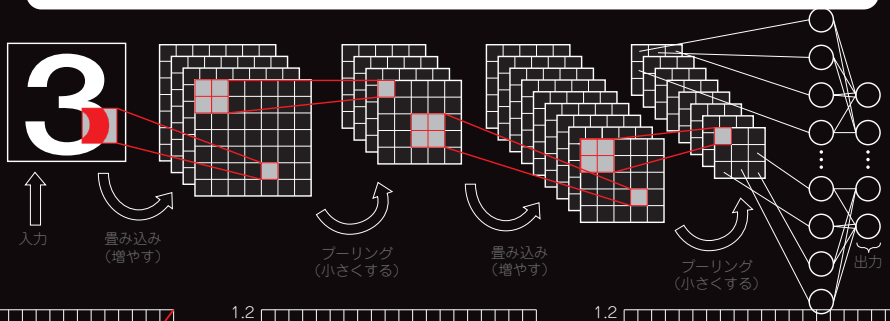
改訂 TensorFlow版

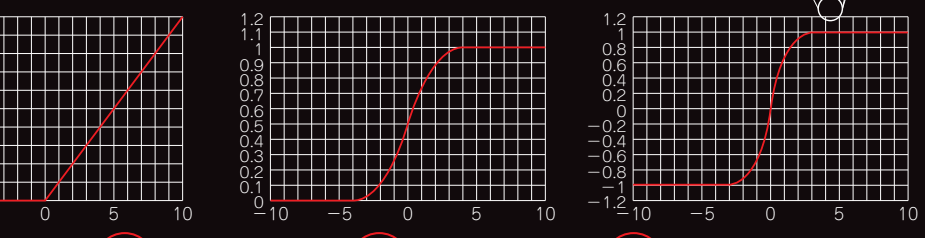
データサイエンス・シリーズ

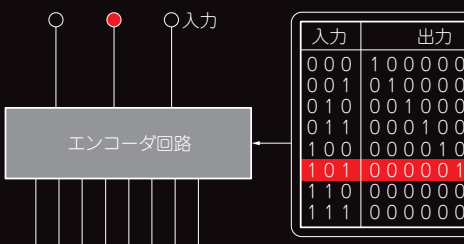
# 算数&ラズパイから始める ディープ・ラーニング

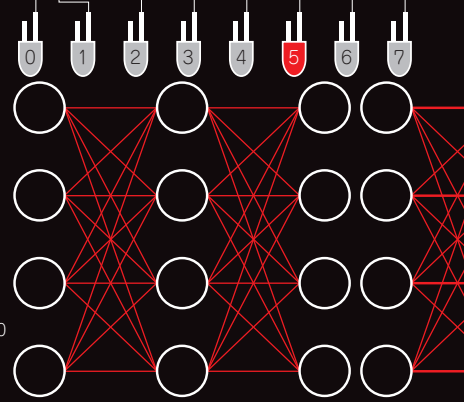
## CNN/RNN/AE/DQNで 画像・音声・データ分析

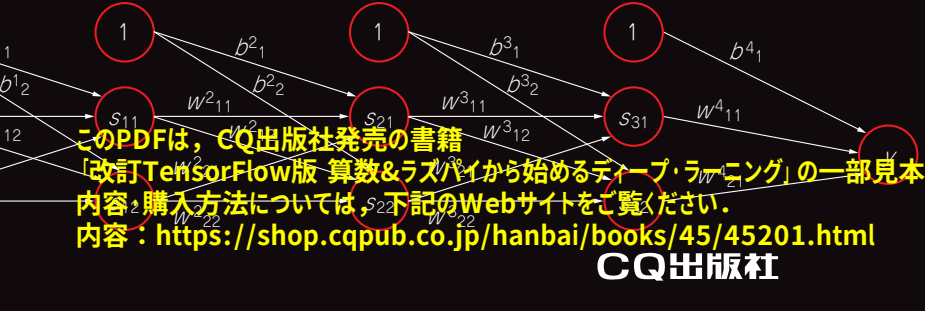
牧野 浩二，西崎 博光 共著

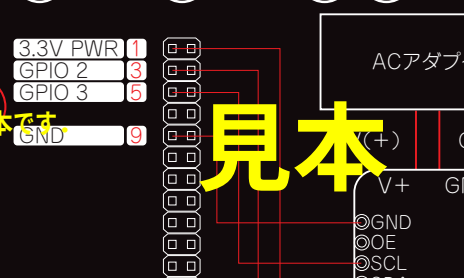






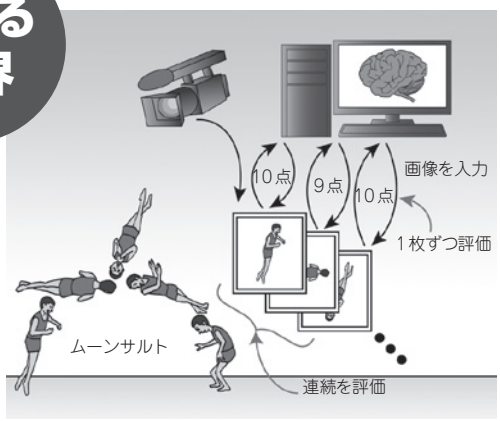
全プログラム付きですぐに

# 本書の歩き方

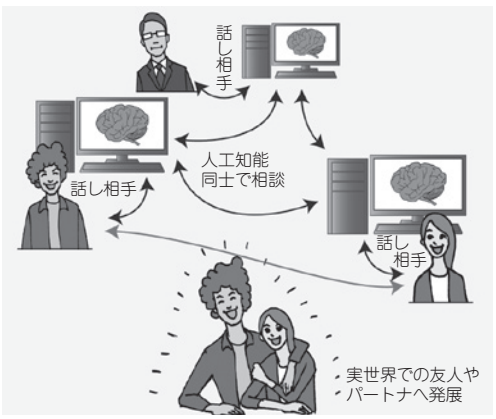
## 第1部　ディープ・ラーニングの世界へようこそ

ディープ・ラーニングはいったい何がすごいのでしょうか．そして，読者の皆さんは何ができるようになるのでしょうか．第1部ではディープ・ラーニングで広がる世界を紹介します．併せて，知らないと始まらない基本中の基本とも言える3大アルゴリズムを紹介します．

### 広がる世界

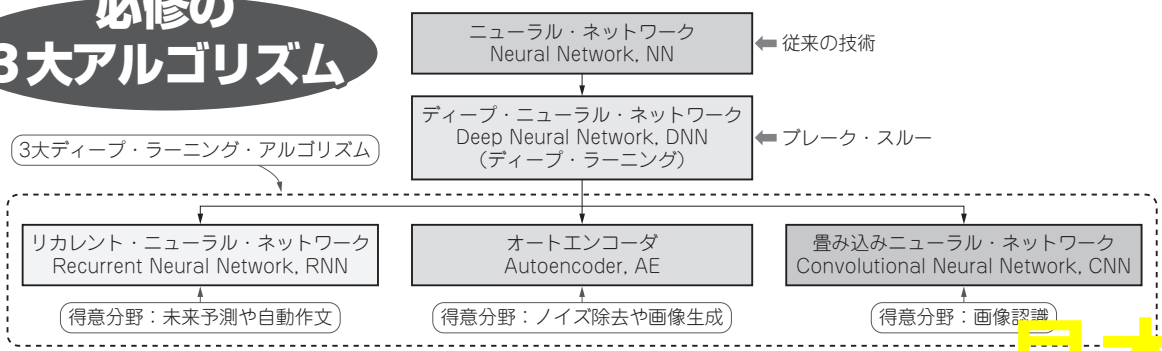



採点競技への応用…見た目に惑わされることがなくなるかも




自分と趣味や感性が合う人を紹介してくれるかも

### 必修の3大アルゴリズム



# 第1章

## エンジン音で車種判定やMyロボの異常検知などに

# 体験①…音でお菓子認識

写真1　ディープ・ラーニングにより音でお菓子の中身（種類）判定をラズベリー・パイを使って体験してみた

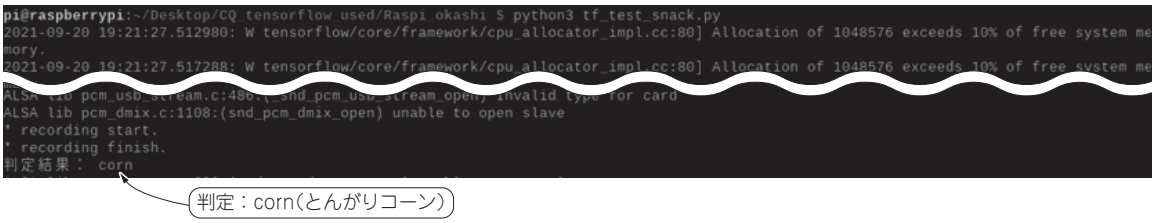



写真2　ラズベリー・パイでディープ・ラーニングしてお菓子の種類を判定

## トライすること

### ●第3部の構成

第3部では，人気の人工知能「ディープ・ラーニング」を使ったプログラミングができるようになるために，ステップ・バイ・ステップで知識を習得していきます．

体験すると，やる気が出やすいと思いますので，まず人工知能をラズベリー・パイ上で動かしてみます．具体的には，PC上でディープ・ラーニングの学習済みモデル（判定用データ）を作ります．それをラズベリー・パイ上で動かし，リアルタイムに対象物を分類してみます．第3部では次の実験を行います．

# 第2章

## 画像処理を知らなくてもOK!<br>きのことたけのこを判別してみる

# 体験②…画像認識1<br>(お菓子の種類)

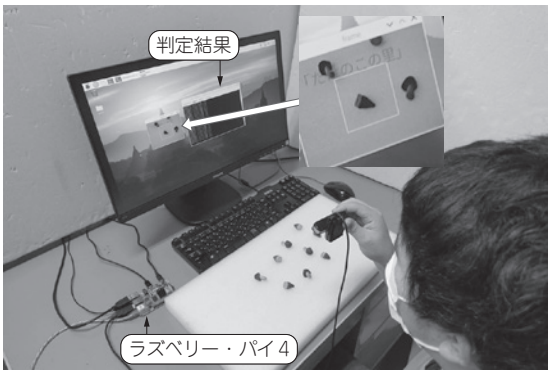



写真1　画像ディープ・ラーニングによるお菓子判定をラズベリー・パイで行う

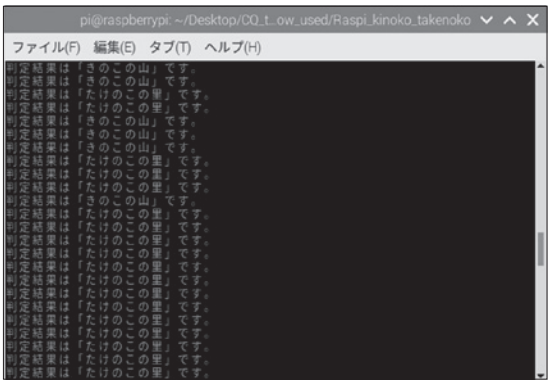



写真2　ラズベリー・パイによる画像ディープ・ラーニングでお菓子判定

### ●実験すること

お菓子「きのこの山」と「たけのこの里」をカメラで撮って，見分ける装置を作ります(**写真1**)．この装置は，一定時間ごとに画面の画像上に文字が出ます(**写真1**参照)．さらに，**写真2**のように判定結果の文字が表示されます．

### ●用意するもの

**▶ハードウェア**

1. ラズベリー・パイ4
2. USB接続のカメラ
3. キーボード
4. マウス
5. PC(学習済みモデルを利用する場合は不要)
6. ディスプレイ

**▶ソフトウェア(著者提供)**

1. 学習済みモデル(自分で作る場合は不要)
2. ラズベリー・パイ上で動く判定用プログラム

# 第3章

## ペットの判定や果物の出荷検査に
# 体験③…画像認識2（本物/偽物）

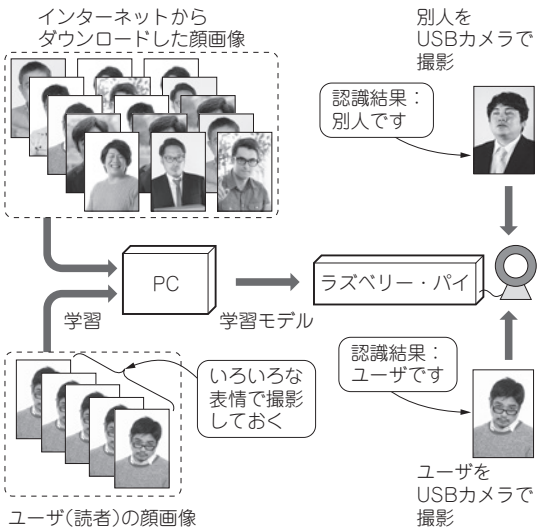



**図1　製作する装置は本人と別人とを見分けられる**

### ●実験すること

3大ディープ・ラーニングの1つ，畳み込みニューラル・ネットワーク（CNN）を使って，顔を認識する装置を作ります[注1].

ここでは**図1**のように，インターネットからダウンロードした大量の顔画像とユーザ（読者）の顔画像を一緒に学習し，ユーザの顔にだけ反応する装置を作ります．なお，学習はPCで，判定はPCとラズベリー・パイで行います．

### ●用意するもの

**▶ハードウェア**

1. ラズベリー・パイ4
2. ディスプレイ

注1：畳み込みニューラル・ネットワークのアルゴリズムは第5部で解説する．

# 第4章

## 第6部で解説する自動運転や対戦AIのもとをまずは

# 体験④…迷路脱出

### ●実験すること

これから注目のディープ・ラーニングである深層強化学習を実装する方法の1つである「ディープQネットワーク」を使って迷路をクリアします.

3大ディープ・ラーニングは必ず教師データが必要ですが，ディープQネットワークは半教師付き学習と呼ばれ，良い行動と悪い行動を教えておくと，良い行動をとるように自動的に進化するものです.

そこで，壁にぶつかると「悪い行動」，ゴールすると「良い行動」として教えることとします．アルゴリズムは本書の第6部で詳しく解説します.

ここではPC上で迷路を解きます．PCでの学習結果を使って，ラズベリー・パイに取り付けた2つのRCサーボモータを動かし，迷路を脱出します．なお，学習時間をかければ，ラズベリー・パイ上でRCサーボモータを動かしながら，実際の環境を学習することもできます.

### ●用意するもの

▶ハードウェア

1. ラズベリー・パイ4
2. ディスプレイ
3. キーボード
4. マウス
5. RCサーボモータとマウント，モータ・ドライバ
6. PC（学習用）

▶ソフトウェア

1. ラズベリー・パイ上で動く移動用プログラム
2. PC上で動く学習用プログラム

### ●プログラム

ディレクトリ名：`Raspi_maze`
動作確認用プログラム（ラズベリー・パイ用）：`rp_tf_play_maze_DQN.py`

# 第5章

## スマート・スピーカや自動操縦に
# 体験⑤…話者認識


写真1　話者認識

### ●実験すること

3大ニューラル・ネットワークの1つ，リカレント・ニューラル・ネットワーク（RNN．第1部，第5部で解説）を使って，話している人を声から認識する装置を作ります（**写真1**）．

声の質に着目し，どんな言葉であっても声を発した人（話者）を特定するというものです．本章はすべてラズベリー・パイで行うことを前提にします．

### ●用意するもの

▶ハードウェア（図1）

1. ラズベリー・パイ4
2. ディスプレイ
3. USB接続のマイク
4. キーボード
5. マウス

▶ソフトウェア

1. ラズベリー・パイ上で動く判定用プログラム
2. ラズベリー・パイ上で動く録音用プログラム（ラズベリー・パイに取り付けたUSBマイクで録音する場合）

# 第1章

## 定番データセットの文字認識で体験
# 画像向きCNN①…手書き認識

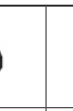
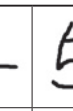
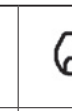
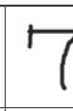
（a）入力画像

```
> python tf_eval_myimage_cnn.py -i tegaki/tegaki2.png -m MNIST_CNN.h5
入力画像ファイル: tegaki/tegaki2.png
訓練済みモデル: MNIST_CNN.h5
判定結果は「2」です
```

（b）実行結果

**図1　自分で書いた手書き文字を分類する**

**表1　自分で書いた手書き文字の分類結果**

| 入力画像 |  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|
| ファイル名 | tegaki0.png | tegaki1.png | tegaki2.png | tegaki3.png | tegaki4.png | tegaki5.png | tegaki6.png | tegaki7.png | tegaki8.png | tegaki9.png |
| 分類結果 | 0 | 1 | 2 | 9（誤り） | 4 | 5 | 6 | 7 | 8 | 9 |

ディープ・ラーニングの仕組みとフレームワークTensorFlowの動かし方を理解したら，いよいよ実践に移りましょう．

畳み込みニューラル・ネットワーク（Convolutional Neural Network：CNN）は，画像処理に強いディープ・ラーニングです．

最近では，リカレント・ニューラル・ネットワーク（RNN）やオートエンコーダ（Autoencoder）に組み入れられて使うケースが増えています．

MNIST[(1)]は，手書き文字を学習したり，正しく認識できるかどうかをチェックしたりする際に利用できるデータセットです．画像認識ですので，畳み込みニューラル・ネットワークの方が向いています．

ここでは，畳み込みニューラル・ネットワークで手書き文字を認識してみます．

### ●やること…画像向きCNNを使って精度良く手書き文字を認識する

手書き文字の画像データを入力し，文字を認識します．実行結果を**図1**に示します．10個の画像を分類した結果を**表1**に示します．3だけうまく分類できませんでした．

第4部第5章の自作ディープ・ニューラル・ネットワークでは，0，3，6，9の画像がうまく分類で

# 第2章

## 画像の収集や学習を体験
# 画像向きCNN②…感情認識

ここでは，畳み込みニューラル・ネットワークを用いて，たくさんの顔画像から5種類の感情分類（怒・嬉・普通・悲・驚）を行ってみます．自分で集めた画像で学習させてみます．

## 実験

### ●できること…顔写真からの感情認識

入力した顔写真（**写真1**）からの5種類の感情が分類できます．分類の様子を**図1**に示します．

### ●準備

実験には，**表1**のプログラムを使います．これらは本書サポート・ページからダウンロードできます．

感情推定に必要なライブラリを**リスト1**の手順でインストールします．dlibのインストールにはかなりの時間（30分以上）がかかる場合があります．

### ●ステップ1：画像の収集

感情が5つに分類されている画像データのデータベースはありません．著者らは，インターネットから画像を1枚ずつ地道にダウンロードして集めました．

まず「怒り」の顔画像を集めてみます．GoogleやBingの画像検索で「angry face」と検索します．すると，とてもたくさんの「怒っている顔」が表示されます．その中から実際に「怒っている顔」だと思うものを画像として保存します．

同様に，他の感情の画像も集めていきます．保存先はoriginalというディレクトリを作り，その下にtrainというディレクトリを作り，作った各感情のディレクトリとしました．これは学習用データですので，検証用のデータも集めておきましょう．保存先はoriginalディレクトリの下にtestディレクトリを作っておきます．

なお，1枚の画像に2人以上の顔が映っていても問題はありません．例えば，3人映っている場合，1人が怒っていて，2人が驚いている画像の場合は次のように分類結果が表示されます．

# 第3章

## 「予測が得意」なアルゴリズムを体験

# データ分析向きRNN①…値の未来予測

リカレント・ニューラル・ネットワーク（RNN：Recurrent Neural Network）は，過去の情報も使って答えを出すことを行います．そのため，未来の予測や文脈の理解など，つながりがある情報を扱うときによく利用されます．

そこで為替を例にとり，明日の為替の終値は「円安」になるのか「円高」になるのかを予測してみます．

## 実験

### ●できること…為替を80%の精度で予測

学習したモデルを用いて為替が予測できているかどうかを確認します（**図1**）．80%の精度で予測できています．

### ●準備

実験には，**表1**のプログラムとデータを使います．これらは本書サポート・ページからダウンロードできます．

皆さんが用意したデータを検証データとして使うこともできます（作成方法はコラム3を参照）．

### ●ステップ1：モデルの学習

為替を学習します．実行の様子を**図2**に示します．学習終了後は，学習データにおいて87.92%で予測できていました．学習に使わなかった検証データを用いた場合も，84.85%で予測できていました．

### ●ステップ2：予測

学習したモデルを用いて為替が予測できているかどうかを確認します．手順は，**図1**で示したとおりです．

プログラムの実行に当たっては，**表2**に示すオプションを指定できます．また，実際に試す際には，検証データを使うことができます．

# 第4章

## 人間のアシスタントとして一大分野に発展するかも

# データ分析向きRNN②…文章の自動生成

リスト1　リカレント・ニューラル・ネットワークによる自動作文…それなりに意味の通る文章になっている？

```
python tf_make_sentence.py -w LM.h5 -v vocab

単語を入力>>> 吾輩
自動作文結果: 吾輩 は この くらい な 比喩 を 云う 。 </s>
```

（a）自動作文の実行例

```
単語を入力>>> 猫
自動作文結果: 猫 の ごとく 、主人 の 顔 を 見 て 、主人 の 顔 を 見 て 、「 君 は 何 の 事 だ から 、そんな 事 を 云う の は 、あの 通り 将軍家 が 、あの 男 の よう な 事 を 云う の です 」</s>
単語を入力>>> 僕
自動作文結果: 僕 は 何 の 事 だ から 、その 時 は 、その 時 の 方 を 見 て 、その 時 に は 贋札 を 割る 。 </s>
単語を入力>>> 机
自動作文結果: 机 の 上 に は 薄っぺら な メリンス の 座布団 が ある 。 </s>
単語を入力>>> 時計
自動作文結果: 時計 は 何 でも いい 。 </s>
単語を入力>>> 主人
自動作文結果: 主人 は 「 君 の 所 へ 行っ て 、御 逗入ん なさい が 、私 は 御 休み に なり ます 」と 主人 は 主人 の 述懐 で ある 。 </s>
```

（b）さまざまな作文結果

### ●やること＆課題

ここでは，コンピュータに文章を生成させてみます．今回のサンプルは日本語の作文ができるようになっています．

第3章で示したリカレント・ニューラル・ネットワークの初期型では，過去の情報がどんどん薄まっていきます．それはそれで良いこともありますが，次のような場合はうまく答えることができなくなります．

「修学旅行で京都に行ったときに印象に残ったのは，そこに住む人たちの温かさと，歴史ある街並みで，中でも○○のすばらしさに圧倒されました．」

これを読むと，○○に入るのは「金閣寺」や「清水寺」と推測できます．しかし，京都という単語が離れ過ぎているため，初期型のリカレント・ニューラル・ネットワークでは○○をうまく答えることができなくなります．

### ●進化したRNN…LSTM

そこで，以下の2つを実現できる仕組みが必要です．

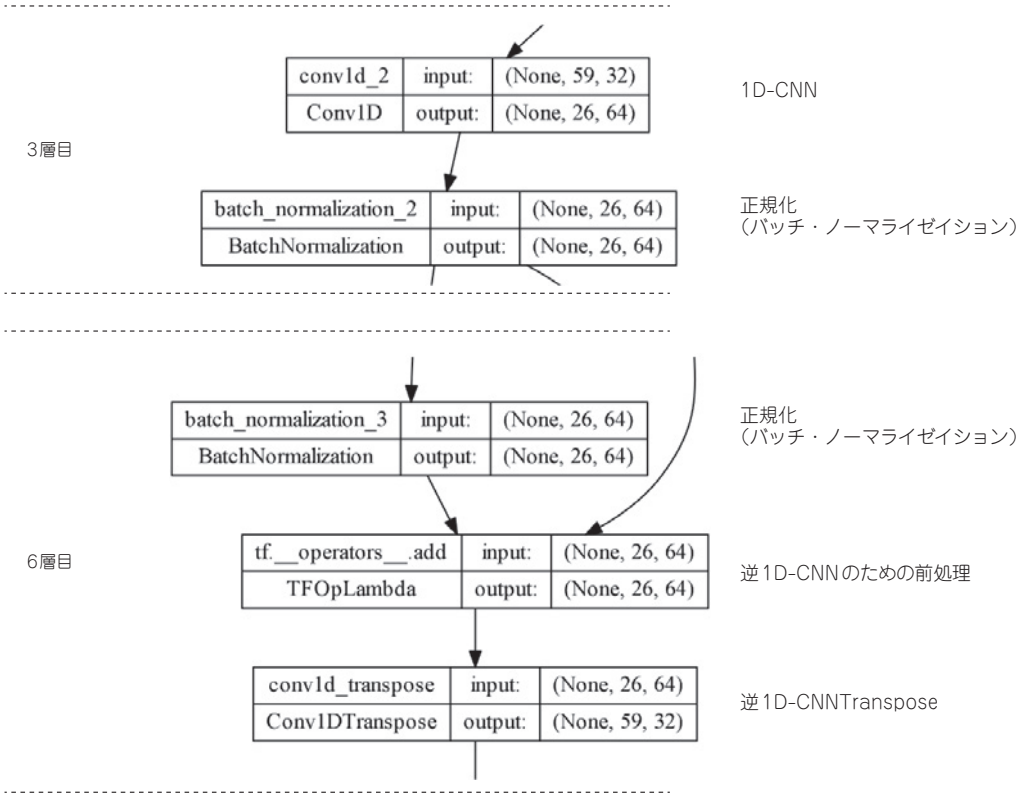



**図6　作成したオートエンコーダの構造**（図5の第3層と第6層部分の拡大）

## ノイズ除去プログラム

今回作成したオートエンコーダの構造を**図5**に示します．オートエンコーダの特徴は，半分を境に逆の構造を持つ層で構成されることです．ここでは，4層で逆の構造になっています．

まず，入力データ（257 × 2）は1D-CNNと呼ばれる1次元の畳み込みニューラル・ネットワーク（`Conv1D`）を用いて処理されます．これを3回続けます．

その後，平滑化処理（1664）を行いニューラル・ネットワークの（Dense）層により128次元のデータとします．このようにオートエンコーダではデータの次元を小さくします．ここまでの処理が4層目です．

そして，それとは全く反対の処理を行います．まず，ニューラル・ネットワークの（Dense）層により，1664次元とし，その後の逆1D-CNN（`Cenv1D`）と呼ばれる1D-CNNの逆の処理を行う1次元の特別な畳み込みニューラル・ネットワーク（`Conv1DTranspose`）を3回行います．これにより，入力と同じ次元を持つ出力データ（257 × 2）を作成します．

3層目と6層目を拡大したのが**図6**です．3層目では1次元畳み込みニューラル・ネットワーク（1D-CNN）を行った後に正規化（バッチ・ノーマライゼーション）を行っています．

6層目は逆で，正規化を行った後，逆1次元畳み込みニューラル・ネットワーク（1D-CNNTranspose）を行っています．ちょうど逆の処理をしていることが分かります．

オートエンコーダでノイズを除去するプログラム（`tf_train_dae.py`）を**リスト1**に示します．プログラムは大きく5つの処理に分かれています．

リスト4　画像評価用プログラム tf_exal_vae.py

```
import numpy as np
import argparse, os
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Model
from tensorflow.data import Dataset
from tensorflow.keras.losses import MSE
from tensorflow.keras.layers import Input, Dense,
Conv2D, Flatten, Conv2DTranspose, Reshape
from tensorflow.keras import losses
from tensorflow.keras.optimizers import Adam

"""
モデルクラス
"""
class ConvVAE(Model):
      def __init__(self, n_zdim):
          super(ConvVAE, self).__init__()
          self.n_zdim = n_zdim
          self.encoder = self.encoder_model(self.n_zdim)
          self.decoder = self.decoder_model(self.n_zdim)

      # 損失計算・誤差逆伝播関数をオーバーライド
      def train_step(self, data):
          with tf.GradientTape() as tape:
              z_mu, z_logvar = self.encoder(data)
                                  # 入力画像の潜在ベクトルを計算
              z = self.reparameterization(z_mu, z_
                                  logvar) # 潜在ベクトルの生成
              reconst = self.decoder(z)
                                  # 潜在ベクトルから画像を生成
              """
              損失計算(MATLABチュートリアル参照)
              """
              reconst_loss = tf.reduce_sum(MSE(reconst,
                                      data)) # 平均二乗誤差
              KL_loss = -0.5 * tf.reduce_sum(1 + z_
          logvar - tf.square(z_mu) - tf.exp(z_logvar), 1)
              total_loss = reconst_loss + KL_loss

          gradients = tape.gradient(total_loss, self.
                          trainable_variables) # 勾配計算
          self.optimizer.apply_gradients(zip(gradients,
                    self.trainable_variables)) # 誤差逆伝播
          return {'loss': total_loss, 'reconst_loss':
                        reconst_loss, 'KL_loss': KL_loss}

      # エンコーダモデルの定義
      def encoder_model(self, n_zdim):
          encoder_in = Input(shape=(28, 28, 1))
          h = Conv2D(32, (3, 3), strides=(2, 2),
                           activation='relu')(encoder_in)
          h = Conv2D(64, (3, 3), strides=(2, 2),
                                    activation='relu')(h)
          h = Flatten()(h)
          z_mu = Dense(n_zdim, name='z_mu')(h) # 平均
          z_logvar = Dense(n_zdim, name='z_logvar')(h)
                                                  # 対数分散
          return Model(inputs=encoder_in, outputs=[z_mu,
                              z_logvar]) # 潜在変数の平均・分散

      # デコーダモデルの定義
      def decoder_model(self, n_zdim):
          latent_in = Input(shape=(n_zdim, ))
          h = Dense(7*7*32, activation='relu')(latent_in)
          h = Reshape(target_shape=(7, 7, 32))(h)
          h = Conv2DTranspose(filters=64, kernel_size=3,
         strides=2, padding='same', activation='relu')(h)
          h = Conv2DTranspose(filters=32, kernel_size=3,
        strides=2, padding='same', activation='relu')(h)
          output = Conv2DTranspose(filters=1, kernel_
    size=3, strides=1, padding='same', name='output')(h)
          return Model(latent_in, output)

      @tf.function
      # 再パラメタライゼーショントリック
      def reparameterization(self, z_mu, z_logvar):
          epsilon = tf.random.normal(shape=tf.shape(z_
               logvar), mean=0, stddev=1.0) # epsilon生成
          return z_mu + epsilon * tf.exp(0.5 * z_logvar)

      # エンコード処理
      def encode(self, x):
          mean, logvar = self.encoder(x)
          return mean, logvar

      # デコード処理
      def decode(self, z):
          return self.decoder(z)

"""
main関数
"""
def main():
      """
      オプション処理
      """
      parser = argparse.ArgumentParser(description='MNI
                                              ST VAE学習プログラム')
      parser.add_argument('--epoch', '-e', default=20,
                                        type=int, help='エポック数')
      parser.add_argument('--zdim', '-z', type=int,
                              default=20, help='潜在変数の次元数')
      parser.add_argument('--batchsize', '-b', type=int,
                              default=50, help='ミニバッチサイズ')
      parser.add_argument('--modeldir', '-m',
              default='model', help='モデルの保存ディレクトリ')
      parser.add_argument('--lrate', '-r', type=float,
                               default=0.0001, help='初期学習率')
      args = parser.parse_args()

      print('# 潜在変数の次元数: {}'.format(args.zdim))
      print('# ミニバッチサイズ: {}'.format(args.batchsize))
      print('# エポック数: {}'.format(args.epoch))
      print('# 初期学習率: {}'.format(args.lrate))
      print('# モデルの保存: {}'.format(args.modeldir))
      print('')

      """
      MNISTダウンロード&準備
      """
      (x_train, _), (_, _) = mnist.load_data()
                                          # 訓練データの画像のみ利用
      x_train = np.expand_dims(np.float32(x_train), -1)
            / 255. # チャンネルを増やして、float32に変換して正規化

      # tf.Tensor形式のデータセットの準備
      train_ds = Dataset.from_tensor_slices(x_train)
      train_ds = train_ds.shuffle(buffer_size=len(train_
                                                      ds))
      train_ds = train_ds.repeat(1)
      train_ds = train_ds.batch(args.batchsize)

      """
      訓練&モデルの保存
      """
      VAE = ConvVAE(args.zdim) # モデルの作成
      VAE.compile(optimizer=Adam(learning_rate=args.
                                      lrate)) # 最適化関数の設定
      VAE.fit(train_ds, epochs=args.epoch) # 訓練

      model_enc = os.path.join(args.modeldir, 'MNIST_
                                                  CVAE_ENC')
      model_dec = os.path.join(args.modeldir, 'MNIST_
                                                  CVAE_DEC')
      VAE.encoder.save_weights(model_enc)
                                          # 訓練済みパラメータの保存
      VAE.decoder.save_weights(model_dec)
      print('The models were saved.')
      # 終わり

if __name__ == '__main__':
      main()
```

ISBN978-4-7898-4520-5
C3055 ￥2800E
CQ出版社
定価 3,080円（本体2,800円）⑩
9784789845205
1923055028005

データサイエンス・シリーズ

改訂TensorFlow版

# 算数&ラズパイから始めるディープ・ラーニング

CNN/RNN/AE/DQNで
画像・音声・データ分析

見本