping を行うために，トレーニングデータの一部をバリデーションデータとして設定します．validation_split = 0.1 とすることでトレーニングデータのうち最後の 10%をバリデーションデータとして用いることができます．（なので実際にトレーニングに使われるのはトレーニングデータの 90%です．）

また，このようにするとトレーニングデータ全てを訓練に用いることはできなくなるので，適切なエポック数が設定できていれば Early stopping は行わなくてもいいかもしれません．最後にテストデータを用いて学習精度を確認するためのコードを入力します．

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

結果は以下のようになりました．CNN は CPU で計算すると時間がいくらあっても足りないので，Google Colab の GPU 機能を使って計算しました．

```
Epoch 1/30

45000/45000 [==============================] -
15s 339us/step - loss: 1.6045 - acc: 0.4111 - val_loss:
1.2246 - val_acc: 0.5628

~（省略）~

Epoch 12/30

45000/45000 [==============================] -
13s 281us/step - loss: 0.5692 - acc: 0.7988 - val_loss:
0.6251 - val_acc: 0.7868

Test loss: 0.6488395797729493

Test accuracy: 0.7786
```

Early stopping を用いた結果，最初に指定した 30 エポックに達する前に終了しています．つまりこれ以上学習を進めても精度が上がらないと判断されたということです．そして，テストデータに対する精度は約 78%でした．

ネットで検索すれば，VGG-16 や GoogleNet，ResNet といった，すでに高い成果を出しているモデルも公開されているので，これらを用いればさらに精度は向上すると思います．

d）コード全文（Keras で CNN）

今回紹介したコード全文を以下に示します．関数などの呼び出し文はコードの先頭にまとめています．

```
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.callbacks import EarlyStopping

(X_train, y_train), (X_test, y_test) = cifar10.load_data()

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255.0
X_test /= 255.0

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)

model = Sequential()
model.add(Convolution2D(32, 3, 3, border_mode='same', activation=
                        input_shape=X_train.shape[1:]))
model.add(Convolution2D(32, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Convolution2D(64, 3, 3, activation='relu', border_mode=
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation= 'softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(X_train, y_train,
                    batch_size=100,
                    epochs=30,
                    verbose=1,
                    validation_split=0.1,
                    callbacks=[EarlyStopping()])

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

2.1.15. ResNet を Fine Tuning して自分が用意した画像を学習させる

a) ResNet とは

ResNet はディープラーニングを行うためのモデルの一つであり，2015 年の ILSVRC（世界的な画像認識コンテスト）で優勝したモデルです．

一般的に，ある程度多層のニューラルネットワークは層が少ないニューラルネットワークよりも精度が高くなりますが，あまりに多くしすぎると勾配消失問題が発生し精度が悪化します．ディープラーニングの学習においては，各層ごとに活性化関数の微分を行い，勾配を計算することで重みを調整しているのですが，層を増やしすぎると微分の積が多くなりすぎて勾配が消えていくという問題があります．

従来のモデルでは各層（Deep path）を通して勾配の計算を行っていますが，ResNet は新たに別の経路（Shortcut Connection）を設けることで，勾配消失を起こしにくくしたモデルです．その結果，ResNet は精度を上げつつ深い階層のネットワークを実現できています．（反面，階層が深いぶん学習時間が長くなりますが…）

さらなる詳しい説明や詳細については，検索したら多くの解説が見つかると思います．

b) Fine Tuning

実際に自前のデータで学習を行おうとした場合，大量のデータを用意するのは困難な場合があります．しかし，すでに学習済みのモデルをベースにしてそこから学習を行えば，少ないデータでも精度の向上を図ることができます．

Fine Tuning は，すでに学習済みのモデルを用いて再学習を行い，新たなモデルを生成することを指します．よく同列に説明される転移学習（Transfer Learning）は，すでに学習済みのモデルを特徴量の抽出器として使い，別の分類をさせる方法なので，モデル自体を再学習させる Fine Tuning とは異なります．

今回は既存のデータセットで学習済みの ResNet の Fine Tuning を行い，自前のデータセットに対して判定を行うことを目指します．

c）データセット

事前学習でのデータセット

今回事前学習に使う（使われている）データセットは，ILSVRC2012 データセットです．ILSVRC2012 データセットは，ImageNet という巨大データセットの中から選ばれた画像データ群で，トレーニングデータ 120 万枚，バリデーションデータ 5 万枚，テストデータ 10 万枚，クラス名 1,000 種類の巨大なデータセットです． 2015 年に ResNet が優勝した ILSVRC は ImageNet のデータを用いて競う大会で, ILSVRC2012 データセットはその 2012 年大会で使われました．

新たに学習させるデータ

新たに学習させるデータとしては，本記事の趣旨的に ImageNet の中にないクラスのデータを使いたいところです．とはいえ，このために新たに学習データを揃えるのも面倒くさいです．(本当は記事のタイトル的にも，自前で調達した画像を使いたいんですが…) そこで，今回は料理のデータセットである Food-101 を使います．Food-101 には，101 種類の料理の画像が，各 1000 枚ずつ含まれています．Food-101 のデータは，検索するか，またはこちらのページからダウンロードできます．今回はあくまで例なので，そのうちの一部（apple_pie，french_toast，hot_dog，pizza，sashimi）を使うことにします． これら 5 種類の料理画像を判別することを目指します．なお，画像は 800 枚をトレーニング用，200 枚をテスト用に用いることにします．

また，今回用いるデータですが，予め以下のような階層のフォルダを構築しておきます．

```
dataset
  ├ train
  |   ├ apple_pie
  |   ├ french_toast
  |   ├ hot_dog
  |   ├ pizza
  |   └ sashimi
  └ val
      ├ apple_pie
      ├ french_toast
      ├ hot_dog
      ├ pizza
      └ sashimi
```

各画像に対応したフォルダを，train，val の 2 つのフォルダに分けています．そして train における各フォルダにはトレーニング用画像を 800 枚ずつ（計 4000 枚），val における各フォルダにはテスト用画像を 200 枚（計 1000 枚）ずつ入れておきます．

以上のようにすれば，Food-101 でなく，自前の画像でも学習ができます．

d）サンプルコード

以下，サンプルコードです．

今回は用意したファイルを Google Drive にアップロードして，Google Colab から Google Drive 内のファイルにアクセスするやり方でデータを読み込みます．こうすることで，同じデータを開発チーム内でも共有することが可能です．

ここで，ややこしいことにフォルダをそのまま Google Drive にアップロードしても，そ

れを素直にフォルダとして読み込むのは難しいので，Google Drive にアップロードする前に一旦このフォルダを zip に圧縮して，dataset.zip を作成し，これを Google Drive にアップロードします．

ではまず，画像の読み込みに必要なコードを書きます．

```
1 !pip install -U -q PyDrive
2 from pydrive.auth import GoogleAuth
3 from pydrive.drive import GoogleDrive
4 from google.colab import auth
5 from oauth2client.client import GoogleCredentials
```

2 行目と 3 行目のコードで Google Drive にアクセスするための PyDrive をインポートしていますが，このモジュールは元々Google Colab に標準で入っていません．

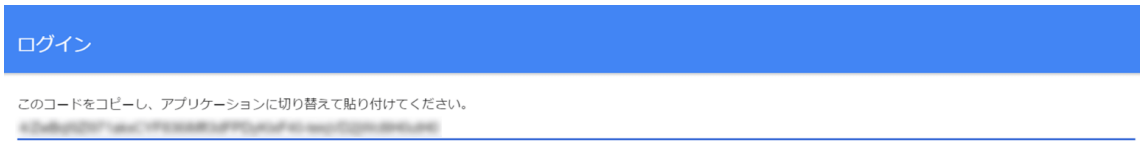
そこで，1 行目のコードで，pyDrive を Google Colab にインストールする必要があります．インポートできたら，Google Drive にアクセスするための認証を行います．以下のコードで行うことが出来ます．

```
1 auth.authenticate_user()
2 gauth = GoogleAuth()
3 gauth.credentials = GoogleCredentials.get_application_default()
4 drive = GoogleDrive(gauth)
```

コードを実行したら，以下のようなフォームが出てくると思います．

```
... Go to the following link in your browser:

        https://accounts.google.com/o/oauth2/auth?redirect_uri=urn%

    Enter verification code:
```

表示されているリンクをクリックして，読み込みたいファイルを保存してある Google Drive アカウントを選択すると，以下のように認証用コードが表示されると思います．



このコードをコピーし，先程のフォームに貼り付けて Enter キーを押します．

特にエラーなどが出なければ，これでデータをアップロードしている Google Drive からデータを読み込む準備が完了しました．

ここで，Google Drive から読み込みたいデータについて，「共有可能なリンク」を有効にしておく必要があります．読み込みたいデータを Drive 上で右クリックすると，以下のように「共有可能なリンクを取得」を選択できます．

そして，共有設定をオンにすると，「https://drive.google.com/open?id=abcdefg…」のような感じの共有リンクが表示されます．（「共有可能なリンクを取得」をクリックしたら自動的にオンになっているかもしれません．）ここで必要なのが「id=」以降の部分になります．今回示した例でいうと abcdefg…の部分です．ここでは仮に，「id=」以降の部分が「abcdefg」だとしてコードの解説をします．

```
id = ' abcdefg '  # 共有リンクで取得したid=以降の部分
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('dataset.zip')
```

これで，Google Colab 内に zip ファイルを読み込むことが出来ました．
つづいて，以下のコードを続けて zip ファイルを解凍します．

```
!unzip dataset.zip
```

上のコードを実行すると，解凍されたデータが以下のように表示されると思います．

```
Archive:  dataset.zip
creating: dataset/
creating: dataset/train/
creating: dataset/train/apple_pie/
inflating: dataset/train/apple_pie/101251.jpg
inflating: dataset/train/apple_pie/103801.jpg
~（省略）~
```

解凍が済んだら，dataset.zip は必要ないです．以下のコードで Google Colab 上から削除できます．（これは Google Colab 上から削除するだけで，Google Drive にある元データを削除するわけではないです）

```
!rm dataset.zip
```

以上で Google Colab を使う際の，だいぶ面倒くさい作業が完了しました．（自前の環境でやる場合はこれまで書いたコードや，それに伴う作業は必要ないので割と楽）

ようやく本題のコードを書き始めます．まず，トレーニングデータ，テストデータの定義を行います．

```
classes = ['apple_pie', 'french_toast', 'hot_dog', 'pizza', 'sashimi'] #分類するクラス
nb_classes = len(classes)

train_data_dir = './dataset/train'
validation_data_dir = './dataset/val'

nb_train_samples = 4000
nb_validation_samples = 1000

img_width, img_height = 224, 224
```

classes に分類したいクラスを定義しています．今回の場合だと，apple_pie, french_toast, hot_dog, pizza, sashimi です．そして，nb_classes にクラス数を定義しています．4 行目，5 行目のコードで，読み込むデータのディレクトリを指定しています．7 行目，8 行目のコードで，トレーニングデータの数，テストデータの数を指定しています．そして,10 行目のコードで読み込む画像をすべて同じ大きさにリサイズします．これは最終的に全結合層に特徴マップを読み込ませるためです．今回は 224×224 にしましたが，あまりに小さいサイズを指定すると画像が縮小されすぎるのでエラーになります．詳細は Keras のドキュメントに書いてあります．

つづいて，読み込んだデータをトレーニングデータ，テストデータとして使えるように定

義します.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale=1.0 / 255, zoom_range=0.2, horizontal_flip=True)
validation_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

今回のような形式で予めディレクトリを作成しておくと，Keras の ImageDataGenerator でトレーニングデータとテストデータを簡単に定義できます.

ここでは，トレーニングデータに対して，rescale=1.0 / 255，zoom_range=0.2，horizontal_flip=True を指定しています．rescale は，これまで同様に各画素値の 0~1 への正規化です．後の 2 つは，トレーニングデータが十分でない場合に画像を水増しするための引数です．zoom_range は画像をランダムにズームします．horizontal_flip は，画像をランダムに左右反転させます．テストデータに対しては，水増しする必要が無いので正規化のみです.

```
train_generator = train_datagen.flow_from_directory(
  train_data_dir,
  target_size=(img_width, img_height),
  color_mode='rgb',
  classes=classes,
  class_mode='categorical',
  batch_size=16)

validation_generator = validation_datagen.flow_from_directory(
  validation_data_dir,
  target_size=(img_width, img_height),
  color_mode='rgb',
  classes=classes,
  class_mode='categorical',
  batch_size=16)
```

続いて，.flow_from_directory で，ディレクトリへのパスを受け取り，拡張/正規化したデータのバッチを生成します．train_generator には train_data_dir を指定しています．target_size は最初に指定した画像サイズで，今回は 224×224 です．カラー画像なので color_mode='rgb'，分類するクラスを classes として指定，分類は多クラス分類なので class_mode='categorical'を指定しています．学習の際のバッチサイズは 16 にしました．テストデータについても，同様に設定しています．以上のコードを実行すると，下のような文が出てくると思います.

```
Using TensorFlow backend.

Found 4000 images belonging to 5 classes.

Found 1000 images belonging to 5 classes.
```

きちんと，トレーニングデータ 4000 枚，テストデータ 1000 枚が定義されているようです．では，学習を行うモデルを作っていきます.

```
from keras.applications.resnet50 import ResNet50
from keras.models import Sequential, Model
from keras.layers import Input, Flatten, Dense
```

Keras には ImageNet データセットで学習済みの ResNet50（50 レイヤの ResNet）が最初から用意されているので，インポートするだけで読み込めます．

```
input_tensor = Input(shape=(img_width, img_height, 3))
ResNet50 = ResNet50(include_top=False, weights='imagenet',input_tensor=input_tensor)
```

入力画像は 224×224 の RGB 画像なので，これを input_tensor として指定しています．今回 FineTuning を行うにあたって，出力結果は 5 クラスの分類にしたいので，全結合層を変更します．そのためにまず，include_top=False とすることで，全結合層を除いた ResNet50 をインポートします．そしてこのとき， weights='imagenet' とすることで学習済みの ResNet50 が読み込めます．逆に，weights=None とすると，ランダムな初期値から始まります．

つづいて，除去した全結合層の代わりに用いる新たな全結合層を作成します．

```
top_model = Sequential()
top_model.add(Flatten(input_shape=ResNet50.output_shape[1:]))
top_model.add(Dense(nb_classes, activation='softmax'))
```

2 行目のコードで，ResNet50 の出力を 1 次元化しています．これを出力クラス数が 5 の全結合層に入力する形で，新たな全結合層を定義します．

```
model = Model(input=ResNet50.input, output=top_model(ResNet50.output))
```

あとは，全結合層を取り払った ResNet50 と，新たに作った全結合層をくっつけてモデルの形づくりは完了です．続いて，学習処理を決めるコンパイルです．

```
from keras import optimizers
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=1e-3, momentum=0.9),
              metrics=['accuracy'])
```

今回は最適化アルゴリズムに確率的勾配降下法を用いてみました．それでは，これまでに構築したモデルを用いて学習を開始します．

```
history = model.fit_generator(
    train_generator,
    samples_per_epoch=nb_train_samples,
    nb_epoch=5,
    validation_data=validation_generator,
    nb_val_samples=nb_validation_samples)
```

ジェネレータにより生成されたデータで訓練するには，.fit_generator を使います．samples_per_epoch をトレーニングデータ数（4000）にしているので，各エポックで 4000 個のサンプルが用いられます．バッチサイズを 16 としたので，1 エポックにおけるステップ数は 4000/16 の，250 になります．Fine Tuning では，すでにある程度重みが定まっているので，エポック数は少なめの 5 としてみました．ということで今回は Early stopping は行

なわず，トレーニングデータ全てを訓練に用います．

結果は以下のようになりました．

```
Epoch 1/5

250/250 [==============================] – 340s 1s/step
– loss: 0.6239 – acc: 0.7707 – val_loss: 0.6378 – val_acc:
0.7749

Epoch 2/5

250/250 [==============================] – 326s 1s/step
– loss: 0.2571 – acc: 0.9113 – val_loss: 0.5076 – val_acc:
0.8280

Epoch 3/5

250/250 [==============================] – 326s 1s/step
– loss: 0.1720 – acc: 0.9415 – val_loss: 0.5026 – val_acc:
0.8349

Epoch 4/5

250/250 [==============================] – 326s 1s/step
– loss: 0.1280 – acc: 0.9555 – val_loss: 0.4658 – val_acc:
0.8610

Epoch 5/5

250/250 [==============================] – 326s 1s/step
– loss: 0.0843 – acc: 0.9738 – val_loss: 0.4351 – val_acc:
0.8788
```

最終的に，テストデータに対する正解率が約 88%まで上昇しています．

一方，ResNet の読み込み時に weights=None として学習を行った場合（ImageNet の重みを初期値とした Fine Tuning を行わなかった場合）の結果を以下に示します．

```
Epoch 1/5

250/250 [==============================] – 328s 1s/step
– loss: 1.8689 – acc: 0.2727 – val_loss: 2.4236 – val_acc:
0.2587

Epoch 2/5

250/250 [==============================] – 315s 1s/step
– loss: 1.7146 – acc: 0.3550 – val_loss: 1.5603 – val_acc:
0.3740

Epoch 3/5

250/250 [==============================] – 315s 1s/step
– loss: 1.5204 – acc: 0.4150 – val_loss: 1.4414 – val_acc:
0.4109

Epoch 4/5

250/250 [==============================] – 316s 1s/step
– loss: 1.3531 – acc: 0.4803 – val_loss: 1.4661 – val_acc:
0.4162

Epoch 5/5

250/250 [==============================] – 315s 1s/step
– loss: 1.3628 – acc: 0.4753 – val_loss: 1.2330 – val_acc:
0.5167
```

ImageNet による重みから学習をスタートした場合は 1 エポック目からすでに約 77%の正解率が出ていますが，普通に学習させると 1 エポック目の正解率は約 25%となっています．5 エポック目でも，約 52%までしか上昇していません．このことからも Fine Tuning の効果が伺えます．

せっかくなので，今回 Fine Tuning で学習したモデルに，テストデータの一部を読み込んで予測確率を出力してみます．今回は下に示すアップルパイの画像を判定させてみます．

図－３８　アップルパイの画像

```
from keras.preprocessing import image
import numpy as np

image_data = 'dataset/val/apple_pie/1005649.jpg'
img = image.load_img(image_data, target_size=(img_width, img_height))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x / 255
pred = model.predict(x)[0]

# 予測確率を出力
top = 5
top_indices = pred.argsort()[-top:][::-1]
result = [(classes[i], pred[i]) for i in top_indices]
print(result)
```

上のコードを実行すると，結果は以下のようになりました．

```
[('apple_pie', 0.9993316), ('french_toast', 0.0006522994),
('pizza', 8.400712e-06), ('sashimi', 5.791974e-06), ('hot_dog',
1.8393388e-06)]
```

高い確率でアップルパイと判定されています．

e）コード全文

以下に，今回のコード全文を示します．

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

id = ' abcdefg '  # 共有リンクで取得したid=以降の部分.左のコードは一例.
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('dataset.zip')

!unzip dataset.zip
!rm dataset.zip
```

以上のコードは Google Colab でデータファイルを読み込ませるために行うコード．自前の環境で行う場合は不要.

```
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet50 import ResNet50
from keras.models import Sequential, Model
from keras.layers import Input, Flatten, Dense
from keras import optimizers


classes = ['apple_pie', 'french_toast', 'hot_dog', 'pizza', 'sashimi'] #
nb_classes = len(classes)
train_data_dir = './dataset/train'
validation_data_dir = './dataset/val'
nb_train_samples = 4000
nb_validation_samples = 1000
img_width, img_height = 224, 224

train_datagen = ImageDataGenerator(rescale=1.0 / 255, zoom_range=0.2, ho
validation_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_generator = train_datagen.flow_from_directory(
  train_data_dir,
  target_size=(img_width, img_height),
  color_mode='rgb',
  classes=classes,
  class_mode='categorical',
  batch_size=16)

validation_generator = validation_datagen.flow_from_directory(
  validation_data_dir,
  target_size=(img_width, img_height),
  color_mode='rgb',
  classes=classes,
  class_mode='categorical',
  batch_size=16)

input_tensor = Input(shape=(img_width, img_height, 3))
ResNet50 = ResNet50(include_top=False, weights='imagenet',input_tensor=i

top_model = Sequential()
top_model.add(Flatten(input_shape=ResNet50.output_shape[1:]))
top_model.add(Dense(nb_classes, activation='softmax'))

model = Model(input=ResNet50.input, output=top_model(ResNet50.output))

model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.SGD(lr=1e-3, momentum=0.9),
              metrics=['accuracy'])

history = model.fit_generator(
    train_generator,
    samples_per_epoch=nb_train_samples,
    nb_epoch=5,
    validation_data=validation_generator,
    nb_val_samples=nb_validation_samples)
```

↓テストデータの，./apple_pie/1005649.jpg をテストしてみた場合のコード.

```
from keras.preprocessing import image
import numpy as np

image_data = 'dataset/val/apple_pie/1005649.jpg'
img = image.load_img(image_data, target_size=(img_width, img_height))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = x / 255
pred = model.predict(x)[0]

# 予測確率を出力
top = 5
top_indices = pred.argsort()[-top:][::-1]
result = [(classes[i], pred[i]) for i in top_indices]
print(result)
```

2.1.16. Keras で回帰

a）回帰問題

これまで，Keras を用いて分類問題を扱ってきましたが，Keras を使ってニューラルネットワークを構築し，回帰問題を解くことも可能です．すなわち，入力データに対して何らかのクラスを出力するのではなく，連続値を出力します．

入力画像から別の画像を生成するような高度な回帰問題（高解像度化や自動着色）も CNN を使えば可能ですが，まずはより簡単な，複数の特徴量から 1 つの数値を予測するような回帰問題をやってみたいと思います．

b）データセット

今回は，機械学習の回帰問題でよく用いられるデータセットである，boston_housing を用いて，ボストンの住宅価格の推定を行ってみたいと思います．これは，Keras のなかにも標準で入っているデータセットです．このデータセットには，犯罪発生数や住居の平均部屋数など，14 個の特徴量を有するデータが 506 件入っています．これらの特徴量を用いて住宅価格の回帰を行います．今回は，データセットの 80%を学習に用い，残りの 20%を評価に用います．

c）サンプルコード

以下，サンプルコードです．

```
import keras
from keras.datasets import boston_housing
```

まずは Keras のインポートです．そして，boston_housing のデータセットもインポートします．

```
(X_train, y_train), (X_test, y_test) = boston_housing.load_data()
```

そして，boston_housing のデータをトレーニングデータとテストデータに分割します．元から 4:1 に分けられているので，データセットの 80%が学習に用いられます．X_で始まるのが各特徴量を有するデータで，y_で始まるのが対応する住宅価格です．

```
#トレーニングデータの正規化
X_train_mean = X_train.mean(axis=0)
X_train_std = X_train.std(axis=0)
X_train -= X_train_mean
X_train /= X_train_std

y_train_mean = y_train.mean()
y_train_std = y_train.std()
y_train -= y_train_mean
y_train /= y_train_std
```

次に各データを正規化しています．これは，各データの属性がそれぞれ異なるためです．

属性がそれぞれ異なると，計算するときにこれらのスケールの違いに結果が引っ張られてしまう恐れがあるので，これを防ぐために，同じ尺度に調整します．具体的な内容としては，各変数ごとに平均値を引いて，これを標準偏差で割ります．

```
#テストデータの正規化
X_test -= X_train_mean
X_test /= X_train_std
y_test -= y_train_mean
y_test /= y_train_std
```

また，テストデータに対しても，トレーニングデータの平均と標準偏差を用いて正規化します．これは，テストデータとトレーニングデータを同じ尺度で正規化するためです．でないと，精度が一気に低下してしまうので気をつけましょう．

さて，データの準備が一通り終わったのでニューラルネットワークモデルを構築します．

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
```

今回の回帰では，画像処理をしたりするわけではないので，畳み込み層などは特に設けず，全結合層のみで構成しています．また，今回は最終的に住宅価格を出力したいので，最後の全結合層のニューロンは 1 つだけになっています．

```
model.compile(optimizer='adam',
              loss='mse',
              metrics=['mae'])
```

今回は最適化アルゴリズムに Adam を用いました．また，分類問題では損失関数に categorical crossentropy を使っていましたが，今回は mse（平均二乗誤差）を使っています．これは回帰でよく用いられる損失関数で，なめらかな関数を学習によって計算します．評価関数は学習に直接関わってきませんが，今回は mae（平均絶対誤差）を用いています．
それでは，これまでに構築したモデルを用いて学習を開始します．

```
history = model.fit(X_train, y_train,  #トレーニングデータ
                    batch_size=1,  #バッチサイズの指定
                    epochs=100,       #エポック数の指定
                    verbose=1,        #ログ出力の指定．
                    validation_data=(X_test, y_test))  #テストデータ
```

今回の場合はバッチサイズ 1 でエポック 100 として学習を行います．
結果は以下のようになりました．

```
Train on 404 samples, validate on 102 samples

Epoch 1/100

404/404 [==============================] – 3s 7ms/step
– loss: 0.3963 – mean_absolute_error: 0.4105 – val_loss:
0.2950 – val_mean_absolute_error: 0.3866

~（省略）~

Epoch 100/100

404/404 [==============================] – 2s 5ms/step
– loss: 0.0176 – mean_absolute_error: 0.0905 – val_loss:
0.1393 – val_mean_absolute_error: 0.2480
```

エポックが進む毎に少しずつ誤差が低下していることが確認できると思います.

2.1.17. 画像キャプションの自動生成

a）キャプション生成

今回は，画像を入力して，その画像の内容を説明する文章を出力するディープラーニングモデルについて解説します．これまでとは違って，かなり長い解説になると思います．

文章を生成する場合，それまでの単語あるいは文章から，次の単語を出力することを繰り返し行う必要があります．そこで，文章生成などのタスクでは時系列データを取り扱うことの出来る RNN（リカレントニューラルネットワーク）が一般的に用いられます．

RNN は，それまでに計算された情報を保持し，次の計算に用いることができます．そして，次々と単語を出力して文章を生成した結果，累積されていった損失から，各出力過程の重みを調整します．このようにすることで，文章などの連続したデータの学習が可能になります．

今回は，RNN の一種である，LSTM（Long short-term memory）を用いてキャプション生成を行います．LSTM は RNN よりも複雑な構造ですが，Keras で実装する場合は，これまでに行ったディープラーニングモデルと同様にモデルの途中に LSTM レイヤを挟み込むだけなので，あまり細かい中身を意識しなくても容易に実装することが可能です．

b）データセットの準備

キャプション生成用のデータセットとして，ここでは MS-COCO を用います．MS-COCO には元々英語のキャプションが付いていますが，ありがたいことに日本語のキャプションも用意してくださった方々がいるので，今回は日本語キャプションで学習と生成を行いましょう．日本語のキャプションデータは，STAIR Lab / ステアラボ千葉工業大学人工知能・ソフトウェア技術研究センターが作成・公開しています．

このキャプションデータは，MS-COCO の 16 万枚の画像に各画像 5 文ずつ，計 80 万文が用意されています．ですが，今回は雰囲気を掴むために，バリデーション用のデータである『stair_captions_v1.2_val_tokenized.json』のさらに一部，2000 枚（1 万文）を用いて学習および評価，テストを行いましょう．

MS-COCO および STAIR Lab によるキャプションデータのダウンロードは，こちらのリンクから可能です．MS-COCO のデータは，『2014 Val images [41K/6GB]』を選んでください．（http://cocodataset.org/#download）（http://captions.stair.center/）

また，このキャプションデータは JSON 形式で記述されていることから，少々見づらく，さらに今回ここで用いるにあたっては余計なデータも多く含まれているので，まず最初により使いやすい形に成形したいと思います．そのためのコードを以下に示します．

```
import json
import pandas as pd
import numpy as np

f = open("stair_captions_v1.2_val_tokenized.json", 'r', encoding='utf-8')
json_data = json.load(f)
```

最初に，必要なモジュールをインポートし，JSON ファイルをインポートして読み込みます．日本語を扱う都合上，文字コードに関するエラーが出る可能性は往々にしてあるので，万一出てしまったら随時対応して下さい．

```
id_file = {}
for i in range(len(json_data["images"])):
    file_name = json_data["images"][i]["file_name"]
    id = json_data["images"][i]["id"]
    id_file[id] = file_name
```

続いて，この JSON ファイルにおいては画像のファイル名が ID に紐づけられていることから，画像のファイル名と ID を対応させたディクショナリを作っておきます．

```
annotations_size = len(json_data["annotations"])
df = pd.DataFrame(data = np.zeros((annotations_size,2)))

for j in range(annotations_size):
    image_id = json_data["annotations"][j]['image_id']
    file_name = id_file[image_id]
    caption = json_data["annotations"][j]["tokenized_caption"]
    df.loc[j, 0] = file_name
    df.loc[j, 1] = caption
```

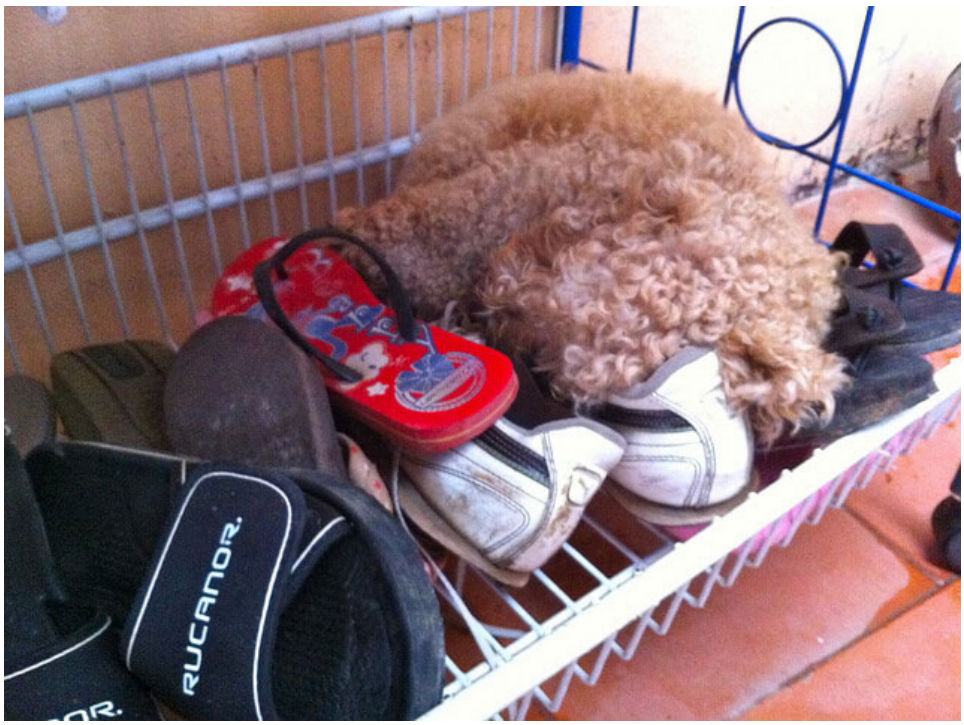
続いて，総キャプション数に合わせた pandas のデータフレームを作り，データフレームに各画像とキャプションを対応させていきます．（どうせ 2000 枚しか使わないのに全画像を処理するのは無駄感がありますが，大した量ではないのでザザっとやっちゃってます）

```
data_size = 10000
df.sort_values(0, inplace=True)
df= df.reset_index(drop=True)
df= df.loc[:data_size-1]

df.to_csv("caption.txt", sep=" ", header=False, index=False, encoding='utf-8')
```

最後に，データフレームの上から 10000 行を txt ファイルとして出力します．生成された txt ファイルの，上から 6 行分が以下のようになっています．

COCO_val2014_000000000042.jpg“白い靴や赤や黒の草履が乱雑に置かれた棚”
COCO_val2014_000000000042.jpg“玄関のかごの中に靴と一緒に紛れこんでいるトイプードル”
COCO_val2014_000000000042.jpg“ケージに置かれた靴に犬が頭をのせて寝ている”
COCO_val2014_000000000042.jpg“かごの中に靴が入れられている”
COCO_val2014_000000000042.jpg“スニーカーやサンダルが入ったバスケットに犬が寝ている”
COCO_val2014_000000000073.jpg“アスファルトの上に停めてある黒いバイク”

このように，画像 1 つにキャプションが 5 つ，画像名［半角スペース］”説明文”の形で指定されていることが確認できます．ちなみに，COCO_val2014_000000000042.jpg の画像は以下のような画像です．（気づいたかもしれませんが，本記事のサムネイルの AI はこの正解データと同じ文をドンピシャで想像しているので，ちょっとやりすぎですね，こんな風に文を出力してくれるといいんですが．）



図－３９　解析に用いた画像

また，説明文を見ると分かると思いますが，単語ごとに文章が区切られています．これは，単語ごとに学習・出力を行う必要があるからです．英文ではそもそも各単語間にスペースがあるため心配ないのですが，日本語でキャプション生成を行う場合はこのような配慮（形態素解析）が必要です．なお，STAIR Lab が用意したキャプションの形態素解析は，形態素解析を自動で行うツールである MeCab（めかぶ）を用いて行われているとのことです．
そして，今回用いるデータセットについて，トレーニング用に 8 割（1600 枚），バリデーション用に約 2 割(399 枚)，最後の 1 枚がテスト用，になるように事前に指定しておきます．また同時に，今回使う 2 万枚の画像も，ダウンロードした COCO データセットが入っているフォルダ（val2014）から，新しい他のフォルダにコピーしておきます．以下にそのためのコードを示します．

```
# -*- coding: utf-8 -*-
import pandas as pd
import os
import shutil

file = open('path/to/caption.txt', 'r', encoding='utf-8')
caption_data = file.read()
file.close()
```

最初に，必要なモジュールをインポートし，先ほど出力した txt ファイルを読み込みます．

```
image_list = []
for line in caption_data.split('\n'):
    tokens = line.split()
    if len(line) < 2:
        continue
    image_id, image_desc = tokens[0], tokens[1:]
    if image_id not in image_list:
        image_list.append(image_id)
```

続いて，txt ファイルに書かれていた画像名の部分だけをリストに格納していきます．

```
os.mkdir('path/to/photo_data')
for list in image_list[:-1]:
    shutil.copy('path/to/val2014/{}'.format(list),
    'path/to/photo_data')
```

そしてまずは，トレーニング画像とバリデーション画像をコピーして入れておくフォルダ「photo_data」を作成し，COCO データセットのフォルダ「val2014」から，最後の 1 枚以外を「photo_data」にコピーします．適宜自分の環境に合わせてパスを設定してください．(参照ディレクトリ内に同じ名前のフォルダが既に存在するとエラーが出ます．)

```
os.mkdir('path/to/test_data')
shutil.copy('path/to/val2014/{}'.format(image_list[-1]), 'path/to/test_data')
```

同じ要領で，最後の 1 枚だけを新しく作ったフォルダ「test_data」にコピーします．

```
train_data = pd.Series(image_list[:1600])
val_data = pd.Series(image_list[1600:1999])

train_data.to_csv("train.txt", header=False, index=False)
val_data.to_csv("val.txt", header=False, index=False)
```

最後にデータをトレーニングデータとバリデーションデータに分けて，train.txt，val.txt の 2 つのファイルを新たに生成します．

なお，tarin.txt の上から 3 行分は，以下のようになっています．

```
COCO_val2014_000000000042.jpg
COCO_val2014_000000000073.jpg
COCO_val2014_000000000074.jpg
```

このように，使用する画像の名前が羅列してあります．

以上でデータセットの準備は完了です．以上のような形式でデータを用意さえすれば，今回解説するコードでキャプション生成が可能です．ですから，自前のデータを Excel に入力して txt 形式で出力・保存するなど，自分にあった形でデータセットを構築して構いません．

それでは，本題のキャプション生成モデルの作成について，以下に説明します．

c）サンプルコード

以下，サンプルコードです．

```
# -*- coding: utf-8 -*-
image_directory = 'path/to/photo_data'#画像データのパスを指定すること
caption_data = 'path/to/caption.txt'#キャプションデータのパスを指定すること
train_data = 'path/to/train.txt'#トレーニングデータのパスを指定すること
val_data = 'path/to/val.txt'#バリデーションデータのパスを指定すること
```

まず，全てのデータのパスを指定しておきます．今後，適宜呼び出します．

画像の特徴抽出

画像からキャプション生成を行うにあたっては，各画像の特徴ベクトルに対応してキャプションを学習・生成します．そのために，まず最初に CNN に画像を通して，最終層でクラス分類される手前で出力される特徴量を抽出します．そのためのコードが以下になります．

```python
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16,preprocess_input
from keras.preprocessing.image import load_img,img_to_array
from keras.models import Model

# 指定したディレクトリ内の各写真から特徴を抽出する関数
def extract_features(directory):
    model = VGG16()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    features = dict() #特徴を格納するためのディクショナリ
    for name in listdir(directory):
        filename = directory + '/' + name #ファイルから画像を読み込む
        image = load_img(filename, target_size=(224, 224)) #VGG16用に224*
        image = img_to_array(image) #numpy配列に変換
        image = image.reshape((1, image.shape[0],
        image.shape[1], image.shape[2])) #モデルに読み込ませるために成形
        image = preprocess_input(image) #VGGモデルに画像を読み込ませる
        feature = model.predict(image, verbose=0) #特徴抽出
        image_id = name.split('.')[0] #画像の名前を取得
        features[image_id] = feature #画像の名前と特徴を紐付け
    return features

#特徴抽出
features = extract_features(image_directory)

#特徴をpklファイルとして保存
dump(features, open('features.pkl', 'wb'))
```

今回は特徴抽出用の CNN モデルとして，VGG16 を用いています．VGG16 の最終層を取り払い，画像を VGG16 に入力できる形に成形し，モデルに通して抽出した特徴と画像名を紐づけて，全ての画像のデータを pkl ファイルとして保存します．今後，文章の学習・生成をする際には，これらの特徴量を初期入力とします．なお，このようにすると学習時に VGG16 モデルそのものの学習は行われませんが，学習できるように設定してもあまり効果的でないことが知られています．

キャプションの読み込みと成形

```python
#ファイルを読み込む関数
def load_doc(filename):
    file = open(filename, 'r', encoding='utf-8')
    text = file.read()
    file.close()
    return text

#キャプションデータの読み込み
doc = load_doc(caption_data)
```

まずはファイルの読み込みのための関数を定義し，全てのキャプションを読み込みます．この関数は今後も適宜使用します．

```
#キャプションと画像名を紐づけする関数
def load_descriptions(doc):
    mapping = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(line) < 2:
            continue
        image_id, image_desc = tokens[0], tokens[1:] #最初の単語を画像名、残
        image_id = image_id.split('.')[0] #ピリオドより手前を画像名とする
        image_desc = ' '.join(image_desc) #キャプションの単語を文字列に戻す
        if image_id not in mapping: #その画像名が一つ目ならリストを作成
            mapping[image_id] = list()
        mapping[image_id].append(image_desc) #画像名にキャプションを紐づけてデ
    return mapping

#キャプションと画像の紐づけ
descriptions = load_descriptions(doc)
```

つづいてディクショナリを定義し，各画像とキャプションを紐付けています．

```
import string

#余計な記号を除去する関数
def clean_descriptions(descriptions):
    table = str.maketrans('', '', string.punctuation)#記号をリストアップ
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            desc = desc.split() #キャプションを単語に区切る
            desc = [w.translate(table) for w in desc] #記号を消去
            desc_list[i] = ' '.join(desc) #キャプションの単語を文字列に戻す

#余計な記号を除去する
clean_descriptions(descriptions)
```

今回用いるキャプションの中には，文頭文末のダブルクォーテーションなど，不要な記号が含まれています．これらをここで除去します．string.punctuation で［!”#$%&'()*+,-./:;<=>?@[¥]^_`{|}~］←これらをまとめて指定可能です．

```
#語彙が縮小されたキャプションを保存する関数
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w', encoding='utf-8')
    file.write(data)
    file.close()

#語彙が縮小されたキャプションをtxtファイルとして保存
save_descriptions(descriptions, 'descriptions.txt')

#画像数のチェック
print('Loaded: %d ' % len(descriptions))
```

語彙が縮小されたキャプションは，descriptions.txt として新たに保存されます．

最後に，画像数のチェックを行っています．今回は 2000 枚あるので，2000 と出力されると思います．

トレーニングデータとバリデーションデータの準備

ここでは，トレーニングデータとバリデーションデータを，いくつかの段階を踏まえて，学習・評価に使える形に成形します．

```
#データセットの画像名のリストを作成する関数
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    for line in doc.split('\n'):
        if len(line) < 1:
            continue
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)

#トレーニングデータの画像名のリスト作成
train = load_set(train_data)
#バリデーションデータの画像名のリスト作成
val = load_set(val_data)
```

まず，上記の関数を使って，トレーニングデータとバリデーションデータの画像名が格納されたリストを作成しています．なお，引数に用いている train_data と val_data は，本サンプルコードの一番最初に指定したものです．

```
#画像名とキャプションを紐付けたディクショナリを作成する関数
def load_clean_descriptions(filename, dataset):#引数datasetはtrainとかvalとか
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):#一行ずつ読み込む
        tokens = line.split() #空白で区切る
        image_id, image_desc = tokens[0], tokens[1:] #最初の単語を画像名、残り全てをキャプションとして読み込む
        if image_id in dataset: #画像名がデータセット中に指定されていれば以下を実行
            if image_id not in descriptions: #その画像名が一つ目ならリストを作成
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq' #キャプションを開始語と終了語で囲む
            descriptions[image_id].append(desc) #ディクショナリに格納
    return descriptions

#トレーニングデータのキャプションと画像名を紐付ける
train_descriptions = load_clean_descriptions('descriptions.txt', train)
#バリデーションデータのキャプションと画像名を紐付ける
val_descriptions = load_clean_descriptions('descriptions.txt', val)
```

上記の関数の引数 filename には先ほど作成したファイル descriptions.txt を指定し，引数 dataset には先ほど作成したリスト train およびリスト val を指定します．この関数を使って，トレーニングデータとバリデーションデータの画像名とキャプションが紐付けられたディクショナリを作成しています．

ここで，各キャプションを何らかの開始語と終了語で挟みます（ここでは startseq および endseq としています）．文章生成の際には，画像と開始語を入力として与えて，そこから続く単語を次々と予測し，終了後が出力されたら文章生成を打ち止めます．

```
from pickle import load
#画像の特徴量を読み込む関数
def load_photo_features(filename, dataset):
    all_features = load(open(filename, 'rb'))
    features = {k: all_features[k] for k in dataset}#画像名と特徴量を紐づけてディクショナリに格納
    return features

#トレーニングデータの特徴量と画像名を紐付ける
train_features = load_photo_features('features.pkl', train)
#バリデーションデータの特徴量と画像名を紐付ける
val_features = load_photo_features('features.pkl', val)
```

上記の関数の引数 filename には最初に作成したファイル features.pkl を指定し，引数 dataset には先ほど作成したリスト train およびリスト val を指定します．この関数を使って，トレーニングデータとバリデーションデータの画像名と特徴量が紐付けられたディクショナリを作成しています．

```
from keras.preprocessing.text import Tokenizer

#キャプションのディクショナリをリストにする関数
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#キャプションをKerasのTokenizerで扱うために変換する
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

#tokenizerを準備する
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
```

機械学習において文字列を扱うためには、単語をベクトル表現する必要があります．単語をベクトル表現することで，各単語間の関係性を学習することが可能になります．Keras には Tokenizer という便利なものがあり，これを使うことで単語をベクトル配列に変換することができます．ここではまず最初にキャプションのリストを生成し，これをもとにベクトル化します．なお，キャプションのリストには train_descriptions を指定するので，トレーニングデータのキャプションに含まれている単語のみベクトル化されます．

```
#最も多くの単語を含むキャプションの長さを計算する関数
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

#最大シーケンス長を計算する
max_length = max_length(train_descriptions)
```

次に，トレーニングデータの文章のうち，最大の文の長さを取得します．これは，モデルの入力層のノード数は一定にしなくてはいけないので，それをトレーニングデータの文の最大長に合わせるためです．

```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from numpy import array
#画像と出力単語を紐づける関数
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()#X1が入力画像、X2が入力語、yがX1とX2に対応する出力語
    #各画像名でループ
    for key, desc_list in descriptions.items():
        #各画像のキャプションでループ
        for desc in desc_list:
            #シーケンスをエンコードする
            seq = tokenizer.texts_to_sequences([desc])[0]
            #1つのシーケンスを複数のX、Yペアに分割する
            for i in range(1, len(seq)):
                #入力と出力のペアに分割する
                in_seq, out_seq = seq[:i], seq[i]
                #行列のサイズを最大の単語数に合わせる
                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                #出力シーケンス
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                #全てをarrayに格納
                X1.append(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)

#トレーニングデータの入力画像、入力語、出力語を紐付ける
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)
#バリデーションデータの入力画像、入力語、出力語を紐付ける
X1val, X2val, yval = create_sequences(tokenizer, max_length, val_descriptions, val_features)
```

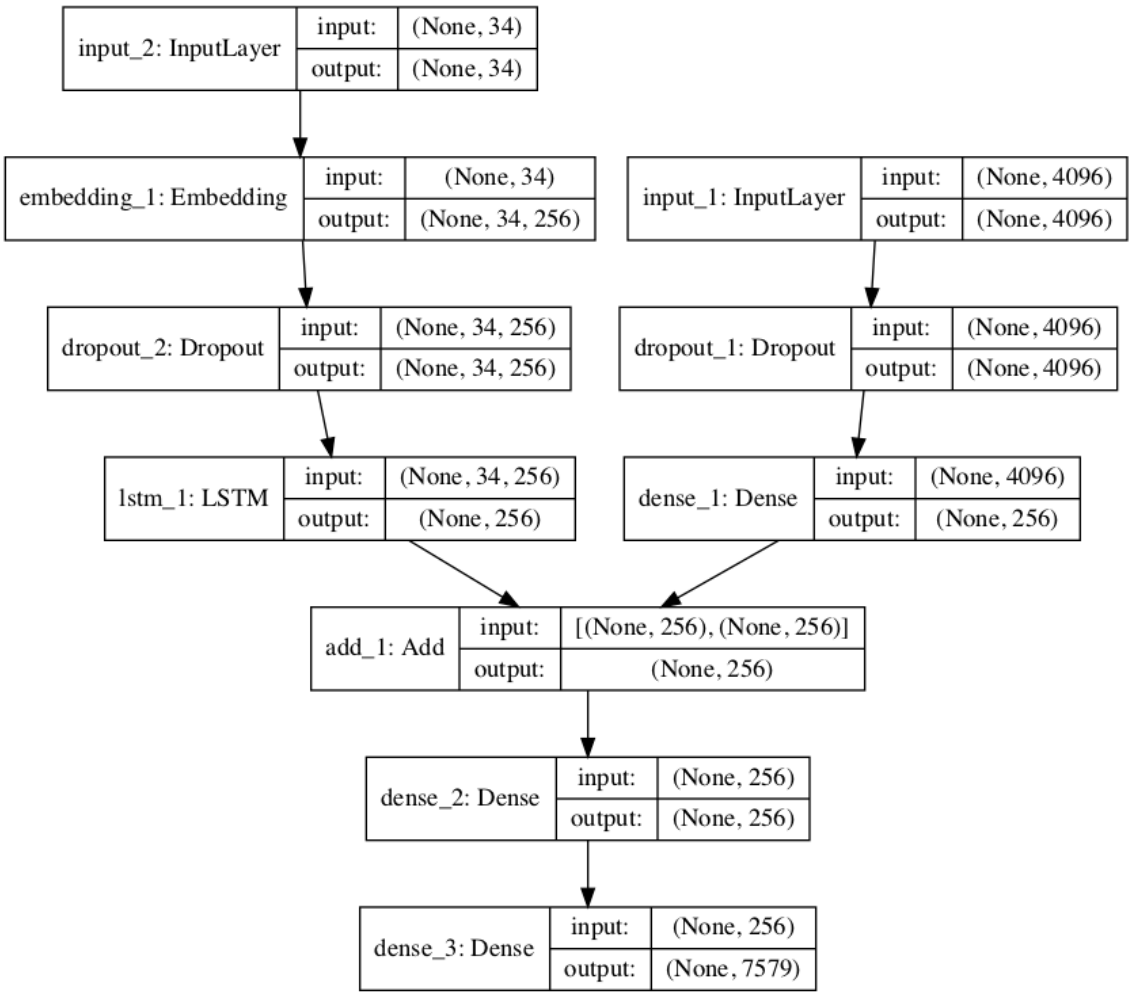
そして最後に，キャプションデータ，画像データを紐づけ，モデルに入力するためのデータセットとします．ここでは，画像とそれに対応する入力語，入力語から予測されるべき出力語を，そのすべてのパターンにおいて設定します．以上でモデルに入力できる形のデータの準備は完了です．

モデルの構築

さて，ここからやっとメインのモデル構築です．

```
from keras.layers import Input,Dense,LSTM,Embedding,Dropout
from keras.layers.merge import add
#モデルを定義する関数
def define_model(vocab_size, max_length):
    #画像の特徴を入力するレイヤ
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    #文章を入力するレイヤ
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    #上の二つの出力を統合する部分
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    #モデルの定義．二つを入力にとって一つを出力する形になる
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    return model
```

今までのモデルは単一のレイヤが一列に積み重なっていただけでしたが，今回のモデルは以下のような構造になっています．



図－４０　モデルの構造

この構造を見て目につくように，このモデルは入力を 2 つとって 1 つの出力としています．すなわち，input1 から画像（の特徴量），input2 から文章を入力し，output で続く単語を出力します．また，時系列データを処理する LSTM レイヤは，input2 からなる層に配置されていることが確認できます．

input1 の入力ノード 4096 は, VGG16 の出力した特徴量を受け取ることからです. また，input2 の入力ノード 34 は，トレーニングデータにおける文章の最大長です．input2 には，それまで生成された文章が入力として与えられるのですが，その文章を最大長に合わせてパディングして入力として与えます．output の 7579 は，出力されうる単語数を示しています．当たり前ですが，学習していない語彙は出力することはできません．（なお，34 や 7579 というのは一例で，今回のデータセットでは，また違った数値になっていると思います．）では，定義したモデルを使って学習を行います．

```
from keras.callbacks import ModelCheckpoint

#モデルの定義
model = define_model(vocab_size, max_length)
#コールバックを定義する
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
#学習
model.fit([X1train, X2train], ytrain, epochs=10, verbose=2, callbacks=[checkpoint], validation_data=([X1val, X2val], yval))
```

今回は Keras のコールバックを定義しています．エポックごとに val_loss をモニターすることで，val_loss がそれまでの最小値だったらそのときの重みのモデルをファイルとして保存します．

テストする際や，新規に画像のキャプションを生成する際は，最後に出力されたモデルを読み込み実施するだけで良いです．データ量が多いので，実行には少し時間がかかると思います．上記のコードを実行すると，以下のような結果が出力されると思います．

```
Train on 108400 samples, validate on 25833 samples
Epoch 1/10
- 188s - loss: 4.0848 - val_loss: 3.4455

Epoch 00001: val_loss improved from inf to 3.44553, saving
model to model-ep001-loss4.085-val_loss3.446.h5

～省略～

Epoch 00005: val_loss improved from 3.13871 to 3.13543, saving
model to model-ep005-loss2.581-val_loss3.135.h5

～省略～

Epoch 10/10
- 184s - loss: 2.2064 - val_loss: 3.3320

Epoch 00010: val_loss did not improve from 3.13543
```

エポックの val_loss が最小の場合は新しくモデルを保存しているのが分かります．それでは，テストデータの 1 つを使って実際に文章を生成してみます．

```
#ディレクトリ内の写真から特徴を抽出する
def extract_features_from_image(filename):
    model = VGG16()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    image = load_img(filename, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    feature = model.predict(image, verbose=0)
    return feature

photo = extract_features_from_image('path/to/test.image')#テスト画像のパスを指定すること
```

まず，本サンプルコードで最初に実行した，全ての写真から特徴を抽出する関数を少し書き換え，個別の画像から特徴量を抽出する関数を定義します．この関数を使って，テスト画像の特徴量を抽出します．ここで，テスト画像のパスを指定してください．

```
from numpy import argmax
from keras.models import load_model

#整数を単語に変換する関数
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

#画像からキャプションを生成する関数
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text

model = load_model('model-ep005-loss2.581-val_loss3.135.h5')#最後に出力されたモデルを指定する
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

続いて，整数を単語に戻す関数を定義します．モデルを学習する際に，各単語は番号に置き換えているので，実際に出力される際にも番号で出力されます．これを，この関数で対応する単語に戻すことができます．

その次に定義しているのが，実際に画像からキャプションを生成する関数です．

yhat = model.predict([photo,sequence], verbose=0)の部分を見ると，画像（の特徴量）とそれまでに生成された文章を入力していることが分かります．なお，一番最初に入力されるのは開始語（この場合は startseq）です．

結果としては各単語の確率が出力されるのですが，yhat = argmax(yhat)で，最も確率が高い単語を選択します．そして，in_text += ‘ ‘ + word の部分で，それまでの文章と新たに出力された文章を繋げます．これを，次のループで新たな入力として使います．これを終了語が出力されるまで繰り返します．すなわちここでは，写真と，それまでに生成された文章から，次に来る確率が最も高い単語を繰り返し出力して文章を生成しているということです．

最後に，今回学習によって得られたモデルを変数 model に定義します．上記のコードは一例で，実際には最後に出力されたモデルのパスを入力してください．そして，キャプション生成の関数を実行し，その結果を表示します．以下のような結果が表示されました．（シード値の固定などは行っていないので，人によって答えは少し違うかもしれません．）

```
startseq 男性 が スキー を し て ジャンプ し て いる endseq
```

図－４１　男性がサーフィンしている画像

残念ながらこれは間違った文章です．スキーじゃなくてサーフィンだったらそこそこ良かったのですが．とはいえ，日本語として間違っているレベルのおかしな文章ではないですし，モデルを調整するか，データ量を増やすことで，精度向上の期待が持てそうです．

追記

今回のコードは説明を簡単にするために，学習データを小分けにするミニバッチ学習を行っていないので，これよりさらにデータ量を増やすとメモリエラーが発生する可能性があります．（環境によっては，今回のデータ量でもメモリエラーが発生するかもしれません．）ミニバッチ学習の方法をこの記事に追加するとあまりに内容が膨大になるので，次回以降に説明したいと思います．

また，今回の出力では最も確率の高い単語を順次つなげていったので，文章が 1 文のみ出力されました．とはいえ，場合によっては 2 文，3 文出力すれば，1 文では言い表せなかった内容も出力できる可能性があります．こちらについても，次回以降に併せて説明したい

と思います.

d）コード全体

以下に，今回のコード全文を示します.

データの準備

```python
import json
import pandas as pd
import numpy as np

f = open("stair_captions_v1.2_val_tokenized.json", 'r', encoding='utf-8')
json_data = json.load(f)

id_file = {}
for i in range(len(json_data["images"])):
    file_name = json_data["images"][i]["file_name"]
    id = json_data["images"][i]["id"]
    id_file[id] = file_name

annotations_size = len(json_data["annotations"])
df = pd.DataFrame(data = np.zeros((annotations_size,2)))

for j in range(annotations_size):
    image_id = json_data["annotations"][j]['image_id']
    file_name = id_file[image_id]
    caption = json_data["annotations"][j]["tokenized_caption"]
    df.loc[j, 0] = file_name
    df.loc[j, 1] = caption

data_size = 10000
df.sort_values(0, inplace=True)
df= df.reset_index(drop=True)
df= df.loc[:data_size-1]

df.to_csv("caption.txt", sep=" ", header=False, index=False, encoding='utf-8')
```

上のコードは，STAIR Lab のデータセットを，扱いやすいように成形するコードです.

```python
# -*- coding: utf-8 -*-
import pandas as pd
import os
import shutil

file = open('path/to/caption.txt', 'r', encoding='utf-8')
caption_data = file.read()
file.close()


image_list = []
for line in caption_data.split('\n'):
    tokens = line.split()
    if len(line) < 2:
        continue
    image_id, image_desc = tokens[0], tokens[1:]
    if image_id not in image_list:
        image_list.append(image_id)

os.mkdir('path/to/photo_data')
for list in image_list[:-1]:
    shutil.copy('path/to/val2014/{}'.format(list),
    'path/to/photo_data')

os.mkdir('path/to/test_data')
shutil.copy('path/to/val2014/{}'.format(image_list[-1]), 'path/to/test_data')

train_data = pd.Series(image_list[:1600])
val_data = pd.Series(image_list[1600:1999])

train_data.to_csv("train.txt", header=False, index=False)
val_data.to_csv("val.txt", header=False, index=False)
```

上のコードは，成形したキャプションデータをもとに，画像をフォルダ分けして，トレーニングデータとバリデーションデータの画像名を指定したファイルを出力するコードです．

モデル構築

```python
# -*- coding: utf-8 -*-
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16,preprocess_input
from keras.preprocessing.image import load_img,img_to_array
from keras.models import Model
import string
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from numpy import array
from keras.layers import Input,Dense,LSTM,Embedding,Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint

image_directory = 'path/to/photo_data'#画像データのパスを指定すること
caption_data = 'path/to/caption.txt'#キャプションデータのパスを指定すること
train_data = 'path/to/train.txt'#トレーニングデータのパスを指定すること
val_data = 'path/to/val.txt'#バリデーションデータのパスを指定すること

# 指定したディレクトリ内の各写真から特徴を抽出する関数
def extract_features(directory):
    model = VGG16()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    features = dict() #特徴を格納するためのディクショナリ
    for name in listdir(directory):
        filename = directory + '/' + name #ファイルから画像を読み込む
        image = load_img(filename, target_size=(224, 224)) #VGG16用に224×224に成形
        image = img_to_array(image) #numpy配列に変換
        image = image.reshape((1, image.shape[0],
        image.shape[1], image.shape[2])) #モデルに読み込ませるために成形
        image = preprocess_input(image) #VGGモデルに画像を読み込ませる
        feature = model.predict(image, verbose=0) #特徴抽出
        image_id = name.split('.')[0] #画像の名前を取得
        features[image_id] = feature #画像の名前と特徴を紐付け
    return features

#特徴抽出
features = extract_features(image_directory)

#特徴をpklファイルとして保存
dump(features, open('features.pkl', 'wb'))

#ファイルを読み込む関数
def load_doc(filename):
    file = open(filename, 'r', encoding='utf-8')
    text = file.read()
    file.close()
    return text

#キャプションデータの読み込み
doc = load_doc(caption_data)

#キャプションと画像名を紐づけする関数
def load_descriptions(doc):
    mapping = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(line) < 2:
            continue
        image_id, image_desc = tokens[0], tokens[1:] #最初の単語を画像名、残り全てをキャプションとして読み込む
        image_id = image_id.split('.')[0] #ピリオドより手前を画像名とする
        image_desc = ' '.join(image_desc) #キャプションの単語を文字列に戻す
        if image_id not in mapping: #その画像名が一つ目ならリストを作成
            mapping[image_id] = list()
        mapping[image_id].append(image_desc) #画像名にキャプションを紐づけてディクショナリに格納
    return mapping

#キャプションと画像の紐づけ
descriptions = load_descriptions(doc)
```

```
#余計な記号を除去する関数
def clean_descriptions(descriptions):
    table = str.maketrans('', '', string.punctuation)#記号をリストアップ
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            desc = desc.split() #キャプションを単語に区切る
            desc = [w.translate(table) for w in desc] #記号を消去
            desc_list[i] = ' '.join(desc) #キャプションの単語を文字列に戻す

#余計な記号を除去する
clean_descriptions(descriptions)

#語彙が縮小されたキャプションを保存する関数
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w', encoding='utf-8')
    file.write(data)
    file.close()

#語彙が縮小されたキャプションをtxtファイルとして保存
save_descriptions(descriptions, 'descriptions.txt')

#画像数のチェック
print('Loaded: %d ' % len(descriptions))

#データセットの画像名のリストを作成する関数
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    for line in doc.split('\n'):
        if len(line) < 1:
            continue
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)

#トレーニングデータの画像名のリスト作成
train = load_set(train_data)
#バリデーションデータの画像名のリスト作成
val = load_set(val_data)

#画像名とキャプションを紐付けたディクショナリを作成する関数
def load_clean_descriptions(filename, dataset):#引数datasetはtrainとかvalとか
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):#一行ずつ読み込む
        tokens = line.split() #空白で区切る
        image_id, image_desc = tokens[0], tokens[1:] #最初の単語を画像名、残り全てをキャプションとして読み込む
        if image_id in dataset: #画像名がデータセット中に指定されていれば以下を実行
            if image_id not in descriptions: #その画像名が一つ目ならリストを作成
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq' #キャプションを開始語と終了語で囲む
            descriptions[image_id].append(desc) #ディクショナリに格納
    return descriptions

#トレーニングデータのキャプションと画像名を紐付ける
train_descriptions = load_clean_descriptions('descriptions.txt', train)
#バリデーションデータのキャプションと画像名を紐付ける
val_descriptions = load_clean_descriptions('descriptions.txt', val)

#画像の特徴量を読み込む関数
def load_photo_features(filename, dataset):
    all_features = load(open(filename, 'rb'))
    features = {k: all_features[k] for k in dataset}#画像名と特徴量を紐づけてディクショナリに格納
    return features
```

```

#トレーニングデータの特徴量と画像名を紐付ける
train_features = load_photo_features('features.pkl', train)
#バリデーションデータの特徴量と画像名を紐付ける
val_features = load_photo_features('features.pkl', test)

#キャプションのディクショナリをリストにする関数
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#キャプションをKerasのTokenizerで扱うために変換する
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

#tokenizerを準備する
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1

#最も多くの単語を含むキャプションの長さを計算する関数
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

#最大シーケンス長を計算する
max_length = max_length(train_descriptions)

#画像と出力単語を紐づける関数
def create_sequences(tokenizer, max_length, descriptions, photos):
    X1, X2, y = list(), list(), list()#X1が入力画像、X2が入力語、yがX1とX2に対応する出力語
    #各画像名でループ
    for key, desc_list in descriptions.items():
        #各画像のキャプションでループ
        for desc in desc_list:
            #シーケンスをエンコードする
            seq = tokenizer.texts_to_sequences([desc])[0]
            #1つのシーケンスを複数のX, Yペアに分割する
            for i in range(1, len(seq)):
                #入力と出力のペアに分割する
                in_seq, out_seq = seq[:i], seq[i]
                #行列のサイズを最大の単語数に合わせる
                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                #出力シーケンス
                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                #全てをarrayに格納
                X1.append(photos[key][0])
                X2.append(in_seq)
                y.append(out_seq)
    return array(X1), array(X2), array(y)

#トレーニングデータの入力画像、入力語、出力語を紐付ける
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features)
#バリデーションデータの入力画像、入力語、出力語を紐付ける
X1val, X2val, yval = create_sequences(tokenizer, max_length, val_descriptions, val_features)
```

```

#モデルを定義する関数
def define_model(vocab_size, max_length):
    #画像の特徴を入力するレイヤ
    inputs1 = Input(shape=(4096,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
    #文章を入力するレイヤ
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)
    #上の二つの出力を統合する部分
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    #モデルの定義。二つを入力にとって一つを出力する形になる
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    return model

#モデルの定義
model = define_model(vocab_size, max_length)
#コールバックを定義する
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
#学習
model.fit([X1train, X2train], ytrain, epochs=10, verbose=2, callbacks=[checkpoint], validation_data=([X1val, X2val], yval))
```

テストデータでキャプション生成

```
from numpy import argmax
from keras.models import load_model

#ディレクトリ内の写真から特徴を抽出する
def extract_features_from_image(filename):
    model = VGG16()
    model.layers.pop()
    model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
    image = load_img(filename, target_size=(224, 224))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    image = preprocess_input(image)
    feature = model.predict(image, verbose=0)
    return feature

photo = extract_features_from_image('path/to/test.image')#テスト画像のパスを指定すること

#整数を単語に変換する関数
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

#画像からキャプションを生成する関数
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo,sequence], verbose=0)
        yhat = argmax(yhat)
        word = word_for_id(yhat, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text

model = load_model('model-ep005-loss2.581-val_loss3.135.h5')#最後に出力されたモデルを指定する
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

上記のコードは，テストデータを使ってキャプションを生成するコードです．（今回使ったテストデータ以外にも，自前のほかの画像でもすぐにキャプション生成できるので，色々と試してみてください．）

Jupyter notebook などを使っている場合，学習に引き続いてこのコードを入力すれば，モジュールのインポートは上記のものだけで大丈夫です．しかし，別々の py ファイルとして扱う場合は，モジュールを再度インポートするほか，tokenizer を別途ファイル保存して読み込みする必要があります．

2.1.18 ひび割れ検出チュートリアル

土木構造物の点検においてひび割れ評価は重要な指標である．その一方で，現在では手作業＋スケッチで調べられており，膨大な手間がかかっている．そこでひび割れ検出を AI で自動化しようという研究は多い．

ここでは，基本的な手法に絞り，ひび割れ検出手法のソースコードを作ったので公開する．本プログラムはあくまでも理解用の基本的なものであるが，本プログラムを発展させれば（手法を最新のものにするなど），さらなる精度向上も期待できる．

プログラムのコードについては小委員会の HP にリンクを掲載しているが，ここではコードの内容について紹介する．

はじめに

このチュートリアルでは、DeepCrack データセットを使って、学習済みの Mask R-CNN モデルを転移学習・ファインチューニングします。

Mask R-CNN は物体検出・領域分割を行う深層学習モデルであり、DeepCrack データセットはおよそ 700 件のコンクリート画像とそのひび割れに対応した二値マスクを含んでいます。これらを使ってコンクリート構造物のひび割れを探すというタスクの自動化を試みます。

まずはじめに、pycocotools をインストールします。モデルの性能評価には、ひび割れとして予測された領域と正解領域の重なりを指標として使いますが、こちらはそれを一般的な COCO 形式に準拠した形で測定するために必要なツールとなります。

```
%%shell
pip install cython
pip install -U 'git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI'
```

コンクリートひび割れデータのためにカスタムクラスを準備する

DeepCrack データセットのために、カスタムのデータセットクラスを記述してみましょう。

まずはじめに、データセットを Github からダウンロードします。

※ 当該データセットは非商用・教育目的の利用に限って使用を許可されていますのでご注意ください。

```
%%shell
# DeepCrack データセットをダウンロードする
mkdir ./DeepCrack
wget https://github.com/yhlleo/DeepCrack/raw/master/dataset/DeepCrack.zip

# "DeepCrack" フォルダに解凍する
unzip -q DeepCrack.zip -d ./DeepCrack
```

データセットの中身を見てみましょう。
データは以下のようなフォルダ構成になっていると思います。

```
DeepCrack/
  train_img/
    11111.jpg
    11112.jpg
    11113.jpg
    11114.jpg
    ...
  train_lab/
    11111.png
    11112.png
    11113.png
    11114.png
```

どんな画像が入っているのか、一つ見てみることにします。

```
[ ] from PIL import Image
    Image.open('DeepCrack/train_img/11115.jpg')
```

```
Image.open('DeepCrack/train_lab/11115.png')
```

一つ一つの画像にひび割れに対応するマスクが存在していることがわかります。

torch.utils.data.Dataset に沿った形でデータセットのクラスを記述してみます。

```
import os
import torch
import numpy as np
from numpy.lib.function_base import delete
from PIL import Image
from PIL.Image import Image as Pilimg
from typing import Tuple
```

```
class ConcreteCrackDataset(object):
    def __init__(self, root:str, transforms=None):
        self.transforms = transforms

        # 全ての画像を読み込み、すべてそろっていることを確認するためにソートする
        self.img_dir = os.path.join(root, "train_img")
        self.mask_dir = os.path.join(root, "train_lab")

        self.masks = list(sorted(os.listdir(self.mask_dir)))
        self.imgs = list(sorted(os.listdir(self.img_dir)))

    def __getitem__(self, idx:int) -> Tuple[Pilimg, dict]:
        # 画像とマスクを読み込む
        img_path = os.path.join(self.img_dir, self.imgs[idx])
        img = Image.open(img_path).convert("RGB")

        mask_path = os.path.join(self.mask_dir, self.masks[idx])
        mask = Image.open(mask_path)
        mask = np.array(mask)

        # 複数クラスがある場合の対応
        obj_ids = np.unique(mask)
        # 最初のクラスIDは背景なので削除する
        obj_ids = obj_ids[1:]

        # 色で分けられたマスクの場合は、二値化する
        masks = mask == obj_ids[:, None, None]

        # マスクごとの矩形領域を計算する
        num_objs = len(obj_ids)
        boxes = []
        for i in range(num_objs):
            pos = np.where(masks[i])
            xmin = np.min(pos[1])
            xmax = np.max(pos[1])
            ymin = np.min(pos[0])
            ymax = np.max(pos[0])
            boxes.append([xmin, ymin, xmax, ymax])

        boxes = torch.as_tensor([b for b in boxes if self._valid_bbox(b)], dtype=torch.float32)
        # 検出対象のクラスは一つ（ひび割れ）だけ
        labels = torch.ones((num_objs,), dtype=torch.int64)
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        image_id = torch.tensor([idx])
        try:
            area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])
        except:
            print("You need to remove this sample: ", img_path, boxes)
            area = None
        # 全部のクラスが iscrowd ではない形にする
        iscrowd = torch.zeros((num_objs,), dtype=torch.int64)
```

```
            iscrowd = torch.zeros((num_objs,), dtype=torch.int64)

            target = {}
            target["boxes"] = boxes
            target["labels"] = labels
            target["masks"] = masks
            target["image_id"] = image_id
            target["area"] = area
            target["iscrowd"] = iscrowd

            if self.transforms is not None:
                img, target = self.transforms(img, target)

            return img, target

        def _valid_bbox(self, box):
            return box[0] < box[2] and box[1] < box[3]

        def __len__(self):
            return len(self.imgs)
```

データセットの定義は以上です。出力されるデータがどんな構造になっているか、確認してみましょう。

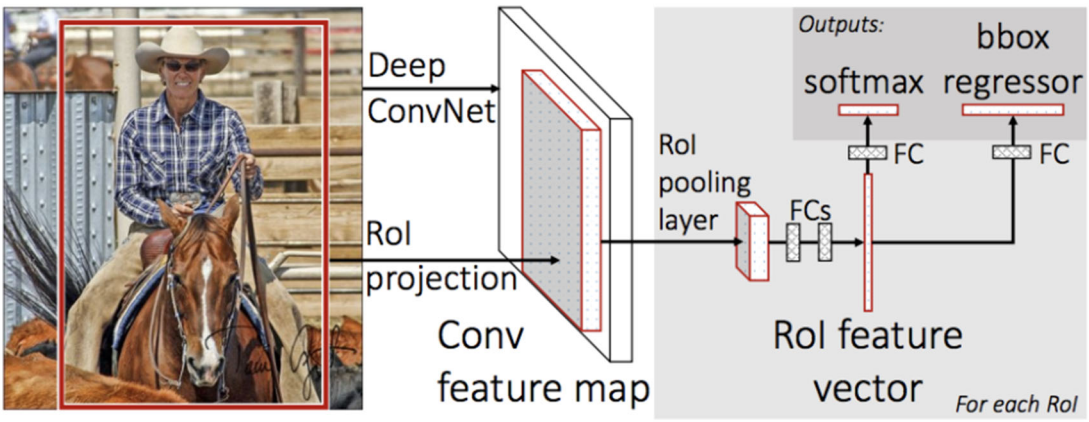
```
[ ] dataset = ConcreteCrackDataset('DeepCrack')
    dataset[0]
```

```
(<PIL.Image.Image image mode=RGB size=544x384 at 0x7FB4B9B88210>,
 {'area': tensor([61359.]),
  'boxes': tensor([[  0., 112., 543., 225.]]),
  'image_id': tensor([0]),
  'iscrowd': tensor([0]),
  'labels': tensor([1]),
  'masks': tensor([[[0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           ...,
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0],
           [0, 0, 0,  ..., 0, 0, 0]]], dtype=torch.uint8)})
```

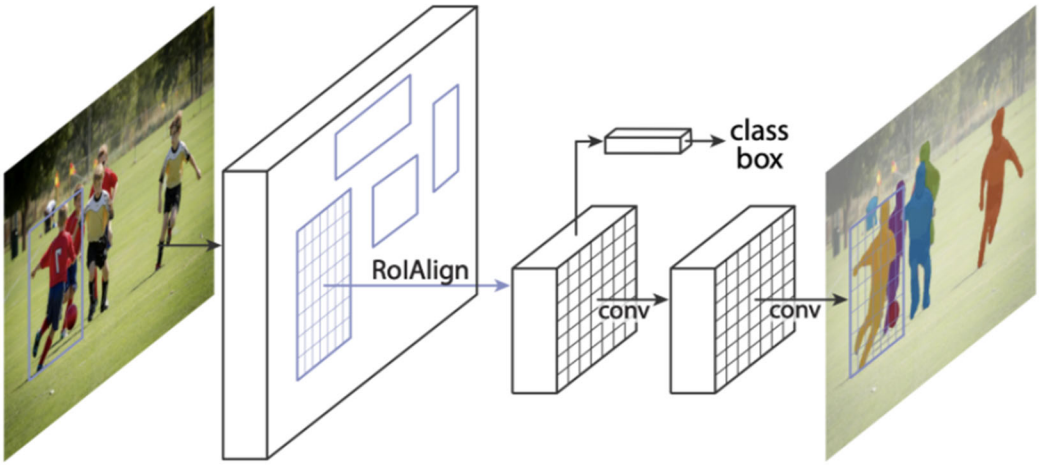
データセットのクラスは `PIL.Image` と、 `boxes`, `labels` や `masks` といった様々なフィールドを含む辞書を戻り値として返していることがわかります。

モデルを定義する

本編で扱う Mask R-CNN は Faster R-CNN を基にしています。Faster R-CNN は画像内に矩形領域を探し、その領域が囲む検出対象が何かを予測します。



Mask R-CNNは論理的な分岐をFaster R-CNNに追加することで、それぞれの対象に対して領域分割マスクも同時に予測するという特徴があります。



Torchvisionのmodelzooには様々なプリセットのモデルが含まれていますが、これを改変して使う目的としては、大きく二つ考えられます。

一つはすでに学習済みのモデルの最終層をファインチューニングするという用途ですが、それ以外にも、モデルのバックボーンを別のものに取り換えて使うという用途にも活用可能です。

少し詳しく見ていきます。

学習済みモデルをファインチューニングする

COCO データセットで学習済みのモデルの最終層だけを取り換え、自分のデータセットで転移学習させたい状況であれば、以下の形が考えられます。

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

# COCOデータセットで学習済みのモデルをロードする
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

# 分類器（最終の全結合層）を新しいものに取り換え
# num_classes を自分のデータセットのクラス数に変更する
num_classes = 2  # 1 class（ひび割れ）+ 背景

# 分類器への入力の次元数を取得する
in_features = model.roi_heads.box_predictor.cls_score.in_features

# 学習済みのモデルのヘッドを新しいものに取り換える
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
```

別のバックボーンを追加して物体検出モデルを改変する

さらにもう一つ、一般的なケースとして物体検出モデルのバックボーンを取り換えたいという状況が考えられます。例えば、現状のバックボーン(ResNet-50)が大きすぎ、もっと小さいモデルに付け替えたいという場合などがあり得ます。

以下は、torchvision にもともとある機能を活用してバックボーンを変更するやり方です。

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# 学習済みの画像分類モデルを呼び出し、特徴抽出部分だけを
# 取り出す
backbone = torchvision.models.mobilenet_v2(pretrained=True).features

# FasterRCNN にバックボーンの出力次元数を伝える必要がある
# mobilenet_v2 の場合、1280次元なのでそれを記述する
backbone.out_channels = 1280

# 画面上の位置ごとに5 x 3のアンカーを生成するRPN（領域提案ネットワーク）
# を5つの違うサイズ、3つの違う縦横比で作成する。
# Tuple[Tuple[int]] にしているのは、それぞれの特徴マップが別のサイズ、
# 別の縦横比になってしまう可能性があるため
anchor_generator = AnchorGenerator(sizes=((32, 64, 128, 256, 512),),
                                   aspect_ratios=((0.5, 1.0, 2.0),))


# 重複領域の切り取り、リサイズ後の切り取りを次項する特徴マップを
# 定義します。
# もし上記で定義したバックボーンが Tensor を戻り値として返す場合、
# featmap_names は [0] でなければいけません。一般的にバックボーンの
# 戻り値は単一の OrderedDict[Tensor] であり、 featmap_names で
# どの特徴マップを使うかを指定することができます。
roi_pooler = torchvision.ops.MultiScaleRoIAlign(featmap_names=[0],
                                                output_size=7,
                                                sampling_ratio=2)

# 全てを一つの FasterRCNN モデルにまとめます
model = FasterRCNN(backbone,
                   num_classes=2,
                   rpn_anchor_generator=anchor_generator,
                   box_roi_pool=roi_pooler)
```

DeepCrack データセットを使ったインスタンス・セグメンテーション（領域抽出）

今回はデータセットが小さいため、上の例の 1 番に沿った形で学習済みのモデルをファインチューニングします。また、領域を抽出したいため、Faster R-CNN ではなく Mask R-CNN を使います。

```
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
from torchvision.models.detection.mask_rcnn import MaskRCNNPredictor


def get_instance_segmentation_model(num_classes):
    # COCOで学習済みのインスタンス・セグメンテーションモデルを読み込む
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    # 分類器の特徴量の次元数を取得する
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # 学習済みのモデルのヘッドを新しいものに取り換える
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # マスク用の分類器への入力となる、特徴量の次元数を取得する
    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    # マスク推論部分を新しいものと入れ替える
    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                       hidden_layer,
                                                       num_classes)

    return model
```

以上で、用意したカスタムデータセットを使う、モデルのトレーニングと評価の準備が整いました。

モデルトレーニングと評価関数

Torchvision の references/detection/ には、物体検出モデルのトレーニングと性能評価をより容易にする機能が色々と含まれています。 今回は、 references/detection/engine.py、references/detection/utils.py と references/detection/transforms.py を使用します。
まず、そういったファイルと必要なライブラリを使えるようコピーしていきます。

```
%%shell

# TorchVision の references/detection を使うため
# 一部のファイルをコピーする
git clone https://github.com/pytorch/vision.git
cd vision
git checkout v0.3.0

cp references/detection/utils.py ../
cp references/detection/transforms.py ../
cp references/detection/coco_eval.py ../
cp references/detection/engine.py ../
cp references/detection/coco_utils.py ../
```

その次に、コピーした reference/detection を基に、データ増強（オーギュメンテーション）やデータ変換の補助関数を書いてみましょう。

```
from engine import train_one_epoch, evaluate
import utils
import transforms as T


def get_transform(train):
    transforms = []
    # PIL image の画像を PyTorch Tensor に変換する
    transforms.append(T.ToTensor())
    if train:
        # トレーニング時にデータ増強を行うため、ランダムに入力画像と
        # 正解を反転させる
        transforms.append(T.RandomHorizontalFlip(0.5))
    return T.Compose(transforms)
```

尚、画像に対してのリサイズ、ピクセル値の平均・標準偏差を使った正規化は Mask R-CNN モデル内部で行われるため、こちらで記述する必要はありません。

実際に動かしてみる

データセットクラス、モデルとデータ変換がそろいましたので、早速それらをインスタンス化してみましょう。

```
# 定義されたデータ変換とカスタムデータセットを使う
dataset = ConcreteCrackDataset('DeepCrack', get_transform(train=True))
dataset_test = ConcreteCrackDataset('DeepCrack', get_transform(train=False))

# トレーニングとテスト用にデータセットを分割する
torch.manual_seed(1)
indices = torch.randperm(len(dataset)).tolist()
dataset = torch.utils.data.Subset(dataset, indices[:-50])
dataset_test = torch.utils.data.Subset(dataset_test, indices[-50:])

# トレーニングと評価用のデータローダーを定義する
data_loader = torch.utils.data.DataLoader(
    dataset, batch_size=2, shuffle=True, num_workers=4,
    collate_fn=utils.collate_fn)

data_loader_test = torch.utils.data.DataLoader(
    dataset_test, batch_size=1, shuffle=False, num_workers=4,
    collate_fn=utils.collate_fn)
```

モデルとオプティマイザー（最適化）もインスタンス化します。

```
device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

# 今回のデータセットでは2クラスのみ - 背景とコンクリートひび割れ
num_classes = 2

# 補助関数でモデルを取得する
model = get_instance_segmentation_model(num_classes)
# モデルをデバイスに移動・配置する
model.to(device)

# オプティマイザーを設定する
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

# 学習係数のスケジューラーを用意し、3エポックごとに学習係数を
# 1/10に減らしていく
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

10 エポックのトレーニングを開始し、各エポックの最後に性能を評価します。

```
# 10 epoch学習させる
num_epochs = 10

for epoch in range(num_epochs):
    # 1エポックごとにトレーニングし、30バッチごとに状況をprintする
    train_one_epoch(model, optimizer, data_loader, device, epoch, print_freq=30)
    # 学習係数の更新
    lr_scheduler.step()
    # 性能評価を行う
    evaluate(model, data_loader_test, device=device)
```

トレーニングが終わりました。テストデータセットの画像で問題なく推論ができているかを確認します。

```
# テストセットから一つ画像を拾う
img, target = dataset_test[0]
# モデルを評価モードに設定し、入力として画像を与える
model.eval()
with torch.no_grad():
    prediction = model([img.to(device)])
```

推論結果を print してみると、辞書の配列になっていることがわかります。リストの要素は、それぞれの画像に対応しています。今回、画像は一枚のみを使って推論していますので、配列の中の辞書は一つです。 辞書は渡した画像に対する推論結果を保持しています。今回のモデルでは、辞書が boxes、 labels、 masks 、 scores をフィールドとして持っていることがわかります。

イメージの中身と推論結果の領域分割マスクも見てみます。中身の確認のためには、0-1 の値で表現されている画像のスケールを変更し、[C, H, W] の形式で表現されているチャネルを反転させる必要があります。

```
Image.fromarray(img.mul(255).permute(1, 2, 0).byte().numpy())
```

次に、推論結果の領域分割マスクを視覚化してみましょう。マスクは［N, 1, H, W］の形式で表現されており、N は推論結果の数を表しています。また確率も 0-1 の範囲で表現されています。

```
Image.fromarray(prediction[0]['masks'][0, 0].mul(255).byte().cpu().numpy())
```

＋ コード　＋ テキスト

正解のマスクも確認します

```
Image.fromarray(target['masks'][0].mul(255).byte().cpu().numpy())
```

うまくいっているようです！

まとめ

このチュートリアルでは、独自のデータセットを使って、トレーニングのパイプラインをセグメンテーションモデル向けに作成する方法をご紹介しました。独自データセットを使うためには、torch.utils.data.Dataset クラスを記述し、画像と正解の矩形、領域分割マスクを戻り値として返す実装を行いました。また、今回は新しいデータセットで転移学習を行うために、COCO train2017 で学習済みの Mask R-CNN モデルを活用しました。

いかがだったでしょうか？尚、より完成されており、複数マシン・複数 GPU のトレーニングにも対応した references/detection/train.py が、Torchvision の Github レポジトリにありますので、是非参考にされてみてください。

2.2 AI 習得セミナー

2.1 節で述べた教材は習得の上で非常に有用であるが，どうしても自学自習では解決できない課題も多い．そこで，ハンズオンセミナーの設計も行った．特に，2.1.1 節から 2.1.4 節までの内容を基本と考え，これらを習得できるようなプログラムを以下のように設定した．

○セミナータイトル：AI 入門セミナー「橋梁形式を判定する装置（CNN）を作ってみよう！」
○プログラム

13：00〜13：15	「AI とは何だろう？」
13：15〜14：00	「AI を用いたデータ分析とは？」
14：00〜14：10	（休憩）
14：10〜15：10	「データを分析する装置を動かしてみよう」
15：10〜15：20	（休憩）
15：20〜16：20	「装置の中はどのようになっている？」
16：20〜16：30	（休憩）
16：30〜17：30	「複雑化した装置でも難しくはない！」
17：30〜17：40	（休憩）
17：40〜18：00	「AI 活用に向けて」

データ収集の重要性と，機械学習の基礎，ディープラーニングの適用，そういったことを，手を動かしながら学ぶものであり，実際に土木学会の HP より募集も開始していたが，COVID-19 の影響で開催を中止した．しかし今後，情勢が落ち着き次第行う予定である．

また，舞鶴工業高等専門学校　社会基盤メンテナンス教育センター（ｉＭｅｃ）と共同で，本セミナーの内容を下敷きに e-learning 教材を作ることも計画しており，現在準備を進めている．

2.3 研究成果共有のためのシンポジウム／論文集の立ち上げ

AI 技術の土木工学分野への応用は黎明期にあるため，現状では適用例・アイデアの共有や意見交換を積極的に進めないといけない状況にある．そこで，AI・データサイエンスシンポジウム，AI・データサイエンス論文集（雑誌英語名：Intelligence, Informatics and Infrastructure）の立ち上げを行った．それぞれについて以下に述べる．

2.3.1. AI・データサイエンスシンポジウム

研究成果の共有および意見交換を目的として，AI・データサイエンスシンポジウムを企画し，令和 2 年 11 月 18 日〜20 日に開催した．様々な適用の方向性を指向しているため，幅広なテーマでの発表を以下の案内文のように募集した．

案内文：

ＡＩ・データサイエンスの実社会への導入が進んでいます．土木工学における利活用も，構造・材料・建設関係のみならず，維持管理，防災，交通，環境等多彩な領域で活発に進められています．

構造工学委員会構造工学におけるＡＩ活用に関する研究小委員会では，ＡＩ・データサイエンスの土木工学における発展に資することを目的として、多方面の研究・開発、調査・試験等の成果や現状について分野横断的・俯瞰的に議論するシンポジウムを開催します。

コンセプト・将来ビジョン・ロードマップ／ニーズ・ユースケース／データ取得／オープンデータ・データ共有／データプラットフォーム／モデリング・BIM・CIM・i-Construction／ＡＩ手法／データサイエンス手法／自律システム・ロボット／画像・時系列・言語等の解析／プログラミング／開発・実装／利活用／ビジネス・イノベーション／普及・啓発・人材育成／事例など，関連したあらゆる観点から，現在進行中の研究開発を含めて，幅広く取り上げます．

本シンポジウムは，本重点課題の母体である「構造工学委員会　構造工学でのＡＩ活用に関する研究小委員会」が主催となって行った．また，様々なアプリケーションを視野に入れていることもあり，以下の団体から協力，後援をいただいた．

協力：

土木情報学委員会　IoT/AI 活用モデル研究小委員会，地震工学委員会　AI・IoT 技術の地震工学への有効活用検討小委員会，応用力学委員会　データ駆動型の信頼性設計およびリスク評価実装研究小委員会，鋼構造委員会　鋼構造物の状態情報取得のためのイノベーション技術に関する調査研究小委員会

後援：

（一社）人工知能学会，（一社）日本ロボット学会

プログラムは（https://committees.jsce.or.jp/struct10/system/files/AI データサイエンスシンポジウムプログラム（修正）.pdf）に示す．COVID-19 の感染拡大のため，ZOOM によるオンラインシンポジウムとせざるを得なかったが，それでも 3 日間にわたって合計８０件の発表，４００名を超える参加者に参加いただき，また土木研究所の西川和廣理事長より「道路橋メンテナンスサイクルへの AI 導入－システム構築に必要なこと－」という基調講演をしていただいた．

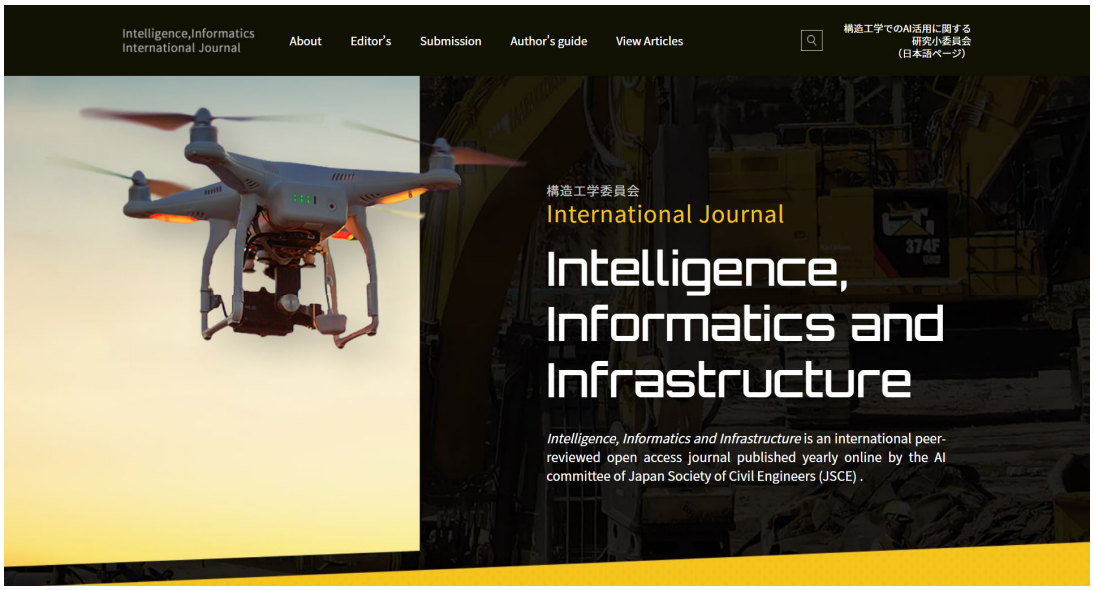
ZOOM をキャプチャした様子を以下に示す．非常に活発な質疑応答もなされ，有意義なシンポジウムとすることができた．第２回以降も継続して行えるよう，体制強化に取り組んでいる．（査読者名簿の整理，編集者の整理など）

図－４２　シンポジウムの様子

2.3.2. AI・データサイエンス論文集の立ち上げ

前節のシンポジウムの立ち上げと同時に，AI・データサイエンス論文集を立ち上げた．従来，このような小委員会が主体となって立ち上げた論文集は，査読なしであり，かつ検索するのも難しい論文集が多かったが，今回は3名の査読者によって審査されるという土木学会論文集に準じた方式を採用した．また，J-STAGE に登録することで，検索などにより発見される可能性を大幅に向上させた．実際に，Google Scholar などで検索すると出てくるようになっており，引用文献情報なども適切に登録されている．

また，海外からの可視性も向上させるため，International Journal として“Intelligence, Informatics and Infrastructure”を立ち上げた．既に数通 Vol.1 として掲載されており，また追加での投稿もいくらかなされており，今後も継続する方針である．



図－4 3　Intelligence, Informatics and Infrastructure の Webpage

3. データプラットフォームについての検討

人工知能技術の中で現在主流となっている教師あり学習，特にディープラーニングは大量のデータを用いた学習により帰納的にモデルを構築するものであり，すなわち，データ量が精度を左右する．特に，データが考えられる全ての境界条件のもとで網羅的にあり，かつ入力の形式と出力の形式が固まっていれば，的確な内挿の実現により高い精度の結果が得られるところにディープラーニングの有効性がある．しかし，土木工学分野においては，活用可能なデータは，一般的にディープラーニングの入力として求められる量よりはるかに（おそらく数オーダーは）少なく，しかも偏ったものしか得られないことが多い．インフラ構造物を例にとると，例えば以下のような原因が考えられる．

- インフラ構造物は，建設される環境が異なるために一品生産物であり，工場製品と比べて個体差が大きいこと
- 管理者間や，業務担当者間でデータの共有がなされないこと

これらは，データの網羅性を高める上での障害となる．この状況では，ディープラーニングの「学習範囲外の状況に弱く，実世界状況への臨機応変な対応ができない」という欠点の影響を避けることが出来ない．そこで，ディープラーニングに代表される AI 技術を土木工学分野で活用するためには，データの共有，管理，活用の仕組みであるデータプラットフォームを整えることが必須となる．

ここまで述べた背景を踏まえ，本節ではまずデータ連携・共有が土木の各分野でどうなっているか，複数分野の研究者にインタビューを行った結果について述べる．ついで，そこから，データプラットフォームのあるべき姿について，いくらかの例をもとに考察する．またその際に必要な技術開発についても述べることとする．

3.1. 複数分野の研究者へのインタビュー調査

ここでは，土木の各分野においてデータ連携に取り組んでいる研究者を対象にインタビューを行った．研究者の一覧を以下の表－9 に示す．このうち，浅本准教授と宮本准教授には SNS ベースでのインタビューを，その他の研究者の方々については ZOOM でのインタビューを行った．また，文字起こししたものについて，同意を得られたものは付録につけたので，必要に応じて参考にされたい．

表－9　インタビュー対象者

分野	インタビュー対象者
環境工学分野	西村文武准教授（京都大学）
地盤工学・地盤防災分野	岡村未対教授（愛媛大学）
地盤工学分野	竹山智英准教授（神戸大学）
河川工学・水文学分野	一言正之様（日本工営株式会社）
河川工学・水文学分野	森脇亮教授（愛媛大学）
河川工学・水文学分野	赤松良久教授（山口大学）
交通工学分野	吉井稔雄教授（愛媛大学）
構造工学・コンクリート工学分野	秋山充良教授（早稲田大学）
コンクリート工学分野	浅本晋吾准教授（埼玉大学）
構造工学・耐震工学分野	北原武嗣教授（関東学院大学）
構造工学・耐震工学分野	宮本崇准教授（山梨大学）

各研究者のインタビューの中で，複数分野にまたがり，以下のような課題が浮かび上がってきた．

○組織論的な難しさ

- データを管理する部署や企業が違う
- データ連携をするインセンティブがそれぞれに低い
- 公開すれば社会的な価値があることはなんとなくわかるが見返りを想定できない（オープンデータ）

○実務的な難しさ

- データを連携することの価値が，概念的には理解できても，具体的には見えない（ユースケースがよくわからない）
- 今発生するデータは将来にしか役立たないことが多い

- データを提供する主体と利益を受ける主体が相違することが多い
- データは連携してこそ価値が増大することが多いが，手元のデータだけで即ノウハウになると思い，データを出したがらない傾向がある．（そして結局死蔵される）
- データを出そうと思っても，出す仕組みがない．

このように，組織論的な難しさ，実務的な難しさが存在する．これらの難しさは，解決が難しいにも関わらず解決しても短期的なメリットが得られないことを意味している．その結果，以下の技術的な難しさを乗り越えるためのモチベーションの醸成が難しくなり，また体制構築や予算獲得も当然難しくなる．

○技術的な難しさ

- 陳腐化することが少ないデータフォーマットを定める必要がある．
- また，陳腐化した場合には柔軟にフォーマットを変える必要がある．
- 土木工学分野では画像などを含む非構造化データも扱う必要があるが，そのような枠組みが，構造化データの場合よりもさらに整っていない．
- 付加価値を生み出すには，画像とテキストデータなどマルチモーダルなデータを連携する仕組みを整える必要がある．しかし現状では，連携を想定したフォーマットもしくは API が想定されていない．

これらは，工夫や根性で乗り越えられるような課題ではなく，学術的研究開発，情報システムの構築，そしてそのために必要な予算や体制が求められる．しかし上記のように組織論的・実務的難しさがあるので，協調領域として関係者全員で取り組むことが求められる．特に，行政による旗振りや方針決定，大学による研究開発および教育，民間によるニーズの提供や試行，ブラッシュアップが必須となる．

また，AI への活用という面でも示唆を得た．AI の学習においてはデータが揃っていることが重要であるが，現状では，①AI が使うデータを入手できない，②せっかくデータを入手しても活用までの前処理に膨大な時間を要する，という課題を抱えている分野が多いことがわかった．

以下の３．２では，AI を含むデータドリブンアプローチの高度化という観点から考えた場合に，データプラットフォームに求められる機能をまず示す．そして，データプラットフォームが，上記の課題にも挙げた非構造化データを扱うためには，構造化データに変換するための機能が必要であることを説明し，そしてそのプロトタイプを示す．

３．３では，「データを出そうと思っても，出す仕組みがない．」という仕組みの改善を考え，J-STAGE Data への取り組みを先駆的にはじめようとしている事例を紹介する．AI の学習のためにデータが必要であるというのは上に記載したが，一方で，論文などで発表された手法などについて，データがないと追試できないが，データをアップロードする場所が整

備されていないという，学術的な観点からの課題がある．そこで，データアップロードの場所として, JST が構築した研究データの共有と利活用を促進するためのリポジトリである J-STAGE Data と，2.3.2 で示した AI・データサイエンス論文集を連携させた．先駆的な取り組みとして本報告書で報告する．

3.2. 目指すべきデータプラットフォームの構造と開発すべき機能

3.2.1. AI にマッチしたデータプラットフォームの構造

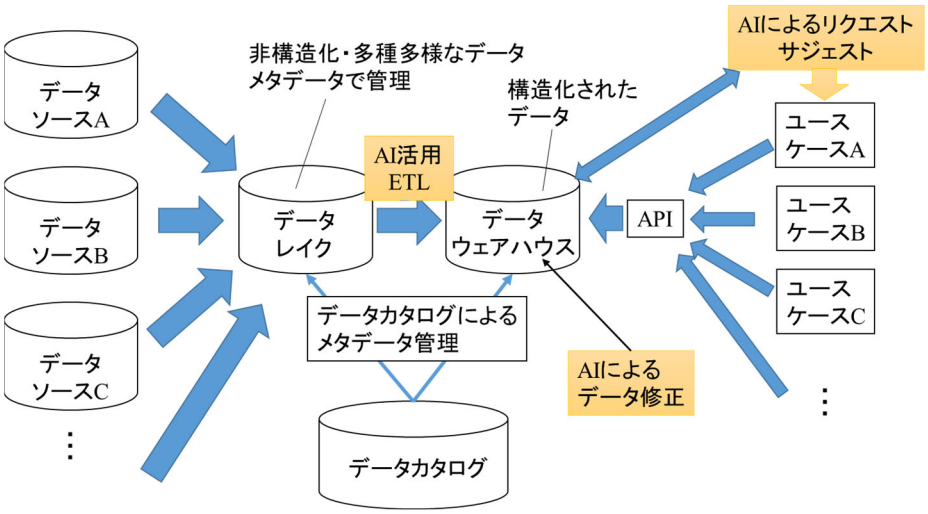
データプラットフォームは，ユースケースに応じて様々な構成が考えられ，また実現されている．特に，データプラットフォーム活用による AI の高度化という観点から考えた場合，実現すべき機能は以下の通りである．

- 入手可能なデータの種類がデータカタログのような仕組みで整理され，そして個々のデータにはメタデータが付与されている．
- データプラットフォーム内のデータで，恐らく入力時にミスがあったと考えられるものについて，AI が修正案を提示する，あるいは自動で修正する．
- AI がユーザーの目的に応じて，非構造化データを自動で構造化し，その上で必要と思われるデータの種類をサジェストする．
- データをアプリケーション側で簡単に用いることができるように，API が整備されている．

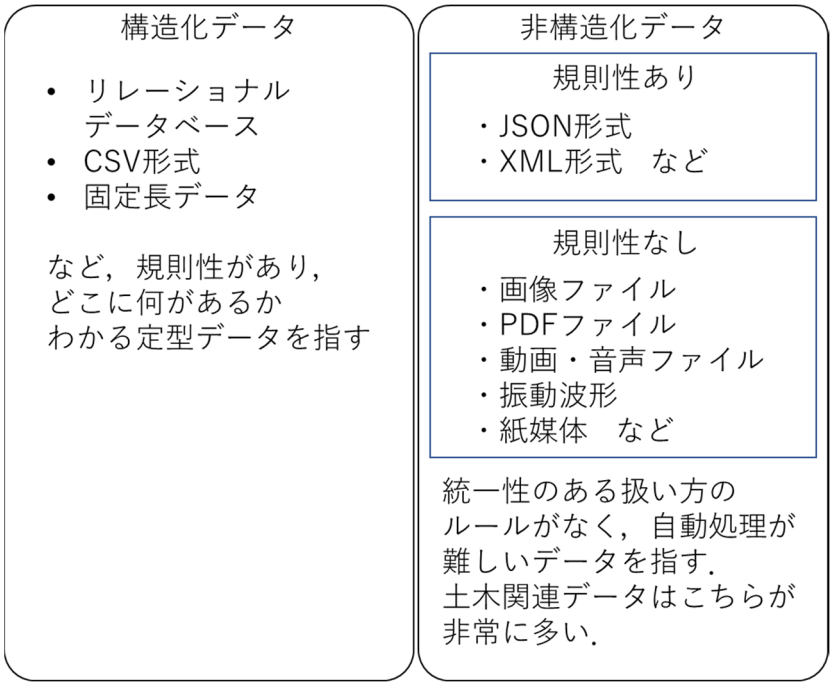
この要件は，上記の①AI が使うデータを入手できない，②せっかくデータを入手しても活用までの前処理に膨大な時間を要する，という課題の解決を視野に入れたものである．

上記の機能を実現するための仕組みとして，例えば図－４４のような構成が考えられる．データレイクとは，あらゆる構造化データと非構造化データを保管する役割を担うもので，データウェアハウスとはデータレイクのデータを変換して構造化したものを格納し，また計算リソースやインタフェースを提供する役割を担うものである．

なお，土木分野データの特徴としては，非構造化データが多いという点が挙げられる．非構造化データを具体的に挙げると，動画データや画像データ，音声データ，振動データ，あるいは統一ルールなく PDF 化された資料など，データ構造が定義されていないデータを指す．構造化データと非構造化データの違いを図－４５に示す．土木関係業務で納品されるデータは PDF ファイル，あるいは画像データが多く，これら非構造化データをどう扱うかというのはデータプラットフォームにおける大きな課題である．



図－４４　データプラットフォーム構成の一例



図－４５　構造化データと非構造化データの違い

データレイクなしでデータウェアハウス（あるいは類したもの）のみが設置されているような運用も多いが，整理された構造化データの作成過程で情報を不可逆的に失ってしまい，後に必要となるデータ構造の変化に対応できず，後から惜しむような事例は枚挙にいとまがない．データレイクは一旦全てのデータを非加工で置いておくことで，後のデータ構造の変化に対応でき，また必要な情報が失われてしまうという事態も防ぐことができる．ただし，単純に保存しているだけでは結局使えないデータが溜まっていくだけになってしまうため，

ガバナンスの効いた状態で保存する必要がある．そのためにはデータに関する情報であるメタデータを付与することが必要となる．また，どのようなメタデータが存在し，そしてデータウェアハウス内でどのように構造化されているかを整理するためのデータカタログの整備も重要となる．

このようにデータプラットフォーム内のデータが構造化されていて，データカタログにより適切なメタデータ情報を得ることが出来るのであれば，AI を含むデータドリブン解析の設計が非常に容易となる．さらに，メタデータとして，「誰に」「いつ」「どのように」作られたかという情報があれば，構築した AI の適用範囲や，信頼性などの評価も可能となる．そしてメタデータが適切に付与されているデータを API 経由でユースケースに合わせて取得できれば，一般に工程の 8 割〜9 割を占めるとも言われる前処理をユーザー側がさほど行わずとも解析を行うことが出来ると期待される．

しかし上記のように，土木分野データには非構造化データが多い．データレイクのデータからデータウェアハウスを構築するためには，適切に情報を抽出し，構造化して格納する必要があるが，必ずしも容易ではない．一般に ETL 処理（Extract; Transform; Load の頭文字）とも言われるが，例えば統一ルールなく PDF 化された資料，特にその中のラスタデータとして表現されている図面からデータを抽出するのは非常に難しい．また，大量に蓄積されている写真などからメタデータを抽出することも難しい．そのデータ処理には必然的に画像処理や AI が求められる．

また，もともとのデータを作成する際に，ミスが生じていることもある（例えば橋梁の幅員 20m と書くべきところを 20km と書いてしまうなど）．メタデータが適切に整備されていれば，そのデータが持ちそうなデータ範囲を与えることも可能となるので，そこから大幅に外れたデータについて警告を出すことは AI がデータプラットフォームで果たすべき役割として重要である．さらに，データウェアハウスでデータが構造化されていれば，AI により目的に応じて必要と考えられるデータをサジェストすることも可能となる．このように，データプラットフォーム構築と，AI 開発はお互いに強化し合う関係性にあり，結びつきは非常に強い．だからこそ，データプラットフォームを，シームレスに構築するための技術が必要であり，そのための根源にあるのは ETL 機能の徹底した充実化である．次節では，非構造化データから，適切にデータを抽出し，構造化する ETL 処理について，特に画像へのメタデータ付与に着目し，そのプロトタイプを構築した．

3.2.2. ETL 処理の例：画像データからのメタデータ抽出

ETL 処理とは，「抽出（Extract）」「変換（Transform）」「書き出し（Load）」の各プロセスの頭文字をとったものであり，データベースやデータレイクなどからデータを抽出し，扱いやすいフォーマットに変換し，データウェアハウスなどに格納する処理を指す．この ETL 処理が容易なフォーマット，難しいフォーマット，様々にあるが，図－４５の構造化データの場合は易しく，非構造化データは難しい．画像データは難易度の高いものの 1 つであるが，

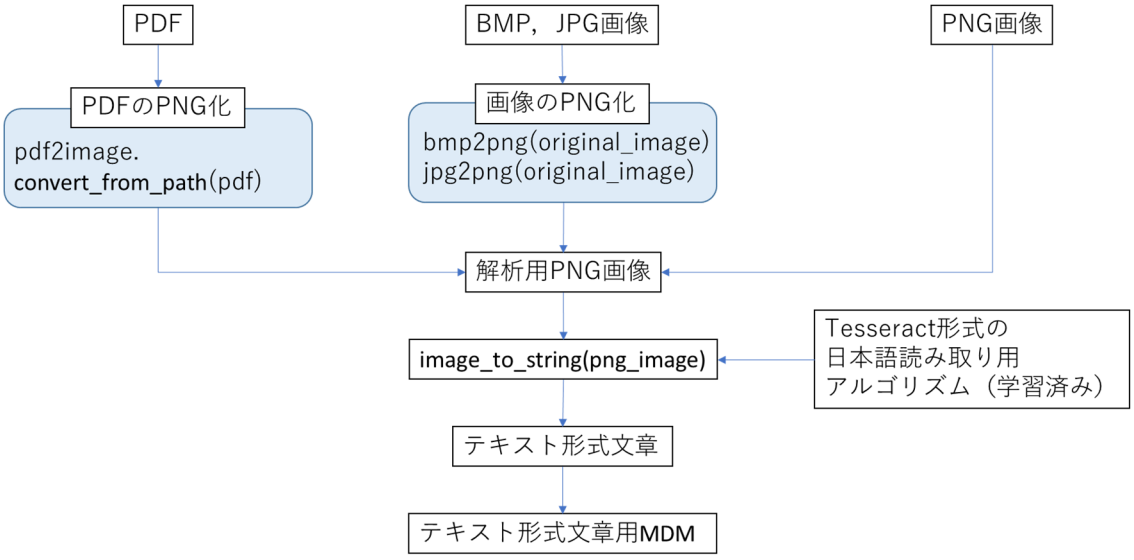
一方で土木関連業務ではスキャン書面や写真など，画像データが多く，もっとも ETL 処理が求められるデータでもある．

本節では，そのような ETL 処理の概要を理解する，あるいは性能を理解するため，画像データからメタデータを抽出するような ETL 処理を行うプロトタイプを構築したので，それを示すこととする．

1）図面や文書などをスキャンした書類

工事書類などが電子的に作成され，その上で文章情報などが適切に保持されるような形式で保存されていれば，そこから文章情報を読み取ることが出来る．しかし，例えばスキャンして PDF 化，あるいは画像化して保存したケースなどではそこから文章を読み込むことは容易ではない．また，文章情報を残すように PDF 化したケースなどでも，意図せず文章情報が喪失しているケースも多く，その場合はやはり画像化して読み込む手法が有力な解決策となる．

整理すると，以下のようなフローとなる．



図－４６　スキャン書類からメタデータを抽出するフロー

このフローに従い，例えば国土交通省による H31 橋梁定期点検要領の付録―１　対策区分判定要領の P5, P15 を，敢えて印刷した上でスキャンしたものについて，テキスト形式文章生成と，メタデータ抽出を行った．

①H31 橋梁定期点検要領　付録―１　対策区分判定要領（P5）：
橋梁定期点検要領　付録―１　対策区分判定要領の P5 を以下に示す．

② 亀裂

【一般的性状・損傷の特徴】

鋼材に生じた亀裂である。鋼材の亀裂は，応力集中が生じやすい部材の断面急変部や溶接接合部などに現れることが多い。

亀裂は鋼材内部に生じる場合もあり，外観性状からだけでは検出不可能な場合がある。

亀裂の大半は極めて小さく，溶接線近傍のように表面性状がなめらかでない場合には，表面きずや錆等による凹凸の陰影との見分けがつきにくい場合がある。なお，塗装がある場合に表面に開口した亀裂は，塗膜われを伴うことが多い。

アーチやトラスの格点部などの大きな応力変動が生じることのある箇所については，亀裂が発生しやすい部位であることに加えて，損傷した場合に構造全体系への影響が大きいため，注意が必要な場合がある。

ゲルバー構造などにある桁を切り欠いた構造部分では，応力集中箇所となり，疲労上の弱点となる場合がある。

同一構造の箇所では，同様に亀裂が発生する可能性があるため，注意が必要な場合がある。

【他の損傷との関係】

- 鋼材の亀裂損傷の原因は外観性状からだけでは判定できないことが多いので，位置や大きさなどに関係なく鋼材表面に現れたわれは全て「亀裂」として扱う。
- 鋼材のわれや亀裂の進展により部材が切断された場合は，「破断」として扱う。
- 断面急変部，溶接接合部などに塗膜われが確認され，直下の鋼材に亀裂が生じている疑いを否定できない場合には，鋼材の亀裂を直接確認していなくても，「防食機能の劣化」以外に「亀裂」としても扱う。

【対策区分判定】

○判定区分Ｅ１；橋梁構造の安全性の観点から，緊急対応が必要な損傷

亀裂が鈑桁形式の主桁ウェブや鋼製橋脚の横梁のウェブに達しており，亀裂の急激な進展によって構造安全性を損なう状況などにおいては，緊急対応が妥当と判断できる場合がある。

アーチやトラスの格点部などの大きな応力変動が生じることのある箇所及びゲルバー構造などにある桁を切り欠いた構造部分の亀裂は，構造全体系への影響が大きいため，亀裂の急激な進展のおそれがある状況などにおいては，緊急対応が妥当と判断できる場合がある。

○判定区分Ｅ２；その他，緊急対応が必要な損傷

鋼床版構造で縦リブと床版の溶接部から床版方向に進展する亀裂が輪荷重載荷位置直下で生じて，路面陥没によって交通に障害が発生する状況などにおいては，緊急対応が妥当と判断できる場合がある。

○判定区分Ｓ１，Ｓ２；詳細調査又は追跡調査が必要な損傷

亀裂が生じた原因の推定や当該部材の健全性の判断を行うためには，表面的な長さや開口幅などの性状だけでなく，その深さや当該部位の構造的特徴や鋼材の状態（内部きずの有無，溶接の種類，板組や開先），発生応力などを総合的に評価することが必要である。した

図－４７　H31 橋梁定期点検要領　付録―１　対策区分判定要領の P5

橋梁の鋼材の亀裂に関する文章である．これについて，テキスト化した結果は以下の通りである．

変換結果：
亀裂
【一般的性状・損傷の特徴】
鋼材に生じた亀裂である。 鋼材の亀裂は，応力集中が生じやすい部材の断面急変部や溶接接合部などに現れることが多い。
亀裂は鋼材内部に生じる場合もあり，外観性状からだけでは検出不可能な場合がある。
亀裂の大半は極めて小さく，浴接線近傍のように表面性状がなめらかでない場合には，表面きずや錆等による凹凸の陰影との見分けがつきにくい場合がある。 なお，塗装がある場合に表面に開口した亀裂は，塗膜われを伴うことが多い。
アーチやトラスの格点部などの大きな応力変動が生じることのある箇所については，亀裂が発生しやすい部位であることに加えて，損傷した場合に構造全体系への影響が大きいため，注
意が必要な場合がある。ゲルバー構造などにある桁を切り欠いた構造部分では，応力集中箇所となり，疲労上の弱点となる場合がある。
同一構造の箇所では，同様に亀裂が発生する可能性があるため，注意が必要な場合がある。
【他の損傷との関係】・ 鋼材の亀裂損傷の原因は外観性状からだけでは判定できないことが多いので，位置や大きさなどに関係なく鋼材表面に現れたわれは全て「亀裂」として扱う。・ 鋼材のわれや亀裂の進展により部材が切断された場合は，「破断」として扱う。・ 断面急変部，溶接接合部などに塗膜われが確認され，直下の銅材に亀裂生じている疑いを否定できない場合には，鋼材の亀裂を直接確認していなくても，「防食機能の劣化」以外に「亀裂」としても扱う。
【対策区分判定】〇判定区分１：橋梁構造の安全性の観点から，緊急対応が必要な損傷亀裂が飯桁形式の主桁ウェブや鋼製橋脚の横梁のウェブに達しており，亀裂の急激な進展によって構造安全性を損なう状況などにおいては，緊急対応が妥当と判断できる場合がある。アーチやトラスの格点部などの大きな応力変動か生じしることのある箇所及びゲルバー構造などにある桁を切り欠いた構造部分の亀裂は，構造全体系への影響が大きいため，亀裂の急激な進展のおそれがある状況などにおいては，緊急対応が妥当と判断できる場合がある。
〇判定区分２：その他，緊急対応が必要な損傷鋼床版構造で綻リブと床版の溶接部から床版方向に進展する亀裂が輪荷重載荷位置直下で生じて，路面陥没によって交通に障害が発生する状況などにおいては，上皿急対応が妥当と判断できる場合がある。
〇判定区分Ｓ１Ｓ２：詳細調査又は追跡調査が必要な損傷亀裂が生じた原因の推定や当該部材の健全性の判断を行うためには，表面的な長さや開口幅などの性状だけでなく，その深さや当該部位の構造前特徴や鋼材の状態（内部きずの有無，溶接の種類，板組や開先），発

生応力などを総合的に評価することが必要である。した

これについて，元の文章と差分をとった結果が以下の通りである．

図－４８　元データとの差分

いくらか異なる部分は存在する．例えば縦リブを綻リブ，鈑桁を飯桁としてしまうなどといった課題が生じている．これらについては専門用語のため，学習が現状では未だ不十分であるためだと考えられるが，必要に応じて図－４６に示す学習を行うことで，改良可能である．

そしてここから，例えばメタデータの候補として橋梁定期点検要領に定められている２６種の損傷などをリストアップし，それと合致するものを選ぶと，「亀裂」「破断」「防食機能の劣化」などというメタデータを付与できる．あるいはリストとして他のものを準備すると他のメタデータを付与できる．

②H31 橋梁定期点検要領　付録―１　対策区分判定要領（P15）：
橋梁定期点検要領　付録―１　対策区分判定要領の P15 を以下に示す．

⑦　剥離・鉄筋露出

【一般的性状・損傷の特徴】

コンクリート部材の表面が剥離している状態を剥離，剥離部で鉄筋が露出している場合を鉄筋露出という。

【他の損傷との関係】

・剥離・鉄筋露出とともに変形・欠損（衝突痕）が生じているものは，別途，それらの損傷としても扱う。
・「剥離・鉄筋露出」には露出した鉄筋の腐食，破断などを含むものとし，「腐食」，「破断」などの損傷としては扱わない。
・床版に生じた剥離・鉄筋露出は，「床版ひびわれ」以外に本項目でも扱う。

【対策区分判定】

○判定区分Ｅ１；橋梁構造の安全性の観点から，緊急対応が必要な損傷

塩害地域において床版下面でＰＣ鋼材が露出し，断面欠損にまで至っており，今後も損傷進行が早いと判断され，構造安全性を著しく損なう危険性が高い状況などにおいては，緊急対応が妥当と判断できる場合がある。

○判定区分Ｅ２；その他，緊急対応が必要な損傷

剥離が発生しており，他の部位でも剥離落下を生じる危険性が極めて高く，第三者被害が懸念される状況などにおいては，緊急対応が妥当と判断できる場合がある。

○判定区分Ｓ１，Ｓ２；詳細調査又は追跡調査が必要な損傷

鉄筋の腐食によって剥離している箇所が見られ，鉄筋の腐食状況によって剥離が連続的に生じるおそれがある状況などにおいては，詳細調査を実施することが妥当と判断できる場合がある。

○判定区分Ｍ；維持工事で対応が必要な損傷

全体的な損傷はないものの，部分的に剥離が生じており，損傷の規模が小さく措置のしやすい場所にある状況などにおいては，維持工事で対応することが妥当と判断できる場合がある。

なお，露出した鉄筋の防錆処理は，モルタル補修や断面回復とは別に，維持工事で対応しておくことが望ましい。

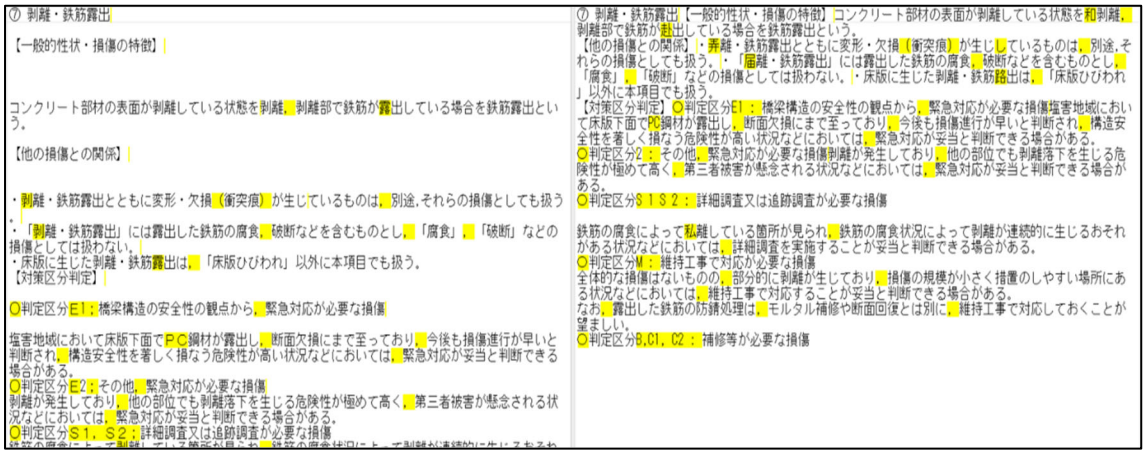
○判定区分Ｂ，Ｃ１，Ｃ２；補修等が必要な損傷

図－４９　H31 橋梁定期点検要領　付録―１　対策区分判定要領の P15

橋梁のコンクリート部材の剥離・鉄筋露出に関する文章である．これについて，テキスト化した結果は以下の通りである．

変換結果：
⑦ 剥離・鉄筋露出【一般的性状・損傷の特徴】コンクリート部材の表面が剥離している状態を和剥離，剥離部で鉄筋が赴出している場合を鉄筋露出という。
【他の損傷との関係】・弄離・鉄筋露出とともに変形・欠損（衝突痕）が生じしているものは，別途,それらの損傷としても扱う。・「届離・鉄筋露出」には露出した鉄筋の腐食，破断などを含むものとし，「腐食」，「破断」などの損傷としては扱わない。・床版に生じた剥離・鉄筋路出は，「床版ひびわれ」以外に本項目でも扱う。
【対策区分判定】〇判定区分 E1：橋梁構造の安全性の観点から，緊急対応が必要な損傷塩害地域において床版下面で PC 鋼材が露出し，断面欠損にまで至っており，今後も損傷進行が早いと判断され，構造安全性を著しく損なう危険性が高い状況などにおいては，緊急対応が妥当と判断できる場合がある。
〇判定区分 2：その他，緊急対応が必要な損傷剥離が発生しており，他の部位でも剥離落下を生じる危険性が極めて高く，第三者被害が懸念される状況などにおいては，緊急対応が妥当と判断できる場合がある。
〇判定区分 S 1 S 2：詳細調査又は追跡調査が必要な損傷
鉄筋の腐食によって私離している箇所が見られ，鉄筋の腐食状況によって剥離が連続的に生じるおそれがある状況などにおいては，詳細調査を実施することが妥当と判断できる場合がある。
〇判定区分 M：維持工事で対応が必要な損傷
全体的な損傷はないものの，部分的に剥離が生じており，損傷の規模が小さく措置のしやすい場所にある状況などにおいては，維持工事で対応することが妥当と判断できる場合がある。
なお，露出した鉄筋の防錆処理は，モルタル補修や断面回復とは別に，維持工事で対応しておくことが望ましい。
〇判定区分 B,C1, C2：補修等が必要な損傷

これについて，元の文章と差分をとった結果が以下の通りである．



図－５０　元データとの差分

①よりもやや精度が低い．特に，剥離という単語の認識精度が低いが，これについても①と同様に学習をすれば改良可能な範囲である．

このように文書データから色々なデータを抽出できれば，例えば「剥離」「鉄筋露出」「腐食」「破断」「ひびわれ」「塩害」のようにメタデータを付与でき，検索可能性も向上する．また，近年の AI は文章データそのものを学習データとするような開発が進んでおり（GPT-3 と呼ばれる AI など），これらを AI に導入することで，例えば文章として表現・蓄積されているような経験知・専門知を AI に取り入れることができると期待できる．

2）業務で得られた写真

土木関連業務では写真を撮影することが多い．例えば橋梁点検・診断業務においては，写真撮影なしでは現場の状況をわかりやすく調書に残しておくことが出来ず，1 橋で数百枚の写真を撮影することは決して珍しくない．あるいは今後 UAV による点検が普及すれば，1 橋あたり数万枚となることも想定される．これが，複数橋梁になるとさらに乗算的に増加していくこととなる．近年の AI が，画像処理に強力な性能を発揮していることを考えた場合，このような写真を活用することが望ましい．

そのような大量の写真群の中から，探したい写真を見つけるのは大変な労力を要する．そこで，写真に対してメタデータを自動付与することができれば，そのメタデータを頼りに検索することが可能となるため，大幅な省力化となる．そこで本委託研究では，撮影画像を元に画像に写っている物体の状況を説明する文章を生成する Image Captioning 技術を活用し，メタデータ自動付与プログラムを構築する．ここでは特に，橋梁点検画像に絞って適用例を示すが，他のアプリケーションにもそのまま横展開可能である．

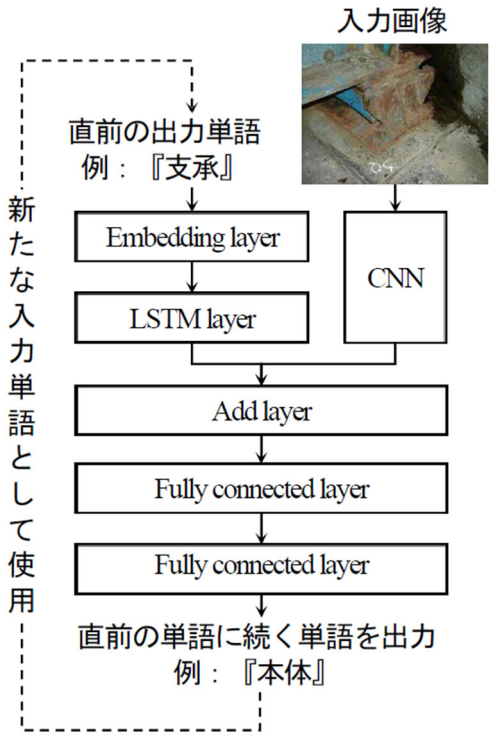
近年，特に Deep learning の発展に伴い，Image Captioning 技術の精度も大幅に向上している．例えば Microsoft などもそのような API を提供しているが（Computer Vision API），土木の専門知が含まれていないため，そのままでは活用できない．例えば以下の図は橋梁点

検の際に得られた画像に対して解析した結果である．

図－５１　Computer Vision API による解析結果例

左図は腐食した支承，右図はひび割れが生じている竪壁の写真であるが，全く正しくない出力となっていることがわかる．何故なら，上記のデータセットは特に土木工学分野固有の用語や概念が含まれたものになっていないためである．そこで本研究では，土木工学分野の学習データセットを構築し，そのうえで Image Captioning を行う Deep Learning モデルの学習を行う．

ここでは，ニューラルネットワークライブラリの Keras を用いて文章生成モデルの構築を行い，また，Keras のバックエンドライブラリには Tensorflow を用いた．構築した文章生成モデルの概要を図－５２に示す．



図－５２　文章生成モデルの構造

本モデルは，主に画像を認識するための CNN（Convolutional Neural Network）と，時系列データである文章を取り扱うための LSTM（Long short-term memory）で構成されている．本モデルでは，CNN に画像を入力するとともに，LSTM を組み込んだネットワークにモデル自身が直前に出力した単語を入力する．LSTM は過去の出力を行った際の状態を内部で保存するため，過去に出力された単語を考慮した上で続く単語を予測して出力することが出来る．その後，結果の結合などを行い，最終的な出力が得られる．このようにして，モデルの出力として直前に出力された単語に続く単語が出力される．モデルの CNN には，ILSVRC-2014 model with 16 weight layers（VGG16）の学習済みモデルを使用した．

文章検出結果の事例を以下に示す．

支承本体において、腐食が発生している。→メタデータ：支承，腐食

床版において、ひび割れが発生している 。→メタデータ：床版，ひび割れ

竪壁にひび割れが発生している。→メタデータ：竪壁，ひび割れ

主桁において、腐食が発生している。→メタデータ：主桁，腐食

横桁において、防食機能が劣化している。→メタデータ：横桁，防食機能の劣化

図－５３　Image Captioning とメタデータ付与

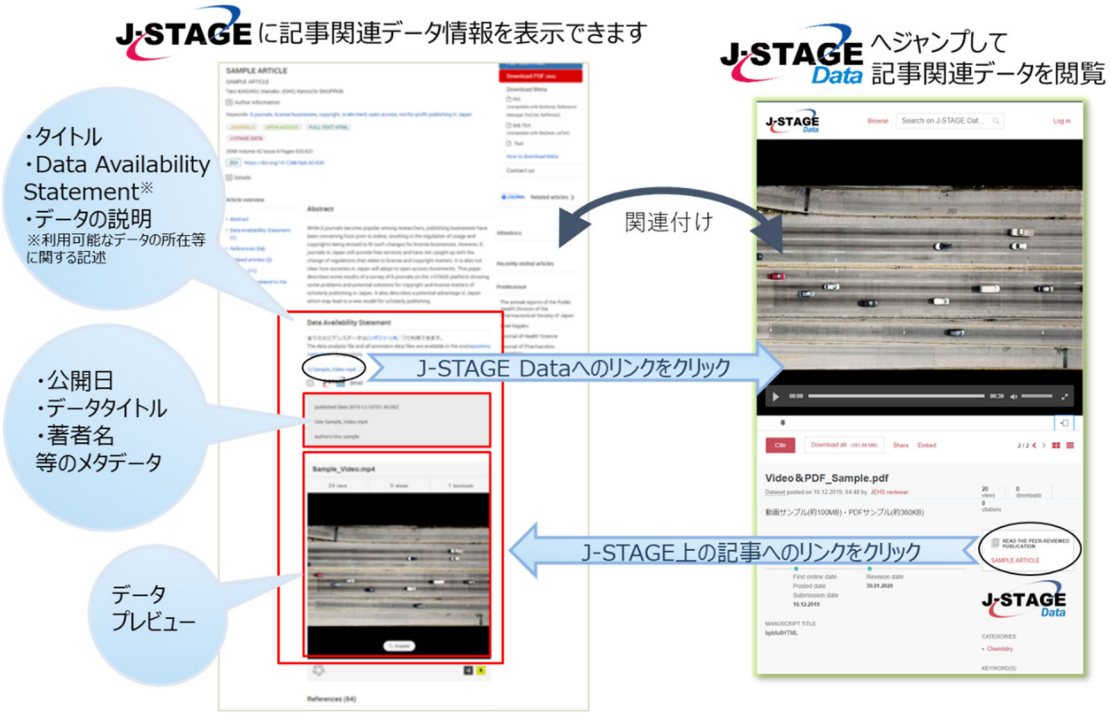
このようにメタデータを付与することが出来れば，画像のファイルパスとメタデータのリスト構築といった構造化が可能となる．ここで示したのはあくまでも ETL 処理の例であるが，少なくとも難易度が非常に高い画像に対しての例ということで，参考になる事例かと考えプロトタイプの結果を示した．

3.3. J-STAGE Data と AI・データサイエンス論文集との連携

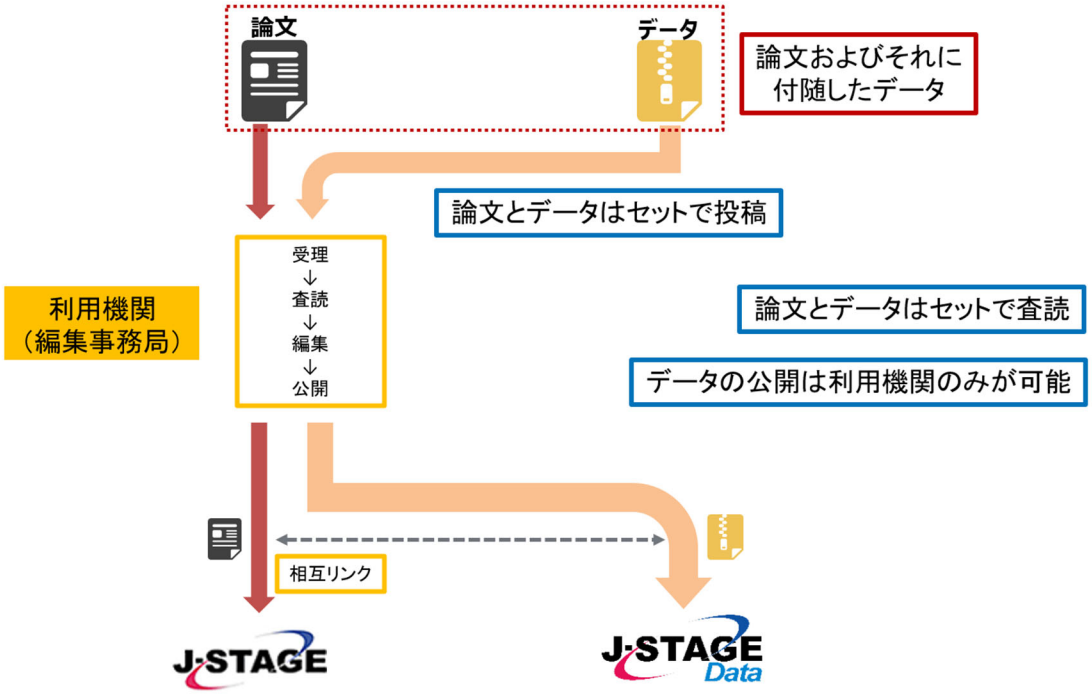
AI の学習のためにはデータが必要である．しかし，論文などで発表された手法などについて，データがないと追試できないが，データをアップロードする場所が整備されていないという，学術的な観点からの課題がある．つまり，論文の結果の信頼性が担保されない状態にある．そこで，データアップロードの場所として，JST が構築した研究データの共有と利活用を促進するためのリポジトリである J-STAGE Data と，2.3.2 で示した AI・データサイエンス論文集を連携させた．先駆的な取り組みをはじめたので本報告書で報告する．

J-STAGE Data とは，J-STAGE の登載記事に関連するデータを登載・公開するデータリポジトリである．令和 2 年 3 月に試行的にリリースされたものであり，AI・データサイエンス論文集についてもこの取組を活用することとした．J-STAGE Data は図－５４のように，J-STAGE の記事とデータをリンクできる．

J-STAGE Data に掲載されたデータには，独立して DOI およびメタデータが付与され，また CC ライセンスが付与される．これにより引用，再利用，検索がされやすくなり，また適切な形でのデータの国内外への流通を促進するものである．J-STAGE Data では，GB サイズの多種多様なファイル形式のデータを公開できる．なお，J-STAGE Data での公開が難しいケースも多くありえるので，実際にどのデータを J-STAGE Data に収載するかは著者および編集委員会が決めれるような形にしてある．図－５５に論文とデータの査読プロセスを示す．論文とデータはセットで投稿，査読がなされることとなる．



図－５４　J-STAGE 掲載論文と J-STAGE Data との連携



図－５５　論文とデータの査読プロセス

JST が示している J-STAGE Data の仕様を以下の表－１０に示す.

表－１０　J-STAGE Data の仕様

項目	仕様
費用	無料（将来的には有償の可能性あり）
データ容量（1誌当たり）	100GB/1誌　（要調整）
データ容量（データ当たり）	上限なし（但し、インターネットでのデータ送信の目安として20GB）
データ種別	データセット、画像、動画、その他、登載可能なデータに制限無し。プレビュー可能なデータについても、一般的な形式は幅広く網羅。
ライセンス	オープンアクセスで、基本的にCCライセンスなど
連携サービス	Dimensions，Google Dataset Search　等

この J-STAGE Data と連携することで，データつきで手法や結果を示すのが当然である（そうでないと追試ができない）という文化を醸成したいと考えている.

4. まとめ

近年のAI技術の進化はめざましく，土木工学分野においても取り入れる余地は多くある．しかし現状では，教育システムが未整備でありプレイヤーが増えていない，データの共有，管理，活用の仕組みが整っていない，という課題がある．また，プレイヤーが増えないからデータ共有が進まず，データ共有が進まないからプレイヤーが増えないという悪循環も見いだされた．そこで本重点課題研究では，土木工学における人工知能技術の活用のために必要な基礎的知識・技術を習得できる教材の開発と，人工知能技術の活用を見据えたデータ収集・記録の基点となる，データプラットフォームのあるべき姿の提案・プロトタイプ構築を同時に対象として行った．これにより，AI 技術の土木工学分野での活用という観点で，方向性を提示することができ，今後すすめるべき研究課題や教育システムの開発を考える起点となった．

ただ，データプラットフォームや教育システム開発についての起点になったとはいえ，今後行うべきことはまだまだ多い．データプラットフォームについては，そもそも土木工学ならではのデータの扱いを考える必要がある．土木工学分野でのデータは，例えば長期間に渡り蓄積される（インフラ構造物の寿命は長い場合は１００年を超える），それゆえデータの品質がバラバラであるなど，他分野と比較して不利な点が非常に多い．これらについて，例えば IT ベンダーに丸投げなどをしても解決しないため，本質的な取り組みが求められる．その意味では，特に研究者の参入が求められる．ただ，現状ではなかなか学術として認められづらいため論文になりづらい．そこで，土木工学分野でのデータの性質や扱いなどを体系化して，学術として整理することが今後のデータプラットフォームの発展，AI 活用の推進に極めて重要であると考えている．また，論文として発表する場も必要であり，「構造工学でのAI活用に関する研究小委員会」で研究代表者らが立ち上げた「AI・データサイエンス論文集」はその受け皿として機能させるような枠組みとしている．

教育システムについても同様で，本重点研究課題ではAIの教育システムについて開発を行ったが，もう少し視野を広げてデータの扱い，あるいはシステムズエンジニアリングの観点まで含めた教育システムを樹立する必要があると考えている．本重点研究課題での Web 上での単発での教材も重要であるが，それと並行して，土木工学分野での ICT 活用の全体像を俯瞰できるような，体系化された教科書，あるいは示方書のようなものが必要であり，それは土木の学問を取り扱う土木学会が先導して行っていくべきものである．以上，この提案をもって，本報告書の締めとする．

付録

本重点研究課題では，様々な分野の研究者にインタビューを行い，データ共有やデータ連携における現状や課題について調べた．それらを文字起こししたもののうち，掲載の同意を得られたものについて，以下に記載する．

付.1. 西村文武准教授へのインタビュー

――西村先生の専門分野で、データを収集して何らかの知見を生み出そうとする取り組みはありますか。今回、共同研究（編注：平成 30 年度下水道技術研究開発（GAIA プロジェクト）「深層学習を活用した流域での人間活動に応じた流入下水負荷変動予測と既往処理システム運転管理の最適化」において研究代表者と西村教授は共同研究している）で一緒に取り組んでいるものもそうかと思いますが。

まず、あの共同研究については、どちらかいうと国交省が旗振りをしている「i-Construction」の下に、下水道でも「i-Gesuido」をしているわけですから、国交省の方針として動いていると言えます。

当然 GAIA（下水道技術研究開発）もそうですが、そうした中で、その上のレベルの最も大きな B-DASH という下水道の実用化技術などに、AI を使った下水道管理、あるいは現在抱えている諸問題について情報を抽出して何か解決できないかと、さまざま取り組んでいます。もっとも、それらに付く予算に魅力を感じて皆さんが取り組むという面があります。また、下水道事業はずっと以前から官が旗振りをしてきましたので、国交省が方針を打ち出すと、民間企業はビジネスチャンスのように捉える傾向が見られます。そうしたことを背景に「なんとかやっていきましょう」という状況です。

――そういったものの中で、すでに実用化に至っているものはありますか。

実用化はこれからです。私も B-DASH の評価委員の一人ですが、技術的な説明を聞いていると、「AI や機械学習を組み入れた」とは言っているものの、果たしてどうかと首をかしげることがあります。どちらかというと「とりあえずやってみました」という印象を非常に強く受けます。たとえば「１ヵ月分のデータでやりました」などです。もっと長いスパンでデータを取っているのですから、そのデータを活用すればいいと思うのですが、そこまで取り組まないような印象があります。機械学習と言っても、「Random Forests でやりました」というものも多くあります。全先生のような、本当の若手技術者が加わっていると、もっと広がると思うのですが……。

ただ一方で、皆さん、関心はすごく持っていると思います。よく理解していないからこそ、

何かバラ色の素晴らしいことができるのではないかと、ものすごく大きな期待を持っているのかもしれません。ですから、現状ではどこまでできて、どこからは難しいなどの実際上の見極めは、なかなか難しい点があるように感じます。以上は、私の個人的な印象です。

――西村先生の GAIA の研究もそうですが、今お話に出てきたデータについては、皆さん、個々にハードディスクなどに保管しているのでしょうか。

はい、個別に持っています。

――共有するような仕組みはできていないのですか。

共有とは、どの程度の範囲を指していますでしょうか。

――どこか 1 ヵ所に、というイメージです。たとえば、まず国交省に集約された上で、関係者が共有できるようにハンドリングされているなどです。下水道のデータの場合、どのような状況でしょうか。

下水処理施設では、「きちんと処理できているか」あるいは「運転管理上できちんとオペレーションできているか」など、チェックのためのデータは取っています。水質や排水などの基準がありますので、その基準が守られているか、基本的な日常点検などで定期的にデータを取っています。これは、自主的な取り組みというよりも、マストの業務です。ですから、データ自体は蓄積されています。

ただ、過去のデータを何かに活用しているかというと、そうした試みはまったく見受けられません。点検のため、1 ヵ月ごとに管理業者が下水処理場に報告はしますが、それで終わりです。

――たとえば、数施設の下水処理施設についてまとめたデータなどはないのでしょうか。

ないと思います。処理場ごとのデータです。

――データは処理場ごとに保管されているのですね。

大きい自治体で複数の処理場があればまとめて持っているかと思うのですが、それもありません。フォーマットすら統一されていない状況です。

――下水処理場は何ヵ所くらいあるのでしょうか。

全国で 2,200 ヵ所くらいです。

――そんなにあるんですね。

規模の小さいところが大半です。半分ぐらいは本当に小さい処理場です。

――都道府県によって異なるでしょうが、平均すれば 1 県につき 40～50 ヵ所ですね。

はい、そうです。

――実際にできているかいないかは別にして、たとえばフォーマットを統一して都道府県レベルで一元管理できるようになると、何かメリットはあるのでしょうか。あるいは、メリットがないから一元管理されないのでしょうか。

おそらく、メリットがあるかないかもわからないのが実態だと思います。そもそも、そうした発想まで至っていないと思います。

――なるほど。

「自施設がきちんと水の水質を守っていれば問題ないので、ほかの施設がどうしているかは関係ない」という意識が強いのではないでしょうか。

――西村先生は、将来的に一元管理できるようになると、何かいいことがあるとお考えでしょうか。

それはあちこちで吹聴しています。

――その内容を教えていただけますか。

水環境管理は、自治体ごとに行うというよりも、本来は流域ごとに行うべきことです。たとえば、河川の右側が一生懸命頑張っていても、左側が頑張らなければ水環境を保全することはできません。

――そうですね。

ですから本来は、流域単位でしっかりとオペレーションしなければいけません。ただ、現時点ではさまざまな問題があります。たとえば、老朽化、管理者あるいは技術者の不足、予算の不足など、下水道に限らず土木系やインフラ系で共通している問題が挙げられます。

それ以外にも、たとえば水質面では「有機物だけでなく、窒素やリンをきちんと除去してきれいな水を流しましょう」という基準や規制はあるのですが、瀬戸内海などでは下水処理場をきれいにしすぎたために漁獲量が減ったという話があります。まだ因果関係までは明確になっていませんが、下水側は頑張って処理をしているのに、漁業従事者から「そんなことをしてくれたら困る」とまったく評価してもらえない。そうすると、「頑張らなくてもいいじゃないか」ということになってしまいます。

――瀬戸内海の問題はニュースで見ました。

今は基準が決められているため、それに向けて、皆さんはきちんとやろうとしています。しかし、本当にそれがトータルでいい方向に向かっているかというと、必ずしも整合性がとれていないかもしれません。そういう面があります。本来は、トータルの「どうあるべきか」が共有されていて、「そのためにはここまででいいよね」などと考えるべきかと思います。

――たしかに、そうした因果関係は、データをまとめて取って考えると見えてくるかもしれません。少なくとも相関関係はいろいろと見えてくるのではないでしょうか。ただ現状では、そういうこともできないということですね。

そうですね。データをきちんと取れて、それを解析できる技術があり、解析できる人がいれば、さまざまなことが見えてくると思います。ただ、現状はそうしたレベルにまでまったく至っていません。

――集めるニーズはあるけれども、何らかの理由でそこまで業界が成熟していないということですね。

はい。管理者は自施設の業務をしっかりと行うことに専念しています。結果、他の施設についてはあまり関心がないようです。

さらに言えば、おそらくまだ、現場での日常業務を紙でチェック・管理して、それを事務所に戻ってからパソコンに入力しているところもあるかと思います。国交省の B-DASH 事業では、「それをタブレット化する」ことを革新的技術として取り上げています。紙に逐一チェックしていたのを、タブレットを使ってその場でネット上に入力する。一から書くのではなく、前回書いたデータをとりあえず上げておくなど、省力化するというわけです。

しかし、その程度のことであればすぐに導入すればいいことです。少し考えればわかるこ

とです。そうしたことを「導入します」「革新的技術だ」と声高に言っている程度の段階なのです。

――どの業界でも似たようなものですね。それに関連して、下水道の何らかの解析のために、たとえば降水量や気象データなど下水処理場が持っていないようなデータをまとめて収集していると便利ですよね。

そういうことができれば、すごく良くなると思います。そうしたデータをきちんと一元化して、すでに存在している解析技術などを使えばいいと思います。

適切な喩えかどうかわかりませんが、現状は、「もしこういうハードウェアができたらこれができるかもしれない」「量子コンピュータができたら予測できるかもしれない」という次元ではなく、すでに解析技術やＡＩは別の業界の技術者によって開発されているので、それを導入すればできるはずだけれども、実際はそこまでできていないという状況です。その気になればできるはずだと思うのですが、先ほど先生が言われたように統一フォーマットもできていません。

九州大学の先生も指摘していましたが、私が「データを使って省力化する技術なんて、その気になればできるじゃないですか」という話をしていたら、「いやいや、全国でも県単位のデータでも、それをやろうとするとフォーマットを整えなければいけないでしょ」と言われました。もちろん、それはそうなのですが……。いろいろな業者がある中で、どこのフォーマットを使うかによって、特定の業者がアドバンテージを持ってしまうようです。

――ほかの業界でもよく聞く話ですね。フォーマットを整えない主な理由はそこにあるのでしょうか。

いいえ。他業者にスタンダードを取られたくないということではなく、もっと手前のレベルです。単に、もっとデータを広く使うという発想が基本的にないからだと思います。そもそもデータを使うという発想がないと思います。

――では逆に、下水道業界の場合は、そうした発想が広まって国がやる気になれば実現するということでしょうか。

おそらくそうだと思います。

――フォーマットは国が決めればいいわけですからね。

そうなのです。ただ、国が決めると言っても、どこかの業者に依頼することになります。

また、どういったフォーマット形式がいいかについても、いろいろあるかと思います。

――ちなみに、取ることができるデータは下水処理場ごとに異なるのでしょうか。たとえば大きな下水処理場ではこういうデータが取れるけれども、小さな下水処理場ではこれが取れないなどです。あるいは、法律で決められているものが同じだから、あまり変わらないのでしょうか。

どうでしょうか。小規模の処理場については定かでありませんが、管理者、事業者は取るべきデータをきちんと取っているはずです。ただ、システムが微妙に違います。また、維持管理のためのデータはそれぞれの運転管理者と事業者が維持管理業者に発注して取らせているという状況でもあります。

――それは、独自のデータを取らせているという意味ですか。

独自のデータを取る義務はありませんが、なかにはいろいろと取っているケースもあります。

――きっと、そういうところもあるでしょうね。

はい。マニアックと言うと失礼ですが、やらされている仕事だけではなく、熱意を持って一生懸命積極的に運転管理などをされているオペレーターの方もいらっしゃいます。そういう方はいろいろなデータを取っているかもしれません。あるいは、記録はしていないけれども「こういう時には、だいたいこうなります」など、ご自身の経験として蓄積されているかもしれません。そうした熟練の技術者がいらっしゃるという話は聞いたことがあります。

――なるほど。

なかには研究的なことをやりたいという人もいらっしゃいます。実際、自治体の方でも、ご自身の私費で研究されている方がいらっしゃいます。私も知っている方で、横須賀にいらっしゃいます。

――それはすごいですね。では、そのデータはあまり使おうとしないのでしょうか。

データでもっといろいろな情報が引き出せるということを、たぶん認識されていないと思います。もしかすると「自分はやっている」と言う人がいるかもしれませんが、全体的に多くはいらっしゃらないと思います。

――では、オープンデータが必要などの段階にも行っていませんね。

そこまで行っていないと思います。

――ちなみに、窒素などいわゆる管理基準のような値があるかと思いますが、それはどのような経緯で決まっているのでしょうか。昔から決まっている値なのですか。

環境基準というものがあって、それを満足するために、国交省の計画がきちんとできています。略称「流総（りゅうそう）」と呼ばれる流域別下水道整備総合計画です。全体的な計画はそれによって決められています。その中で「この河川はどういう環境基準で、どのレベルの環境基準にしないといけない」とされています。こうした計画を立案するような専門家は、「流域ごとに考えるべき」という考えをもちろん持っておられます。ただ、個々の事業者は「ここではこういう基準で決められているから、それを守ります」という意識です。そもそも役所的な発想ですから、法律などで「こうしなければいけない」と決まればしっかりと守られます。

――私は環境基準についてよく知りませんが、おそらく環境基準も元々はデータか何かを基に決められているのですよね。

そうだと思います。行政のことは私も詳しくありませんが、単なるさじ加減ではないと思います。ただ一方で、全部を把握できているかと言うと、必ずしもそうではないと思います。値は、人間の活動によって出る分もあれば、自然環境から出る分もあるのですが、その量をきちんと把握できていないと思います。ある調査結果はあるのですが、妥当性・信頼性は定かでありません。

――なるほど。

先ほどの「下水処理場がきれいにしているから漁獲量が減った」という瀬戸内海の話も、「本当に下水処理場が原因か？」というクエスチョンがまだ残っています。ですから本当は、自然由来のものも含めて、もっときちんとデータを取らないと有効な対策も取れないという話にもなります。

――本来はそうあるべきですよね。そもそも環境基準が決められて、それを守るためにこの基準を満たすべきという流れかと思いますが、その値自体が本当に適切かどうかも判断が難しいのではないでしょうか。しかも、それを満たすための管理基準も、どんぶり勘定で決

められているのではないかと疑いたくなります。

どんぶり勘定が入っていると思います。環境基準があって、排水基準があるのですが、排水基準は環境基準の約 10 倍の値ですから。つまり、10 倍くらい希釈されるだろうということです。本来はケースバイケースにすべきはずなのですが、それがとりあえずの基準になっています。実は、ほかにもいろいろな問題があります。たとえば、COD という水質項目。あれは学術的には何の意味もない指標なのです。

――えっ、そうなのですか。

COD には、CODMn と CODCr という 2 種類があるのですが、CODMn は汚濁物が 100 あっても、100 と検知できないのです。状況によっては 20〜30 くらいにしかなりません。ですから、研究論文で「CODMn を使いました」などと書くと、「こいつアホか」というような扱いを受けます。けれども、水質項目はマンガンなのです。ですから、自治体などはマンガンで値を測ります。

――乖離があるのですね。

ただ、それを変えるのは、おそらくかなり厳しいだろうと思います。もう何十年も、その指標が続いていますから。よほどの何かがない限り、あるいは行政か政治の偉い人がトップダウン的に言わないと変わらないでしょう。いくら研究者が束になっても難しいと思います。行政には行政の領分がありますから。

――そうなんですね。

矛盾や無意味なことがそれなりにあるということです。本来はそうしたことも、データを示してきちんとやればいいのでしょうが、実際に何かが動くのは問題が生じてからです。だから本当に、私の個人的な意見では、予見的に何かをやれるのはよほどのことがないと難しいのです。

――なるほど。いろいろと理解が進んできました。ここまでの話からすると、下水分野には、データベースなど情報プラットフォームみたいなものは存在しないということでよろしいでしょうか。

現在はまだまあそういう段階です。情報を取ろうという動きはありますが、GAIA や B-DASH などで新技術として実装化したいという思いがあり、国交省もそちらに研究開発費

を付けていますので。

――特にデータベースなどもあるわけではなく……。

まったくありません。逆にそこを早く主張すると、それなりの……。ですから全先生を巻き込んで……（笑）

――産業分野でデータベースが本当に大事なのであれば、先行者利益を大きく取れる気がするのですが。

そうなんですよ。

――それに関連して、たとえば、だいぶ前の質問と少し被りますが、先生の専門分野である下水などにおいて、たとえば「△△、□□みたいなデータを活用すれば、○○という問題が解決できるのではないか」とした場合、△△、□□、○○に入るような言葉は何でしょう。将来的な内容でも結構ですし、実現可能性も考えなくて結構です。それぞれ２～３個挙げていただけるとうれしく思います。

草味のことを言うと、先ほどの「海の栄養塩が足りない」は上手にあてはまります。

――海の栄養塩？

栄養です。下水処理が進み、栄養塩除去を積極的にやっているがために、海自体が貧栄養になって生物生産性が少なくなり、漁獲量も少なくなったと、瀬戸内海の漁業従事者は言っています。海苔の養殖も、やはり栄養塩が足りないので十分育たないと指摘されています。

――それは、どのようなデータを見れば、たとえば濡れ衣かどうかがわかるのですか。

実は、ものすごくきれいな相関関係が表れているデータが出ています。下水処理場下の負荷量の減少と漁獲量の減少だったでしょうか、記憶が定かでありません。ただ、妥当性・信頼性は怪しいように感じられます。

――負荷量の減少というのは、要するに、あまり下水処理しなくても済むようにきれいな水だから、ということですか？

いや、きれいにしすぎたということです。負荷量というか、栄養塩の量のことです。

――下水処理場から出る栄養源のことですか。

　栄養塩の「塩」とは、窒素、リンのことです。

――下水処理場から出る栄養塩の減少と漁獲量の減少……。

　たしか、どこかのホームページ上のデータですから少し怪しいと思っています。ちらっと見ただけですが、すごくきれいな相関というか、経年変化でした。ただ、漁獲量の減少とそこの水域にいる生物量の減少は直接的な関係ではありませんよね。何かの別の理由があって漁業ができなかったということも考えられますから。

――栄養塩の減少以外にも漁獲量が減少する理由はありそうですね。

　いろいろあると思います。たとえば、栄養塩の周期的なものだったり……。

――栄養塩の周期？

　生物としての魚が増えたり減ったりする原因には、何かあるとは思います。ただ、栄養塩が少なくなるからすぐ魚の量が減少して、しかも漁獲量にまで直結するかというと、どうでしょうか。冷静に考えれば、そこまでにはかなりのプロセスがありますから、因果関係をそんなに直に言えるかなあと疑問に感じます。おそらくいろいろなプロセスがある中で都合のいいデータだけを持ってきて、あたかも因果関係があるような図を出してきたのではないかと思われます。

――何か別のデータがあるかもしれないですよね。

　それは十分にあると思います。

――では、ほかに何を見たらいいのでしょうか。たとえば、自然から出てくる栄養などは見ることできるのですか？

　それはよくわかりません。

――そうなのですね。

海域に流入する量のうち下水処理水が大半かと言うと、そうではありません。おそらく半分ちょいくらいです。また、「本当に栄養塩が影響しているのか？」という疑問もあります。ファクターはそこではないかもしれません。

――そうですよね。

ですから、そうしたところである程度の因果関係がわかれば、そうした問題が起こらず、必要十分なことをやればいいということが見えてくるかもしれません。

――そうですね。ほかに何か事例はありますか。

もう実際にいくつかのメーカーがやっているのですが、今、いろいろなセンサーが出てきています。たとえばアンモニアなど、リアルタイムで測定できるセンサーが開発されています。

下水処理場の多くは、空気を送るエアレーションのための電気代が結構かかります。逆に、それを少し絞れば、電気代がかなり節約できます。ただ、そうしたセンサーの値をリアルタイムで把握すれば、かけるべきエネルギーの量がわかるのです。現状では、そのセンサーをすべて付けているわけではないので、過剰なエネルギーをかけて処理をしていることになります。処理場の従事者は、当然ですが、処理が不十分な水を出すことをものすごく嫌いますので、適切な数値がわからず「少なくともこれだけ入れておけば大丈夫だろう」とどうしてもオーバーワークになってしまいます。また、濃度は一定ではなく、日変動や日周期もあります。たとえば 10 万人くらいの下水処理場であれば、朝の 10 時前後がピークになったり……。

しかし、センサーを使えば、下水に汚いものが入ってきた時は多くの空気を送り、あまり汚くない時は送る空気を絞ることができます。下水自身が汚くないので、それほど頑張って空気を送らなくてもいいのです。そういう制御ができます。その制御も、経験などに頼るのではなく、集めたデータを元にできます。現在、東京の町田などで実証実験をしています。

――空気を送ることを、エア……何とおっしゃいましたか。

エアレーションです。曝気とも呼ばれます。

――その曝気は、アンモニア以外でも必要なのですか。空気を送って処理をすべきものは、アンモニアだけなのでしょうか。

いいえ、基本的には全部です。オーバーオールに見ると酸化処理ですから、酸化剤として

の酸素を必要とします。非常にざっくりと言えば、有機物が酸素を使って二酸化炭素と水になるという反応を起こしています。生物を使っているので酸素が要ります。詳しく言えば、窒素や酸素のコントロールもあります。

ですから、それをきめ細やかにすれば空気を減らす余地はあるのです。ただ、今まではセンサーなどがなく、過剰気味に行っていました。空気量を増やしたり減らしたりできない処理場がまだあります。

以前、大阪府で溶存酸素計というセンサーを付けていたので、「これで制御すればいいじゃないですか」と言ったら、「これは酸素があるかどうかをチェックするためのセンサーで、空気量の制御はできない」と言われたことがあります。なぜ制御できないかというと、曝気する、空気を送る装置、いわゆるエアーポンプが１個しかなく、配管がその先で４つの池に分かれていて、手動でちょうど４等分になるようにバルブを調整しているのです。だから、１つの配管への量を変えると、ほかの３つも絶対バランスが崩れて、たくさん出る池と出なくなる池が生じてしまう。今は、微妙なバランスでうまく決めてやっているのだから、ちょこちょこ変えることは絶対無理だ。そう言われました。

――なるほど。

ですから、昔の装置を使っている処理場だと制御ができないのです。素人発想では、独立した小さいエアーポンプを導入すればいいように思うのですが、そういうことができないのです。

――今回、GAIA で取り組んでいるものも、その一種ですよね。

まさしくそうです。

――そもそもの問題設定を忘れ気味になっていますが、先生としては、GAIA は最終的にどういう問題の解決を目指しているのでしたでしょうか。出てくる下水の質でしたでしょうか。

一つは、どういう操作の時にどういう結果が出ているかということを、過去何年間のデータで因果関係などを調べます。その現象はさまざまで、「これが起こるとこれが起こる」などが複雑なので、そういうパターンを認識させ学習させて、もし空気量を下げてエアレーションを下げるとどれぐらいの処理数になるかということを求めます。実際の処理場でオペレーションをしているわけではないので難しいのですが、昨年、全先生に計算してもらったら、きれいな傾向が見られました。

――理屈はそうですね。今の話を聞くと、空気は中の微生物を元気にさせるためだったのですね。

そうです。われわれと同じ代謝です。だから、富士山のような酸素の薄い場所だと少しへたばりますし、空気を与えると元気になります。ただ、空気を与えるためにはエネルギーが必要でお金がかかります。あまり与えすぎず、必要十分な量をきちんと与えることができれば、少しでもエネルギーを削減することができます。

基本的に、下水処理場は市町村がやっています。「もっと広い範囲でやらないといけない」というところは市町村単位ではなく県が行いますが、基本的には市町村です。そうした自治体では、最近「エネルギーをもっと削減しましょう」など CO_2 削減対策をしていますが。たとえば、県庁や市役所などの役所内や事務所では、こまめに消灯をこまめにするなどして、それなりに電気代を低減させる余地があります。しかし、先ほど述べたように、下水処理場では装置的にもなかなか減らすことができない状況です。下水処理場は、自治体の中でも多くのエネルギーを使うところなので、それが削減できればかなりの効果があるのですが、現状のシステムだと削減できない状況なのです。ですから、改築・更新の時期に差し掛かっている今こそ、そうしたことに取り組めばいいのではないかと思っています。

――それをやるにしても何らかのデータによるエビデンスが必要なため、GAIA も重要なのですね。

そうです。そういう点でも非常に重要だと意味付けしています（笑）。

――ありがとうございます。お伺いしたいことはおおよそわかった気がします。お忙しいところたいへんありがとうございました。

付け加えますと、今までの話は下水の話でしたが、上水道は下水よりも早く整備されましたので、老朽化も下水以上に早く進んでいます。また、下水管が使えなくなっている箇所を見つけたいというニーズもあるのですが、地下に埋まっているためよくわからない状況です。そういうことをアメリカかどっかのベンチャーが……。

――何かやっていましたね。

はい。そのいろいろな情報、水道管の太さや管網など物理的な情報から計算して、どれだけの水量が流れている。あるいは、水道管が地中に埋められた地盤情報、たとえば土壌の情報などを入力して、おそらく過去のデータと照らし合わせて、機械学習によって「どういうところに埋設されているどういう状態の水道管が老朽化しやすい」などを抽出して、心当た

りを重点的にチェック・修理するといい、というものもあるようです。

――はい、私も見ました。

　たしか、カルフォルニアかどこかでやっているのではないでしょうか。

――日本でも最近やろうとしているようなことを言っていました。

　そうですね。水道管はかなりいろいろな町ですでに完成されていて、毎年限られた予算しか取れませんから、同時に全部を更新できません。ですから、古いものから更新などをしていますが、老朽化が早く進む場所とそうでない場所があります。そういうことを見つけて、限られた年度予算の中で緊急を要するところを重点的に整備するなどに生かすいう話も聞いたことがあります。

――私もすっかり忘れていましたが、そういう内容を読みました。ありがとうございます。

　ベンチャーみたいな企業はそういうことをできるのです。怒られるかもしれませんが、お役所はそういう発想が足りないように思います。

――そういうところはありますね。

　ですから私は、やはりデータを取って情報を取って生かす余地がものすごくあると思っています。今後も先生のご協力を仰ぎます。

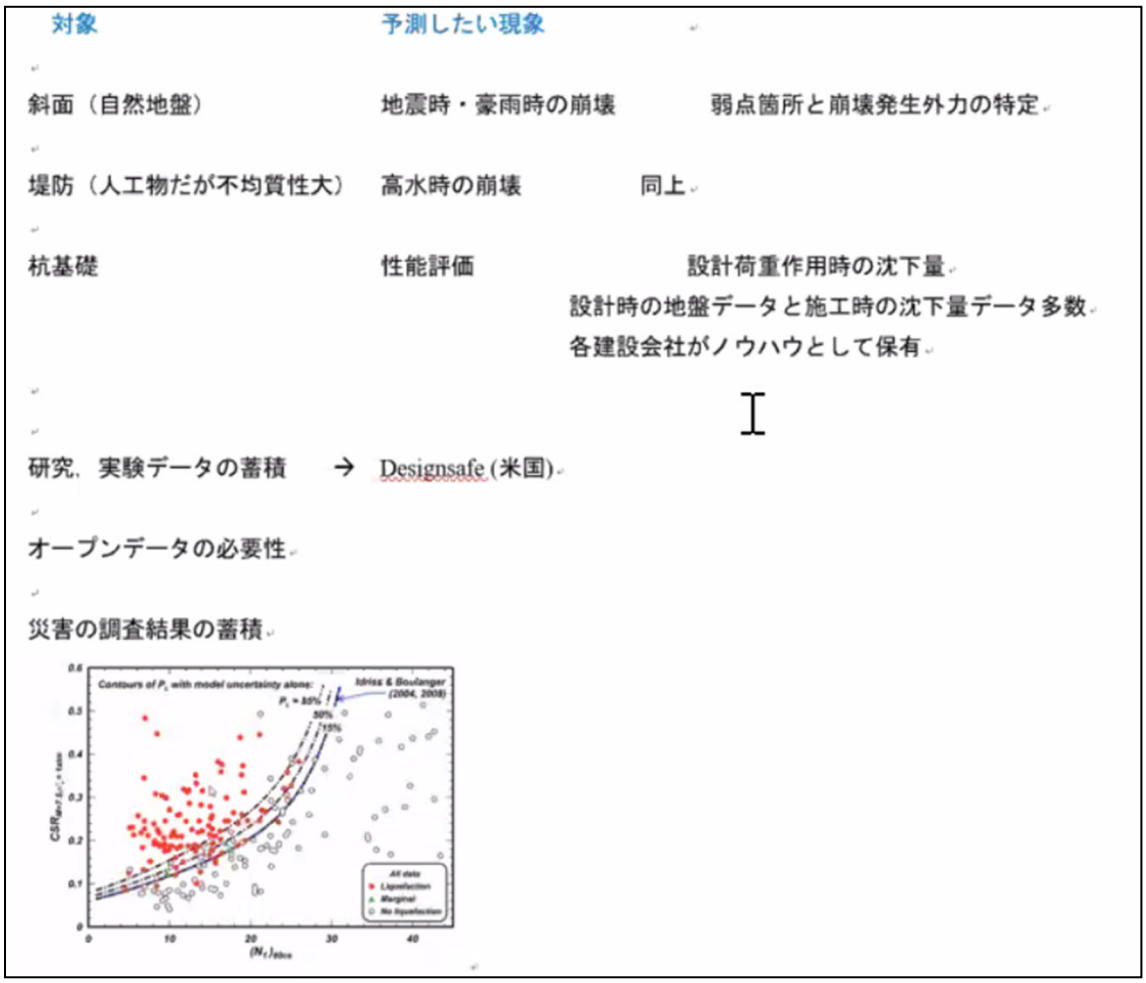
――ありがとうございました。

付.2. 岡村未対教授へのインタビュー

――先生のご専門は地盤工学ですよね。最近、地盤分野において、さまざまな分野からデータを収集・連携して何らかの知見を生み出すような取り組みが行われているだろうと思いますが、どういったものがありますでしょうか。

　実は私もあまり存じ上げていないのですが、説明のため、私のメモを画面共有させてください。

――はい、ぜひお願いします。



図－付１　岡村先生による共有資料

このメモには、よく聞くと思われるものについて、「対象」と「予測したい現象」に分けて羅列してあります。

「対象」としては、まず「斜面」が挙げられます。これは最近の災害がらみで斜面の防災ということです。それから、かねてからご相談している、河川堤防をはじめとする「堤防」です。堤防も斜面と同様、予測の精度が非常に悪いので、多くの崩壊事例の中から精度を上げる何かがないか、蜘蛛の糸にもすがりたい思いです。

３つめは「杭基礎」。杭の基礎の性能評価は、実はもうかなり古くから行われています。ですから、ある意味でかなり質のいいデータが揃っています。ほとんどの場合、杭を施工する前に地盤調査を行い、設計荷重作用時の沈下量、つまり上部構造物ができると荷重がかかって沈下しますので、その性能を評価できるのです。そうしたデータのノウハウは建設各社が保有していて、「このくらいの面積でこれくらいであれば、このくらいで大丈夫ですね」というデータをかなり持っています。ただ、データをオープンにしようという働きはあまり

ありません。

次に「研究、実験データの蓄積」です。日本の地盤分野でも、細かいことはおそらくたくさんされているのでしょうが、大きな動きとしてはあまりありません。しかし、さすがアメリカではデータを一生懸命活用しようとしています。論文集も、特に ASCE では出典データの公開を半分義務付けているというようなことをずっとやっています。また、共同研究をしていますと、必ず同じフォーマットでデータを出してきます。これは、アメリカの土木系の防災データプラットフォーム、DesignSafe というものを彼らがつくっているからです。どのように活用しているのかはよく知りませんが、この 10 年ほどはとにかく DesignSafe にさまざまなデータを蓄積していて、「ここに入れるから、フォーマットを揃えてデータを出せ」と必ず言われます。

また「災害調査結果の蓄積」。アメリカは GEER（Geotechnical Extreme Events Reconnaissance）という組織を作り、世界中の地震や台風，洪水等の自然災害の調査に調査隊を送っていて、そのデータもこの DesignSafe に入れています。このように、アメリカではさまざまなデータを入れようとする動きが以前からあります。

そうした中でもう一つ、最近着目しているのは、画面にお示ししたグラフです。これは液状化判定法のグラフです。日本においても、たとえば斜面崩壊でも、降雨の強度、QL カーブなどというのがあり、指数がいくつになると崩壊するというものがあります。それと同じように、液状化判定法のグラフは、縦軸が外力、つまり地震の強さ、横軸が地盤のＮ値となっています。そうするとグラフのような関係になり、地震度が強くて曲線よりも上に位置すると液状化する、ということになります。

また、グラフでは、現地調査で「実際に液状化した」という証拠があるものを赤点、液状化しなかったものを白点で示しています。理想的には、曲線を挟んで赤と白が明確に分かれるとうれしいのですが、実際には曲線の上にも多くの白点があります。要するに、地震外力が強くて液状化するはずなのに、液状化せずに生き残っている地盤がたくさんあるということです。たとえば、Ｎ値が 20 で地震外力が 0.2 だと液状化するはずなのに、地震外力が２倍の 0.4 以上でも液状化しないケースがあるのです。そうした地盤をアウトだと見なしてしまうと、対策用のお金がもったいないので、以前からずっと液状化判定は続けられています。古くて新しい問題です。

ただ最近は「データの質が悪いからダメだ」「数じゃない、データの質だ」と言い始めていて、この数年は一生懸命「質への回帰」をアメリカ人がやり始めています。アメリカでは、こうした質も量も含めて積極的に行われています。

――「アメリカでは」ということは、日本ではされていないということかと思いますが、このアメリカのデータは世界の誰でも使えるのですか。

DesignSafe ですか？　これは登録すると一応使えるようですが、おそらく使える部分と

使えない部分があって、必ずしもすべてオープンではないと思います。

――日本で同様の動きが出てこないのは、どの辺りが理由になっていると思われますか。

この間も言いましたが、東京大学の生産技術研究所で 1980 年くらいから小池俊雄先生がデータベースをつくっています。たしか DIAS という名称です。あれは立派だと思います。ただ、地盤の分野であまり動きが出てこないのはなぜかというと、うーん……。

――少し視点を変えます。たとえば地盤の分野でデータが DesignSafe のように蓄積がされていると、何かに役に立つのですか。

問題の一つはそこですね。その部分の認識が、おそらく日本人は結構甘いと思うのです。小池先生の DIAS もまさにそうですが、なぜ、アメリカ人がそこまでデータを信奉して蓄積していく努力を一生懸命するのかを考えるべきです。日本人にはあれほどの熱意がありません。目先のことで役に立たなかったらすぐに飽きてしまう傾向があります。

――そうですね。

また、地盤の分野では、杭については、データ共有という意味ではまだ実現されていませんが、うまくいっている成功パターンだと思います。

しかし、斜面や堤防、特に堤防は管理者が国や県などの役所です。ですから、何か問題が生じるとすぐに瑕疵などの管理責任を問われるため、データをあまりオープンにしたくないという文化がずっと続いてきました。最近は必ずしもそうではありませんが、2000 年頃まではずっとそうでした。建設省などは裁判で負けたりすると、殻のようにデータを出さなかったりしました。報告書や論文を出す時も、国交省から「事前に見せてください」と言われ、「都合の悪いことは困ります」という態度です。だから、今後使えるかもしれない正直なデータを皆さんにオープンにして自由に使おうという文化がないのです。そんな感じがします。

――地盤の分野で自由に使おうという文化があると、何か役に立つのですか。たとえば、行政がデータを完全オープンにした場合、最も改善されそうなところはどの辺りでしょうか。それとも、実はあまり役に立たないのでしょうか。

私は、斜面にも堤防にもデータが役に立つだろうと思っています。

――斜面や堤防には、たとえばどのようなデータがあるのですか。壊れた、壊れていないと

いうのは当然あるのでしょうが。

国や自治体が管理する道路脇の斜面については，道路の管理上，斜面カルテが整備され点検結果が蓄積されています。それ以外ですと，斜面には国有地もありますが、私有地が多い。そうした私有地のデータは全然ありません。斜面の傾斜や植生など、空から見えるような衛星データはもちろん多くあるのでしょうが、それ以上のデータとなると……地上のデータもありません。

――ちなみに、これは別件で後ほど相談しようかと思っているのですが、斜面の崩壊は何が計測されていればわかるのですか。たとえば、三次元点群か何かを押さえていればわかりそうなものなのですか。三次元点群とＮ値、ほかに降水量などですか。

抵抗力については、地形、つまり斜面の形状と、Ｎ値でも何でもいいのですが地盤のデータがあればわかります。ほかにもちろん、雨の量などの外力も必要になりますが。

――抵抗力はそれでわかるのですか。

はい。

――ただ地盤情報は、ボーリングデータもそれほど密に取られていませんよね。

密にどころか、山をボーリングする人はいません（笑)。

――そうですね、あまり使い途がありませんね。

はい。

――では、ボーリングデータ以外はどうしたらいいのですか。

山の場合、すべてを調べるのではなく、谷に近いところだけを重点的に深く調べるのです。土砂が谷を下っていくわけですから。ただ、谷から離れているところから深層崩壊のように大きくドーっと滑ってくるようなところは、なかなか調べきれていません。

――では、全面的なボーリングデータが取られていて、三次元形状がわかっていれば、わかるものなのですか。

ほぼわかると思います。

――逆に、ボーリングデータが取られていない場合、何か代替となるようなデータはあるのですか。

現状ではありません。斜面崩壊はある一定の期間の再現現象で、風化が進み、50 年か 100 年経って表層が分厚くなってくると重くなって滑るわけです。そして、滑った後はツルツルの岩盤が出てくるので、表土がありません。斜面崩壊はその繰り返しの現象です。ですから、表面の土の厚さ、表土がどのくらいあるかだけでもデータがあれば相当違うのですが、たとえば 3 年前に滑った場所では、もう滑る土はありません。たとえ 1000mm もの雨が降っても何も起こりません。

――なるほど。そういうものなのですね。勉強になります。

傾斜の激しいところであれば、もっと短いスパン、たとえば 30 年に 1 回滑ったり、土が薄くて滑ったりするかもしれません。傾斜や土の厚さなど、いろいろな相関があります。ただ、そうしたデータさえあれば、おそらく予測はできます。ですから、価格の安いセンサーを日本中に取り付ければ、おそらく予測ができるようになるでしょう。IoT ならぬ、Io ジオテクニカルと言えるでしょうか（笑）。

――堤防はいかがでしょうか。たとえば、堤防で予測したい現象として崩壊現象があるとした場合、どういうデータが寄与するのでしょうか。

土の内部の水の状態です。堤防も、湿った状態では硬いのですが、水を吸ってしまうとブヨブヨに軟らかくなって破堤します。だから水を吸わせてはいけないのですが、表面の舗装にところどころクラックが入っていて、そこから水が入ることがあります。道路に水が入るとそこから傷むのと似たようなものです。弱部あるいは表面の地形などにより水が集まりやすいところは、だいたい崩れます。

――堤防はそんなに崩れるものなのですか。

頑張ってはいるのですが、崩れます。たとえば、たしか一昨年の東日本の台風の時は、140 ヵ所で破堤しました。

――そんなに破堤しているのですね。

140 ヵ所と聞くと多いと感じるかもしれませんが、関東から東北にかけて至る場所で越水し、越水の延長が何百キロもある中で、たった 140 ヵ所で済んでいるのです。破堤の延長としては合計 1 キロか 2 キロもないのではないでしょうか。相当頑張っていると思います。

――なるほど。

イメージとしては、10 キロにわたって非常に大きな外力が作用すると、その中で 1 ヵ所が破堤するという感じです。けれども、1 ヵ所でも破堤すると被害が大きいので、管理者には堤防の更なる信頼性の向上が求められます。

――堤防の内部の水分量は測られているのですか？　測られていないのですか？

測られていません。

――そうすると、今はまったく予測不能な状態ということですか。

そうです。

――なるほど。

堤防は、基本的に形状規定で作られています。

――形状とはカタチのことですか。

高さとのり面勾配。「のり面勾配が 2 割以上，余裕高を 1,2m 設けるといったように。
しかし、ようやく 2000 年ころから「それだけではいけない」ということになりました。そして、1 キロか 500ｍに 1 本くらいのボーリングを全国で実施・評価して、「この辺りは危ないから対策を考えましょう」など、ということをやっています。ボーリングを飛び飛びにしているので、間の部分の状態はもちろんわかりません。

――そのボーリングで調べたのは、水の量ですか？　それとも別の地質情報ですか。

地質の情報です。たとえば、雨が降ったら水が入りやすい砂っぽい成分なのか、基礎地盤から地下水が湧いてきそうな地質か、などです。ただ、所詮数百 m から 1 キロに 1 ヵ所ですから。

――１キロに１ヵ所で十分なのですか。

いやいや、まったく足りません。１キロ先は肉眼で見えないほど遠いですよ。その間にどれだけ悪い箇所が潜んでいるのだろうかわかりません。

――所詮、点的な計測で、倍にしても 500ｍですからね。

そうなんです。物理探査もある程度実施されてはいるのですが、見たい内部欠陥のスケールが違いすぎます。本当は堤防にパイピングで穴が開くなど、センチメートルから何十センチメートルの大きさの欠陥を見つけたいのに、物理探査や１キロに１本のボーリングでは。非常につらい現状です。それでも連続的な地層分布が把握できるという点では物探も重要なツールですが。

――私は専門外なのでわかりませんが、「データ収集の問題点」と関連して、堤防で１キロおきにボーリングすることは、筋のいい話なのですか。今の話だけを聞くと、たとえばそのシステムに 100 倍のコストをかけてもまともな調査になるのかどうか疑念を抱くのですが、そもそも筋のいいアプローチなのでしょうか。

そう言ってしまうと元も子もないのですが、形状規定で「中が何であろうが、形があればいいよ」という段階から考えれば、力学の観点が入ったという意味では大前進だと思います。

――それは同意できるのですが……。

しかし、１キロに１本のボーリングでどれだけ実際の予測精度が上がるのかという点では、今のところは筋が悪いと言わざるを得ません。たかが１キロに１本ではありますが、ほぼ全国ですから使った予算は膨大で、その割に飛躍的に精度が上がったというわけではありませんから。でも、向かうべき方向性は、やはり力学を入れていくベースでやるしかありません。

――その点は極めて同意します。

方法はともかく、力学を入れる方向でデータを取っていっていくしかありませんので、私は「筋が悪い」と言いたくないのです（笑）。

――「筋が悪い」という表現が不適切でした。私も、力学に行くという方向性はいいと思っているのですが、一方で、ほかの方法は考えられるのだろうかとも思うのです。地盤の情報

を調べるために、たとえば面的に測る、あるいは線的に測るような技術があればいいのでしょうが……。その技術がなければ、それ自体が問題点になりますよね。

それはそうですね。大量のデータが集まってくると、データドリブン的に何かが画期的にわかってくるかどうか、その辺りの道筋は、はっきり言って私にもよくわかりません。ではなぜ、たとえば斜面や堤防に関するデータを集めなければいけないと思っているかと言うと、データドリブンももちろんそうですが、学問・学術の進歩のためです。

現状、たとえば今国交省では、堤防の中の水について数百 m から 1 キロに 1 ヵ所、「こんな計算をして、安全率が一定値以下の箇所には表面に遮水シートを張るなどの対策工をしなさい」などのような照査・点検・対策を行っています。そして、災害が起きるたびに、「われわれの点検法では安全率 1.2 だったが、たまたまここの地点は 0.9 だったのでアウトになった。これで説明がつきます」というような、お役所的に瑕疵責任を問われないような説明ができる報告書がダーッと出てくるわけです。お役所的にはそれで結構なのでしょう。しかし、学問・学術の進歩の面では、もう少しきちんと正直なデータが蓄積されていないと、新たなアイデアがある人が検証の材料として使おうとした時にせっかくのデータが役立ちません。質のいいデータをとにかく蓄積していく必要があります。崩壊にはおそらくさまざまなパターンがありますので、信頼できるデータをなるべくたくさん蓄積していきたい。そういうデータの蓄積もしたいという思いが、片方にあります。

――極めて同意します。それこそ DesignSafe の液状化の話のように、まず「データを集めればいいだろう」と始めて、次に「質が悪いから何もできない」と質への回帰がなされるのは、極めて自然なことで、同意します。そういう意味では、今日お伺いした内容は質問の意図を満たしている気がします。

とりとめのない話でたいへん恐縮です。斜面は、雨が降って崩壊します。堤防は、越流をして表面が削られたり、あるいは下から水が通ってパイピングしたり、いくつかのパターンが複雑に絡み合っています。そうしたいろいろな被害パターンを知りたいと思っています。

――なるほど。

斜面については、先ほど全先生が「N値がたくさんあればわかるのですか」とおっしゃった通り、確かにそうなのです。もちろん細かく言えば、地盤にはさまざまな層があって、「この地盤はたまたまこの途中に火山灰の層があったから、ここが何か悪さをした」など特殊な事例もありますが、そうした地盤の層状の構造も含めて、データさえあれば斜面の安定に関してはかなりわかります。もちろん、「山の上のほうから、どこかの水が伝わって、たまたまここから出てしまって壊れた」というような特殊事例もあり、全部を説明できるわけでは

ありませんが。かなり体系的にわかるはずです。
　一方、堤防のほうは、果たしてどういうメカニズムで壊れているのかがよくわからない点がありますので、どこでどのようなデータを取れば本当の解決につながるのかもわかりません。そうした苦しさが依然としてあります。

――そうですか。そういえば以前、先生が東京にいらっしゃった時、災害が起こった時にデータがどうのこうのという話をされていました。あれはどのような状況なのでしょうか。

　すでに動かし始めました。あの時にお話ししたことは、たしか二つです。まず、なるべくたくさんのデータを取りに行くためマンパワーが必要なので、とにかくたくさんの人にコミットしてほしいということ。それで最初は、スマホを片手に LINE で簡単に写真を撮りながら、「この程度の報告をしてくれるだけで災害調査団の名称を与えます」みたいな、軽いトーンから入りました。その上で、「もう少しきちんとした被害調査をなるべくたくさんの地点でして、データを取りましょう」と二段構えにしました。それは地盤工学会で動かし始めています。
　ただ実は後半部で、どういうデータを取っていけばいいのかと悩んでいます。取れるデータはなるべく全部取ったほうがいいのでしょうが、なかなか難しい点もあります。そこは皆さんといろいろ知恵を絞りながら、どうしようかと考えているところです。

――比較的、地盤よりわかりやすいかもしれないインフラにおいても、結局どういうデータを取ればいいのかわからないということは頻発する問題です。もちろん、全部のデータは取れませんので。ですから、さらに不確定性の高い土の場合は、もっと難しいことがあるのだろうなあと想像します。そういう話だけでもすごく助かり、勉強になります。

　「それを東大の生研の DIAS にどんどん入れてくれていいですよ」と言っていただいているのですが、彼らはデータの専門家なので、彼らの話のレベルについていくのが大変です。こちらがまだまだ勉強しないと会話がなかなか成り立たないシチュエーションも多くあります。でも、先ほどお話ししたように、将来の検証材料も含めて、さまざまなデータがプラットフォーム上にあって、スススッとタグ付けされて出てくるような形にしたいわけですよね。そういうデータの蓄積の仕方もわれわれはよくわかっておらず、今後勉強しなければいけません。

――データ管理の仕方は、特に私はたいへん弱く、誰も教えていないし、誰も勉強していないように思います。今日はいろいろな話をお聞かせいただき、最初の質問に対しても聞きたかったことをほぼ聞かせていただきました。オープンデータのお話もしていただきました。

データはオープンにしたほうがいいと思います。ただ、どうするのか。すべてオープンにするのがいいのですか。

――いや、わかりません。

先ほどお話しました国交省の立場のようなこともありますし……。

――場合によってはテロにも関係するかもしれません。

そうですね。洪水の時に利根川の弱点箇所にミサイルを撃ち込まれたら、東京は終わってしまうかもしれません（笑）。

――インフラでも、弱いところにボーンとやられたら落ちますね。一方で、心配のしすぎというか、それはこちらの範疇ではないという観点もあります。

たしかに。本当に使いやすいデータがふんだんにあると、世界中から研究してくるかもしれませんね。

――おそらく、そちらのメリットのほうが大きそうな気はします。

いずれにせよ、土木がデータの勉強をあまりしていないことは、まさにその通りだと思います。その辺りを勉強するにはどうしたらいいのでしょうか。コツコツ努力するしかありませんね。

――そういう教科書でも書かなければいけないかなあ思っているほどです。

それはいいですね。

――土木の人間からそういうことをしないといけないのではないかと思っている次第です。

付.3. 竹山智英准教授へのインタビュー

――地盤のデータについて，現在どういうものがあるのでしょうか。

地盤ではやはりボーリングデータが一番に来るかと思います。

――ボーリングデータに関して、データを集めて何らかの知見を出そうとする取り組みはあるのですか。先生や先生のグループはすごく積極的に取り組まれているようなイメージがありますが……。

我々のグループでは、ボーリングデータを集めて何をするかと言うと、地盤モデルを作ります。たとえば、地盤モデルを作って数値計算をするというようなことです。

――ボーリングデータを集めて地盤モデルを作る際、ボーリングデータは簡単に手に入るものなのですか。

ご存じかもしれませんが、「KuniJiban」というサイトに上がっているものは簡単に手に入ります。誰でもクリックすればダウンロードできます。ただ、数は多くありません。

――もちろん場所によるかと思いますが、密度的にはどれくらいなのでしょうか。

場所によります。東京は結構あります。ただし、国交省管轄の工事のデータですから、国交省の工事に関わるところしかボーリングデータが上がっていません。また、幹線道路沿いや、河川堤防を造るときの河川沿いなどの線状のデータは結構ありますが、まんべんなく広がっているという感じではありません。

――（「KuniJiban」に上がっていない場所で、）先生がたとえば研究で地盤モデルを作りたいと思うことはないのですか？

あります。そういう時には、国だけではなく、地方自治体の持っているデータをいただくなどしています。東京都の場合は共同研究の形でデータをいただきました。「KuniJiban」に上がっているデータよりもかなり密でした。
また、神戸であれば、「神戸 JIBANKUN」というウエブベースのものがあります。

――では、それこそインフラデータプラットフォームのようなものが必要なわけですね。

はい。インフラデータプラットフォームに、国のデータだけでなく、地方自治体のデータが紐付けされていると非常に便利になります。

――そうですよね。

――そもそもボーリングデータには、どのような情報が入っているのですか？

基本的には場所。また、坑口の標高。あとは、N値、土質の分類、たとえば砂や粘土などの情報です。

――よくある柱状図のようなものですか。

そうです、柱状図をイメージしていただければわかりやすいと思います。

――そういったデータは、たとえば「KuniJiban」であれば、Excel 形式で入っているのですか。それとも図面か何かですか。

XML です。それをおそらく自動で処理した柱状図の PDF も置いてあります。両方が置いてあります。

――「KuniJiban」以外のところはいかがですか。

ほかのところも基本的に XML です。

――では、フォーマットはだいたい統一されているのですか。

バージョンがいろいろとあるのですが、だいたい統一されています。

――大変参考になります。それはきっと、データを集めて何らかの知見を出そうとする取り組みですね。

そうですね。

――ありがとうございます。先ほどのお話と関係しますが、データが必要であろう問題に対して、収集時に不都合が生じることはありますか。

不都合があるのは、ボーリングデータではやはりデータが足りないということです。簡単に手に入るデータは足りません。加えて、ボーリングデータにはN値や土層区分くらいしかないものが多くあります。ほかにも、たとえば土の密度（または単位体積重量）も必要ですし、計算しようと思ったら透水係数も必要、圧密試験も必要です。でも、現状ではありません。

――そういうことを知りたい場合はどうするのですか。

お金をかけて室内試験するしかないのですが、そこまでやっているのが少ない状況です。

――たとえば、ほかのところでＮ値や土層区分がわかっているものの室内試験をした結果があるとした場合、そうしたデータはオープンにされるものなのですか。

われわれがやっていることは、Ｎ値とほかのパラメータとの相関のようなデータをたくさん集めて、「Ｎ値とこのパラメータはこのような関係です」みたいな経験式（計算式）を作って、それを基にほかのパラメータを推定するというようなことです。

――そうしたことは地盤屋さんの中では一般的で、誰もがするようなことなのですか。

そうですね。たとえば、ヤング率とＮ値のようなものは基準にも書かれているくらいですから、よくあります。「Ｎ値からいろいろと推定します」ということは一般的です。

――でも、直感的には、Ｎ値と土の種類がわかったら透水係数などもある程度……。

透水係数は厳しいのです。Ｎ値は「何回叩いたら 30 ㎝沈むか」というようなものですので、強さや硬さには関係がありますが、透水性とはあまり関係がありません。
あと過圧密比は地盤の荷重履歴を表すパラメータなのですが、先ほどお話しした、叩いたＮ値を調べてもよくわかりません。

――その辺りが全然わかりませんので勉強になります。「収集に不都合がある」ということは、たとえば透水係数のようにそもそも測っていないということと、必要なのにデータを揃えていないということの、２つを意味していますよね。

そうですね。

――でも、透水係数はきっと必要なのですよね。

はい。

――　一般的な建物を作るなどの場合も必要でしょうか。

そうした場合や、たとえば「ダムを造る」というような場合は詳細に試験しなければいけ

ないので、そういう時には試験をしているはずです。

――これまでのお話ですと、まず、簡単に手に入るデータが足りない。一方で、フォーマットは統一されている。ただ、たまに違うこともある。必要なデータがあったりなかったりする。データについてはそのような状況でしょうか。

はい、そうですね。

――「KuniJiban」はオープンデータ扱いなのですか。

誰でも見れますからオープンデータではないでしょうか。

――そういったオープンデータの必要性や重要性があると思うのですが、なぜオープンにされないのでしょうか。

なぜなのでしょう。

――たとえば、自治体はなぜオープンにしないのですか。

しているところもあると思いますが，システムを作ることなどにお金をかけられないからではないでしょうか。たとえば試験をして、そのボーリングをデータにするということにも結構お金がかかるようです。そうやって調査してデータ化したものを共有してくれたら役に立つと思います。

――そうした知見の共有が可能になれば、重複した業務、たとえば周りに建物を建てる時にも使えるかもしれません。

その通りです。

――お聞きしたかったことはそのくらいでしょうか。今回、いろいろな分野の人に聞いてるのですが、「データなんて使ったことがない」というか、「重要性はわかるけど使えない」という人が多い状況です。「そうだなあ」と思っています。自分の分野のこと、構造物の維持管理を考えても、「そうだなあ」と思っています。

こんなことを言うと身も蓋もありませんが、やはり、土木の構造物は単体物だからではないでしょうか。

――私もそうだと思います。一品生産なので、機械と比べてデータを集めることのメリットが見えにくいというか、大量のデータを集めないとメリットが出てきません。機械であればある程度同じようなものをたくさん作りますので、データを集めるのも短期的な効果がありそうですが、土木はなさそうですから。

そうですね。

付.4. 一言正之様へのインタビュー

――基本的にメールに書いたとおりですが、河川工学分野での現在のデータの扱いや、特に構造分野以外でどのような状況かを知りたいと思い、インタビューさせていただきます。

はい、わかりました。私の分野は水文や河川になると思いますが、私の話せる範囲でお話しします。

たとえば水文の解析などをしようとすると、AI などに限らず、必ずデータが必要になります。まず、地形データから始まり、土地利用、水文データ、降雨のデータ、観測水位、観測流量などです。また、地形と言っても標高データもあれば、河道断面の測量データもあります。そうしたデータがたくさん必要になります。

データの情報源は、プラットフォームというほどではありませんが、一つには国土地理院の情報公開があります。たとえば今、基盤地図情報などでほぼ全国を対象に、５ｍか 10ｍの解像度で標高データが手に入ります。また、土地利用や水域のポリゴン、河道のライン情報なども基本的に国土地理院が公開していて、それが今のところ、プラットフォームの一つになっているかという状況です。

ただ、まだまだ足りないものも結構あります。最近は三次元点群データのようなものがありますが、たとえば河道断面の測量データや LP データなど、国交省の河川事務所が独自に調査しているようなものはなかなか公開されません。特に河道断面は、河川の分野でかなり重要になることが多いので、一番物足りないと言いますか、ネックの一つになっているかと思います。

また、水文データに関しては、国交省が観測している雨量データと水位データに関する「水文水質データベース」というデータベースがあり、基本的にはそこでほぼ全部を見ることができるようになっています。これも一つのプラットフォームです。ただ、これもまだ完璧ではなく、実際には 10 分ピッチで観測されているのですが、公開されるのは 60 分ピッチでしかありません。

――それはどうしてなのでしょうか。

私の聞いたところでは、1時間データについては、事後に専門家が水文照査をして、「このデータは間違いないね」「この期間は工事の影響で少しおかしい」などときちんとチェックをしてから公開することになっているようです。ただ、10 分ピッチまでは照査しきれないということで、1時間データまでだけを掲載している状況です。また、そもそも 10 分データは保管せず、もう残っていないということもあるようです。

――10 分ピッチのデータについては、ある程度はサーバか何かに残っているけれども、公開していないということでしょうか。

そうですね。公開はされていません。数年分くらいは残っていることが多いのですが、それ以上は残っていないことも少なくありません。

――専門外なので筋違いな質問かもしれませんが、結構なデータ量ではありませんか。

結構なデータ量ではありますが、たとえば三次元標高データなどに比べれば、それほど大したデータ量ではありません。

――河川によって異なるでしょうが、いわゆる大河川の場合、だいたい一つの河川でどれくらい計測しているのでしょうか？

利根川や信濃川のような大河川では、数百や 300 くらいです。

――たとえば 300 とすると、10 分ピッチなら 150 くらい。ということは、300 を掛けると 45,000 くらいですか。

はい。

――桁が 45,000 くらいなら、それほど大したデータではないのですか。

そうですね。現時点ですでに 60 分ピッチが公開されていますので、それが 6 倍になるという計算ですが……。

――直感で言うと、データ量としてはせいぜいテラくらいですか。

おそらくそうだと思います。テラまで行くかどうかというくらいだと思います。

――そのサーバの管理は、国交省がやっているのですか。国土地理院ですか。

　水文水質データベースは国交省です。

――別件で愛媛大の森脇先生という水の先生にばったり会った時に伺ったのですが、河川は結構データが公開されているようですね。

　そうですね。かなり進んできてはいます。

――河道断面が公開されないのはなぜでしょうか。ただ単に、河川事務所が公開するのが手間がかかるからでしょうか。

　そうだと思います。公開する理由がないのだと思います。

――なるほど。手間を除けば、別に公開して困るものでもありませんよね。

　困る理由は、私にはわかりません。

――河道断面が知られたところで何もできないですからね。

　そうですね。

――危険性はなかなかなく、テロなどもできないですからね。

　そうですね、ちょっと考えられないと思います。

――では別の質問です。一言さんは、いろいろと機械学習や強化学習をされていますが、たとえば「こんなデータがあればいいのになあ」あるいは「データの成形がものすごく必要になっている」というようなことはありますか。

　そうですねえ、うーん。そこについては特にありませんね。

――それは素晴らしいですね。

　水位や観測データが時系列に並んでいるだけですし、地形データもせいぜいラスターかベクターで並んでいるくらいですので、構造の分野と比べればかなりデータは統一されて

いるのではないかという気がします。

――それはいいですね。また、これは別件ですが、最近、ダムの操作の話をされていますが、ダム系のデータは出るのですか。

国が管理しているダムについては出てきます。それもだいたい、水文水質データベースに載っています。

――そこで質問が二つあります。データが揃っていて公開されているのに、いわゆるプラットフォーム的なものはあまり聞かれません。

プラットフォーム……そうですね。

――プラットフォームにまとめたらきっと便利ですよね。

そうですね。ただ今のところ、水文水質データベースと国土地理院のデータベースがかなり整っていますので。

――その二つだけなので、別にプラットフォームにする必要がないということでしょうか。

はい。また最近、東京大学でDIASというものに取り組んでいますが、DIASに集約しようという話は一応聞いてはいます。
現状では、レーダーの雨量などがDIASに集約されています。ですから今のところ、プラットフォームが三つあるような感じになっています。国交省と国土地理院とDIAS。

――なるほど。もう一つの質問ですが、私から見ると非常にうらやましい話に聞こえるにもかかわらず、一言さん以外から、あまりデータドリブンのアプローチをしているという話を聞きません。

そうですか。いや、意外とそうでもなく、最近になってデータドリブンでやっている方がかなり増えました。

――最近は増えつつあるのですね。

ええ。まず、民間の競合が飛びついてきて、比較的すぐにキャッチアップしてきました。ほかには、たとえば首都大学の河村明先生や、室蘭工業大学の中津川先生などです。

芳村先生のところも取り組んでいます。芳村先生ご自身はそれほどやっていらっしゃらないと思いますが、学生さんがやっているのかもしれません。

――そうですよね。関係する論文を見ているのですが、なぜ一言さん以外はあまり取り組んでいないのか不思議に思いまして。それで話を聞くと、「データはそれなりに揃っている」と森脇先生も言っていましたが、そんなうらやましい状況でなぜやらないのかなあと思っているのです。すでに取り組んでいる一言さんに質問しても仕方がないのですが。

私が民間の立場で論文を書けているのも、やはりある程度データが公開されているからだと思っています。データが出てこなければ、民間は手も足も出ないことになりますので。
ちなみに、データドリブンはこの１～２年で本当に増えています、水工学でも。

――この１～２年でようやくですか。

水工学では最も大きい水工学講演会というものがあるのですが、昨年 11 月の水工学講演会でも、深層学習による水文分野の何かの研究のようなセッションができていました。
ほかに、普通のセッションの中でも、機械学習の研究、アプローチがかなり増えています。

――それだけデータが揃っていたら、そうなるのが自然というか……。

そうだと思います。

――データプラットフォームの重要性について今お伺いしましたが、今は必然的にオープンデータが存在しているということですよね。

はい、そうですね。

――逆に、オープンデータがないと何が困るのでしょうか。

オープンデータがなかった場合ですか。

――たとえば、水文水質データベースが存在しないと、河川の分野では何が起こるのでしょうか。

私たちのような民間のコンサルタントの立場では、発注者や河川事務所などの経由でデータをもらえますので、ひと手間は増えますが、特に何も……。

――研究ベースとしてはいかがでしょうか。

　研究ベースとしては、かなり不都合になるかと思います。

――データドリブンの研究以外の目的にも、たとえばモデルの研究などでも、こういったデータを利用されているという認識でよろしいのでしょうか。

　はい、そうです。物理ベースのモデルでも、たいていは国土地理院や国交省のこうしたデータを使っています。
　少し話がずれるかもしれませんが、河川には、最後の壁となる気象業務法という法律があります。たとえば「オープンデータができて、洪水予測ができるようになりました」という状況になっても、気象庁がストップをかけて、民間は勝手に公表してはいけないということになっているのです。

――何を、ですか。

　洪水予測などです。省庁以外の組織が勝手に洪水予測をしてはいけないということになっているのです。

――え、どうしてですか。

　国民を惑わせるから、ということだと思います。

――なるほど、たしかに。占い師みたいに。

　そうです。そこはずっと岩盤規制があって、ようやく昨年、今年くらいから、気象庁以外にも門戸を開いてはどうかという議論がスタートしたところです。

――なるほど。たいへん参考になりました。ありがとうございます。

　いえいえ。

――河川などの専門の人、たとえば森脇先生に聞くと、「え？　データはオープンなものだろう」みたいな反応になります。

そうですね。あと残るのは、事務所でやっている測量データなどでしょうか。また、洪水の時には浮子などを投げて記録して流量観測などもやりますが、そうした生データのようなものはやはりありませんね。

――それがないのは、その場でデータを取って公開する仕組みがないからか、それとも、公開するのを嫌がる人がいるからでしょうか。

おそらく仕組みがないのだと思います。

――おっしゃるとおりなのだと思います。ありがとうございました。

付.5. 森脇亮教授へのインタビュー

水文学や気象学の分野では、現地での観測データをシミュレーションにデータ同化することは既に行われていますし、今後も行われると思います。ルーチンで観測されている川の水位や気象データはデータ同化として用いられていますが、今後さらにデータが増えれば増えるほど、さまざまな種類のデータ、あるいは大量のデータをデータ同化するためのいろいろなアイデアが出てくるだろうと思います。

――具体的な問題としては、どのような問題があり得るのでしょうか。

たとえば、水文学系では河川水位などです。

――水位予測ということですか。

水位を予測するモデルに、水位の実測データを同化させるということです。それは、すでに実務レベルでも運用されているようです。

――水位予測にはどのようなデータが必要になるのですか。

現在時刻までの水位です。

――ほかに、降水量などですか。

降水量に関しては、データ同化という意味では必要ありません。降水量に関しては、河川水位の機械学習的なアプローチで使われます。

――物理的なモデルで、インプットは何で、アウトプットは何でしょうか。インプットが水位ですか。

降雨流出モデルでは、インプットは雨で、アウトプットは河川流量または河川水位です。ただ、河川のある区間を切り取ってその区間の河川の流れを扱う場合は、その区間の上流端での流入流量と、下流端での水位が境界条件として必要であり、つまりインプットであり、アウトプットは任意地点の川の水位や流量になります。

――なるほど。

ですから、水位はアウトプットです。そのアウトプットされた水位の実測データがあれば、過去における予測データから出てきた水位と、実際にその時点で測られた水位とを比べて、データ同化をして結果を少し補正します。データ同化とはそういうことです。

また気象予測では、さまざまなものがデータ同化に使われています。たとえばGPS化降水量が挙げられます。GPSの電波は衛星から地上に到達するまでの間に大気中で遅延します。その遅延量が、水蒸気の量によって変わるのです。ですから、その情報を使って、伝播の遅延量から大気中の水蒸気を求めることができます。それを「GPS可降水量」と言います。GPSの観測点は多地点にあり、そこから常にデータが集まってきますので、全国の大気中に含まれる水蒸気の量などがすぐにわかってしまうのです。そういったものを気象予測のためのシミュレーションにデータ同化すると、気象予測の精度を上がると言われています。すでに気象庁が使っている数値シミュレーションに組み込まれているようです。

――そうなのですね。初めて聞きました。

今お話しした GPS 可降水量以外にも、たとえば衛星データや気象レーダーの情報を使えば上空の雲や雨のデータも得られるので、様々なデータがデータ同化に用いられていくものと思います。

――そういったデータ連携のようなことをして作ったデータをたくさん集めて、何をするのでしょうか。機械学習ということですか。

データ同化により数値シミュレーションの精度を上げることができると思いますし、機械学習の学習データとして活用できる可能性も十分あると思います。

河川の話に戻りますが、最近は豪雨災害が多くなってきたため、中小河川にも水位計をた

くさん設置しようという動きがあります。安価に設置できて電源の必要もなく、水位が高くなってきた時だけ水位を観測して携帯電話の電波で送るような「危機管理型水位計」というものです。そうした水位計が増えてきているので、それらのデータが活用されて河川水位の予測精度が上がることが期待されます。

――なるほど。今お話しいただいた例でも別の例でもいいのですが、「こういったデータがあればもっと精度が上がるのに取りにくい」というような事例はありますか。

今は低料金の通信サービスも充実してきていて、スタンドアローンでセンシングデータをリモートで確認できるようになってきていますので、以前に比べると障害はかなりなくなってきているように思います。

ただ、本質的に取りにくいデータはあります。たとえば、風です。風は、地上でデータを取ろうとしても、周辺の建物の影響を強く受けますので、強さも向きも正確に測れないのです。ですから、建物の影響を受けない、建物より十分に離れた上空で測らなければいけません。そういう意味で風は測りにくいですね。

――ほかの分野では、データの隠蔽体質があってデータを出さないという話がよくあります。水文分野でもそういった話はありますか。ウェザーニューズなどはすごくデータを囲っていそうですが。

どうでしょうか。私が経験した中で言えば、太陽光パネルのデータは日射量データに近いので活用できると期待しているのですが、公共機関に設置されている太陽光パネルのデータをくれと言ってもなかなか出てきません。そもそも時々刻々のデータを残していないというのもあるのですが、別の理由もあるようです。

――それはなんですか。

太陽光パネルは、次第に劣化したり、壊れたり、停止したりする時があります。そうしたことがリアルタイムにオープンになってしまうと、きちんとメンテナンスをしていないじゃないかなどいろいろな管理責任を問われるからです。そういう意味で取りにくいデータはあります。

――よく考えたらインフラの維持管理でも一緒の話ですね。

はい、インフラの維持管理と似ているかもしれません。

――ミスだと見られたくないという話ですよね。

はい。気象データにもそういう話があります。気象庁のデータは人が相当な手間をかけてメンテナンスしていますから質として保証されているのですが、それ以外の放置されている気象センサーは次第に質が落ちていくため、信頼性に関わります。また、気象データは、測定機器をきちんと検定していなければいけないことが法律で定められています。きちんとしたルールがあります。研究目的であれば緩和されますが、業務などでやろうと思うとルールに従う必要があります。ですから、IoT でどんどんデータが取りやすくはなってはいるものの、データ本来の質的な保証については留意し続ける必要があります。そうしたことがいつも付きまといます。

――そうなのですね。それでは次の質問ですが、水位データは大量にあると思いますし、気象データも時系列がすごく長いので大量にあると思いますが、そうした大量のデータの扱いはどのようになされているのでしょうか。どこかにまとめて蓄積されているのでしょうか。

それはどうでしょうか。

――水位データをどれくらいの頻度で取っているのかわかりませんが、たとえば分ごとに取っているとしたら、膨大なデータになりますよね。

なります。水位データの場合は、国交省の「川の防災情報」というサイトで誰でも見られるようになっています。「川の防災情報」には、水位だけではなく、たとえばカメラの画像のデータや雨のデータ、ダムの貯水量など、さまざまな全国のデータが見られるようになっています。データはおそらく、そういったところのサーバーに蓄積されているのではないかと思います。また、古いデータになればなるほど見られなくなるなどしますので、そうしたアーカイブの蓄積についてどうしているのかは、私も知りません。

また、「川の防災情報」とは別に、「水文水質データベース」というサイトがあります。これも全国の河川の水位の情報やダムの情報などについて、直近のものは見られます。そうした形で、国が割としっかりデータにアクセスできるように整備してくれています。

ただ、気象データのほうに関しては、より膨大なデータになるので、有料なのです。一般財団法人気象業務支援センターという機関が、昔のこれまでのさまざまな種類のデータを、注文があれば提供するようになっています。ただ、基本的なデータは気象庁のホームページからダウンロードできますので、どこからどこまでが有料サービスで、どこからどこまでがフリーのサービスかということはどんどん変わっていくようです。私も全部は追い切れて

いませんが、今はそうした感じになっています。
　また、東大の DIAS でしたでしょうか。あそこでもそうしたデータプラットフォームのような形のサービスをされていますよね。

――小池先生でしたでしょうか。

　そうです、小池先生です。観測データではありませんが、特に気候変動の予測のデータはこの分野ではよく使われます。

――わかりました。今までのお話では、データがかなりオープンにされている感じがしますが、だいたいそういう認識でよさそうでしょうか。

　そうですね。もうかなりオープンです。

――なるほど。たとえば、DIAS みたいなものはデータプラットフォームの一環だと思いますが、そういったデータプラットフォームはほかにどのようなものがありますでしょうか。「川の防災情報」もデータプラットフォームの一環なのかもしれませんが。

　どうでしょうか。　あと、衛星画像などに関しては NICT、国立研究開発法人情報通信研究機構です。NICT は、ひまわり画像などのデータアーカイブのようなことをしています。それについても、研究目的で申請すれば過去のデータなどを公開してくれます。衛星画像は、可視画像から赤外画像まで結構いろいろあります。たとえば赤外画像では、水蒸気量の推定、雲頂、雲の高さ、地表面の海面温度など、さまざまな情報を含んでいます。土壌水分なども含むため、この分野ではかなり重要な情報として使われます。そういったひまわりの画像のデータは、たしか、NICT のサイトから利用申請すればデータ公開されていたと思います。
――よくわかりました。ありがとうございます。欲しかった情報はおおよそ得られた気がします。

　さまざまなデータがたくさんあり、いろいろな方がいろいろな工夫でデータを活用されていますので、まだほかにもたくさんあるのだと思います。私の話は断片的でしかないと思います。

――森脇先生は数年前からインフラのメンテナンスをよく見られています。お話を聞くと、明らかにインフラメンテナンスよりもデータが整然と公開されているのだと思うのです。その辺りの要因は何でしょうか。

センサーで物理的・客観的に測れるものだからではないでしょうか。物理量なのであまり人間の判断が入る余地がないため、オープンなのではないでしょうか。

――なるほど。水位が高いものは高いですからね。

そうだと思いますね。

――ありがとうございます。大変よくわかりました。

付.6. 吉井稔雄教授へのインタビュー

（本インタビューは，
https://www.oriconhd.jp/project/post/traffic/921bfdb381a9df6446edab1def726fa991691864.html の R-アラート WEB のサイトを共有していただいた上で進めています）

――事前に資料をお送りいただき、ありがとうございます。

役に立つかどうか、わかりませんが。

――ちなみに、このシステムはどの程度まで動いているのですか。

今、動いています。ですから、ホームページ上では、それがリアルタイムで動いています。現在から過去 15 分以内に走行したプローブのデータを使って、現状の交通状況を出しています。そして、過去のデータと比べて異常を検知されたらアラートが出るシステムになっています。

――何のアラートでしょうか。道が切れているなどのことでしょうか。

先日、大雪で自動車が立ち往生したニュースがありました。そうした立ち往生などの直前に、「今、少し車の動きがおかしいですよ」というような情報が出ます。

――そうした情報が、たとえば気象条件と合わせてより精度が上がるというイメージなのですか。

そうです。交通だけではなく、気象条件や地形の情報、アジア航測から出た土砂災害の情報なども加味しながら、アラートを出しています。

このアラートは、「ここは何か普段と違う」ということを示しています。例えばアラート

が出た周りに、近くに濃い雨雲が見られるとします。そうなると、この辺りは動きが普段と少し違うということを意味するのです。１台１台の車の動きはわかっていますので、それが非常に遅くなったり、いつもに比べて極めて遅くなったりすると、こうしたアラートが出ます。もちろん、天候の情報なども考慮しています。たとえば、雨が降って速度低下などが生じているのではないか、という感じです。これはリアルタイムの情報です。

――こうした取り組みがあるのですね。

　そうです。すでに行っています。

―そして、ＳＮＳなどと連携して「ヤバイよ」ということに……。

　そうです。

――このシステムでは、車両の動きが遅くなっている以外には、どのような情報提供をするのですか。

　アラート自体は、何を言っているのかわからないのです。とにかく「非日常だ」と言っています。

――とにかく非日常……一種の異常状況のようなことですか。

　そうです。「何か異常だよ」ということを示しているだけで、その内容はそれぞれ違います。たとえば、冠水の情報なども出そうとしています。降水量があるレベルを超えて、しかも車両の速度が低下したような場合です。いわば冠水の情報ですね。

――それは「冠水」という主旨で出るのですか。それとも、あくまで「異常だよ」なのですか。

　「異常だよ」と言うだけです。ただ、「道路冠水じゃないかなあ」という推測はできます。

――「異常」だと言われて、天気を見れば……。

　大雨が降っていますからね。

――人間は「冠水かな」とわかりますね。

人間というか、道路管理者にわかればいいのです。

――はい、当然、道路管理者も推測しますよね。

「この辺はもしかすると冠水しているかもしれない」と思ってくれればいいということです。

――なるほど。

さらに、研究段階なので、まだこのシステムには入っていませんが、今取り組んでいるのは、ドラレコ活用です。各車に付いているドラレコの映像をＡＩで処理をして直接冠水の状況を捉えようと、今やっているところです。

――できそうですね。

ただ、ＡＩを使って冠水を検知するのは結構難しいようです。

――そうなのですか。

学習画像が少ないのです。

――たしかに、学習データ自体は少ないと思います。

ですから、学習データを自分で作る必要があるのです。AI の専門家が取り組んでいますが、それがなかなか難しい。

――そんな気がします。

あまり画像がありませんから、正解なデータが少ないのです。

――そうですね。

だから難しいのですが、今、そうしたことにも取り組んでいます。

――すごいですね。たいへん有益な情報、助かる情報をありがとうございます。こうした取

り組みは交通分野に多いのですか。

この手の取り組みで、情報提供まで考えている大学の研究者は、桑原先生だけではないでしょうか。

――プローブデータなどがあり、いわゆるGISデータとも相性がいい話ですから、もっと……。

民間では、いろいろと取り組まれていますが、基本的に行政が使うようなデータを作るよりは、むしろ、ドライバーが利する、個人に流す情報を作ります。このような行政に流す情報をつくっても利益が得られる保証がないので。そのためHondaなども、「インターナビ」と言ってドライバーに情報提供するシステムを作っていて、大規模災害時には「とおれた道マップ」のような有益な情報を提供していますが、「この道路が通れない」といった情報や、気象情報と組み合わせて「そろそろ危ない」といった情報までは提供していません

――たしかに儲からないのかもしれませんが……。

このシステムの中には、日本気象協会という会社の天候情報、アメダスのデータなどを入れています。気象庁とは異なる組織です。その会社は、気象庁が作った雨のデータを加工して、四国地方整備局などに提供されているそうですが、このアラートシステムについても開発費が回収できるような適切な値段で買ってもらえるとよいのですが……。

――でも、できているシステムがすごいですね。しっかりしています。

これがオンラインで動いています。そういうシステムがあるのです。

――こういった取り組みをしようとする際、不都合になることは何でしょうか。たとえばプローブデータを収集する際の不都合があるとすれば、それはどのようなものでしょうか。たとえば関係機関が何かを手放さないとか……。こうした取り組みは誰がどう考えても有益なので、もし簡単にできるのなら、もっと取り組まれているもいいはずだと思うのですが。

プローブのデータがなかなか入らないということが一つ挙げられます。
少なくとも一般には出てきません。

――ところで、先日の座談会でしたでしょうか。雑誌が無事届きました。

ありがとうございました。

――その時、オープンデータの話をされていました。オープンデータの重要性については私も重々理解していますが、吉井先生はかなり強めにオープンデータの重要性を語っていた気がします。もしそうであれば、どのように考えられてそうなったのか、お聞かせください。

私が「オープンデータが必要だ」と言っていたのですか。別の人ではありませんか。

――吉井先生です。

そうでしたか。もちろん必要だと思います。

――私も必要だと思うのですが、吉井先生は妙にオープンデータに意識が強いなあと思っていました。誤解でしょうか。

いえ、データを開示してもらうことは大事です。データを開示してもらえれば、さまざまな交通管制や制御の高度化につながります。われわれの研究レベルだけではなく、複数のデータをいろいろと合わせれば実務的にもできることがあると思います。細かな制御や管制の話になりますが、どんどん使うべきだと思います。

――今の話に関連したものでは、オープンにしたほうがいいデータで、なおかつまだオープンになっていないデータには、どのようなものがありますか。

まず、プローブはまったくオープンになっていません。

――プローブデータですね。あとは何でしょうか。

感知器のデータもリアルタイムではオープンになっていません。

――感知器というのは何ですか。

道路に付いている車両感知器です。高速道路にはたくさん設置されています。

――カシャカシャやられるものですか。

カシャカシャするのはオービスです（笑）。それではなく超音波式の感知器などです。超

音波を路面照射しておいて、超音波が路面に反射されて帰って来るまでの時間によって車が通っていることを検知します。超音波が行って帰ってくる時間は一定なので、それが短くなったら車が通っているということです。そうした感知器の情報は、少なくともオープンではありません。

――なるほど。

ただ、結構使ってはいます。たとえばVICS（ビックス）情報など、高速道路会社の感知器の情報を使って、「ここは今、速度何キロです」というように皆さんに情報提供されています。けれども、生のデータは使われていませんね。

――そうしたデータがオープンになって、APIのような感じでデータを取ることができてリアルタイムで解析できたらいいということですね。

リアルタイムでできるといいですね。今利用されているのは、短くても５分集計などのデータなのです。

――５分集計ですか。

５分で集計しています。オキュパンシーという「５分の間で何秒下に車がいたか」という情報と交通量とで集計しているわけです。それを計算すると速度が出ますから。そうした速度のデータのために、５分平均速度と５分交通量のデータが提供されていますが、生データは基本的に出ていません。しかし、オンラインはさておき、生のデータも使うと車間距離などがわかり、事故リスクの研究などでも使えるわけです。

――そうですね。

でも、そうしたデータは特別に頼めば提供していただける可能性はありますが、なかなか入手することが困難です。

――先生は「複合させればいい」「連携させればいい」などの旨をおっしゃっていたような気がしますが、まさに今の事故リスクの話もそういった話ですか。

事故リスクは、とにかく多くのデータを複合させないとどうしようもないのです。全先生にお送りした資料には、プローブデータと事故データ、モバイル空間統計についても書きました。そのうちモバイル空間統計は、歩行者や自転車など、車以外の動きがわかるものです。

そして、車の動きはプローブでわかります。それらを組み合わせると、「ここは歩行者が危ない場所だ」「ここは車が危ない場所だ」などということが見えてくるのです。そうした話はあります。

――ところで、いろいろと話が飛んでしまいますが、桑原先生が作った「データ指向型モビリティ情報生成グループ」というものがあるかと思いますが……。

先ほどの DOMINGO ですね。

――最初のほうの質問とも被りますが、交通分野にはもっとデータを活用して何かをしようとする研究がすごく多くあるイメージがあります。

交通分野……自動運転がらみですか。自動運転は、まだ車載センサで把握できる範囲のデータしか使っておらず、インフラデータなどの活用はこれからの取り組みでしょう。

――では、いわゆる計画分野でも土木計画分野でも構いません。ほかの分野と比べてデータドリブンなイメージを強く持っていたのですが、そうでもないのでしょうか。

それはモデル分析の話ではないでしょうか。交通行動に限らず、いろいろなモデル分析のことでしょう。

――たとえばデータから……。

最もベーシックなものは、需要を読むものです。たとえばデータに基づいてＯＤ交通量を出すなどの研究は、昔からたくさんあります。でも、すべてオンラインではありません。

――きっと、データプラットフォームのような話ではないのですね。

あるに越したことはありませんが、そういうモデル分析には絶対にデータが必要です。モデルを作るためにデータは必要ですから。
「重力モデル」のことは知っていますか、最もシンプルな。

――聞いたことはありますが、耳覚えがあるくらいです。

東大では教えていないのかもしれませんね。

――東大では基本的に何も教えません。

たとえば、Ａ市、Ｂ市の人口がそれぞれＰ、Ｑとすると、ＡとＢの間の交通量が重力モデルに従うというものです。距離Ｒに反比例して、それぞれの質量の積に比例しますので、「Ｒ分のＧ、ＰＱ」となるわけです。

――はい。

「二つの都市の距離に反比例して、それぞれの人口の掛け算に比例します」というようなモデルを作るわけです。

――なるほど。

そうすると、少なくともパラメータが一つ入ってきます。人口を掛け算して、距離で割って、交通量になるわけないのですから。万有引力の法則ではＧが定数となりますが、重力モデルではパラメータとなります

――それはそうですね。

一番シンプルなモデルでも、一つのパラメータが入ってきます。実際には、人口を累乗して考えるなどして、二つ、三つのパラメータがあるのですが。そのパラメータを推定するために、データが必要になるのです。モデルが一つでも、その土地土地の状況に応じてパラメータの値の組み合わせは無限にあります。

――そうですね、無限ですね。

はい、無限にあると思います。無限にというか、今までやってきたもの全部がそうですよね。

――交通分野は何かちょっと……。もちろんインフラでもそういった微分、微分方程式のパラメータ、係数を決めるのには使いますが。

物理の法則と違って、交通の場合は傾向がわかるだけです。人口が倍になれば交通が倍になるというだけです。原単位という言葉はわかりますよね。「何人いたら何トリップ、何回移動する」などという話は、原理原則ではありませんよね。一方、物理では、「質量と加速度を掛け算したら力になります」など、ガチっと固まっています。

――そうですね。

われわれの場合は、「人口が倍になったから交通量も倍」までは考えられますが……。

――人がどう動くかまではわかんない。

そうです。そこは定数をかけるのではなく、パラメータが必要となるわけです。ですから、このパラメータの値を決定するためにデータが必要となります。データがなければ絵空事で、実務には使えません。

――わかりました。また話が飛んで申し訳ありませんが、吉井先生からいただいたPDF資料の１ページや今のお話などで、道路地図はもちろん、土地利用データや行政区域データも使い道がいろいろあり、プローブデータもきっと使い道がいろいろあることがわかりました。一つのデータでも複数の使い道があるのだろうと思います。

それぞれそれなりに使われています。

――事故データも天候情報もその最たるものですね。

実は事故データについては、「どこの都道府県でこんな事故が何件ありました」という情報以外はほとんど使われていません。

――そうなのですね。車両感知器情報の使い道はいかがでしょうか。

これは交通管制に使います。

――交通管制以外ではどうでしょうか。

今ですか。

――はい。

交通管制以外で何に使っているのだろう。

――たとえば吉井先生ならどう使いますか。

われわれは、予測モデルを作る時に使っています。最近は「AI を使って渋滞の発生を予測します」というようなモデル。それは過去の車両感知器のデータを使ってモデルを作り、オンラインで情報を提供します。

――なるほど。

アイトランスポートが開発した DOMINGO のトラフィックスコープもそうですね。トラフィックスコープには車両感知器の情報が入っていませんが、「今こうなっています」という情報は車両感知器からもえられます。

――デジタル道路地図や天候予報はどうですか。

プローブと同様、デジタル道路地図がないと何もできません。プローブとデジタル道路地図はセットです。

――すでに道路地図があるものとして考えてもですか。

はい。道路地図がなかったら、わけがわかりませんからね。

――なるほど。

ただ、DRM とわれわれが呼んでいるデジタル道路地図とプローブデータは、結構リンクしていないのです。プローブデータには座標があるだけというように。

――面白そうな話ですね。

プローブデータは元々、GPS のデータですから、緯度・経度があるだけなのです。それをデジタル道路地図にマッピングするという作業が必要になります。

――プローブデータに緯度・経度はあるのですよね。

位置情報としては緯度・経度しかありません。ですから、「どこの道路の上を走っています」というデータは、加工しなければ出てきません。

――「どこの道路の上を走っているかは、加工しないと出てこない」というのは、たとえば

道路が交差している場合しかり、緯度・経度が少しずれていて少し道から外れている場合しかり、ということですか。

そうです。道から外れていると、右の道を走っているのか、左の道を走っているのかわかりませんから。また、一番難しいのは、高速道路と一般街路が重なるところです。この場合は履歴を辿るしかありません。

――そうですね、

また、速度の情報なども。それはアルゴリズムがあって、地図上に乗せてくれるようなデータに変換する技術は開発されていますが……。

――要するに挟まないといけない。

そうです。それを DRM に、デジタル道路地図上に乗せないことにはわれわれも使えません。

――面白そうな報告書が出せそうです。ありがとうございます。

もう終わりですか。先ほど、トラフィックスコープでアラートなどを出しているという話がありましたが、あれはプローブのデータを使ってメッシュ単位で出しています。

――どのぐらいのメッシュですか。

1km×1km のメッシュです。

――そんなメッシュでいいのですか？

それは地方の場合です。資料にあるアラートが出ているところは山の中ですから、道の上を走っているというよりも「どこのメッシュの上を走っているか」という情報でも十分な情報になります。都市部でも、まずは 1km メッシュで異常を検知、ハイライトしてやり、細かいところはプローブデータや、あるいは SNS などの別の手段で確認する、という使い方になります。

――そうですか。

道路にすると、地図とのマッチング作業が必要になるので時間がかかるのです。

――要するに、「過去のデータを見ると、このメッシュをこのくらいのスピードでみんな行くけれど、何か急にメッシュの動きがおかしい」というような話でしょうか。

そうです。

――それなら、そのくらいのメッシュでも「ここに何かあるぞ」というのがわかる気がしますね。たいへん参考になりました。ありがとうございました。

付.7. 秋山充良教授へのインタビュー

――「データプラットフォームについて現状と事例を整理する」という課題について，まず最初に、「先生の専門分野（コンクリート構造）では、データを集めて何らかの知見を出そうとする取り組みはありますか？」という質問をさせてください.

私にはコンクリートの細かいところを見ていく能力がありませんので、分野横断のほうへ行っています。そういう人間からすると、データを集め、たとえば道路ネットワークの情報や橋梁の細かい情報、地盤の情報、ネットワーク全体に作用するハザードの情報、水害の情報、飛来塩分の情報、強震動の情報などのデータを集め、ネットワークの空間的な広がりをオーバーラップさせて、リスクがどれくらいかということを計算します。そういうほうに行っている人もいるということです。

――いわゆる Frangopol 先生の流れですか。

そうです。私はもちろんそちらですから、そういうふうになります。彼の核にあるのは信頼性などで、どちらかというとソフト的な部分です。私の場合は、そうしたバックグラウンドを持ちながら Frangopol のように一体化して融合している感じでしょうか。

――今のお話には二つの内容があったと思います。一つは、「分野横断」のお話です。もう一つは、データをいろいろ集めてくる必要があるというお話かと思います。

そうですね。

――そうしたデータは、一つひとつ、先生が地道に集めてくることになりますか。それとも

何か統合されているものがありますか。この質問の意図は、何らかのプラットフォーム的なものがあれば、果たして便利なのかどうかということです。あるいは、現状でもすでにあるのでしょうか。

Google ではありませんが、日本全国のインフラのデジタル情報を完全に作るというのは、本当に理想です。たとえば Google マップで、どういう年代の設計基準のどういう構造物があるかがわかるなどです。実際今は、たとえば維持管理の情報についても５年に１回は必ず入るわけですから。

――そうですね。

そういう維持管理の情報が入り、耐震診断をした情報が入り、柱状図やＮ値の情報なども入れば……。今、国交省は、なんとか民間情報も吸収しようとしています。ああいう情報なども同じプラットフォームに入ってくるなどすれば、維持管理や災害のリスクなどについても非常にリアリティを持った情報で評価することが可能になります。

たとえば私が、南海トラフの影響を受ける地域の災害リスクや災害廃棄物の量の推定などをするとした場合、ある程度は GIS で拾えますが、細かい情報を積み上げている割に、やはり最後はどうしてもマクロな検討になってしまいます。ですから、もっとリアルに構造物の情報などが入ってくれば……。

実は、強震動の予測や津波の高さなどをすごく真面目に計算している割に、最後は「エイヤッ！」なのです。

――はい。それは私も感じていました。

そうした点に課題があります。せっかく手法などができていても、最後の部分がそうなのです。ですから、データプラットフォームというか、インフラのそういうものがあるといいかと思います。

――わかりました。先生がおっしゃるように、計算している割に最後は「エイヤッ！」でやっていることについては、私も同様に思っています。私も実際、以前、細かいことやっていた時に、「これは何のためにやっているのだろう？」という思いになったことがあります。

わが国の場合、どうしてもオープンでないところがありますから、すごく真面目に計算しているのに、最後の精度にすべて依存してしまうのです。私は学生とよくこんな話をします。100m 走で（400m リレーで前の）３人が 0.1 秒を争ってものすごく一生懸命走っているのだけれど、最後のアンカーがどうしようもないと。「だったら前の３人は意味がない」という

状況になってしまっている感じです。

――同感です。そういうところが多い気がします。

そこは本当にきちんとやるべきです。本当に、行う価値があると思います。

――これは私の「エイヤッ！」の例として「ミクロの積み重ね」と資料に書きましたが、おそらく秋山先生がおっしゃられていることは、「最後の精度を高めるにはデータプラットフォームが必要だ」ということも含んでいるかと思います。では現在、最後の精度を高めるために、先生は自力で頑張ってデータを集めているのでしょうか。

できることはやりますが、どうしても最後はどんぶりにならざるを得ません。たとえば、ネットワークの改築をやる場合、たとえば三重県A市などと言いつつ、構造物の情報はリアルな情報を持っている疑似情報を使うという感じです。

――「リアルな情報を持っている」というのは……。

たとえば、Google マップに橋があることがわかっているのですが、そこの橋はバーチャルだということです。要するに、自分がきちんと計算できる情報を持っている橋があるとしてやっている、という感じです。

――なるほど。

本来であれば、そこの橋についてきちんとした情報を提供してくれれば全部動くのですが、現状ではその橋の設計基準など細かい配筋情報がわからないので、「こういう橋がそこにあるとする」としているのです。同じ時期に造られた代表的な橋の情報を持っていればそれを使うなどして、自分でリアルなマップな上に偽の橋の情報を置いて解いている、という感じです。

――そうなのですね。たとえば論文などの場合は、そういった方法論を示すのが重要になりますね。

はい。ですから論文では、方法論を示したり、こういう計算が必要なんだと示したりしています。ただ、ジャーナルはそれでいいのですが、本当の防災にするにはどうしても……。

――本当にリアルに役立つためには、当然、リアルな情報が必要なわけですね。

今は本当にそこに壁があります。ただ、今、維持管理では道路の橋とトンネルは全数検査していますので、その情報と防災などをもっと一体化させていけばいいのになあと思います。ですから私は、維持管理から入っていけばいいと思います。国交省もそれなりの情報を持っているのですから、オープンになると本当にいろいろなことができると思います。

また，市町村のデータなどとの連携もあまり出来ていません．アメリカなどでは全部同じフォーマットで情報を吸い上げていますよね。日本はなぜ同じようにできないのかと思います。

――もう１点、たとえばもう少しミクロに構造物という単位で見る際は、どのような情報が必要なのでしょうか。たとえば先生の場合、構造物の腐食などいろいろと取り組んでいらっしゃいますが……。

先ほど言ったような点検・検査などをしているのですから、そうした情報がもっときちんと統一されて、ほかの情報と一緒に入ってくるといいとは思います。

――これは私の問題意識でもあり、まだあまり考えがまとまっていませんのでお伺いしたいのですが、その場合の「統一した情報」とは何でしょうか。たとえば、一つの構造物を見る時に、すべてのデータを何か統一したものに含めることは大事だと思いますし、そうすることで解析も回るかと思うのですが。

たとえば、Google マップなどで調べたい橋をクリックすると橋の全景が出てきて、その橋には何年前にどこにどういうひび割れがあったかが写真とともに出てくる、という感じで全部一体になっているといいと思います。また、設計図面も一緒にあり、配筋情報もわかる。そうしたことができるはずなのです。なぜなら、情報というものは、設計の段階から全部積み重ねてきているわけですから。ところが現状は、設計図面は設計図面、点検・検査は点検・検査、地盤は地盤と、情報が全部分かれてしまっています。その辺りを統一すれば……。また、たとえば補修・補強をしている時にも、おそらく一体になった情報がうまく集まらないと思います。設計したところは設計した情報だけ、耐震補強するところは耐震補強の情報だけなどという状況ですから、全体の解析などをやろうとするとすごく骨折れます。できるようにするためには，例えば情報を吸い上げるところを先に作って「ここに入れろ」という仕組みが必要かとも思います．

――そうですね。

常に後追いになって、「統一したほうがいい」と言う頃にはすでに各々がバラバラにデー

タを持ってるから、「もう今さら……」という感じになってしまっています。

――そうですね。もう少し具体的な話をすると、以前、先生が愛媛にいらっしゃった時に鉄筋の腐食などをいろいろと見せていただきましたが、ああいう予測をたとえばリアルでする際はどのような情報が必要なのですか。

やはり、コンクリートの情報や配筋の情報などが要ります。

――コンクリートの情報というのは、いわゆる配合などの話ですよね。あとは誰が造ったとかでしょうか。

そうですね。今はコア抜きをして検査するものもあると思いますので、そうした情報もひび割れ情報などと一緒に紐づけされているといいですね。

――たとえば一つの構造物であれば、秋山先生がご自身のセンスで適した形式に変えて解析されると思いますが、それは秋山先生でないとできないのですか。

そんなことはないと思います。

――だとすれば、どのように統一的にすればいいのでしょうか。たとえば、そのくらいの問題だとした場合、上手にビジュアライズすれば全部の情報が乗せられるものなのか、それともエクセルか何かで「ここからここはやられてる」などとすればいいのか。どのようにデータを整えるのがいいのか、この１年くらい私の頭の片隅にひっかかっています。

そこを真面目に考える価値はありますね。要するに座標の情報……。

――で、いいのでしょうか。

ただ、本当にきちんとしたデジタルの情報というのであれば、やはり３次元の FEM のインプット情報ではないでしょうか。あのようなスタイルではないかと思います。

――少なくともきちんとした３次元 FEM が復元できるような……。

まあ、FEM ではなくても……。

――要するに、３次元モデルで復元できるものですか。

数値解析に行く人はそのまま数値解析に行けばいいでしょうし、大きいものに行くのであればそちらでいいと思います。ただ情報の集め方としては、メッシュで切って、それぞれにどういうふうに配筋があり、コンクリートになされた部分的な情報をそこの要素に紐づけてというようにすれば、たとえば確率論的な研究もする人なら、そこから空間的な強度の変動を見るなど、さまざまなことができるのではないでしょうか。

――メッシュのサイズや粒度などに関する話もあるかと思いますが、基本的にはそうなりますよね。いや、そうしないといけないのでしょうね、ただ、難しい話だなあとも思います。

たしかに、どこまでの分解能にしていくのか。日本全体のマップからスタートすると、相当の情報量になってしまいますから、かなりたいへんかもしれません。

――そうしたことを今まとめているところですので、今の話だけでもたいへんありがたく思います。参考までに、道路ネットワークなどのテーマで、先生の研究の全体像がわかりやすい論文のようなものはありますか。私の報告書では、「こういう情報を集めないといけないので、こういうデータプラットフォームが必要だ」というような書き方をしようと思うのですが、報告書を書く際に助かりそうなものがあるとたいへんうれしく思います。

どういうものをお伝えすればいいのでしょうか。

――何らかの目的がある時に……何でもいいのですが、たとえば津波の場合は津波避難解析や被害解析でしょうか。

津波であれば、津波でネットワークがどれだけ停滞するか、津波でごみの廃棄物がどれだけ出るかなどでしょうか。最近私はもっぱら面的な計算に関心を寄せています。

――そのように何らかの目的がある時、たとえば目的を一つか二つに定めた時、何を知る必要があって、ネットワーク全体のネットワーク図、あるいは……。

私の場合はリスクを出したいわけで、そのリスクがごみであったり、停滞日数であったり、交通容量の停滞であったりします。そのリスクを出す時には、ありきたりの表現ですが、ハザードとフラジリティの情報が必要です。ハザードは、断層から各地点の地震動強度や津波高さの分布になります。私はスパコンを使うような高度な計算はできませんが。一方のフラジリティとは、要するに、構造物が持っている個々の位置での弱さや強さを表現する情報です。構造物そのものの大きさや設計基準から、それが地震荷重を受けたり津波荷重を受けた

りする時の強さ／弱さを評価する情報です。それらをオーバーラップさせて初めて信頼性が出てきます。そうして、構造物が壊れるとどれだけネットが停滞するか、構造物が壊れるとどれだけごみが出るか、というほうに展開していきます。

――そうしたことをなされた時に、どういう情報を集めたかということを論文に書かれていると思いますが、そのような論文はありますでしょうか。

メールでお送りしますので、あとで見てください。

――ありがとうございます。また、似たような話ではありますが、何らかのデータが必要であろう問題に対して、データ収集時に何か不都合はありますか。たとえば、どうしてもデータが出てこない、そもそもデータがないなど、さまざまあると思いますが、どういったことがありますか。先生の研究では、データを集めなければいけない研究が多そうな気がします。

実験室で行うものはいいのですが、「実験室でやっていることをリアルな構造に」といった時は、先ほどお話ししたような情報を持っているもので仮定するしかありません。

――リアルな構造物の情報を入手する手段がないということですか。

そうです。日本では４段階くらいの基準に分かれます。たとえば橋を例に挙げると、高度経済成長期の前回の東京オリンピック前くらいの時、宮城県沖地震くらいの時、兵庫（阪神・淡路大震災？）の前と後、だいたいこの４つくらいで大きく橋の性能が変わっているのです。ですから、「当時の基準に従って造った橋はこんなものです」という代表的なものがそれぞれあるわけです。それらがネットワーク上に散らばっていると仮定して、今はやるしかないのです。

――それこそ、先生が先ほどおっしゃったマクロ／ミクロの話ではありませんが、これをやってしまう（最後にどんぶりせざるを得ない）場合に、細かく津波の高さなどを調べることの意味が結構消える……。

そう、ありません。本当にそこには大きな壁があります。

――学問では、たとえば「これがわかるようになったら、それぞれの津波の高さを厳密に調べていてよかったね」となりますし、あるいは、方法論をきちんと整備しておけば「これがわかればいいんだよ」というドライビングフォースにもなります。けれども今は結局、そこがボトルネックになっているような感じですね。

本当にそうです。また、おそらくもう一つあるのは、仮に図面情報がデジタルで入手できたら、そのまますぐ解析にいくかというと、そこにも壁があります。たとえば、われわれは地震が起こるたびに被害分析委員会なるものを立ち上げて、耐震解析をして、「この橋はたしかにこの強震動だとせん断破壊する／しない」ということをやります。それをしている時、図面の読み方が大学の人間にはなかなか難しいのです。３次元、それこそ FEM や梁要素でも、モデル化するには一旦それを読み解いてくれるプロがいつも必要です。

たとえば、2003 年に三陸南地震、2004 年に新潟県中越地震が起きて、たくさんの再現解析をしましたが、やはりあの時も常に JR の人が出してくる図面について、解析ができるように読み解いてもらって初めてわれわれは計算できたという感じです。ですからその点も、ただデジタル化するだけではないプラスαの部分として仕事があるような気がします。

――今、よく BIM/CIM あるいは「i-Construction」で３次元モデルと言っていますが、そもそも３次元設計をしていれば解決する問題なのかということも、よくわからない話ですね。

そうなのです。ですから、ああいうことをする時には、きちんと構造屋さんなどそれぞれの使うユーザーが入って作っていかないと、作ったはいいけれども使い物にならない結果になるということは大いにあり得ます。

――そうですよね。

わが国のさまざまなデジタル化の末路を見ると、大いにあり得そうな気がします（笑）。

――そうですね。ものすごく単純な例でも、たとえば支承部をどうモデル化するかというのは、それほど簡単な話ではないはずです。

そうですね。そこをきちんとわかっている人がやらないで、単に３次元のすごいものだけを出されても……。

――そうなるような気がしますよね。

大いにあり得ます。

――「これで FEM ができる」と言ってできても、合わないなどの結果になりそうですよね。

ですから、やはりきちんと使うユーザーがそこに入って作業する必要があります。

――ほかに、データ収集時の不都合はいかがでしょうか。基本的には今おっしゃられたような話ですか。そもそも入手する手段がないからよくわからないことと、たとえば図面とかを入手しても使えるデータにならないということ。

そうですね。鋼（はがね）の場合は、写真などを見て断面の形がわかればある程度は計算できそうな気がしますが、特に鉄筋コンクリートの場合は「その中にある配筋はどうか？」という話になってきますから、もう一段階の壁があります。

――再現する際、配筋はどのような感じなのですか。そもそも配筋が違うじゃないか、ということですか。それとも、配筋がしっかり書かれていない、などのことですか。

いや、配筋はある程度きちんと書かれています。ただ、たとえ同じ帯鉄筋が巻いてあっても、先ほどお話しした４段階の基準によって異なるのです。たとえば、帯鉄筋としてきちんとコアコンクリートに機能する巻き方をしているのもあれば、実は帯鉄筋ではなくて、横荷重を受けたら帯鉄筋がふにゃっと広がってしまうのもあるわけです。ですから、リアルに計算して精密にしようとすると、単に配筋と書かれているだけでは情報不足で、「それがきちんと端部に、どうコアコンクリートにフックされているのか」まで必要になるので、結構たいへんです。

――それはそうですよね。

ですから、そういうレベルの計算までやるのであれば、やはり図面を読み解いて、デジタルにする時にはそこまでのことを考えて読み取ってもらう必要性が出てくるかと思います。

――要するに、何らかの専門知が必要なわけですね。

そうです。ですから、先ほどの話ではありませんが、ユーザーや解析の人がきちんと非線形解析をして、終局限界状態に近いところまでの挙動を追いたいというところまでのニーズを意識するならば、やはりそれなりのものになります。そうではなく、ある程度マクロに「どれだけの耐震性があるか」だけであれば、設計基準と大雑把な構造諸元さえわかれば、「中の配筋はこんなものだろう」くらいのことは、それなりのコンサルタントに聞けばわかるはずですからね。その辺は、いきなり 100 点を求めても仕方ないかもしれませんが。

――なるほど。では、残り時間が少なくなってきましたので、オープンデータの必要性や重要性についてお伺いします。それは、先ほどお話しいただいたとおりでしょうか。

そうですね。

――なるほど。それに関連して、たとえば浅本先生によると、配合比と物性の関係性がわかるデータベースのようなものがあるとのことです。それはわかりやすい発想ですが、コンクリート構造や構造物の維持管理などでは、どういったデータベースあるいは情報システムがあるのでしょうか。そもそも、そうしたデータベースなどはあるのでしょうか。もちろん点検などは記録されていると思いますが。

たとえば、アメリカにはPONTISというものがありますよね。

――あります。

あれにはすでに何十年ものデータが貯まっていますから、「どういう維持管理をやろうとするといくらになる」などについても、インプットすればアウトプットされます。本当はあの辺りが一つの参考になるというか、目指すところなのかもしれません。

――PONTIS自体、別にそれを要求しているわけではありませんが、FHWAのものが機能しているのは２年に１回など、すごい間隔で見ていますし……。

あれは、統一してほぼ全州のはずですよね。ですから、日本の「それぞれのところがそれぞれのデータベースを持っている」というのとは、やはり違いますよね。あの姿になっていかなければ。日本の今のやり方をしていると……。

――ほかにも、コンクリート構造などで設計のためのシステムのようなものがあるのですか。

設計のためのシステムですか。

――ただの設計ソフトウェアですか。

個々の橋を造る時は、橋の形式ごとにそれぞれのソフトウェアがあります。ただその前に、橋の形式を決めなければいけません。実はそこが一番の腕の見せどころのはずなのですが、おそらく多くのコンサルタントでは、橋長やいくつかの項目を入れると「こういう橋梁形式があり得る」というようなものをそれぞれ持っていると思います。そこから３～５案くらいに絞り込んでいくのだと思います。でも、そこにこそAIなどの出番があるかもしれません。

――その辺りは別にデータの話でない気もします。データが集まっていれば、たとえば動解が通りそうな橋など、これを出してくれるなどしてくれればいいのでしょうが、そういう話はそれとして……。なるほど。たしかにPONTISみたいな話はそうですね。

ところで、構造不確実性を伴う問題で日本の話は今お伺いしましたが、こういった構造不確実性を伴う問題では、当然、分布を出す必要があります。分布を出すことに苦労はされませんか。

もちろん、断層の問題などまったくわからないのものについては、仕方がないので「エイヤッ」と決めるしかないのでしょうが、それ以外のものについては結構あるのではないかと思います。

――ものすごく単純な例で、たとえば簡単な床版橋の床版橋の耐荷力を調べなければいけないシーンがあるとした場合、道路ネットワークのリスクを考えると、耐荷力の分布を先生あるいはFrangopol先生のような解析をする必要があるかと思います。しかし一方で、耐荷力の分布を出すのはそれほど簡単な話ではないと思うのですが、そういうのはどうされているのですか。

それは図面ベースで行います。耐荷力は図面ベースである程度の予測がつきます。まず、材料特性もよくわからない中、図面ベースでモデル誤差を考えながら分布を作ります。そのあとはUpdatingです。Bayesianにしても、点検・検査の情報からいかにその分布を正しいほうへ尤度に基づいて変えていくかということが、それこそ Frangopol や私がやっていることではないでしょうか。

――いわゆるBayesian的なUpdatingは当然そうなるのだろうと思いますが、耐荷力の例が良くなかったのかもしれません。それでは、耐荷力以外のことを教えていただきたいのですが、耐荷力の分布のUpdatingのためにはどのような情報があればいいのでしょうか。

たとえばこの間の先生のAIシンポジウムで言うと、私の研究室などでは、「腐食・ひび割れが出ているのなら、耐荷力がこれくらい落ちるのではないか」、そして、「中の腐食はこれくらいだから」と腐食の分布を1,000個くらい予測して、3次元FEMを介して「(耐荷力が)これくらい落ちたところに分布しているのではないか」などと行っています。

たとえばアメリカなどでは、実際にトラックを乗せて、ローリングプルーフテストみたいなことをします。そして、分布の裾野を切ってしまって……。全部を乗せたら本当に危ないので、テール側のある部分までの荷重を乗せて、「そのテールまでは実際に耐えたのだから、それよりも大きいはずだ」というように分布を更新する、というようなことではないでしょ

うか。

――なるほど。

やはり ICOSSAR なのだと……。ああいうところの人たちで言うと、AI ではなく、基本はやはり Bayesian です。ただ、私も今回ずっといろいろとやってきましたが、最近のああいう AI などのほうが、逆問題の精度としてはいいなあなどと思いました。

――それはなかなか難しいと思いますので、むしろ私は、Bayesian が大事だなあと思っています。

私は、学生にも「そこは必ずやりなさい」と言っています。あそこにはきちんと理論もあって、パラメータの認知が必要で、当然、力学や化学も必要です。「それをやった上で、きちんとやりなさい」と言っています。もちろん研究生はしています。ただ、どれだけアグリーしているかわかりませんが、われわれが持てるコンクリートの情報には難しい面があります。たとえばわれわれは、腐食生成物ができて、膨張圧でどんどん腐食ひび割れができて、腐食量が増えるとひび割れが増えるというように計算するのです。けれども、どうしても全部は乗りません。なぜなら、たとえば鉄筋の周りに構造物ができた瞬間に、すごく微細な空隙ができるのです。また、コンクリートの中にあるコアの中に腐食生成物が入って腐食生成物が増えて引張力をもたらすのは、実はそういう空隙の影響などに影響を受けてしまうということもあります。要するに、われわれはそれをモデル化できていないのです。ですからそういう意味で、われわれが計算で捉えられていないパラメータが実はすごく影響しています。そういう捉えられないパラメータの影響が大きい時は、たとえブラックボックスと言われても、あちらのほうが「なんとなく合っているな」というところがあると思います。

――おっしゃることはよくわかります。

私の研究室では、「ベイズにしろパーティクルフィルタにしろ、理論はきちんとすべてやりましょう。その上で、世の中には便利なものもあるから、そちらもやりましょう」という感じです（笑）。

パラメータをちゃんと認知して、情報に基づいて更新をして、確からしいものを置いていくということでは、やはりあそこは基本です。

――極めて同意します。

この間、『Structural Safety』に出した時も、いきなり「AI みたいなもので腐食ひび割

れ幅から腐食量を予測します」と言ったら、早速「なぜベイズを使わないんだ」と意見が来ました。「この雑誌ならそうだよなあ」と思いました。

――たいへん参考になるご意見をいただきありがとうございます。あと、土木では少量のデータを工夫して使うようなシーンがきっと多いかと思いますが、基本的にはそういう状態にありますよね。

まあそうですね。

――一般的に橋梁の解析などではそうだと思います。ただ、少量のデータを工夫して使うことも大事ですが、本来はデータがあるものはあるべきだと思いますが、いかがでしょうか。

もちろんそうです。しかも、あるのです。あるにもかかわらず、集約されていないだけなのです。

――そうですよね。

「共有するだけ損」という方向ではなく、「共有すると得になるんだ」という形に持っていけるようにしないといけません。
ただなかなか難しいですね。ただアメリカでは、PONTIS のようなものとＡＳＣＥなどが個々の橋の状態などについて昔からどんどん積極的に公開しています。

――なぜでしょう。国民性の違いでしょうか。

何か危ないことがあった時に、「積極的にそれをオープンにして危なくないようにしよう」という考えと、「隠してこっそりやっていこう」という違いがあります。それが、2011 年のような大きな事故を起こしてしまうのだろうなあなどとも思います。

――それはそう思いますね。

みんなで「危ないものは危ない」と認知し合う必要があると思います。
本当にもっと情報をオープンにしてほしいと思います。

付.8. 北原武嗣教授へのインタビュー
――北原先生の専門分野は耐震構造ということでよろしかったでしょうか。

一番のメインは耐震構造でしょうか。地震動特性を踏まえた耐震設計や耐震信頼性・リスク評価というものもあるのですが、もともとは構造から入っていますので。

――さて、最初のご質問ですが、データを集めてあるいは連携して何らかの知見を出そうとする取り組みはありますか。きっとあるだろうという認識ですが。

私自身が具体的に知っている取り組みがあるかと聞かれると少し疑問ではありますが、一つは、地震・耐震の分野で最もデータがオープンというか、いろいろ集めて公開されているのは、K-NET、KiK-net の地震観測網だと思われます。

ただ、K-NET、KiK-net は地震データです。そのため、たとえばそのデータを使って距離減衰式を構築する、もしくは確率論的な指標化に使うなどのケースでは活用できると思います。ただ、地震工学の分野では、私自身は本当に地震防災につなげていくことが AI やデータサイエンスで重要だろうと思っています。ただ、地震被害というのは頻度が少なく、そもそもデータが少ないのです。K-NET、KiK-net のデータに関しても、そこが一つの弱点だと思います。地震計もたくさん構築され、すでにかなり整備されていますし、阪神・淡路大震災以降にかなりのデータが蓄積されてはいるのですが、小さい地震のデータが多く、大きな地震のデータがたくさん取れているわけではありません。そうすると、こういう手法は、内挿は当然得意ですが外挿までできるかというとなかなか難しい面があります。われわれが地震防災や耐震設計を考える際、「巨大なケース、過大な地震に対する評価をしたいのに、そのデータが少ない」ということが問題として残っているように思います。

――大きな地震のデータがたくさん取られているわけではないというのは、大きな地震が少ないからということですか。

そうです。

――阪神・淡路大震災以降だと、中越、東日本、熊本……。

本当に大きいのは東日本大震災と熊本地震くらいで、被害の大きさであれば新潟県中越地震や福岡県西方沖地震などいくつかあります。ただ、やはりある程度の被害が出たという点と考え合わせると、東日本大震災と熊本地震くらいしかありません。

――そうした感じで地震データがあるとのことですが、地震防災や地盤のデータなどであれば……。

私自身は地盤が専門ではありませんが、地盤のデータは、世界的にいろいろデータベース

化するという動きがあると聞いています。実際に、シンガポールの先生が中心になってデータベース化をされていると思います。それには、いろいろな世界中のデータベースが割と集まってきていると聞いています。おそらくオープンで使えていて、それこそこういうグループたちが「データドリブンなアプローチに活用していこう」というような研究などの取り組みをされているので、進んではいると思います。ただ、私自身は地盤が専門ではないので、それが本当にどこまで実用化・活用されているのかということはわかりません。また、全世界でデータベース化をすれば当然そのデータ数は増えてくるのでしょうが、地盤にはおそらく地域特性があると個人的には思いますので、それをどう評価するかが課題かもしれません。

――先生がおっしゃっている地盤のデータベースとは、どのくらいの深さのイメージなのでしょうか。

調べればいくつか知見はありますが、具体的にちょっと……。

――オーダー、あるいは……、たとえばボーリングデータで取れるような位置の……。

基本的にはおそらくボーリングデータのイメージだと思います。

――地震のことはよくわかりませんが、あるいは、より深いところも関係しそうな気もするのですが。

そうですね。しかし、そこまでのデータは本当になかなかないという感じではないでしょうか。

――「地盤のデータは減衰などに大きな影響を及ぼすのだろう」ということは、物理メカニズムから想定できますが、きっとそうなのですよね。

そうですね。地震動は、震源によって違いますし、地下構造の伝播特性でも変わってきます。さらに、表層の増幅特性の兼ね合わせとなります。そのため当然、ソースとしての断層のデータがなければいけません。次に、深い地下構造で伝播特性がどうなるかを評価できなければいけません。そして、表層の増幅特性です。

ですから、ここで言っている地盤のデータベースは、基本的に表層の増幅特性の部分で有効だとは思いますが、地下構造の伝播特性の部分ではなかなか活用しきれないのではないかと思います。

――なるほど。ほかには何のデータが必要でしょうか。地盤についてわかったら、あとは構造物や住宅などのデータでしょうか。

地震のデータがある程度揃ったり評価できたりしても、やはり結局は「どういう地震が来たら被害に結びつくか」というところがデータとして必要なのです。動的解析は十分できてきていますので解析すれば応答値は出ますが、解析は結局モデル化したもの同士の結果です。地震応答解析に使う構造モデルはあくまでモデル化したものです。ですからやはり実現象とは全然違ってきますので、実現象と地震被害との相関関係のデータが必要なのですが、現状で十分かと言うと、なかなか少ないのが現実かと思います。

――そういうところで一応必要なのは、おそらく、地盤と構造物と地震、あと被害状況調査結果でしょうか。被害状況調査結果のデータはあるのですか。

東日本、熊本、神戸などの被害状況は調べられていますが、結局はそれくらいしかありません。それらだけでもある程度の定性的な分析をするには結構な量なのでしょうが、先ほどお話ししたデータドリブンのアプローチをしようと思うと、十分な量とは言い難いのではないでしょうか。

――さて、K-NET、KiK-net 以外で、少し観点を変えて、ほかにもありますか。

地震防災の分野であれば、もう一つの方向性として、宮本先生が取り組んでいるような衛星データを使った事後の被害でしょうか。復旧のための早急な被害状況予測、あるいは被害状況というところには使える可能性があるとは思います。ほかにも数名、建築の分野でも取り組んでいる方がいます。おそらく社会インフラだと思いますが。でも、こちらはデータが少ないので、これからのアプローチ先としては十分可能性がある分野だと思います。ただ、現状ではデータが少ないですし、たとえば衛星データだと公開データにはなかなかなりにくそうなところがあります。

――衛星データを使った被害状況予測、宮本先生がやっていたのは建物ですね。ほかには地滑りなどもあるかと思いますが、ほかに対象とするものは何でしょうか。

建物、地滑り、道路通行性の可否といったところでしょうか。あとは交通ネットワークでしょうか。

地滑りについては、衛星データだけではなく、たとえば GPS のデータなどを使うことができるので、おそらくオープンなデータもあるのではないかと思います。

――また、事前資料に「データを集めて（連携して）」と書いていますが、別の質問として、連携という観点では、何かシームレスな連携はできるのでしょうか。それとも、研究者がたとえば「K-NET、KiK-net からデータを集め、地盤データを集め、構造データを集め」というように、研究者の頑張りに依存するような形なのでしょうか。

現状では研究者の頑張りに依存しているような気がします。先ほどお話しした国際的なデータベースについては、おそらく連携をとっていると思います。しかしそれ以外では、現状で具体的にシームレスな連携がとられているというのは……。研究チームごとでのデータ共有はあっても、シームレスな連携がとられているかどうかは、私自身は把握しきれていません。

――防災の分野では、「SIP4D」などはプラットフォームかと見ています。被っているのですが、データが必要であろう問題に対して、データ収集時に不都合はありますか。たとえば先生が研究するにあたって、「こんなデータがあればいいのに、ない」あるいは「入手できない」というようなものはありますか。あるいは、ありませんか。

構造物の被害のデータは入手が難しいですね。土木構造物はそれほどではありませんが、建物構造の被害データはなかなか難しいと感じています。

――それは、データが取られていないからでしょうか。そもそもないからでしょうか。

民間というかオーナーの所有のデータですから、なかなかオープンにはなりにくいのです。

――なるほど。そういったプライバシーの問題は、ほかのさまざまな分野でもちらほら聞きます。構造物の被害のデータは民間だから入手が難しい……それはそうですね。ほかには何かありますでしょうか。

私は十分に把握していないのですが、この間の９月の土木学会全国大会において、宮本先生を中心に「AI・IoT 技術の地震工学への有効活用検討小委員会」が主催で開催していただいた研究討論会で、静岡が何か……。

――３次元点群を取っていますね。

はい、それです。あれはオープンデータベース化しているということですが……。

――していますね。

あれの場合は、そういうことにすごく熱心な方がいるからデータを取れているのですが、そうした個人的な努力がないと取れません。また、そこでの議論の中で伺いましたが、データベース化、オープンデータベースやデータプラットフォームを作っていこうと思うと、「誰がデータを取らなければいけないのか」「誰がデータを維持していくのか」「そのコストはどこが持つのか」ということを考える必要があり、かなり難しい面があるとのことです。また、今は静岡県が予算で取っているのですが、その予算をかけてデータを取ったことが、たとえば県民に対してどれだけのメリットがあるのかということもあります。逆に言えば、県がデータを取ってオープンデータベース化すれば、いろいろな人が使えますし、いろいろなところが使ったほうが当然メリットは大きくなりますが、そのために県民の税金でデータベースを整備することに対する疑問が上がってくるそうです。

――そうでしょうね。少し前に別件で静岡に行った時、たしか杉本さんに伺ったのですが、やる気のある地場の建設業者はすごく得をするらしいのです。そうすると、県民の税金を使って企業を儲けさせることになってしまいます。

そうなのです。でも、その業者がきちんと社会基盤整備に貢献してくれるということであれば、社会全体にもう一回還元されると思います。ですから、そういう社会システムができないと、地震防災に関するデータオープン化というのは難しい分野なのでしょう。それは地震だけではないでしょうが……。

そういった意味で、社会システムや、そこのあり方自身を変えないと、結局「コストなどは誰が負担するんだ」というところで止まってしまうおそれがあるかと思っています。

――そうですよね。また、データプラットフォームやオープンデータの重要性については、どのようにお考えになりますでしょうか。たとえば、データプラットフォームなどがあることによって、研究は進むものなのですか。あるいはオープンデータでも構いません。いろいろなデータがオープンであれば、それによって研究が進み、何か社会的に意義のあるような結果が出せそうなものなのでしょうか。

逆に言うと、オープンデータがないと当然、こういう分野の研究は進まないと思いますので、必要性というよりむしろ絶対必須です。必須だと思います。ないといけません。

研究しているだけであれば、各自が持っているデータの中でやって「こういう結論です」「こういう関係性がある」ということを言えればいいわけです。しかし、それを実際に実用化しようと思うと、結局、説明責任になってきてしまいます。そうなると途端にすごく行き詰まってしまうという共通の課題というか、ジレンマがあるように思います。