深層学習後編（day4）レポート

May 15, 2025

1　Section1: 強化学習

強化学習とは、長期的に報酬を最大化できるように環境のなかで行動を選択できるエージェントを作ることを目標とする機械学習の一分野である。行動の結果として与えられる利益（報酬）をもとに、行動を決定する原理を改善していく。強化学習の全体像を図 1 に示す。エージェントは、方策関数 Π をもとに、ある状態 s における行動 a を出力する。行動 a の結果、行動価値関数 Q（あるいは状態価値関数）によって報酬が与えらる。この報酬を最終的に最大化するような方策を学習することが、強化学習の目標となる。

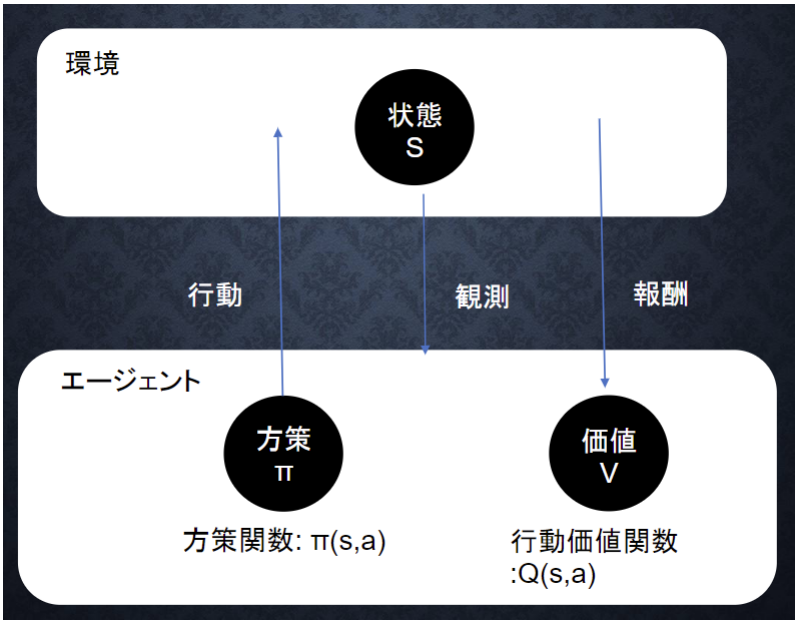

図 1: 強化学習の全体像

1.1 参考図書など関連記事レポート

強化学習のゴールが分からなかったので、その部分を中心に調査。

- [1]
  - 強化学習の目的は、この行動価値関数を最大化する戦略を得ること。目先の報酬ではなく、長期的報酬を最大化するように学習する。
- [2]
  - 方策関数には、一意な行動を出力する決定的方策と、行動の確率分布を出力する確率的方策がある。
  - 価値関数には、状態価値関数と行動価値関数がある。状態価値関数は、方策を Π で固定したとして、状態 s から将来もらえると期待できる割引報酬和を表す関数のこと。行動価値関数は、方策を Π で固定したとして、状態 s で行動 a をとった場合に、将来もらえると期待できる割引報酬和を表す関数のことで、Q 値ともよばれる。長期的報酬を最大化したいので、報酬和を出すことに注意。
  - 強化学習アルゴリズムには、価値ベース、方策ベース、モデルベースの 3 種類の分類が考えられる。価値ベースは、最適価値関数 Q^* を学習し、そこから適当な方策 (ε -グリーディー法や Boltzmann 選択など) にしたがって行動を選択する。方策ベース (方策反復) は、直接最適方策 Π^* を学習し、割引報酬和を最大化する。要は、価値関数・方策関数どちらを起点に学習するかの違い。一方で、モデルベースは、マルコフ決定過程に関するパラメータ（状態遷移確率など）が既知で、環境に関するモデルが構築できる場合に利用するが、多くの場合で環境のモデルを推定することは難しい。

References

[1] https://qiita.com/qiita_kuru/items/2c00a81b4b26bf9ad210

[2] https://qiita.com/shionhonda/items/ec05aade07b5bea78081

2 Seciton2: AlphaGo

DeepMind 社によって開発された、強化学習とディープラーニングを組み合わせた囲碁 AI。方策関数（どこに打つか）である PolicyNet と RollOutPolicy、価値関数（そこに打つと勝敗はどうなるのか）である ValueNet で構成される。なお、RollOutPolicy は探索中に高速に着手確率を出すために使用され、NN ではなく線形の方策関数。その他は NN である。また、打つ手の探索にモンテカルロ木探索法を使用し、基盤の状況認識として NN の一種である CNN を用いてる。AlphaGo の学習は以下のステップで行われる。

1. 教師あり学習による RollOutPolicy と PolicyNet の学習
   PolicyNet について、KGS Go Server（ネット囲碁対局サイト）の棋譜データから 3000 万局面分の教師を用意し、教師と同じ着手を予測できるように学習を行った。具体的には、教師が着手した手を 1 とし残りを 0 とした 19x19 次元の配列を教師都市、それを分類問題として学習した。

2. 強化学習による PolicyNet の学習
   現状の PolicyNet と PolicyPool（PolicyNet の強化学習の 500Iteration ごとのアーカイブ）からランダムに選択された PolicyNet と対局シミュレーションを行い、その結果を用いて方策勾配法で学習を行った。アーカイブとの対局を使用することで、対局に幅を持たせて過学習を防ごうという意図がある。

3. 強化学習による ValueNet の学習
   PolicyNet を使用して対局シミュレーションを行い、その結果の勝敗を教師として学習した。教師データの作成手順は以下のとおり。

   (a) 教師あり学習で作成した PolicyNet で N 手まで打つ。

   (b) $N+1$ 手目の手をランダムに選択し、その手で進めた局面を $S(N+1)$ とする。

   (c) $S(N+1)$ から強化学習で作成した PolicyNet で終局までうち、その勝敗報酬を R とする。

   $S(N+1)$ と R を教師データ対とし、損失関数を平均二乗誤差とし、回帰問題として学習した。

その後、教師あり学習を一切行わず強化学習のみで作成・特徴入力からヒューリスティックな要素を排除し石の配置のみ・ResidualNet（NN にショートカット構造を追加し、勾配爆発や消失を抑える）を導入した AlphaGo Zero が発表された。

2.1 参考図書など関連記事レポート

AlphaGo に用いられる、モンテカルロ木探索について調査。

- [1]
  - 強化学習の問題を解くアルゴリズムとして、モンテカルロ法がある。モンテカルロ法は、何らかの報酬が得られるまで行動をしてみて、その報酬値を知ってから、辿ってきた状態と行動に対してその報酬を分配していく方法。
  - モンテカルロ法で完全ランダムで手を選択した場合、相手の最善手を考慮できておらず、正しい報酬として計算できない場合がある。なら、その時点での平均報酬が高い手を選ぶ方法があるが、本当は勝てる展開でたまたま負けた場合、その手が選ばれなくなってしまう。平均報酬が高い手を選びたいが、平均報酬がまだうまく見積もれていない手は、ある程度見積もれる段階までは選びたいモチベーションがある。

- 上記のトレードオフに対して、理論的にある上界で無駄を抑えられるアルゴリズムとして、UCB（Upper Confidence Bound）というアルゴリズムが知られており、ゲーム木探索に UCB を取り入れた手法として UCT がある。UCT やそれと同様の枠組みの手法が、モンテカルロ木探索アルゴリズムと呼ばれている。
- モンテカルロ木探索は 5 つの要素で構成される。
  - Selection
    初期状態からどの手を選ぶか、モンテカルロ木探索で選択する手順を示す。
  - Expansion
    ある時点でゲーム木の葉ノードに到達したとき、その葉ノードを訪れた回数が閾値以上となった場合に、ノードから先を展開する。
  - Evaluation
    ゲーム木の葉ノードから得られる報酬を評価する。モンテカルロ木探索では、Rollout と呼ばれる方法で評価を行うことが一般的だが、AlphaGo では別の方法で評価を行う。Rollout では、ゲーム木の葉ノード以降は、決着がつくまでランダムに手を進め、その勝敗をそのまま報酬とする。
  - Backup
    得られた報酬の情報を記録する。今回のシミュレーションで訪れたすべてのノードについて、Rollout で得られた報酬を記録し、期待報酬を更新する。
  - Play
    Selection,Expansion,Evaluation,Backup の流れを何度も繰り返すことで、Q 値を更新し、Q 値の推定精度を高める。事前に決めていた試行回数だけ繰り返したら、実際に着手する手を選択する。AlphaGo では、選択数が最も多い行動を実際の着手として選択する。

References

[1] https://www.brainpad.co.jp/doors/contents/01_tech_2018-04-05-163000/

3 Seciton3: 軽量化・高速化技術

深層学習は、多くのデータを使用したり、パラメータ調整のために多くの時間を使用する。特に、最近の深層学習は層が深いなどの理由で調整すべきパラメータの数が膨大となっており、より顕著である。そこで、学習を軽量化・高速化する技術がいくつか提案されている。

- データ並列化
  親モデルを各ワーカーに子モデルとしてコピーする。データを分割し、各

ワーカーごとに計算させる。各モデルのパラメータの合わせ方で同期型か非同期型か決まる。

  - 同期型
    各ワーカーが計算が終わるのを待ち、全ワーカーの勾配が出たところで勾配の平均を計算し、親モデルのパラメータを更新する。
  - 非同期型
    各ワーカーはお互いの計算を待たず、各子モデルごとに更新を行う。学習が終わった子モデルはパラメータサーバに Push される。新たに学習を始める時は、パラメータサーバから Pop したモデルに対して学習をする。

  処理のスピードは、非同期型の方が早い。一方で、パラメータサーバから Pop したモデルで学習を始めるため、最新のモデルのパラメータを利用できず学習が不安定になりやすい。

- モデル並列化
  親モデルを各ワーカーに分割し、それぞれのモデルを学習させる。全てのデータで学習が終わった後で、一つのモデルに復元させる。途中分岐するようなモデルである際は、各分岐ごとのモデルで並列化するのが一般的。基のモデルのパラメータ数が多いほど、スピードアップの効率も上昇する。

- GPU による高速化
  高性能なコアである一方で少数しかない CPU ではなく、低性能であるが多数ある GPU を用いて演算処理を高速化する方法。

- モデルの軽量化
  モデルの制度を維持しつつ、パラメータや演算回数を低減する手法の総称。モバイル、IoT に組み込む際に有用である。代表的な手法として下記の 3 つある。

  - 量子化
    通常のパラメータの 64bit 浮動小数点を 32bit など下位の精度に落とすことでメモリと演算処理の削減を行う手法。ただし、精度が低下するという欠点はある。近年では、大きな精度の差が見られないことから 16bit が使われることが多い。
  - 蒸留（Distillation）
    規模の大きなモデルの知識を使い、軽量なモデルの作成を行う方法。教師モデル（規模が大きく、予測精度の高いモデル）と生徒モデル（教師モデルをもとに作られる軽量なモデル）の 2 つで構成される。教師モデルの重みを固定した状態で、教師モデルと生徒モデルそれぞれの誤差を使って重みを更新する。これにより、生徒モデルに教師モデルのパラメータ情報を継承させることができる。
  - プルーニング
    モデルの精度への寄与が少ないニューロンを削除する方法。ニューロンの削減の手法として、重みが閾値以下の場合にニューロンを削除し、再学習する方法がある。

3.1 参考図書など関連記事レポート

python の tensorflow を用いた分散学習について調査。

- [1]
  - MirroredStrategy を用いて GPU を用いることができる。下記のソースコード 1 では、*n_gpus* 分だけ GPU をセットし、MirroredStrategy でその GPU 分だけ使うようにしている。

ソースコード 1: build_strategy

```
1        def build_strategy(n_gpus):
2            tf.config.set_visible_devices(gpus[:n_gpus],
                 'GPU')
3            logical_gpus = tf.config.
                 list_logical_devices('GPU')
4            strategy = tf.distribute.MirroredStrategy(
                 logical_gpus)
5        return strategy
```

  - strategy.scope() 内でモデルやオプティマイザを構築することで、これらを各 GPU にコピーできる。

ソースコード 2: run

```
1    def run(n_gpus, epochs, batch_size, learning_rate):
2        strategy = build_strategy(n_gpus)
3        ...
4        with strategy.scope():
5            model = build_network(IMAGE_SIZE, N_CLASSES)
6            optimizer = tf.keras.optimizers.Adam(
                 learning_rate)
7            loss = tf.keras.losses.
                 SparseCategoricalCrossentropy(
                 from_logits=True)
8            metrics = tf.keras.metrics.
                 SparseCategoricalAccuracy()
9        model.compile(optimizer=optimizer,loss=loss,
             metrics=metrics)
10       model.fit(...)
```

References

[1] https://www.tensorflow.org/guide/distributed_training?hl=ja#examples_and_tutorials

4 Section4: 応用技術

- MobileNet
  深層学習モデルの軽量化・高速化・高精度化を実現したネットワーク。一般

的な畳み込みレイヤーでは、入力特徴マップを $H \times W \times C$、カーネルサイズを $K \times K \times C$、出力チャネル数を M とすると、出力マップの計算量は $H \times W \times K \times K \times C \times M$ となり、計算量が多くなる。そこで、Depthwise Convolution と Pointwise Convolution を組み合わせて軽量化を実現している。

  - Depthwise Convolution
    入力マップのチャネルごとに畳み込みを実施する方法。具体的には、フィルタ数を 1 として計算する。よって、出力マップの計算量は $H \times W \times C \times K \times K$ となる。各層ごとの畳み込みなので層間の関係性は考慮されていない。そのため、後述する Pointwise Convolution とセットで使うことが前提。
  - Pointwise Convolution
    入力マップのポイントごとに畳み込みを実施する方法。具体的には、カーネルサイズを $1 \times 1 \times C$ として計算する。よって、出力マップの計算量は $H \times W \times C \times M$ となる。

  通常の畳み込みでは空間方向とチャネル方向の計算を同時に行うのに対し、それらを個別に分けて計算を行っている。

- DenseNet
  CNN の一種である。ニューラルネットワークでは層が深くなるにつれて学習が難しくなる問題があるが、DenseNet は DenseBlock と呼ばれるモジュールを用いて解決している。DenseBlock には下記の特徴がある。

  - 出力層に前の入力を足し合わせる。層間の情報の伝達を最大にするため全ての同特徴量サイズの層を結合する。
  - 特徴マップの入力に対し、Batch 正規化、Relu 関数による変換、3×3 畳み込み層による処理を行う。

  ResNet で用いられる RessidualBlock では前 1 層の入力のみ後方の層へ入力するが、DenseBlock は前方の各層からの出力全てが後方の層への入力として用いられる点が異なる。

- Wavenet
  音の音声波形を生成する深層学習モデル。Pixel CNN（高解像度の画像を精密に生成できる手法）を音声に応用したものである。時系列データに対して畳み込み（Dilated convolution）を適用している。Dilated convolution では図 2 のように、既存の畳み込みと異なり、層が深くなるにつれて畳み込むリンクを離している。これにより、受容野を簡単に増やすことができる。

4.1 確認テスト

4.1.1 Depthwise Convolition はチャネル毎に空間方向へ畳み込む。すなわち、チャネル毎に $D_K \times D_K \times 1$ のサイズのフィルターをそれぞれ用いて計算を行うため、その計算量は（い）となる。

$H \times W \times C \times D_K \times D_K$

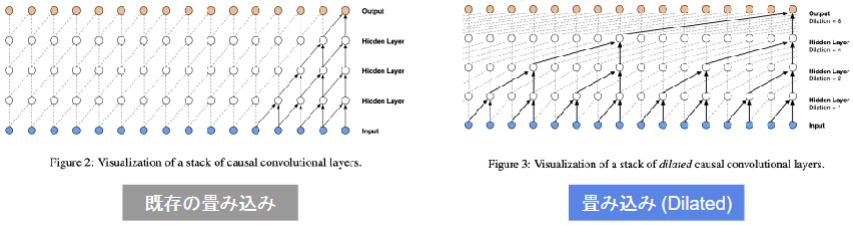

図 2: 既存の畳み込みと Dilated convolution

4.1.2 次に Depthwise Convolution の出力を Pointwise Convolution によってチャネル方向に畳み込む。すなわち、出力チャネル毎に $1 \times 1 \times M$ サイズのフィルターをそれぞれ用いて計算を行うため、その計算量は（う）となる。

$H \times W \times C \times M$

4.1.3 深層学習を用いて結合確率を学習する際に、効率的に学習が行えるアーキテクチャを提案したことが WaveNet の大きな貢献の 1 つである。提案された新しい Convolution 型アーキテクチャは（あ）と呼ばれ、結合確率を効率的に学習できるようになっている。

Dilated causal convolution

4.1.4 Dilated causal convolution を用いた際の大きな利点は、単純な Convolution layer と比べて（い）ことである。

パラメータ数に対する受容野が広い

4.2 参考図書など関連記事レポート

MobileNet の改良として、MobileNet-v2,v3 があるため、それらについて調査。

- [1]
  - MobileNet-v2
    Pointwise convolution の計算量が大きいため、これを減らすために、MobileNet の畳み込み演算に代わって Inverted Residual を導入している。Inverted Residual は、ResNet で使われる Residual Block を応用したものであり、Depthwise convolution を小さな 1×1 の convolution で挟むことで、Pointwise convolution を小さな計算量で近似している。
  - MobileNet-v3
    図 3[2] の Figure 4 のように、Inverted Residual Block 内に 3×3convolution による特徴量抽出と、Squeeze-and-Excitation モジュールを導入している。Squeeze-and-Excitation モジュールの全体像を図 4 に示す。左の入力データからチャネルごとの重みを全結合層で算出して、それを

掛け合わせたデータを新しい入力として用いている。
活性化関数として、式 1 で表される h-swish が ReLU の代わりに使われていることも特徴。

$$h-swish(x) = x\frac{ReLU6(x+3)}{6} \tag{1}$$

$$ReLU6(x) = \min(\max(x,0),6) \tag{2}$$

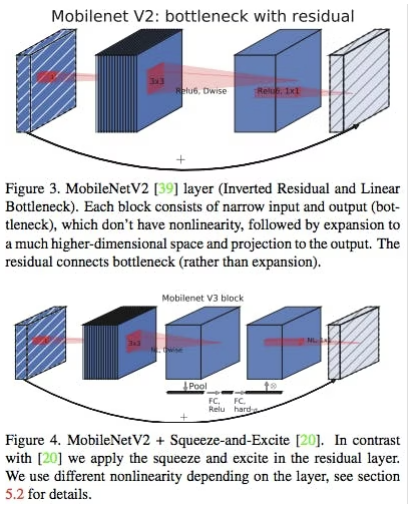

Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).

Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

図 3: MobileNet-v3 における Inverted Residual Block の変更

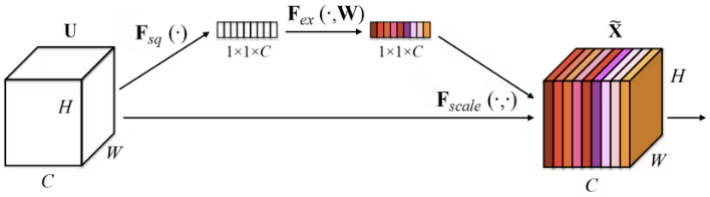

図 4: Squeeze-and-Excitation モジュールの全体像

References

[1] `https://qiita.com/omiita/items/77dadd5a7b16a104df83`

[2] Howard, Andrew, et al. "Searching for mobilenetv3." arXiv preprint arXiv:1905.02244 (2019).

5　ResNet（転移学習）

ResNet は事前学習のモデルとして良く用いられている深層学習モデルであり、派生形も数多く存在する（DenseNet など）。ResNet の構造の全体像を図 5 に示す。ResNet は巨大な構造であるが、SkipConnection により、深い層の積み重ねでも勾配消失や勾配爆発の回避に成功している。また、Bottleneck アーキテクチャと呼ばれる、3×3 の畳み込みを 1×1 の畳み込み層で挟むことで、同一計算コストで 1 層多い構造を作っている。
ResNet の派生形である Wide ResNet は、ResNet のフィルタ数を k 倍（高速・高精度の学習を可能にした）にし、パラメータを増やす方法として、層を深くするのではなく、各層を広くしている。

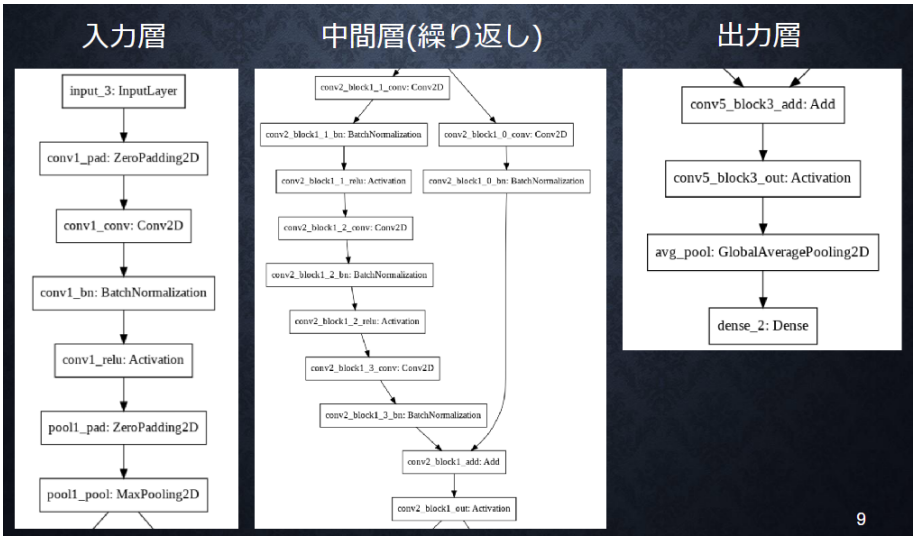

図 5: ResNet の構造の全体像

5.1　実装演習と自身の考察

5.1.1　4_1_transfer-learning.ipynb

ResNet による事前学習を利用して、TFFlowers データセットの学習を行っている。tensorflow において事前学習を利用するには、図 6 のように weights パラメータに事前学習データセットを指定すればよい。また、ファインチューニングを行う場合は、trainable プロパティを True に設定する。図 7 に、事前学習なし、ファインチューニングなし事前学習、ファインチューニング事前学習それぞれの学習用と検証用の正答率を示す。 全体的に、ファインチューニングなし事前学習、事前学習なし、ファインチューニング事前学習の順に正答率が高くなっている。ファインチューニングなし事前学習が最も悪くなったのは、事前学習には ImageNet を用いており、本来のタスクである TFFlowers とはデータの分布が大きく異なったか

らであると考えられる。一方で、ファインチューニングを行うことで、TFFlowers に適した学習が行われ、精度が大きく向上している。

```
resnet = tf.keras.applications.resnet.ResNet50(weights='imagenet')
resnet.trainable = True
```

図 6: tensorflow における事前学習とファインチューニングの利用

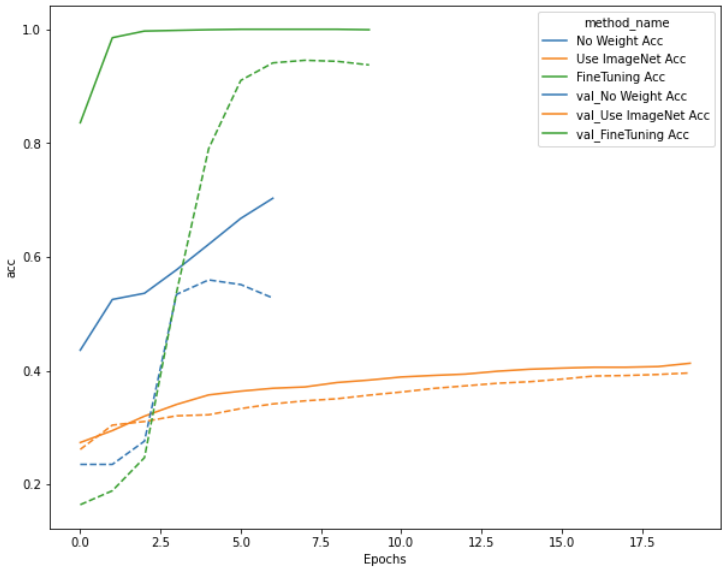

図 7: 事前学習なしとありによる正答率

6 EfficientNet

効率的なスケールアップの規則を採用することで、開発当時の最高水準の精度を上回りつつ、パラメータ数を大幅に減少させた深層学習モデル。幅、深さ、解像度などを何倍増やすかは、複合係数（Compound Coefficient）を導入することで最適化を行っている。シンプルかつ簡潔な構造であるため、転移学習でも性能を発揮している。
複合係数を導入した Compound Scaling Method（複合スケーリング手法）の詳細は以下の通り。

- 幅（d）、深さ（w）、解像度（r）はある程度まで増やすと精度の向上は横ばいになる。また、演算量は d, w^2, r^2 に比例している。そこでモチベーションとして、以下の式のように演算量を $\sim 2^\phi$ で近似できるように制約をかけ

ている。

$$\begin{aligned} \textbf{depth:} \quad & d = \alpha^{\phi} \\ \textbf{width:} \quad & b = \beta^{\phi} \\ \textbf{resolution:} \quad & \gamma^{\phi} \\ \textbf{st.} \quad & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2, \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

- 上記の制約を満たしつつ、モデルの精度を最大化する最適化問題として定式化し、学習を行っている。

7 物体検知と SS 解説

物体検知には、Classification, Object Detection, Semantic Segmentation, Instance Segmentation があり、この順で難易度が高くなっていく。Object Detection 以降（物体検出）は、一枚の画像にいくつ物体があるかもデータセットとして重要となってくる。
物体検出においては、以下の評価指標が使われる。

- IoU: Intersection over Unit
  正解と予測の bouding box の和集合のうち、どれだけ正解を包含しているか。クラスラベルがあってるかどうかだけでなく、検出精度も評価したいため。

- mAP: mean Average Precision
  クラスラベルごとの PR 曲線の下側面積を平均したもの。1 枚の画像に複数のクラスがあると考えられるため。

- FPS: Flames per Second
  検出時の秒単位のフレーム数。検出速度も評価したいため。

物体検出のフレームワークとして、以下の 2 つがある。

- 2 段階検出器（Two-stage detector）
  候補領域の検出とクラス推定を別々に行う。相対的に精度が高い傾向である一方、計算量が大きく推論も遅い。

- 1 段階検出器（One-stage detector）
  候補領域の検出とクラス推定を同時に行う。相対的に精度が低い傾向である一方、計算量が小さく推論も早い。

代表的な物体検出モデルとして、SSD,FCN,U-Net がある。

- SSD
  - VGG16 を利用。
  - 特徴マップ中の一つの特徴量に対して複数 Default Box を配置し、それらをオフセット項によってサイズを変更する。よって、クラスラベルとオフセット項を学習することになる。

  - 多数の Default Box を用意したことによる問題を対処している。同じクラスを検知した Box が複数ある場合は、IoU が高い Box を選ぶ Non-Maximum Suppression。検知対象の物体以外を検知した Box が多くなるため、ある程度の割合まで減らす Hard Negative Mining。

- FCN
  - Semantic Segmentation を行うモデル。CNN を利用。全ての層が畳み込み層であり、全結合層がない。
  - Semantic Segmentation をするため、pooling されて解像度が低くなった特徴マップを元のサイズまで戻す Deconvolution/Transposed convolution が使われている。

- U-Net
  - Semantic Segmentation を行うモデル。Encoder,Decorder があり、モデルの形が U に似ていることから U-Net と名付けれている。
  - 畳み込みの時点で受容野を広げる工夫として、Dilated Convolution を採用している。

7.1 参考図書など関連記事レポート

講義動画では紹介されてなかったスライド内の Exercise を解いてみた。

7.1.1 物体検出タスクには精度評価をはじめ大規模なデータセットの存在が欠かせない。以下の選択肢のうち、各データセットに関する説明として最も妥当な主張と考えられるものを１つ選べ。

1. VOC12 では Ground-Truth BB の情報が中心座標と幅, 高さで与えられていない。
2. フリマアプリの出品画像を入力とする物体検知タスクに取り組みたい。訓練データとして COCO18 を用いることで精度の向上が期待される。
3. ILSVRC17 は ImageNet のサブセットで構成され、Instance Segmentation の学習に必要な情報も与えられている。
4. OICID18 はクラス数が 500 と比較的大きなデータセットであるが、物体検出タスクにおいては常にクラス数の大きいデータセットで学習を行う方が好ましい。

正解は 1。左上隅と右下隅の座標が与えられている。
2 は物体の位置を正確に検知する必要のないタスクであるため、COCO18 を使わない方が良い。
ILSVRC17 は Instance Segmentation の学習に必要な情報は与えられていない。
物体検出タスクにおいては、常にクラス数の大きいデータセットで学習を行う方が好ましいとは限らない。1 枚の画像中にいくつ物体があるかも指標となる。

8 Mask R-CNN

物体検出+物体認識のアルゴリズムの原型として R-CNN（Regional CNN）がある。R-CNN の全体像を図 8 に示す。R-CNN では、関心領域を切り出し候補領域の画像の大きさを揃える物体検出タスクを行った後、CNN により求めた特徴量から SVM で分類を行う物体認識タスクを行う。R-CNN の発表後、関心領域ごとに畳み込むのではなく、画像ごとに 1 回の畳み込みをすることで計算量を削減した Fast R-CNN が考案された。さらにその後、関心領域の切り出しも CNN で行う Faster R-CNN, YOLO, SSD が考案された。
Mask R-CNN は、Faster R-CNN を拡張したアルゴリズムである。特徴は以下の通り。

- インスタンスセグメンテーションに対応している。Faster R-CNN の物体検出機能にセグメンテーションの機能を付加したイメージ。
- bounding box 内の画素単位でクラス分類を行うため、物体の形も推定可能。
- 物体検出の結果として得られた領域についてのみセグメンテーションを行うことで効率アップしている。
- Fast/Faster R-CNN で使われていた RoI Pooling に代わり、RoI Align を導入。これにより、補間処理によってより多くのピクセルの情報を使い、推定精度を上げている。

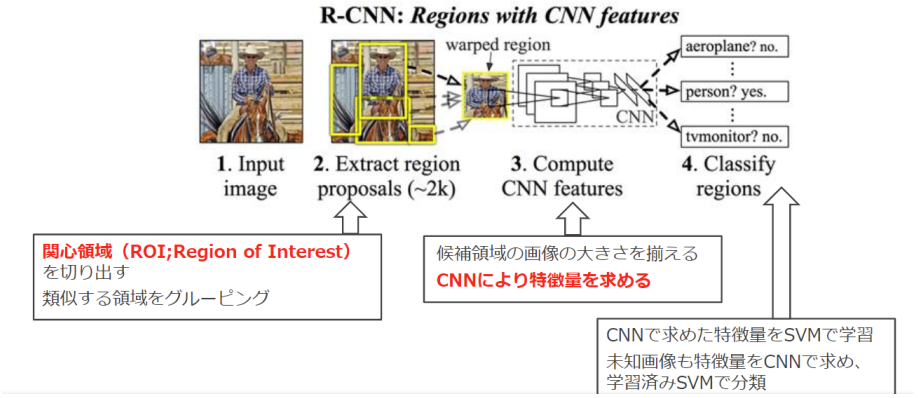

図 8: R-CNN の全体像

9 FCOS

多くの物体検出モデルは、アンカーボックス（大量のバウンディングボックスを生成し、それぞれのボックスが検知したい物体を含むか推定する方法）を使用している。しかし、以下のデメリットがある。

- ハイパーパラメータの設定に敏感である。
- アンカーボックスのサイズやアスペクト比が固定されている。
- ポジティブサンプルとネガティブサンプルのバランスが崩れる。

FCOS では、アンカーボックスを使わないアンカーフリーかつ、one-stage の手法となっている。特徴は以下の通り。

- YOLO と異なり、画像の中央付近の点からのみでなく、全てのピクセルから四次元ベクトルを予測する。
- FPN（Feature Pyramid Networks）を使い、複数のサイズの特徴マップを生成している。つまり、高解像度と低解像度の画像の良い所を両立させている。
- Non-Maximam Suppression で、IoU が閾値より低い box を削除している。具体的には、一番スコアが高いバウンディングボックスを出力とし、他の物の IoU を計算する。そして、IoU が閾値より低い場合削除している。

10 Transformer

Transformar は、論文『Attention is all you need』（Vaswani et al., 2017）で発表された、RNN や CNN を使わない、Attention を使用した Encoder-Decoder モデルである。これにより、当時のモデルより少ない計算量で学習を完了し、かつ、文長が長くなっても精度を落とさないことを可能とした。Transformer の全体像を図 9 に示す。特徴としては以下の通り。

- Source Target Attention と Self-Attention の 2 つの Attention を組み合わせている。
  - Self-Attention は、Attention の input と memory を同じ文章から抽出する。これにより、文章の構造を学習する。
  - Source Target Attention は、Attention の input と memory を違う文章から抽出する。これにより、新しい文章に対して予測を行う。
- Position-Wise Feed-Forward Networks で、位置情報を保持したまま順伝播させる。
- Scaled dot product attention で、全単語に関する Attention をまとめて計算する。
- Muti-Head attention で、重みパラメタの異なる 8 個の Attention の出力を組み合わせ、新たな出力とする。
- Position Encoding で、単語列の語順情報を追加する。input の末尾に位置情報を表す 2 進数ベクトルを付与。
- Masked 処理で、Decoder 時に未来の情報を Mask（未来の情報を知ったうえで学習が行われるのを防ぐため）。

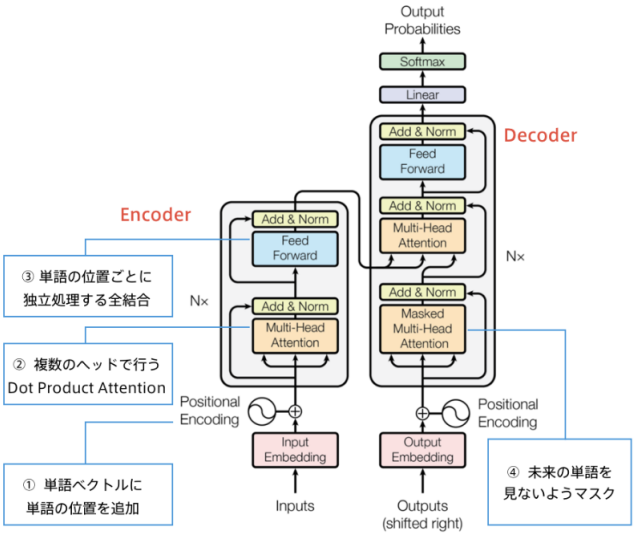

図 9: Transformer の全体像

10.1 参考図書など関連記事レポート

- [1]
  - Attention は、query と key から value を出力する辞書オブジェクトといえる。query は、関連度を検索したい言葉。key は、memory 内の言葉と query がどの程度近いのかを測るための索引。value は、key の索引に対応する値。
  - query と key から、attention_weight とと呼ばれる関連度を計算する。この重みが大きいと、query 中の言葉と key 中の言葉の関連度が高い、ということになる。
  - decorder で mask 処理を行うのは、self-attention で自分自身の文章を学習する際に、先の答えが見えてしまうのを防ぐため。decoder はアウトプットを予測するフェーズのため、答えが分かっている状態での学習は防ぐ。

References

[1] https://www.acceluniverse.com/blog/developers/2019/08/attention.html

11　BERT

BERT は、Google が開発した自然言語処理モデル。汎用性が高いことから転移学習のモデルとして採用されることが多く、Google の検索エンジンなどで利用されている。
双方向 Transformer エンコーダに基づいたモデルであり、Transformer では過去の情報から現在の情報を予測するのに対し、BERT は未来の情報も使って予測することで、より人間の言語理解能力に近い精度で自然言語を処理することができる。具体的には、空欄語予測と隣接文予測を行う。空欄語予測では、ランダムに抜かれた単語を前後の単語から予測する。隣接文予測では、2 つの連なる文章ペアに対して隣接文を 50%の確率でシャッフルし、 2 つの文章を入力として隣接するか否かを予測する。

11.1　実装演習と自身の考察

11.1.1　4_6_bert.ipynb

BERT の学習済みモデルを利用して転移学習をしている。その流れを図 10 12 に示す。なお、この前に、入力データ（夏目漱石書籍の文章データ）の収集を行っている。
まず図 10 より、tokenizer（形態素解析で文章を最小単位に分割し、その形態素ごとに ID を付与することで文章を ID に変換する機能）と BERT を導入している。
次に図 11 より、mecab.parse などを使用して文章を要素ごとに形態素解析している。なお、char2idx が文字から ID に変換する辞書、idx2char が ID から文字に変換する辞書である。
次に図 12 より、文章を 128 文字ごとに分割している。その後、文章生成モデルを構成している。
最後に、「楽しい勉強でした」の分類結果（夏目漱石の「それから」「こころ」「夢十夜」それぞれの文章との類似度）を図 13 に示す。実行環境の制約上、学習回数は 5 回であるが、[0.16646542, 0.32961306, 0.50392145] であることから「夢十夜」と最も類似度が高いといえる。

```
[7]  from transformers import TFBertModel
     from transformers import BertJapaneseTokenizer


     tokenizer = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-japanese-whole-word-masking')

     bert = TFBertModel.from_pretrained('cl-tohoku/bert-base-japanese-whole-word-masking')
```

図 10: BERT を利用した転移学習の流れ 1

12　GPT

GPT は、2019 年に OpenAI が開発した事前学習モデルである。GPT の構造は Transformer を基本とし、「ある単語の次に来る単語」を予測し、自動的に文章を完成できるように教師無し学習を行う。よって出力値は「その単語が次に来る確

```
[8]  import MeCab
     import numpy as np
     import tensorflow as tf
     import os
```

```
[9]  with open('train.txt', 'r', encoding='utf-8') as f:
       text = f.read().replace('¥n', '')
     mecab = MeCab.Tagger("-Owakati")
     text = mecab.parse(text).split()
     vocab = sorted(set(text))
     char2idx = {u: i for i, u in enumerate(vocab)}
     idx2char = np.array(vocab)
     text_as_int = np.array([char2idx[c] for c in text])
```

図 11: BERT を利用した転移学習の流れ 2

```
[10] seq_length = 128

     # 訓練用サンプルとターゲットを作る
     char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
     sequences = char_dataset.batch(seq_length+1, drop_remainder=True)
     def split_input_target(chunk):
         input_text = chunk[:-1]
         target_text = chunk[1:]
         return input_text, target_text

     dataset = sequences.map(split_input_target)
```

図 12: BERT を利用した転移学習の流れ 3

```
text = '楽しい勉強でした。'


encoded = tokenizer.encode_plus(
            text,
            text,
            add_special_tokens=True,
            max_length=128,
            pad_to_max_length=True,
            return_attention_mask=True
        )
inputs = tf.expand_dims(encoded["input_ids"],0)
res = model.predict_on_batch(inputs)
res
```

```
array([[0.16646542, 0.32961306, 0.50392145]], dtype=float32)
```

図 13: 「楽しい勉強でした」の分類結果

率」である。
後継である GPT-2 では、Layer Norm の位置を前にずらし、最後の self-attention ブロックの後にも Layer Norm 層を入れている。GPT-3 では、これまで事前学習後に fine-tuning していたのを行わず、新しいタスクに対しても勾配を更新し直さない。
BEAT の違いとして、GPT は単一方向の Transformer であり、常に次の単語を予測することが挙げられる。

12.1 参考図書など関連記事レポート

近年、GPT の後継モデルとして、GPT-3.5, 4, 4o が発表されたため、それらについて調べてみた。

- GPT-3.5 [1]
  パラメータ数は 3550 億個になり、GPT-3 の約 2 倍になった。それ以外の大きな変更点はないが、ChatGPT と呼ばれる Chat 形式のインタフェースで使われたモデルであるため、多くの人に認知されたモデルとなっている。

- GPT-4, GPT-4 Turbo [1], [2]
  安全性のリスクから、詳細な設計は非公開となっている。精度が劇的に上がっており、司法試験や大学入試テストなどの数々の試験で合格レベルの点数を叩き出している。GPT-4 Turbo は GPT-4 よりもさらに高精度にしたモデルかつ、1 トークン当たりの価格も GPT-4 より半額となっている。

- GPT-4o [2]
  GPT-4 Turbo が主にテキストベースの応答に特化していたのに対し、GPT-4o では音声、画像、テキストをリアルタイムで処理できる、マルチモーダル AI となっている。

References

[1] https://toukei-lab.com/gpt

[2] https://www.multifverse.com/blog-posts/gpt4-gpt4turbo-gpt4o-comparison

[3] https://kotanigawakenji.com/generative-ai-marketing/difference-between-gpt-4o-and-gpt-4-turbo-thorough-comparison-of-the-latest-models#AIGPT-4oGPT-4_Turbo

13 音声認識

- 周波数
  1 秒あたりの振動数 (周期数)

- 角周波数
  周波数を回転する角度で表現。例えば、270° は $3/2\pi$。

- 標本化
  連続時間信号（アナログ）を一定間隔に抽出することで離散時間信号（デジタル）に変換
- 量子化
  標本化で抜き出したデータを基データの振幅に合わせることで数値に変換
- フーリエ変換
  周期的・非周期的波形を正弦波や余弦波の集合に変換すること。これにより、基の波形にどんな周波数成分が、どんな大きさで含まれているかが分かる。
- 窓関数
  ある区間の信号を取り出すために、もとの信号列からある区間を取り出す際にかけ合わせる関数。
- メル尺度
  人は周波数の低い音に対して敏感で、周波数の高い音に対して鈍感であることに基づいた尺度
- 逆フーリエ変換
  振幅・周波数から元の波形を構築する変換
- ケプストラム
  フーリエ変換したものの絶対値の対数を逆フーリエ変換して得られるもの。音声認識の特徴料として利用される。

14 CTC

CTC（Connectionist Temporal Classification）とは、音声認識モデルの 1 つ。音響モデル、発音辞書、言語モデルを 1 つの深層学習モデルで学習する End-toEnd モデル（E2E モデ）となっている。特徴は以下の通り。

- ブランクと呼ばれるラベルを導入し、同一ラベルが連続するようなテキスト系列も表現可能・モデルが無理なアライメント推定を行うことを防いでいる。
- 前向き・後ろ向きアルゴリズム (forward-backward algorithm) を用いた DNN の学習を行う。

15 DCGAN

DCGAN とは、GAN を利用した画像生成モデルである。
GAN とは、生成器と識別器を競わせて学習する生成＆識別モデルである。Generator と Discriminator から構成され、前者は乱数からデータを生成し、後者は入力データが真データ（学習データ）であるかを識別する。Generator は、Discriminator に自分が作った偽データを誤識別させるように生成するデータを学習

し、Discriminator は、偽データが入ったとしても正しく識別できるように学習する。このことから 2 プレイヤーのミニマックスゲームと捉えられ、GAN の価値関数は、Generator ができるだけ Discriminator に対して誤識別させる条件の元で、Discriminator が正しく識別できるように確率を最大化する、バイナリークロスエントロピーとなっている。
DCGAN では、いくつかの構造制約により生成品質を向上している。Generator では、Pooling 層の代わりに転置畳み込み層を使用し、最終層は tanh、その他は ReLU 関数を活性化関数として用いている。一方、Discriminator では、Pooling 層の代わりに畳み込み層を使用し、Leaky ReLU 関数を活性化関数として用いている。共通事項として、中間層に全結合層を使わない・バッチノーマライゼーションを適用している。

16 Conditional GAN

Conditional GAN の特徴は以下の通り。

- Generator と Discriminator にいくつかの追加情報を与え、あるラベル（クラス）のもとで画像を生成＆識別できる。
  - Generator では、あるラベルのもとで画像を生成する。
  - Discriminator では、あるラベルかつ、その画像が真データであることを識別できるように学習する。

17 Pix2Pix

CGAN では条件パラメータとしてラベルが与えられるが、Pix2Pix では画像を用いている。つまり、ある画像のもとで何らかの変換を施した画像を出力する。これにより、画像の生成方法を学習している。
Generator は条件画像をもとに画像を生成し、Discriminator は Generator の画像変換方法と真の変換方法が正しい変換かどうかを識別する。Generator には物体の位置を抽出するために U-Net が用いられている。一方、Discriminator には Ganerator が生成した画像がぼやけることを防ぐために損失関数に L1 正則化項を追加している。共通事項として、条件画像をパッチに分けて、各パッチに Pix2Pix を適応している。これにより、L1 正則化項の効果を向上している。

18 A3C

A3C（Asynchronous Advantage Actor-Critic）とは、DeepMind の Volodymyr Mnih（ムニ）のチームが提案した強化学習の学習法の一つである。A3C の全体像を図 14 に示す。複数のエージェント（Worker）が並列に実行され、勾配計算を行う。その後、学習が終わった段階で勾配情報を用いて非同期に共有ネットワークを更新する。また、各エージェントは定期的に自分のネットワークの重みを共有ネットワークの重みと同期する。A3C のメリットとして、学習が高速化、安定化されることが挙げられる。

同期処理を行うモデルとして A2C が発表された。A2C では、各エージェントは中央司令部から行動の指示を受けて、一斉にネットワークの学習を開始し、同期的に共有ネットワークの勾配情報を更新する。

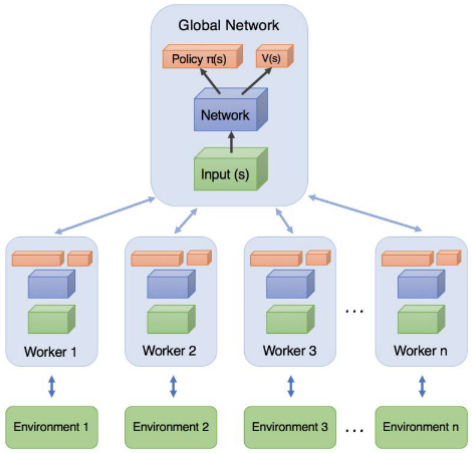

図 14: A3C の全体像

19 Metric-learning（距離学習）

距離学習では、「データ間の距離」を学習する。データ間の距離を適切に測ることができれば、距離が近いデータ同士をまとめてクラスタリングしたり、距離が遠いデータを異常として検知したりなど、様々な応用が可能となるためである。特に、深層学習技術を用いた距離学習の手法は、深層き距離学習と呼ばれる。代表的な手法としては以下の通り。

- Siamese network（シャムネットワーク）
  2 つのサンプルをペアで 2 つの CNN に入力し、それらのサンプル間の距離を計算する。2 つの CNN は重みが共有され、それらの出力を距離を計算するネットワークに入力している。

- Triplet network（トリプレットネットワーク）
  Siamese network と異なり、3 つのサンプルをそれぞれ CNN に入力する。Siamese network の欠点であった、同じクラスのペアと異なるクラスのペアの間に生じる不均衡と、学習時のコンテキストに関する制約を緩和している。

- Quadrupt loss
  4 つのサンプルを入力とするモデル。4 つのデータを 1 組として学習することで、任意のクラス間距離が任意のクラス内距離よりも必ず大きくなる。

20　MAML（メタ学習）

MAML は、深層学習モデルの開発に必要なデータ量を削減するモチベーションで開発された。MAML では、転移学習・ファインチューニングと異なり、複数のタスクで学習し、タスクに共通する重みを学習している。新しいタスクを学習する際は、共通重みからファインチューニングを行い、学習を行う。欠点として、タスクごとの勾配計算と共通パラメータの勾配計算の 2 回計算が必要であり、計算量が多いことが挙げられる。

21　グラフ畳み込み（GCN）

GCN とは、ノードとエッジで構成されたグラフを入力にできる CNN のことである。フィルタの畳み込みをグラフ上で行うことで、グラフやノードの特徴を得ることができる。畳み込みの方法には、グラフフーリエ変換を用いる方法などが挙げられる。

22　Grad-CAM,LIME,SHAP

ディープラーニングはブラックボックスであるため、判断の根拠を説明できないという課題がある。そこで、モデルの解釈性に注目し、ブラックボックス性の解消を目指した手法がいくつか提案されている。代表的な手法は以下。

- CAM
  過学習を防ぐ正則化の役割として使われた GAP を、CNN が潜在的に注目している部分を可視化する役割として用いた手法。ネットワークの大部分が畳み込み層で構成され、最終的な出力の前に GAP を実行すれば CAM を使える。出力層の重みを畳み込み、特徴マップに投影することで、画像領域の重要性を識別できるようにしている。

- Grad-CAM
  最後の畳み込み層の予測クラスの出力値に対する勾配を使用した CAM。勾配が大きい＝予測クラスの出力に大きく影響する重要なピクセルに重みを増やしている。CAM はモデル構造内に GAP がないと可視化できなかったが、Grad-CAM は GAP がなくても可視化できる。また、出力層が画像分類でなくてもよく、様々なタスクで使える。

- LIME
  特定の入力データに対する予測について、その判断根拠を解釈・可視化するツールである。単純で解釈しやすいモデルを用いて複雑なモデルを近似することで解釈を行っている。LIME への入力は 1 つの個別の予測結果であり、対象サンプルの周辺のデータ空間からサンプリングして集めたデータセットを教師データとして、データ空間の対象範囲内でのみ有効な近似用モデルを作成する。

- SHAP
  協力ゲーム理論（協力して得た報酬を、貢献度が異なるプレイヤーにどう

分配するか）の概念である shapley value（シャープレイ値）を機械学習に応用した手法。これにより、ある順序における予測値への特徴量の貢献度を表現する。

22.1 実装演習

22.1.1 4_8_interpretability.ipynb

図 15 は、Grad-CAM による画像分類の出力結果である。トイレットペーパーを分類結果として出力したうえで、ヒートマップにより画像のどの部分が判断根拠となったかを示している。下図のヒートマップから、トイレットペーパー本体より、周りの背景からトイレットペーパーを判断していることが分かる。これは、データセット内で黒の背景である画像がトイレットペーパーに偏っているか、輪郭でトイレットペーパーと判断している可能性がある。

図 15: Grad-CAM による画像分類結果

23 Docker

Docker とは、軽量かつ高速に動作するコンテナ型仮想化ソリューションである。アプリケーションと依存関係をコンテナとしてパッケージ化し、どの環境でも一貫した動作を保証できるようにしている。これにより、開発・テスト・本番環境の動作の一貫性や、コンテナを配布するだけで開発者が同じ環境を使えるといった効率性を向上させている。