機械学習のための
データ前処理の基本と実践法
9

データ拡張、ファインチューニング

徳島大学
デザイン型AI教育研究センター
鳥井 浩平

今回の内容

1. データ拡張
2. ファインチューニング

前回の復習

畳み込みニューラルネットワークの実装

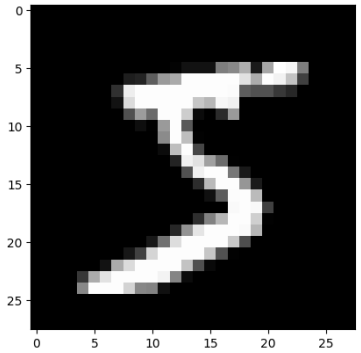
畳み込みニューラルネットワーク

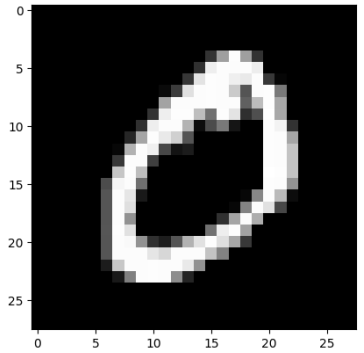
- **Convolutional Neural Network (CNN)**
- **ニューラルネットワーク（NN）**に**畳み込み処理**を導入したもの
- 画像に特化したネットワーク構造をもつ
- 1998年にYann LeCunが提案したLeNetが元祖CNN
  Yann LeCun：2019年にチューリング賞を受賞
- 現代の高精度な画像認識AIの多くはCNNを用いている

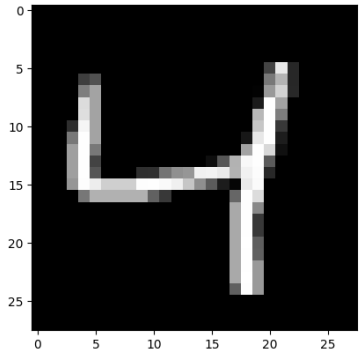
MNISTデータセット

- 手書き文字認識用のデータセット
- 数字の0~9を手書きした画像とラベルのセット
- 画像は28×28のグレースケール画像
- 学習データ60,000枚、テストデータ10,000枚

画像







ラベル

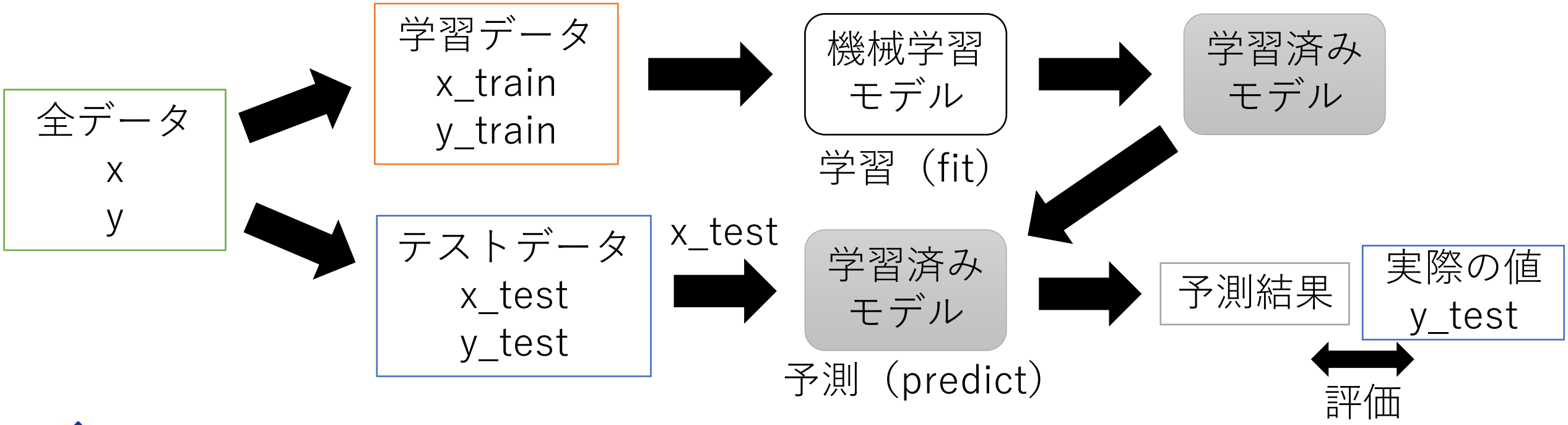
5

0

4

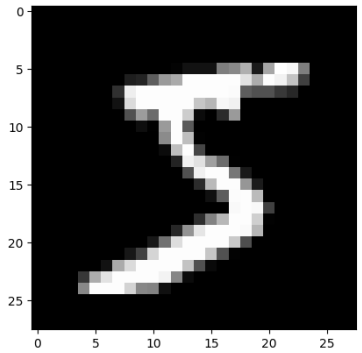
機械学習の実験プロセス

1. 全データを学習データとテストデータに分ける
2. 学習データを用いて機械学習モデルの学習（fit）を行う
3. テストデータを用いて機械学習モデルの評価（predict）を行う



モデルのイメージ

- 画像を入力すると画像に写る数字が何であるかを出力する
- 出力は10次元のベクトル



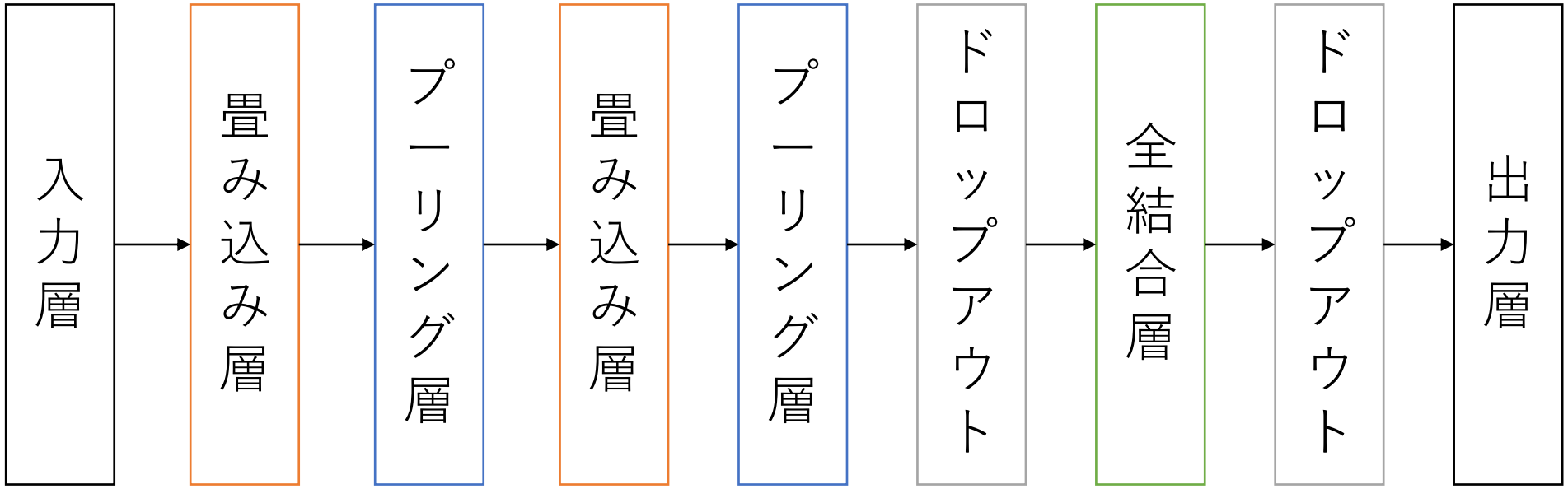
CNN

0 1 2 3 4 5 6 7 8 9

(0, 0, 0, 0.1, 0, 0.9, 0, 0, 0, 0)

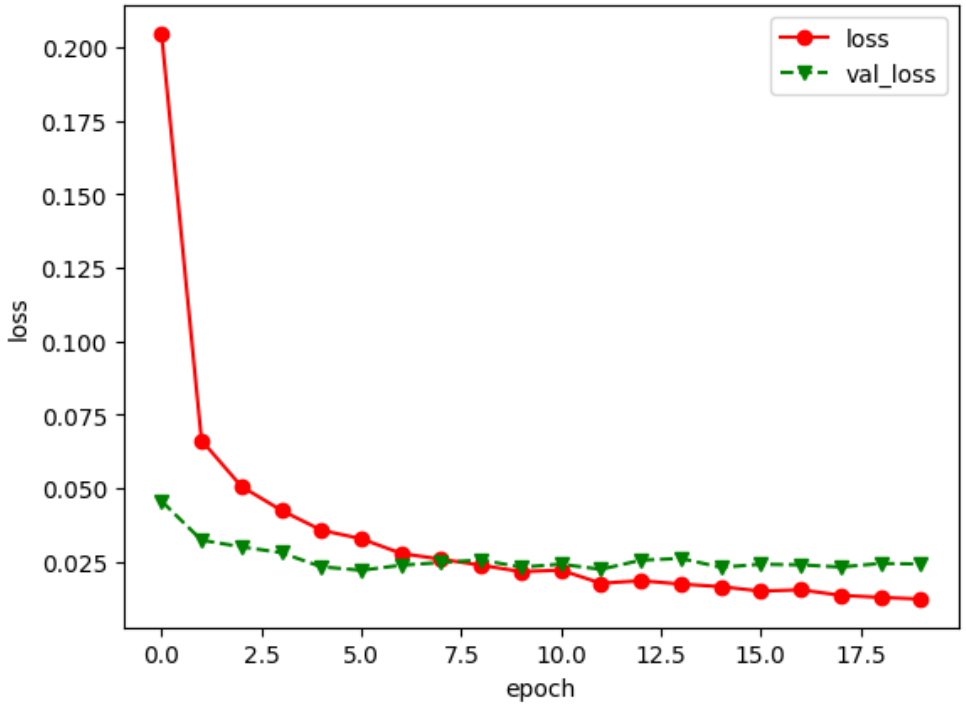
CNNモデルの概要図

今回は以下のような構成のCNNモデルを構築する



誤差グラフの見方

- 赤が学習データに対する誤差
- 緑が検証（テスト）データに対する誤差
- 両方とも右肩下がりがベスト

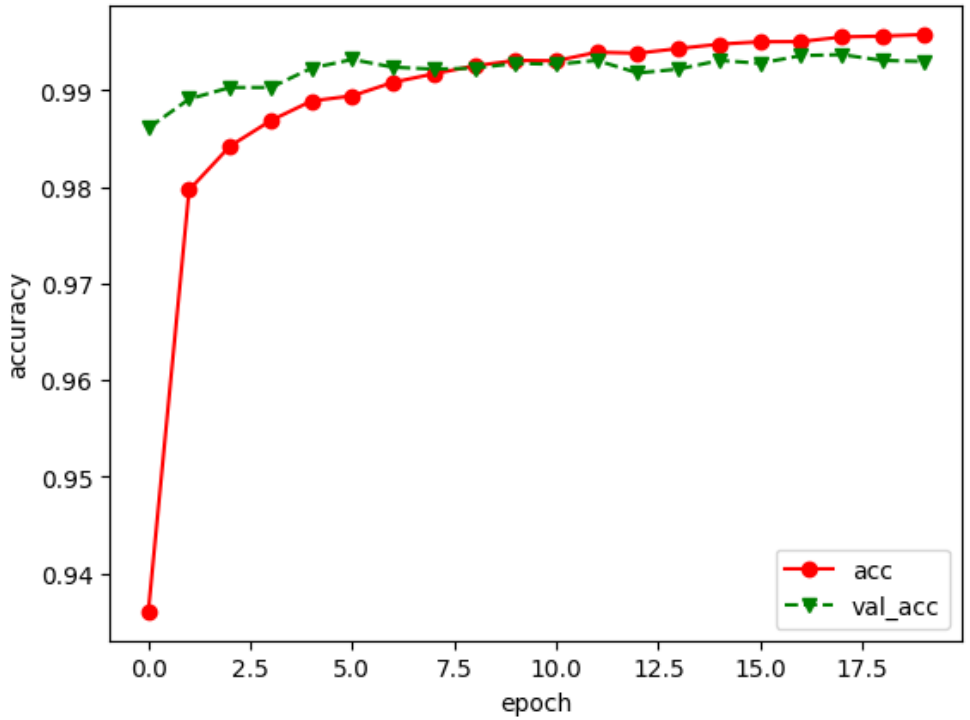


【よくある事例】

- 緑だけ右肩上がり→**過学習**を疑う
- 誤差が安定しない→学習データの質を疑う

Accuracyグラフの見方

- Accuracyは1（100%）に近ければ近いほど良い
- 両方とも右肩上がりがベスト



1. データ拡張

ロバストなモデル構築のために

理想的なデータとは

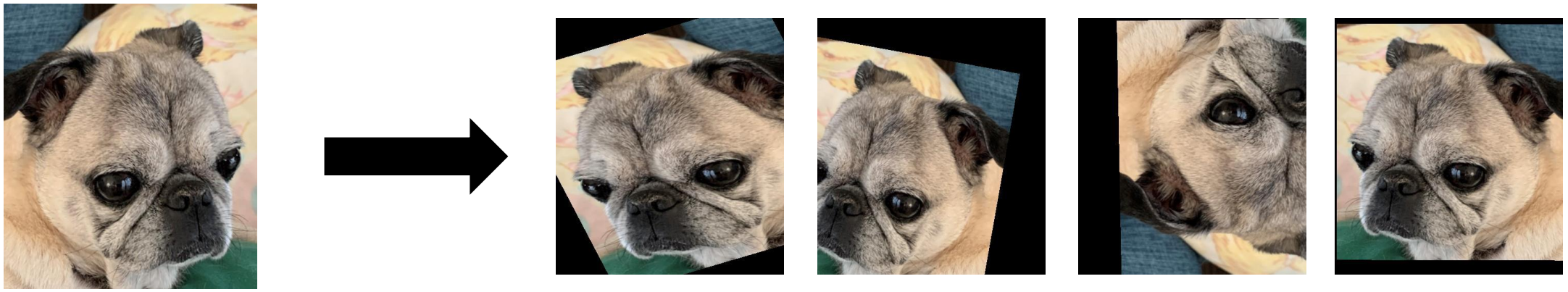
- 機械学習はとにかくデータの質と量が大事
- 大原則　Garbage In, Garbage Out　「ゴミを入れるとゴミが出る」
- ロバストなモデルには多様なデータの学習が不可欠
  ロバスト：未知のデータに対しても頑健で精度が高いこと
- できるだけ学習データのパターンを増やしたい...

【データの増やし方】

- 素直に新しいデータを収集する
- 手元のデータを加工して疑似的にデータを増やす（**データ拡張**）

データ拡張

- Data Augmentation
- データを加工して疑似的にデータを増やすこと
- 画像認識分野では特によく用いられる
- 学習データに対してランダムに適用することが多い

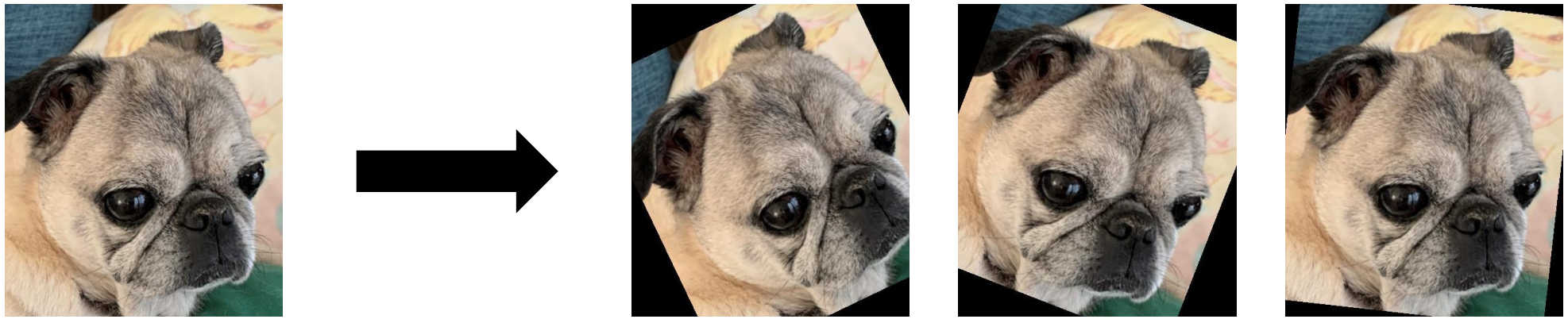


反転

- ランダムに上下反転
- ランダムに左右反転

回転

- この例は-30度から+30度までランダムに回転



拡大・縮小

- この例は元画像サイズの-20%から+20%までランダムに拡大・縮小
- 縦横それぞれに対して拡大・縮小を行っている

移動

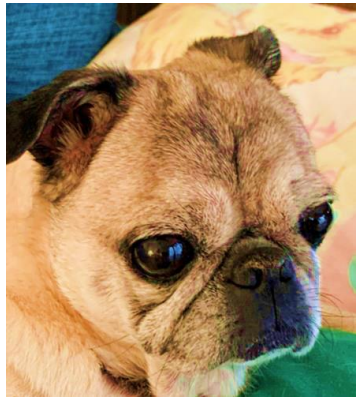
- この例は元画像サイズの-20%から+20%までランダムに移動
- 縦横それぞれに対して移動を行っている

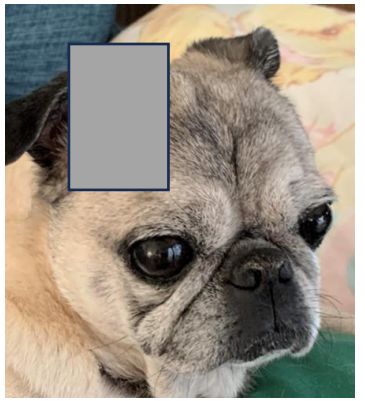
せん断（シアー変換）

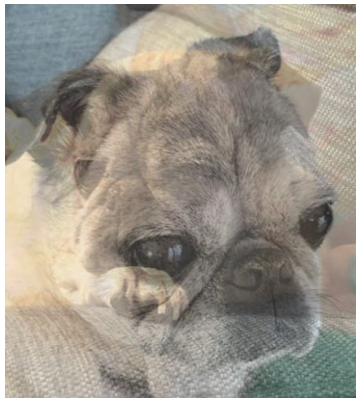
- この例は-10度から+10度までランダムにせん断

その他データ拡張の手法

- 色変換
- 明るさ・コントラスト変換
- Cutout
- Mixup
- ...

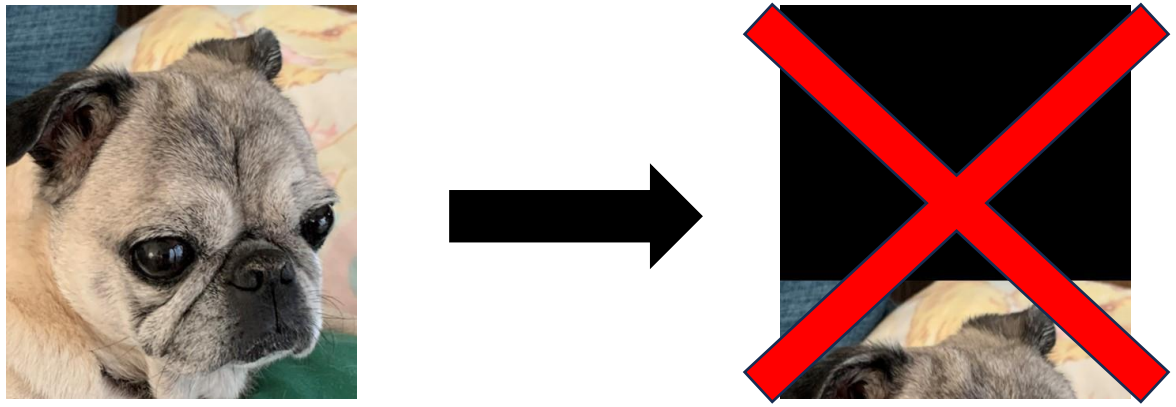






データ拡張の注意点

- 不必要なデータ拡張は避ける
- 目的に応じて適度なデータ拡張を心がける
  ＝何事もやりすぎは良くない



大事な特徴である
顔が見えない！

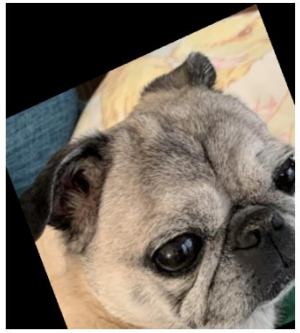
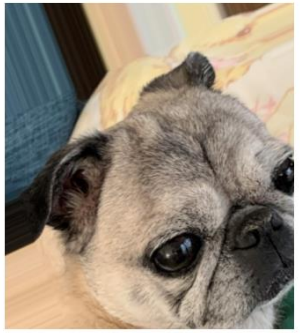
データ拡張（ジェネレータ）の実装

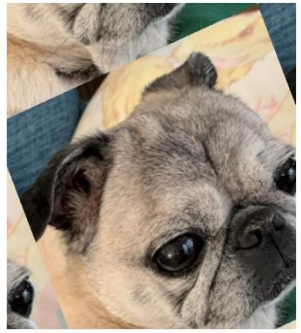
```
datagen = ImageDataGenerator(
    horizontal_flip=True,        # 左右反転
    vertical_flip=True,          # 上下反転
    rotation_range=30,           # 回転（度）
    zoom_range=0.2,              # 拡大・縮小
    width_shift_range=0.2,       # 横方向の移動
    height_shift_range=0.2,      # 縦方向の移動
    shear_range=10,              # せん断（度）
    fill_mode="constant",        # 画素補完の方法
    rescale=1.0/255.0,           # 画素値の定数倍に用いる定数
)
```

fill_mode

- 加工後に空白となる画素を埋める方法を指定できる

"constant"	特定の画素値で埋める（cval=画素値で指定可能）
"nearest"	境界付近の最も近い画素値で埋める
"reflect"	境界付近で折り返した画素値で埋める
"wrap"	境界付近で繰り返す画素値で埋める



rescale

- 画素値を定数倍する
- rescaleに **1.0/255.0** を指定することで正規化と同等の処理ができる

2. ファインチューニング

高精度なモデルを少量データ・短時間で

ファインチューニング

- 学習済みモデルを目的に合わせて再学習すること
- ファインチューニングによって効率的にモデルを構築できる

具体的なイメージ

- 未学習CNNのパラメータ初期値はデタラメな値
- 初期値がすでに学習された値であれば効率良く学習できそう...
- （他の仕事で得た経験を今回の仕事に活かせそう...）

ファインチューニングと転移学習の違い

ファインチューニング

出力層（分類器）以外のパラメータ（特徴抽出器）も含めて再学習する

転移学習

出力層（分類器）のみを再学習する

ファインチューニング実装に向けて

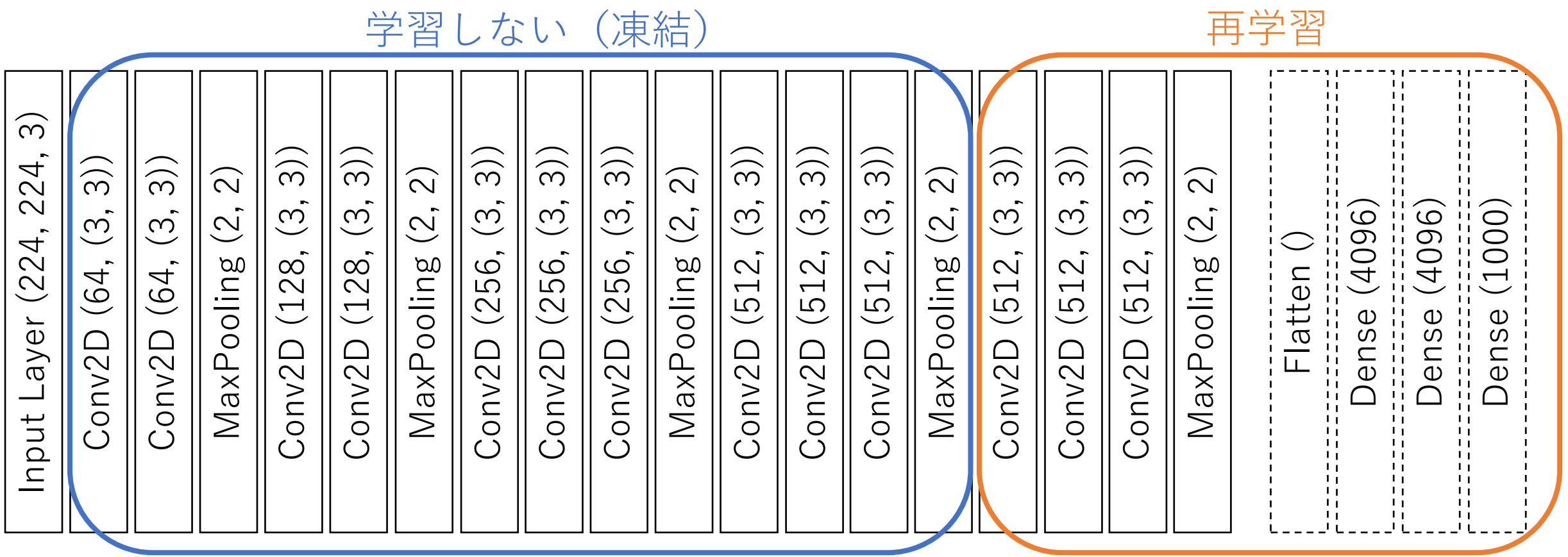
- 今回は馬とシマウマを分類するモデルの作成を目標とする
- データはCaltech 256 Image Datasetを用いる
- 学習済みモデルは**imagenet**を事前学習済みの**VGG16**を用いる
  imagenet: 計1400万枚以上の大規模画像データセット

Caltech 256 Image Dataset (https://www.kaggle.com/datasets/jessicali9530/caltech256)

- カテゴリ数257種、計30,607枚の画像データセット

VGG16の構造とファインチューニング

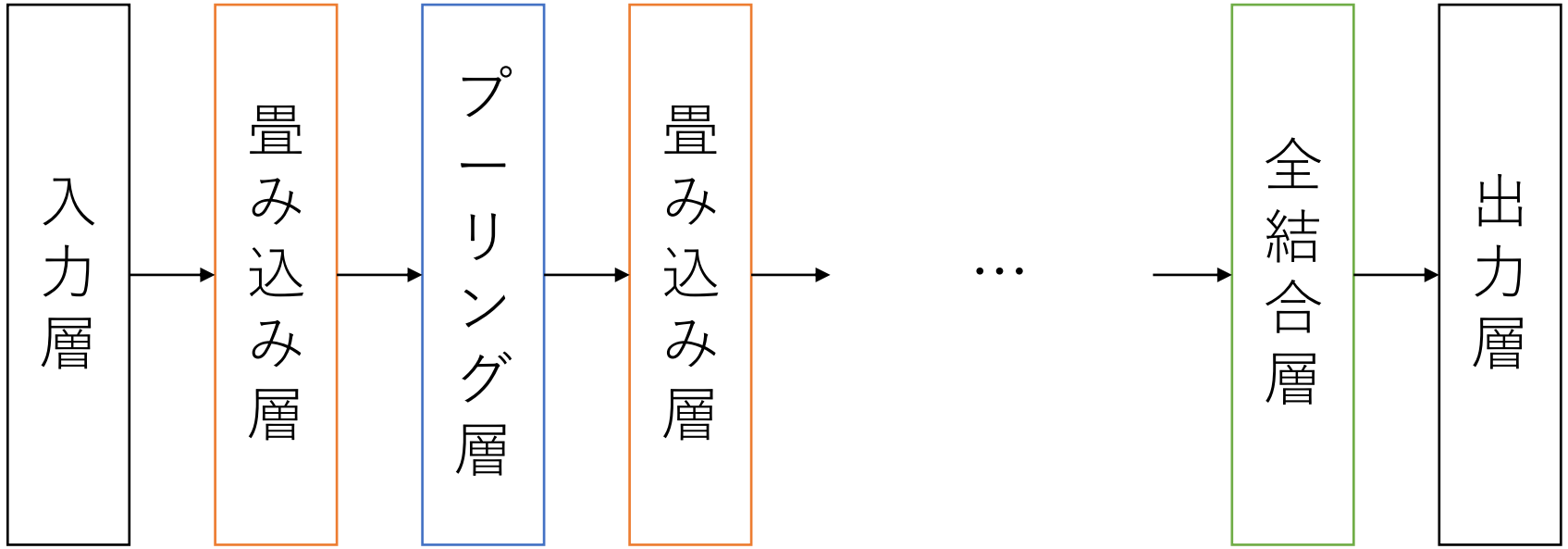
今回は、局所的特徴はそのままに、大局的特徴と出力のみを再学習する



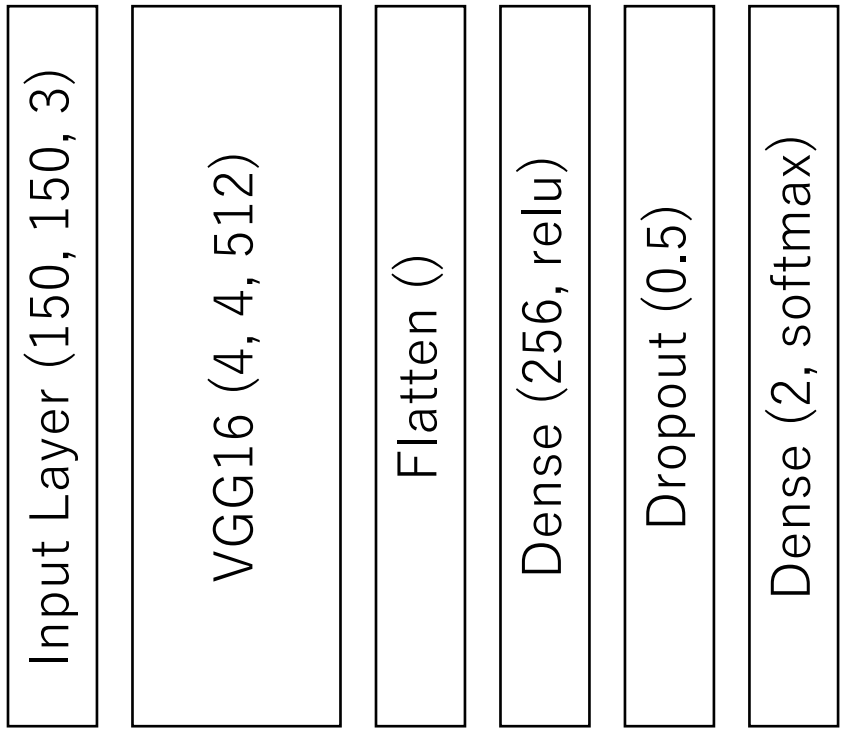
（補足）CNNの特徴

局所的特徴：線分や色相など
大局的特徴：シルエットなど

- 畳み込み層とプーリング層を何度か繰り返して全結合層につながる
- プーリング層により特徴の位置ずれを吸収
- 浅い層では局所的な、深い層では大局的な特徴量を抽出する



今回のモデル構造


Input Layer (150, 150, 3)
VGG16 (4, 4, 512)
Flatten ()
Dense (256, relu)
Dropout (0.5)
Dense (2, softmax)

データの準備

- Caltech 256 Image Datasetをダウンロードする
- データセットの中の馬とシマウマの画像をそれぞれ40枚ずつ選択する
- 40枚を学習用20枚とテスト用20枚に分ける
- GoogleDrive上にデータを適切な場所にアップロードする

※詳しくは後ほど

ジェネレータの定義

```
train_datagen = ImageDataGenerator(
        rescale = 1.0 / 255,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True)
test_datagen = ImageDataGenerator(rescale = 1.0 / 255)
```

- 学習用とテスト用でそれぞれジェネレータを定義する
- テスト用にデータ拡張は不要なため正規化のみ設定する

データの読み込み

```
train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size = (img_height, img_width),
        classes = classes,
        batch_size = 32,
        class_mode = 'categorical')
```

- train_data_dirから画像とラベルを自動的に読み込む
- テスト用も同様

学習済みVGG16の読み込み

```
vgg_model = VGG16(
        include_top = False,
        weights = 'imagenet',
        input_shape = (img_height, img_width, 3))
```

- include_topで出力層（Flatten以降）を含むか否かを選択できる
- weightsで事前学習済みモデルを指定できる
  Noneのときは事前学習済みモデルを読み込まずにパラメータはランダム初期化
- input_shapeで入力サイズを指定できる

モデルの構築

```
model = Sequential()
model.add(vgg_model)
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes, activation='softmax'))
```

- VGG16の後に出力層の部分を繋ぐ

層の凍結

```
for layer in vgg_model.layers[:15]:
    layer.trainable = False
```

- VGG16の15層までは再学習しないようにする

モデルの保存（コールバック）

```
mc_cb = ModelCheckpoint(
        filepath = 'drive/MyDrive/finetuning.h5',
        monitor = 'val_loss',
        verbose = 0,
        save_best_only = True)
```

- ModelCheckpointで学習中のモデル保存に関わる設定ができる
- monitorで監視する評価指標を指定できる
- save_best_only=Trueの場合、監視している値が最小のときだけ保存

学習

```
history = model.fit(
    train_generator,
    epochs=epoch,
    validation_data = test_generator,
    callbacks = [mc_cb])
```

- callbacksにコールバックのリストを渡す

モデルの読み込みと予測

```
model = load_model('drive/MyDrive/finetuning.h5')

pred = model.predict(x)[0]
```

- keras.models.load_modelで保存したモデルを読み込む
- predict(x)で入力に対する出力（各カテゴリに対する確率）が得られる
  xは入力画像テンソル　今回は(1, 150, 150, 3)のテンソル

演習

- データ拡張を実装する
- ファインチューニングを実装する
- 作成したモデルによる画像の分類を行う

github.com/wt501/sample_programs/image_augmentaion.ipynb
github.com/wt501/sample_programs/fine-tuning.ipynb
github.com/wt501/sample_programs/classify.ipynb

まとめ

- データ拡張について学んだ
- VGG16を用いたファインチューニングを行った

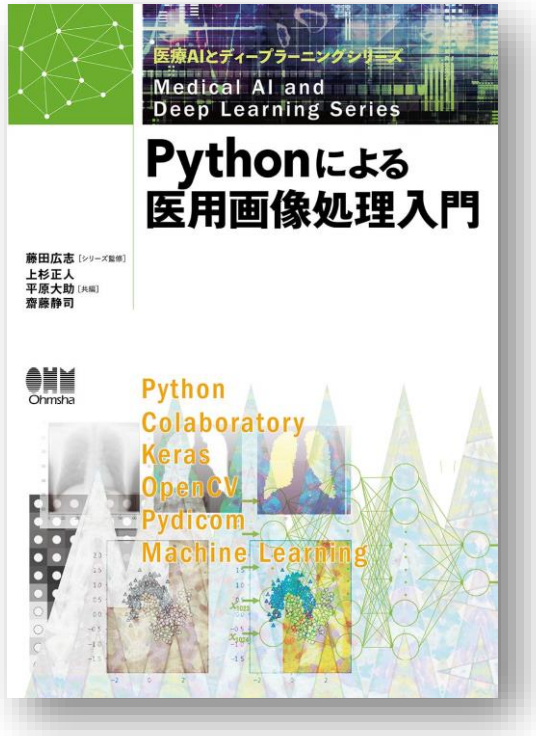
【次回】
- Pythonで医用画像の画像処理を行う

参考書について

今回まで

www.amazon.co.jp/dp/4910558012

次回以降



www.amazon.co.jp/dp/4274225461