# レポート作成技術

電気電子システム工学科 5 年　貝沼遼太郎

2021 年 05 月 11 日

# 1　目的

# 2　課題

## 2.1　ニューラルネットワークの仕組み，モデルの構築から学習，推論の流れについてまとめよ

### 2.1.1　ニューラルネットワークモデル（NeuralNetwork）

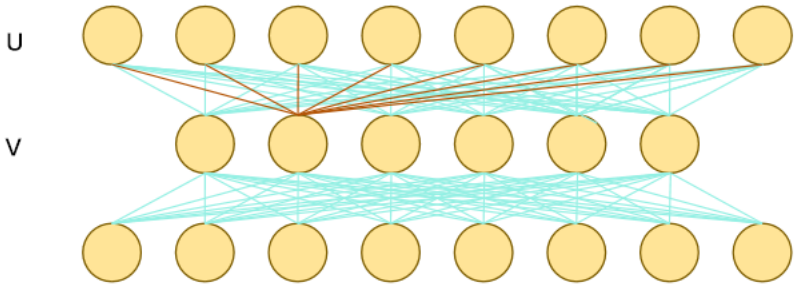
まずは，図のような単純な全結合層を 3 つほど使った多層ニューラルネットワークモデルを見てみる．

いちばん上の行を入力層という．今回のモデルは 8 個の入力ユニット（ノードともいう）を持っている．

上から 2 番目の層を中間層という．今回のモデルでは 6 個のユニットを持っている．

最後の層を出力層という．今回のモデルでは 8 個の出力のユニットを持っている．

中間層のことを隠れ層と呼ぶ場合もある．また，3 層以上のモデルをディープニューラルネットワーク（DeepNeuralNetwork:DNN）と呼ぶ．ユニットとユニットを繋ぐ線の事を，エッジという．各エッジは重みを持つ．つまり，エッジを通るとエッジの重み分の掛け算が実行される．ユニットでは，前の層の各ノードの値がエッジの重みの値をかけたものを総和し，ユニットの値とする．ユニットの値に活性化関数を通したものがそのユニットの値となり次の層に送られる．




$$v_j = \sigma(w_{j0}u_0 + w_{j1}u_1 + w_{j2}u_2 + \cdots + w_{jn}u_n + b_j)$$

図 1　多層ニューラルネットワークモデル

基本は掛けて足す（積和）の繰り返し．

行列・ベクトルを使って表記すると $v = \sigma(Wu + b)$

W は重み行列（WeightMatrix）で，ニューロとニューロを繋ぐ線を通ると，Wjn の係数が掛け算される．

### 2.1.2　活性化関数（ActivationFunction）

$\sigma()$ は活性化関数（ActivationFunction）でカッコの中の値を ReLU 関数や tanh 関数やシグモイド関数として出力する．$\sigma()$ のカッコの中の値をそのまま出力するもの，$\sigma(x) = x$ を恒等関数という．

単純なDNNやCNNでよく使われるのがReLU関数

$$y = \max(0, x)$$

RNNでよく使用されるのが tanh関数（双曲線正接：ハイパーボリックタンジェント）

$$y = \tanh x$$

sigmoid関数は確率値の出力に使われる

$$y = \frac{1}{1 + e^{-x}}$$

この他マルチクラス分類によく使われるのがsoftmax関数（合計が1になる）

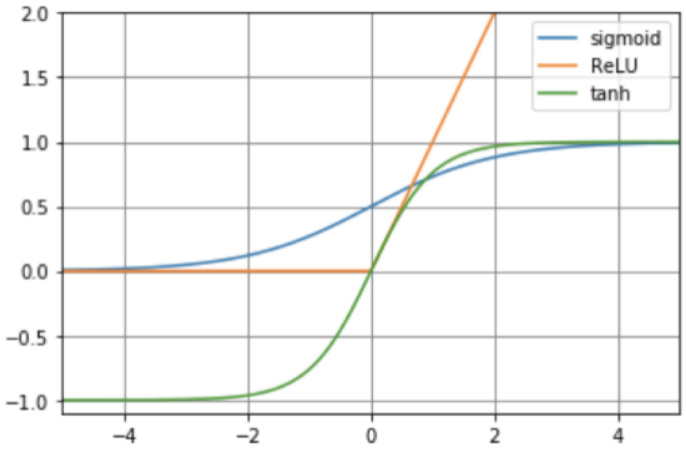



図 2　活性化関数

### 2.1.3　畳み込み（Convolution）

畳み込みは，ニューラルネットワークの入力層に入力する前の前処理として使う．

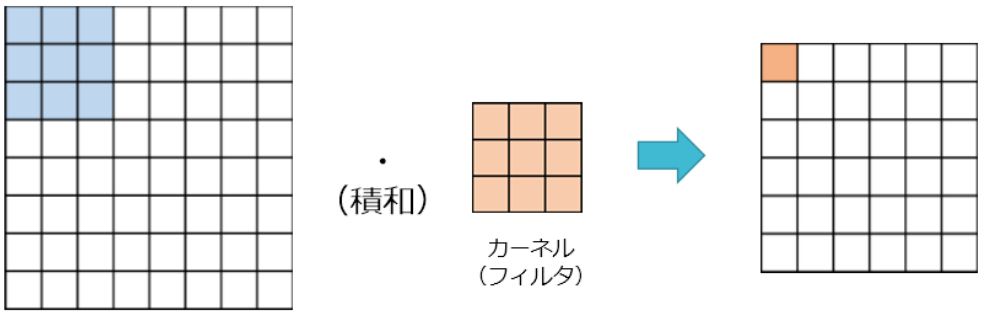



図 3　畳み込みのイメージ

画像処理の中核になっているのが畳み込み層．

3 × 3 や 5 × 5 といったサイズのカーネル（画像処理でいうフィルタ）をスライドさせながら，カーネルの重みを対応する部分行列に掛けて和を取り，バイアスを加えたものを出力の 1 要素とする．

そのままだと出力のサイズが小さくなるがパディングを行ってサイズを変えない場合もある．

カラー画像などの場合には，カーネルは 3 次元になる．RGB なので赤色を表すのに数値として 0～255 の範囲の値をもち，同様に緑色 0～255，青色 0～255 の 3 色があるので 3 次元となる．さらに，PNG 等は透明色の $\alpha$ チャンネルをもっており，RGBA の 4 次元になっている場合もある．

カーネルの数が出力のチャネル数になる．

### 2.1.4　プーリング（Pooling）

プーリングも，ニューラルネットワークの入力層に入力する前の前処理として使う．

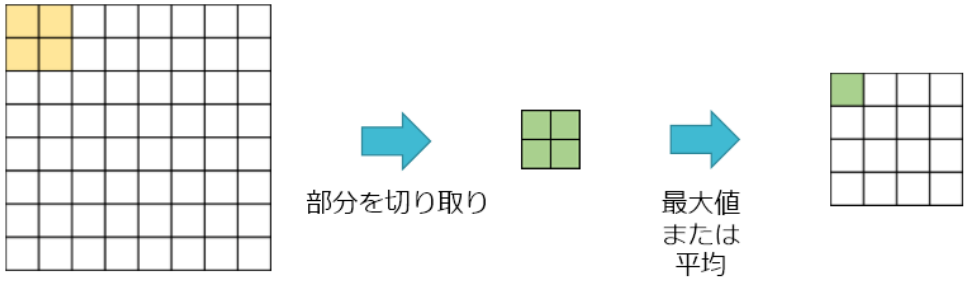



図 4　プーリングのイメージ

特徴を抽出しサイズを縮小するのがプーリング層の役割．

プールサイズは画像処理では 2 × 2 がよく使われる．

2 × 2 のサイズで 2 個ずつスライドさせれば画像は縦横それぞれ半分のサイズになる．

出力のチャネル数は変わらない．

画像処理では最大値を取る Max プーリングをよく使う．

### 2.1.5　畳み込みニューラルネットワーク（ConvolutionDeepNeuralNetwork）

CNN では，下図のように隠れ層は「畳み込み層」と「プーリング層」で構成される．

畳み込み層は，前の層で近くにあるノードにフィルタ処理して「特徴マップ」を得る．

プーリング層は，畳み込み層から出力された特徴マップを，さらに縮小して新たな特徴マップとする．

この際に着目する領域のどの値を用いるかだが，図のように最大値を得ることで，画像の多少のずれも吸収される．

したがって，この処理により画像の位置移動に対する不偏性を獲得したことになる．

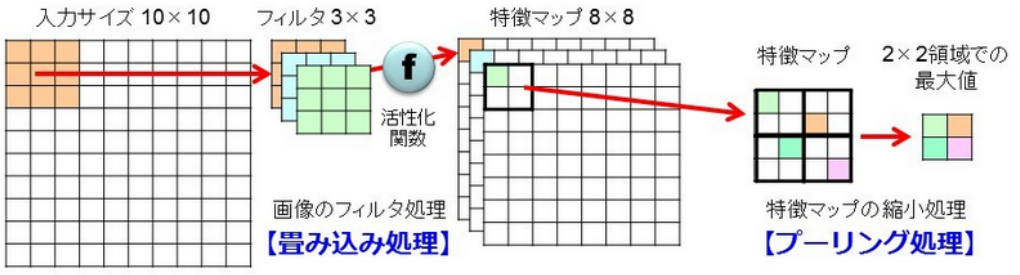



図 5　畳み込みニューラルネットワーク

畳み込み層は画像の局所的な特徴を抽出し，プーリング層は局所的な特徴をまとめあげる処理をしている．

つまり，これらの処理の意味するところは，入力画像の特徴を維持しながら画像を縮小処理していることになる．

今までの画像縮小処理と異なるところは，画像の特徴を維持しながら画像の持つ情報量を大幅に圧縮できるところである．これを言い換えると，画像の「抽象化」ともいう．

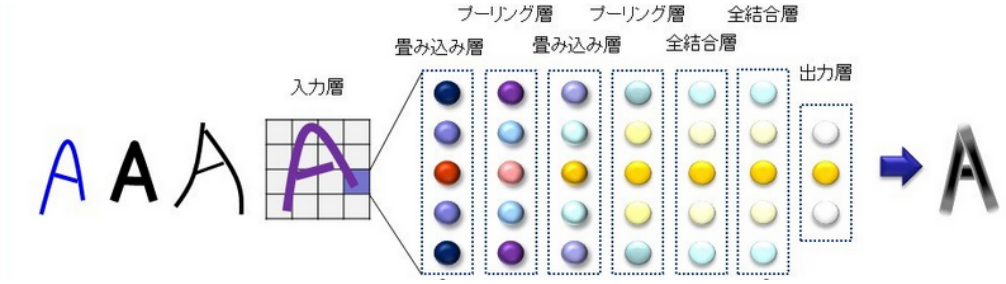



図 6　畳み込みニューラルネットワーク 2

## 2.2　画像分類の精度を上げるため，keras_hands-on_textbook.ipynb のコードを参考にして精度向上を行え．精度向上前と精度向上後とを比較し，改善あるいは劣化を考察せよ

## 2.3　過学習とはどのようなことを言うのか，また今回の実験で過学習が発生しているかを考察せよ

過学習とは，機械学習のモデルが訓練用データに最適化しすぎて，新しデータに対応できなくなることをいう．いわばモデルが訓練用データを丸暗記した状態である．つまり少しの応用問題，イレギュラーデータに対応できない状態である．これはモデルのパラメータ数に対して，訓練用データが少なすぎる場合に発生するとされている．

下図は過学習が発生したと思われる学習のモデルである．この時，学習に用いた training データは 110 ほどであったのに対して，モデルのパラメータ数は 3,321,410 となっている．よってこの学習ではモデルのパラメータ数に対して学習データが少なかったために過学習が発生したと考えられる．

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d (MaxPooling2D) | (None, 63, 63, 32) | 0 |
| conv2d_1 (Conv2D) | (None, 61, 61, 64) | 18496 |
| max_pooling2d_1 (MaxPooling2 | (None, 30, 30, 64) | 0 |
| conv2d_2 (Conv2D) | (None, 28, 28, 128) | 73856 |
| max_pooling2d_2 (MaxPooling2 | (None, 14, 14, 128) | 0 |
| flatten_1 (Flatten) | (None, 25088) | 0 |
| dense_3 (Dense) | (None, 128) | 3211392 |
| dense_4 (Dense) | (None, 128) | 16512 |
| dense_5 (Dense) | (None, 2) | 258 |

Total params: 3,321,410
Trainable params: 3,321,410
Non-trainable params: 0

図 7　過学習が発生した時のモデル

下図は過学習が発生したときの学習推移のグラフである．training データでの学習は正答率が増加し，損失が減少していることが分かるが，validation データでの検証は正答率が減少し，損失が増加していることが分かる．よって，この学習は training データに最適化され，イレギュラーなデータに対応できなくなっている事が分かる．

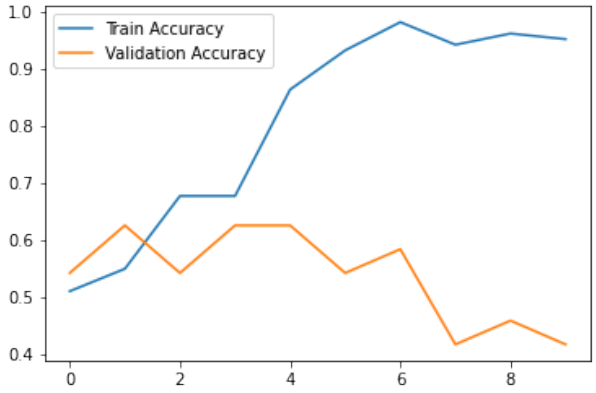



図 8　過学習と考えられる正答率の推移

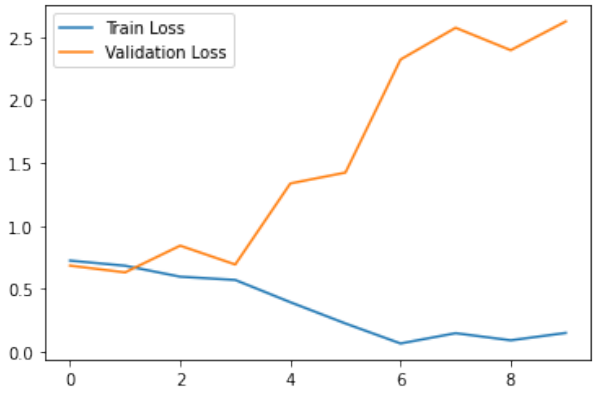



図 9　過学習と考えられる損失の推移

## 2.4　過学習を防止するための手法について考察せよ

前節のような過学習を抑制するための方法として，ドロップアウト層の導入，バッチ正規化，パラメータ正則化の 3 つが挙げられる．

### 2.4.1　ドロップアウトの導入

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               3211392
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 3,304,898
Trainable params: 3,304,898
Non-trainable params: 0
```

図 10　ドロップアウト層を追加したモデル

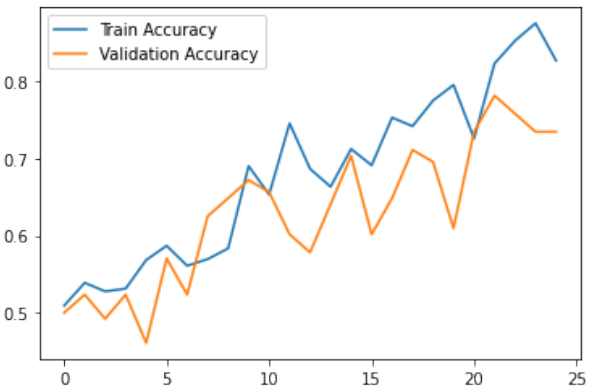



図 11　ドロップアウトの導入による正答率の推移

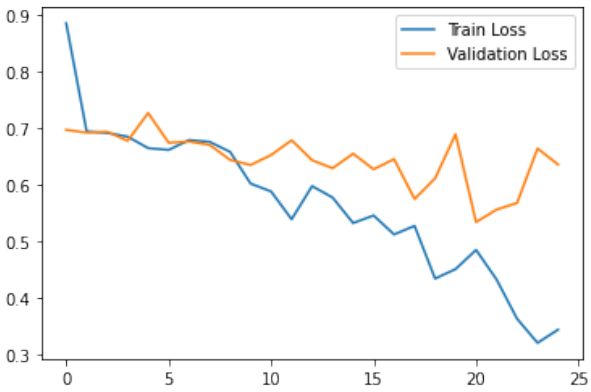



図 12　ドロップアウトの導入による損失の推移

### 2.4.2　バッチ正規化

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_6 (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d_6 (MaxPooling2 | (None, 63, 63, 32) | 0 |
| batch_normalization (BatchNo | (None, 63, 63, 32) | 128 |
| activation (Activation) | (None, 63, 63, 32) | 0 |
| conv2d_7 (Conv2D) | (None, 61, 61, 64) | 18496 |
| max_pooling2d_7 (MaxPooling2 | (None, 30, 30, 64) | 0 |
| batch_normalization_1 (Batch | (None, 30, 30, 64) | 256 |
| activation_1 (Activation) | (None, 30, 30, 64) | 0 |
| conv2d_8 (Conv2D) | (None, 28, 28, 128) | 73856 |
| max_pooling2d_8 (MaxPooling2 | (None, 14, 14, 128) | 0 |
| batch_normalization_2 (Batch | (None, 14, 14, 128) | 512 |
| activation_2 (Activation) | (None, 14, 14, 128) | 0 |
| flatten_2 (Flatten) | (None, 25088) | 0 |
| dense_4 (Dense) | (None, 128) | 3211392 |
| batch_normalization_3 (Batch | (None, 128) | 512 |
| activation_3 (Activation) | (None, 128) | 0 |
| dense_5 (Dense) | (None, 2) | 258 |

Total params: 3,306,306
Trainable params: 3,305,602
Non-trainable params: 704

図 13　バッチ正則化のモデル

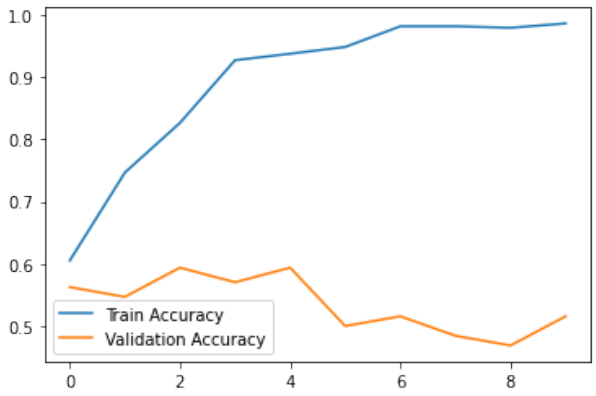



図 14　バッチ正則化したときの正答率の推移

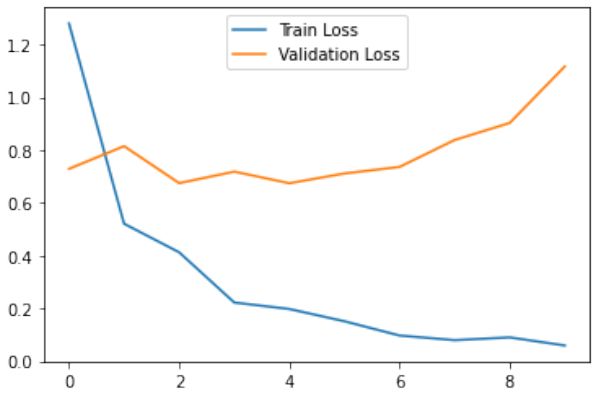



図 15　バッチ正則化したときの損失の推移

### 2.4.3　パラメータ正則化

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_9 (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d_9 (MaxPooling2 | (None, 63, 63, 32) | 0 |
| conv2d_10 (Conv2D) | (None, 61, 61, 64) | 18496 |
| max_pooling2d_10 (MaxPooling | (None, 30, 30, 64) | 0 |
| conv2d_11 (Conv2D) | (None, 28, 28, 128) | 73856 |
| max_pooling2d_11 (MaxPooling | (None, 14, 14, 128) | 0 |
| flatten_3 (Flatten) | (None, 25088) | 0 |
| dense_6 (Dense) | (None, 128) | 3211392 |
| dense_7 (Dense) | (None, 2) | 258 |

Total params: 3,304,898
Trainable params: 3,304,898
Non-trainable params: 0

図 16　パラメータ正則化のモデル

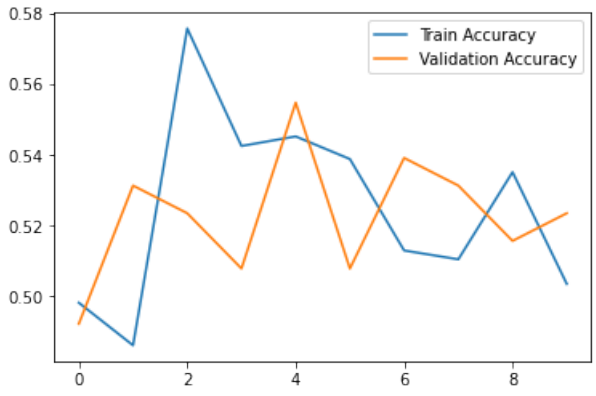



図 17　パラメータ正則化したときの正答率の推移

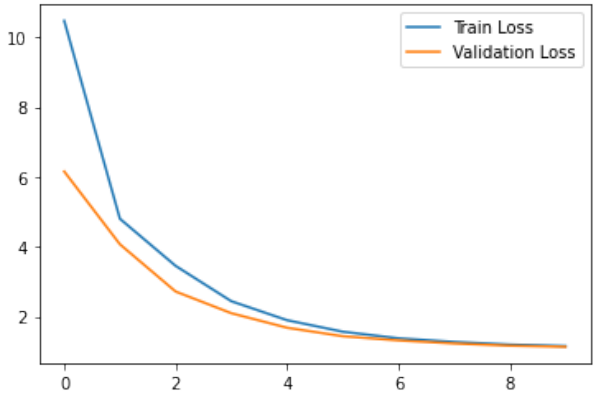



図 18　パラメータ正則化したときの損失の推移

## 2.5　ファインチューニングの手法について考察せよ

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 128, 128, 3)] | 0 |
| block1_conv1 (Conv2D) | (None, 128, 128, 64) | 1792 |
| block1_conv2 (Conv2D) | (None, 128, 128, 64) | 36928 |
| block1_pool (MaxPooling2D) | (None, 64, 64, 64) | 0 |
| block2_conv1 (Conv2D) | (None, 64, 64, 128) | 73856 |
| block2_conv2 (Conv2D) | (None, 64, 64, 128) | 147584 |
| block2_pool (MaxPooling2D) | (None, 32, 32, 128) | 0 |
| block3_conv1 (Conv2D) | (None, 32, 32, 256) | 295168 |
| block3_conv2 (Conv2D) | (None, 32, 32, 256) | 590080 |
| block3_conv3 (Conv2D) | (None, 32, 32, 256) | 590080 |
| block3_pool (MaxPooling2D) | (None, 16, 16, 256) | 0 |
| block4_conv1 (Conv2D) | (None, 16, 16, 512) | 1180160 |
| block4_conv2 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block4_conv3 (Conv2D) | (None, 16, 16, 512) | 2359808 |
| block4_pool (MaxPooling2D) | (None, 8, 8, 512) | 0 |
| block5_conv1 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_conv2 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_conv3 (Conv2D) | (None, 8, 8, 512) | 2359808 |
| block5_pool (MaxPooling2D) | (None, 4, 4, 512) | 0 |

Total params: 14,714,688
Trainable params: 0
Non-trainable params: 14,714,688

図 19　VGG16 モデル

| Layer (type) | Output Shape | Param # |
|---|---|---|
| vgg16 (Functional) | (None, 4, 4, 512) | 14714688 |
| flatten (Flatten) | (None, 8192) | 0 |
| dropout (Dropout) | (None, 8192) | 0 |
| dense (Dense) | (None, 2) | 16386 |

Total params: 14,731,074
Trainable params: 16,386
Non-trainable params: 14,714,688

図 20　ファインチューニングに用いたモデル

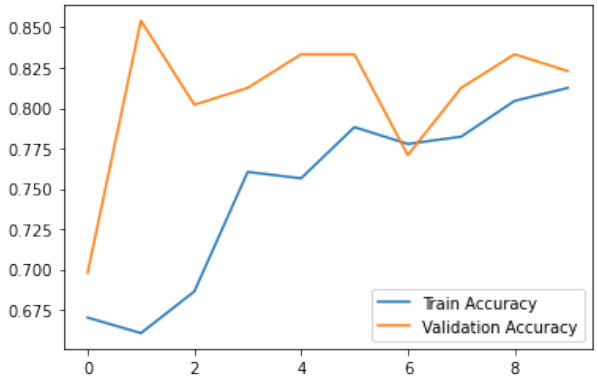



図 21　重みを凍結して学習したときの正答率の推移

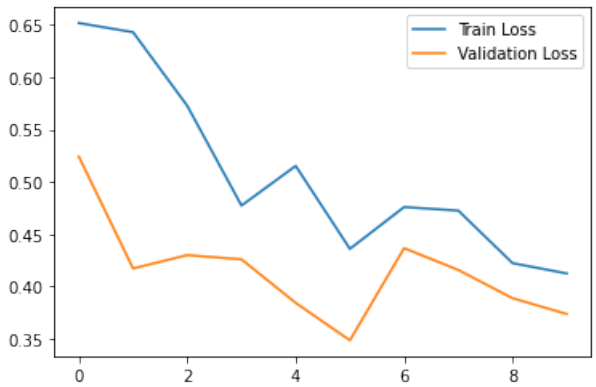



図 22　重みを凍結して学習したときの損失の推移

ファインチューニングを行うために凍結した conv_layers の block5 の層の凍結を解除．Trainable params の値が増えていることが分かる．

| Layer (type) | Output Shape | Param # |
|---|---|---|
| vgg16 (Functional) | (None, 4, 4, 512) | 14714688 |
| flatten (Flatten) | (None, 8192) | 0 |
| dropout (Dropout) | (None, 8192) | 0 |
| dense (Dense) | (None, 2) | 16386 |

Total params: 14,731,074
Trainable params: 7,095,810
Non-trainable params: 7,635,264

図 23　ファインチューニングに用いた学習モデル

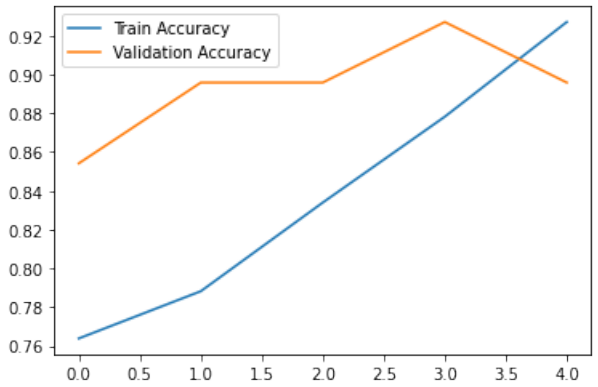



図 24　凍結解除して学習したときの正答率の推移

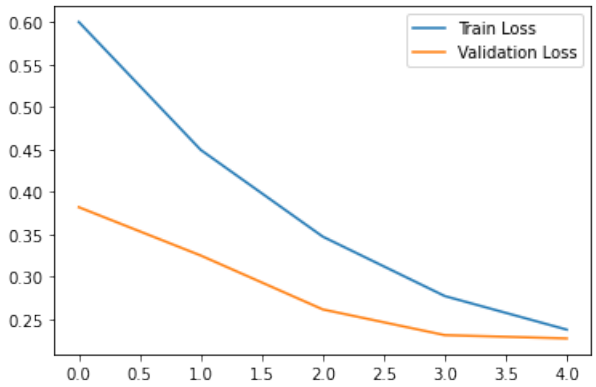



図 25　凍結解除して学習したときの損失の推移

## 3　参考文献


- 「Python によるデータ解析応用編」-矢野 昌平