2021 / 06 / 01

深層学習を用いた画像認識の基礎の基礎

小林 里瑳

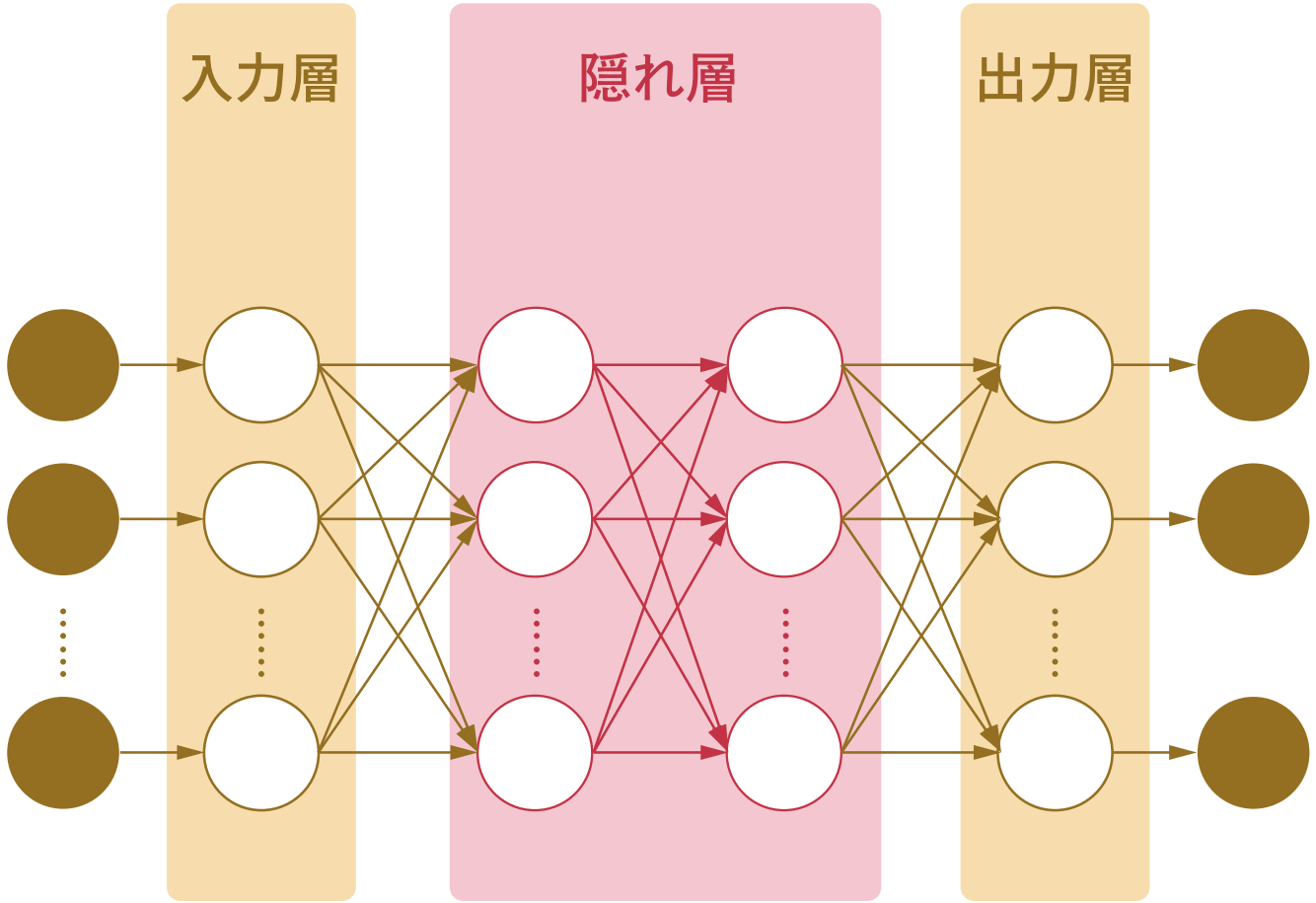
東京大学大学院工学系研究科
社会基盤学専攻　博士課程

目次

ニューラルネットワーク

機械学習モデル

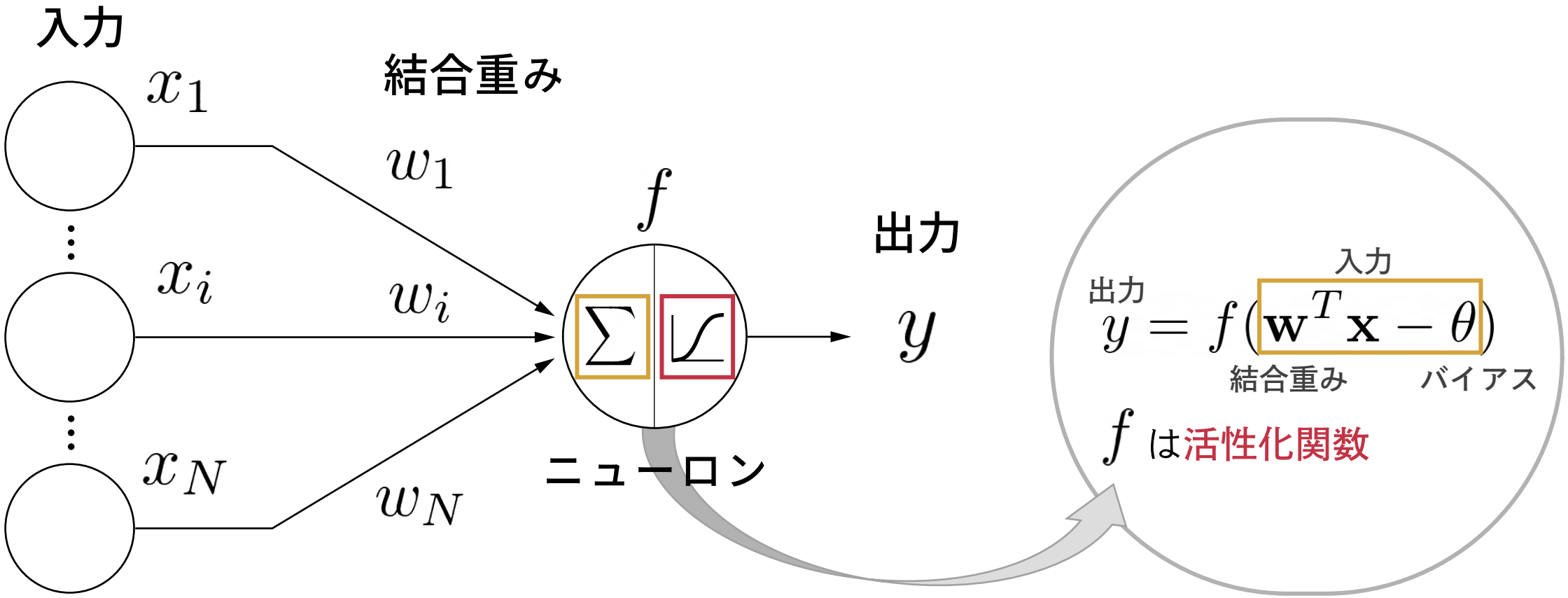
ニューラルネットワークとは



- 機械学習手法の一つ
  - 教師あり学習のモデルとして有名
- **ニューロン**と**シナプス**から成る人間の神経回路網を模した数学モデル
- 今日取り扱うCNN はニューラルネットワークの内の一つ

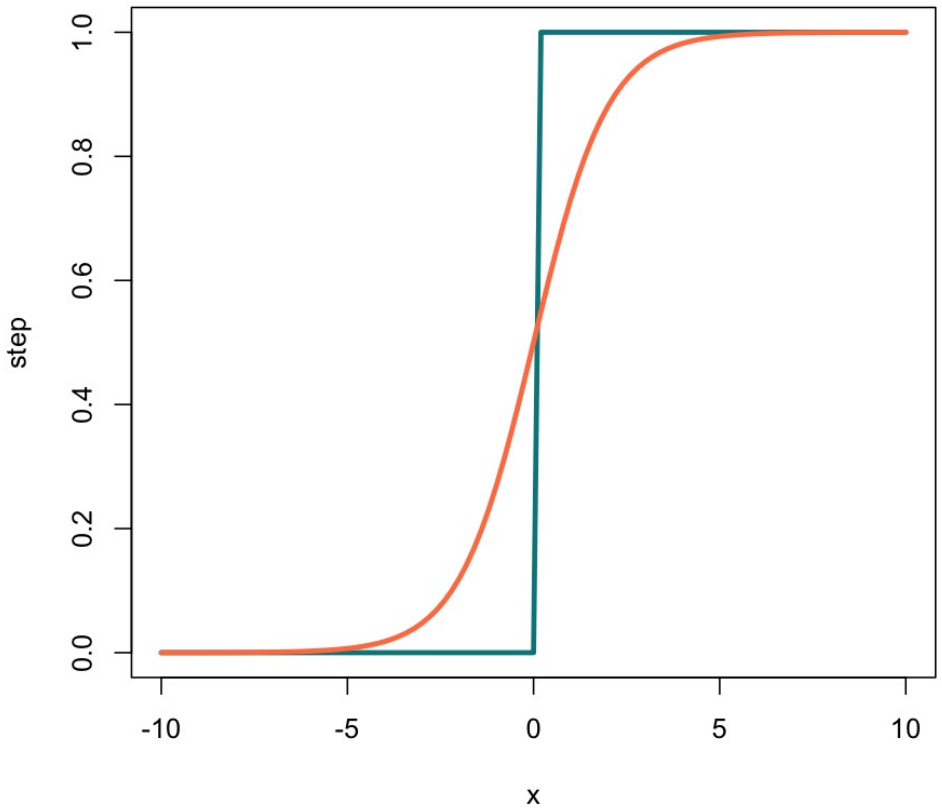
ニューロンモデル

- 入力値に対し重みを乗じて線形結合
- 活性化関数によって変換し，出力を与える



活性化関数

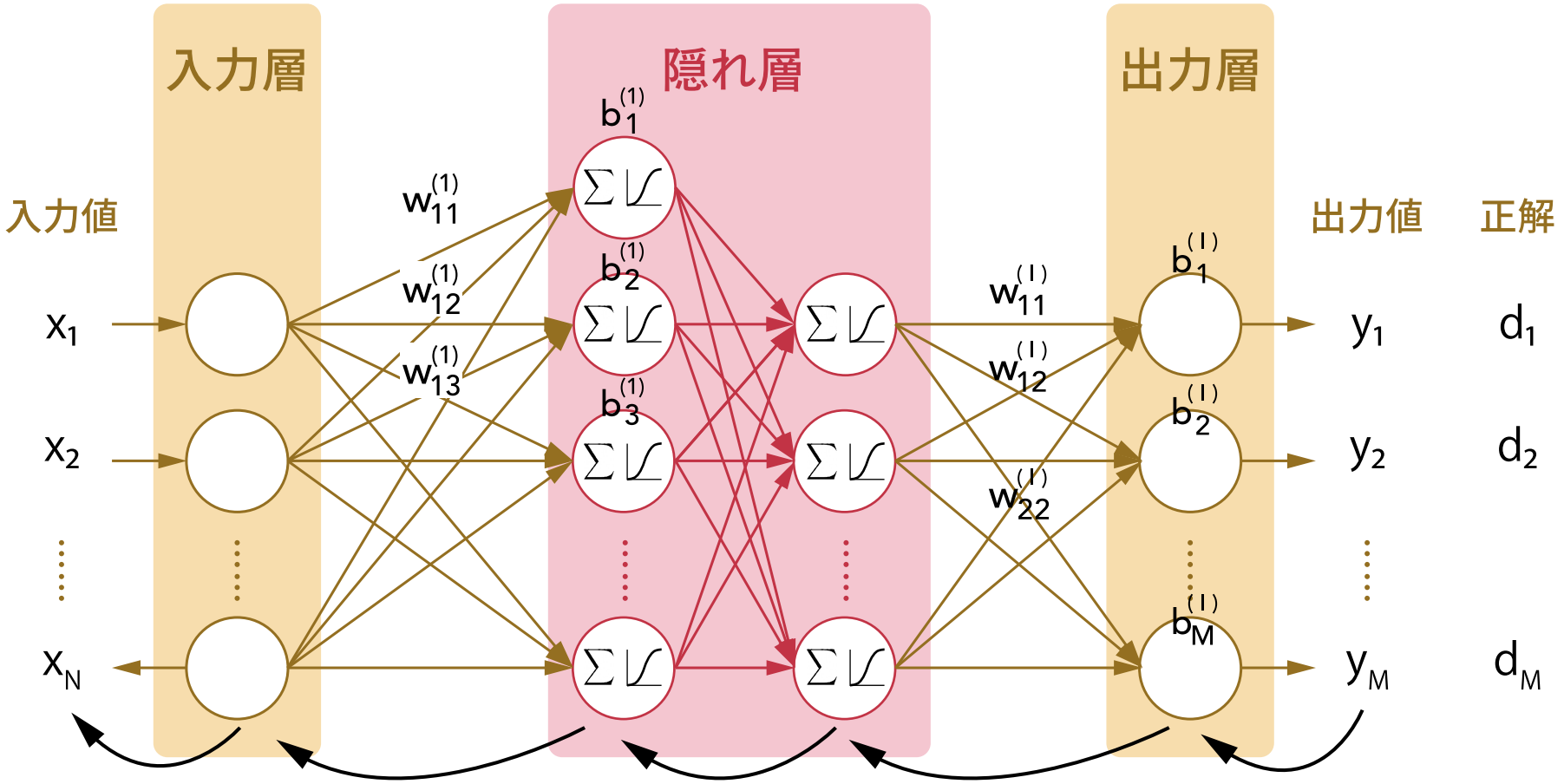
- 内部状態に応じて出力を与える：信号に対して**発火**するかどうか？ を決める関数



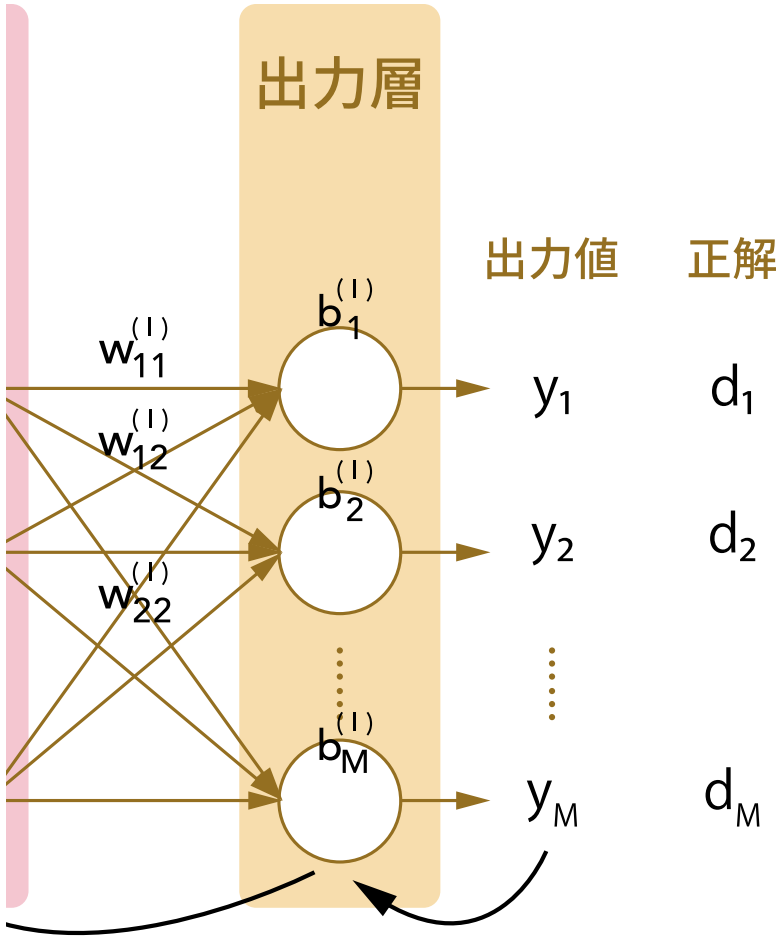
- ステップ関数（図青線）
  - 形式ニューロンで用いられてきた
  - 内部状態が正なら発火する
  - $h = 0$のとき微分不可能
- シグモイド関数（図橙線）
  - ニューラルネットワークで用いられている
  - ステップ関数に比べて連続かつ微分可能
- ソフトマックス関数
  - 多項ロジットモデルと式形が同じ

誤差逆伝播法

- モデル出力$\boldsymbol{y}$と訓練データ$\boldsymbol{d}$の誤差を最小化するように結合重み行列$\boldsymbol{W}$ を調整する



誤差逆伝播法



正解と出力の二乗誤差の総和Eを最小化する

$$E = \frac{1}{2}\sum_{m=1}^{M} \|\boldsymbol{d}_m - \boldsymbol{y}_m\|_2^2 \ (m = 1,2,\ldots,M)$$

M は訓練データの数

全ての結合重み$w_{pq}^{(i)}$の更新式

$$w_{pq}^{(i)} \leftarrow w_{pq}^{(i)} \eta \sum_{m=1}^{M} \frac{\partial E_m}{\partial w_{pq,m}}$$

$\eta > 0$ は学習率（更新量の大きさを制御する要素）

誤差逆伝播法

1. 結合重み行列$W^{(0)}$をランダムに設定
2. パラメータを入力し，出力誤差Eを計算
3. 出力誤差E が許容誤差を下回るまで3-1~2を繰り返す

3-1. 逆伝播：各結合重みに関する勾配を全て求める

$\delta_q^{(i)}$：第i層におけるノードq に関する局所勾配　　$b_q^{(i)}$：第i層におけるノードqの内部状態

$$\delta_q^{(i)} = \frac{\partial E}{\partial b_q^{(i)}}, \quad \delta_q^{(I)} = (y_q - d_q) f'\left(b_q^{(I)}\right), \quad \delta_q^{(i)} = \sum_{p=1}^{M^{(i+1)}} \delta_p^{(i+1)} w_{pq}^{(i)} f'(b_q^{(i)}), \quad \frac{\partial E}{\partial w_{pq}^{(i-1)}} = \delta_q^{(i)} x_p^{(i-1)}$$

3-2. 結合重みの更新

$$w_{pq}^{(i)} \leftarrow w_{pq}^{(i)} \eta \sum_{m=1}^{M} \frac{\partial E_m}{\partial w_{pq,m}}$$

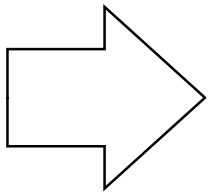
CNN –畳み込みニューラルネットワーク

モチベーション

- 多次元の訓練データの場合，パラメータ間で時間/空間的依存関係にある
  - 例：画像データにおいてあるピクセルの”縦横の情報” を考慮した学習

画像データをピクセルの行列で表現

1	3	9
4	7	0
5	1	4



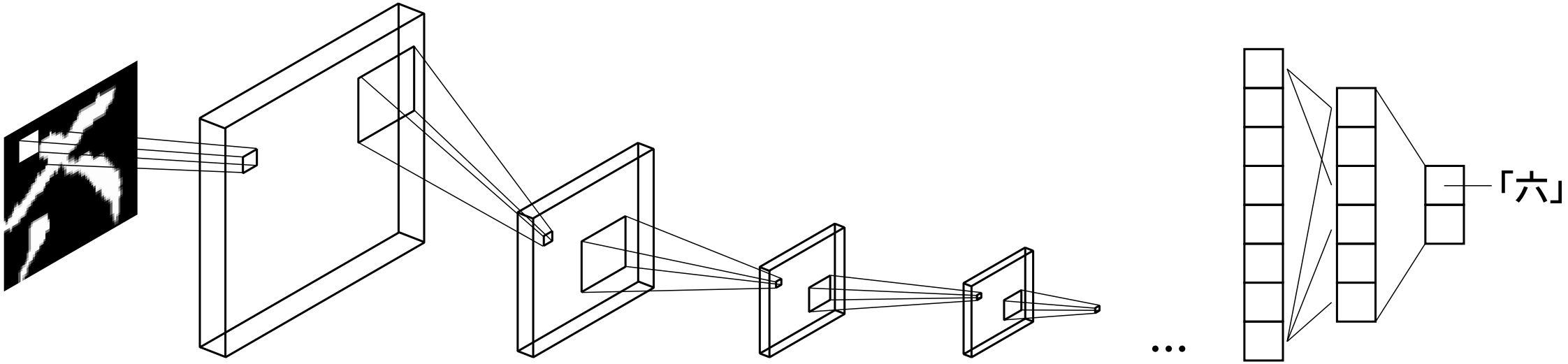
1
3
9
4
7
0
5
1
4

画像の空間依存性が失われている

全体像

- 畳み込み＋プーリングを何回か繰り返してパラメータを抽出する

隠れ層：特徴を学習　　**全結合層：分類**



INPUT　畳み込み convolution　プーリング pooling　畳み込み convolution　プーリング pooling　…　Flatten

局所的な特徴を抽出

空間的にぼかす

畳み込み

フィルタをスライドして

フィルタ

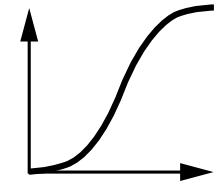
1	0	1
0	1	0
1	0	1

フィルタも
学習の対象

0	0	0	0	0
0	0	1	1	0
1	1	1	0	0
0	1	0	1	0
1	0	0	1	0

畳み込みの適用間隔を”ストライド”といい，
ハイパーパラメータとして与える

活性化関数を適用する
場合もある



特徴マップ

2	2	2
2	4	1
4	2	2

を作成する

畳み込み

- 入力層の一部分を切り取って，フィルタ（カーネルともいう畳み込み演算）をかけて，畳み込み層の1ニューロンに対応するようにする
- 局所的な特徴抽出器の役割を果たす

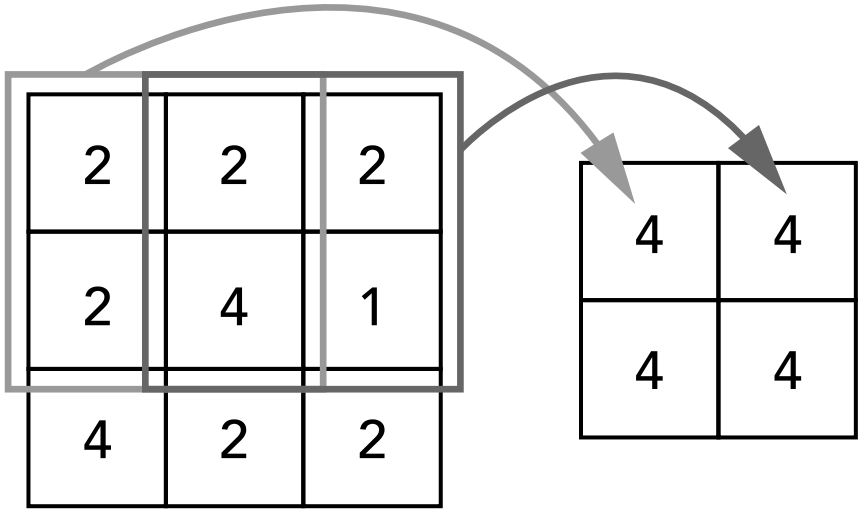
$$\mathrm{J}(\mathrm{u})=\sum_w f(\mathbf{u})I(\mathbf{u}+\mathbf{u}_0)d\mathbf{u}$$

I：入力画像
J：出力画像
u：幅w ピクセルの畳み込み窓における座標
u_0：入力画像における座標の畳み込み演算の中心位置座標

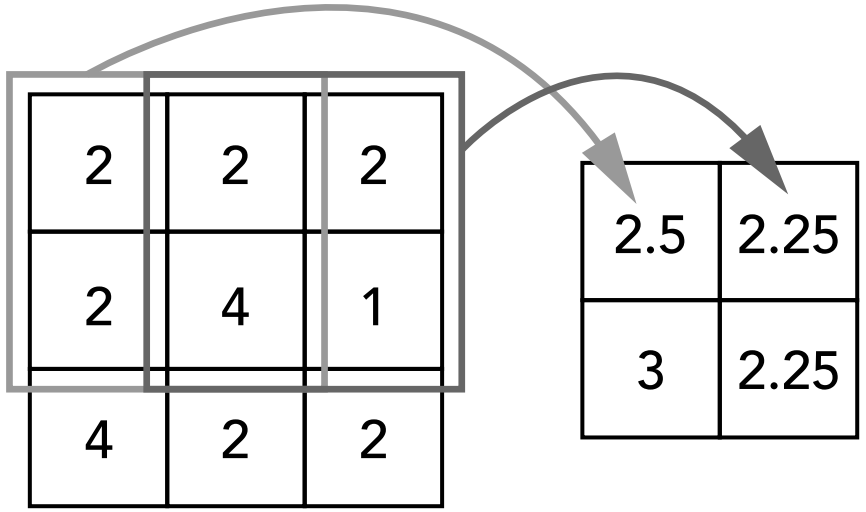
- フィルタごとに内積を計算し，活性化関数を適用する

プーリング

- 畳み込み層によって出力された特徴量を次元圧縮
- 領域内の最大値もしくは平均値を出力
- 平行移動などに関するロバスト性に関係する



Max プーリング



Average プーリング

CNN による画像分類問題の例

モチベーション

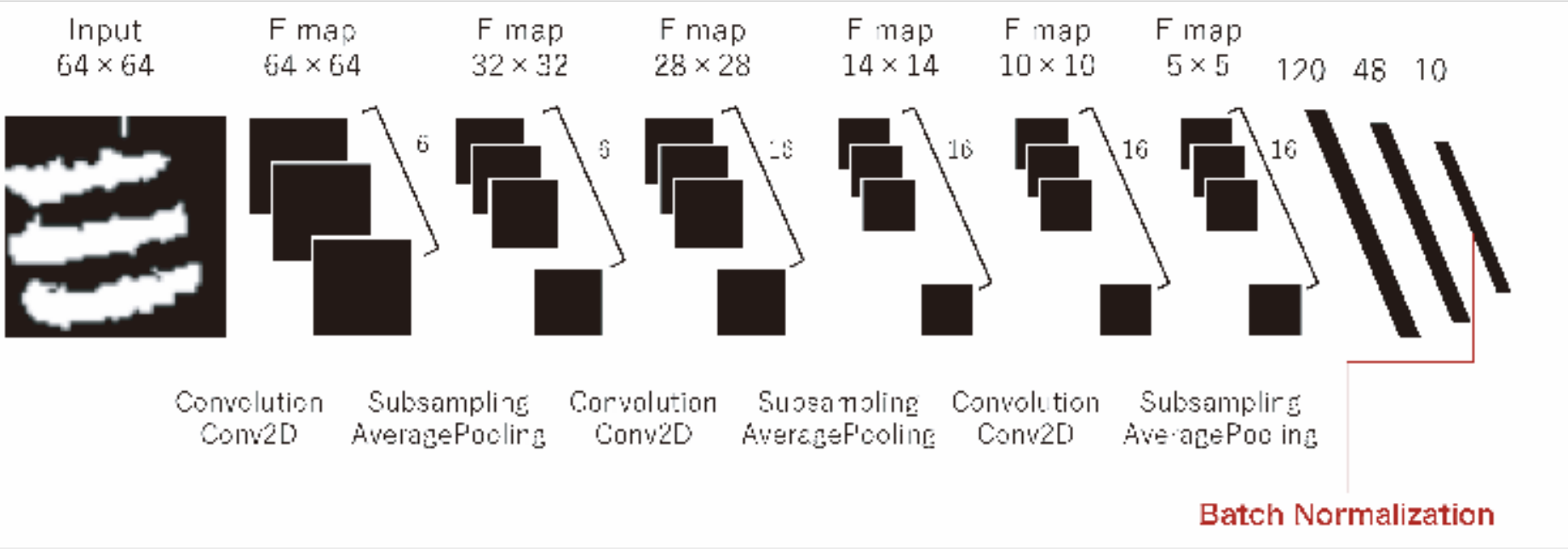
- 史料デジタイズ需要の高まり
  - 史料をインプットデータとして使用することが増加(Paleoclimatology, Cliometrics)
  - 今回は，筆文字で記された史料からモデル実証分析に資するデータを生成できるように**画像認識**のタスクに取り組む
- 既往技術と課題
  - 訓練データ(KMINST)のような鮮明で整形された画像データを得られないことがほとんど
  - 日本語文字認識において一般的な課題：文字の出現回数の偏り
  - →環境に左右された画像を学習分類する必要
  - →訓練データの分布の偏りに頑健な手法が望ましい。

手法

- CNN based feature-extraction による転移学習

  特徴を学習するCNNアーキテクチャを用いて学習を行なったモデル情報（構造を重み）を転移して新たなモデルを構築する転移学習を採用

  今回は訓練データと検証データの観測分布の差異はあるが同じ分類を用いる場合を仮定したため，feature-extractionを採用



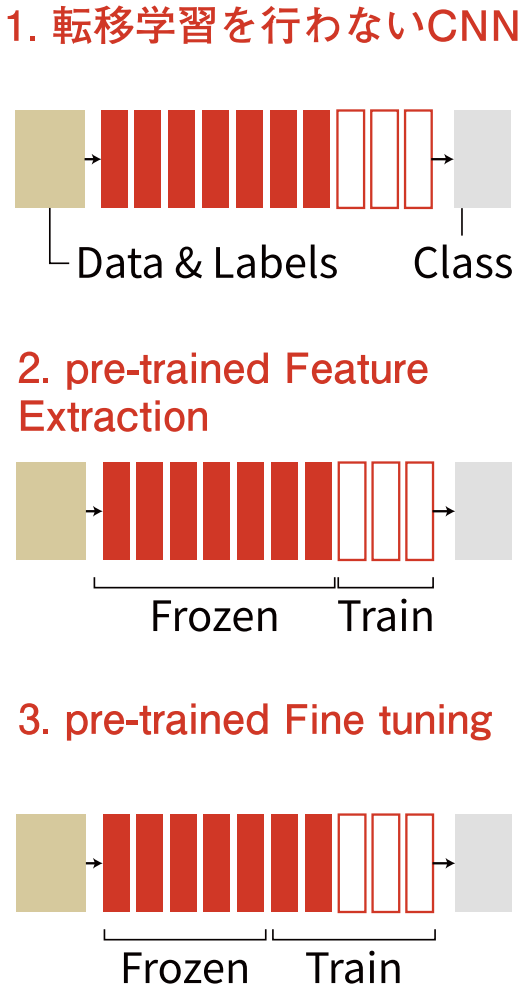
実験

- 訓練データ
  - くずし字データベースで配布されているKMNISTデータセットを使用「〇」のみ土地台帳よりスクレイピング
  - データは 64*64 グレースケール（1チャンネル）


KMNIST データ

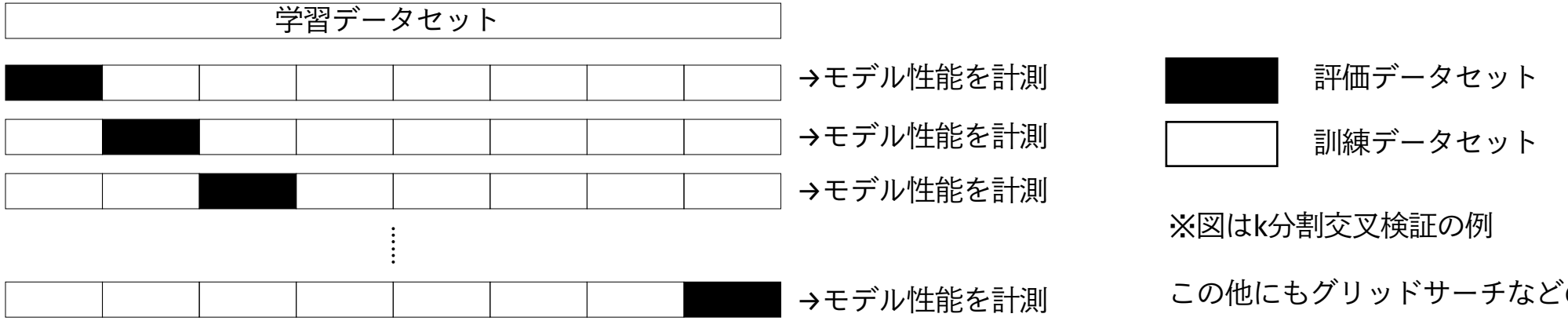

土地台帳データ

- 学習方法
  - 1. 通常のCNN
  - 2. 1で学習したCNNモデルの全結合層のみ学習
  - 3. 1で学習したCNNモデルの畳み込み層と全結合層を学習



補足：モデルの評価

- ホールドアウト法
  - モデルを作る訓練(train)データ，モデルを評価する評価(val)データ，汎化性能を確かめるテスト(test)データに分割して評価する方法
    - 今回は訓練データと評価データのデータセットはくずし字，テストデータは土地台帳とすることで，「くずし字データベースを使った学習モデルは土地台帳の文字認識に使えるのか？」ということを評価
- クロスバリデーション
  - データセットを分割→学習→性能評価　を複数回繰り返し平均を取る方法

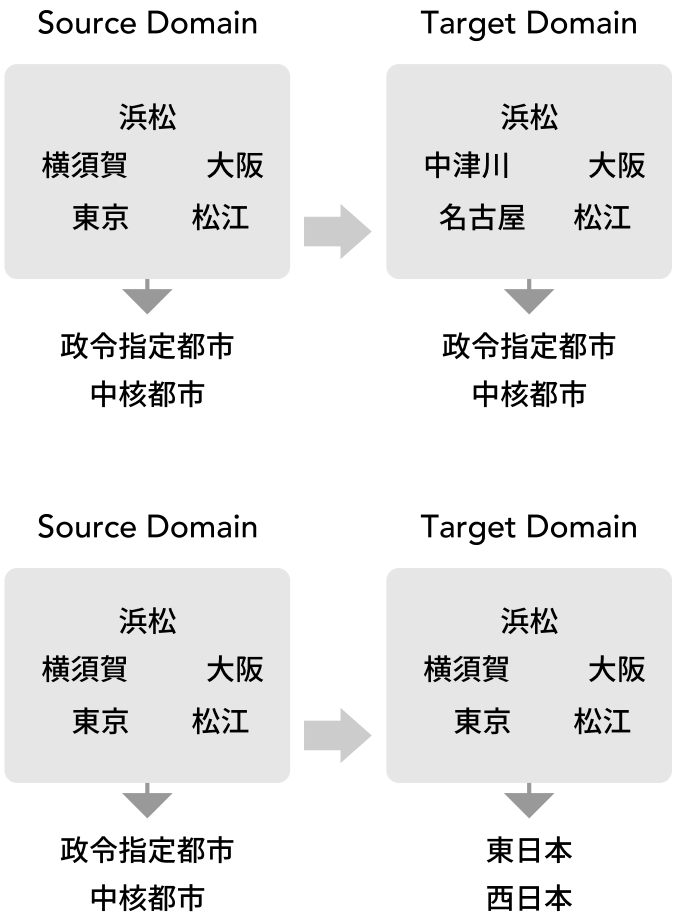


※図はk分割交叉検証の例

この他にもグリッドサーチなどの方法がある

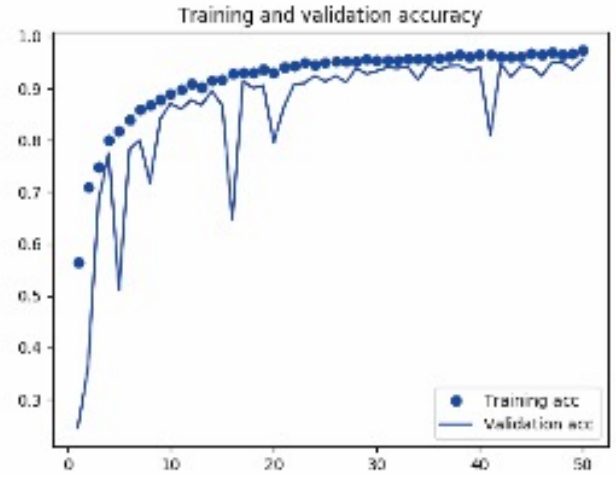
補足：転移学習

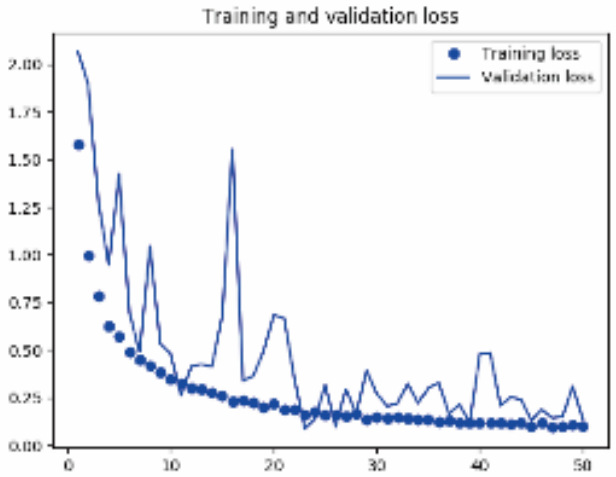
- ある領域(Source Domain)で学習したモデルを別の領域(Target Domain)に適用させる学習技術
  - Source DomainとTarget Domainを比較して，何が違うかによってタスクが異なる
    - Source DomainとTarget Domainの両方にラベルがある inductive transfer learning の例
      - 観測データの定義域：Cross-xxx Adaptation
      - 観測データの分布：Domain Adaptation　(上図)
      - ラベルの定義域：Fine Tuning　(下図)
      - ラベルの分布



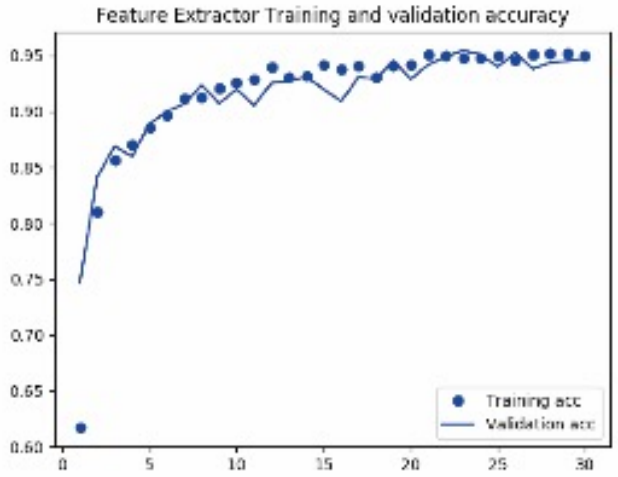
学習結果

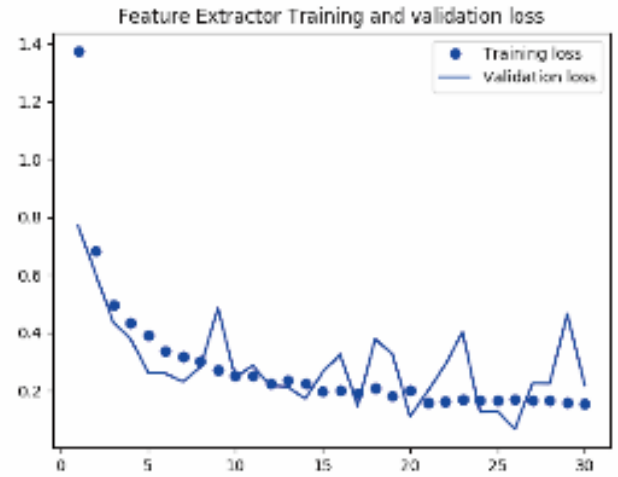
転移学習を行わないCNN



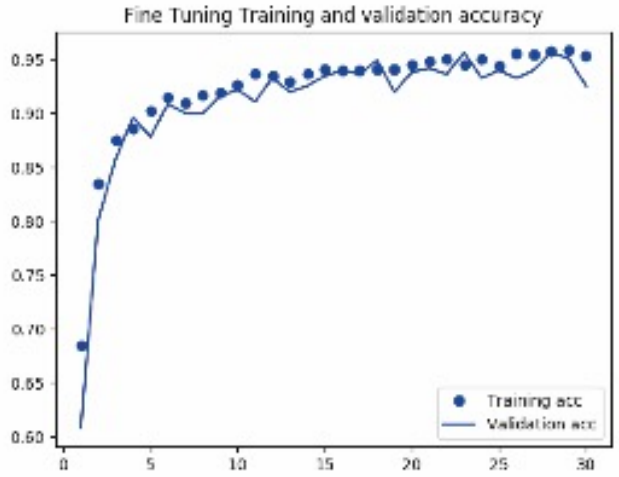


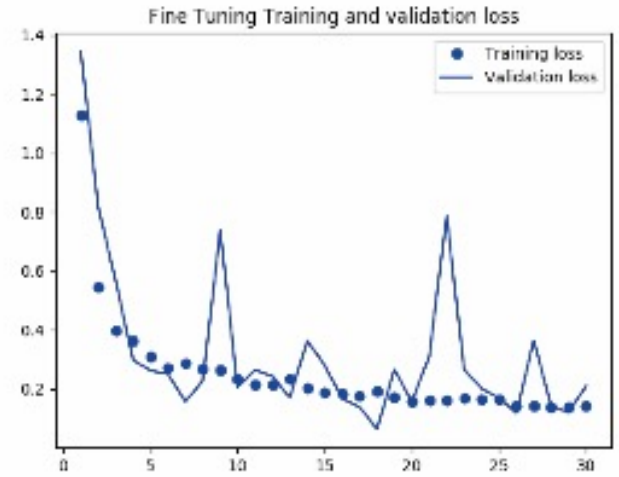
Pre-trained CNN and Feature Extraction





Pre-trained CNN and Fine tuning





Python パッケージを使ったCNN の実装

大まかな手順

機械学習用データセットの作成

- 画像データをベクトル化する

学習

- Pythonパッケージ によってCNN を構築（今回はkeras を使いますが…）

テスト画像を用いて汎化性能を検証

機械学習用データセットの作成（1/2）

- ディレクトリの構造

```
root /
  ├ 00_Make_npz.py              学習用データセットの作成
  ├ 01_Feature.py               学習コード：特徴抽出
  ├ 01_Mizumashi.py             学習コード：画像の水増しのみ
  ├ 02_Predict.py               TestData を使って学習モデルを検証
  ├ gray64_dataset/
  │   └ TestData.npz            検証データセット
  │   └ TrainData.npz 訓練データセット
  └ images/
      └ test/                   検証データ（台帳からスクレイピング）
      └ train/          訓練データ（くずし字データベースから）
```

機械学習用データセットの作成（2/2）

コード：学習データセットを保存

```
import os
import sys
import numpy as np
import cv2

paths = ['./train/' + i for i in p]
label = np.zeros(len(paths)-1)
for cnt_path, path in enumerate(paths):
    files = [filename for filename in os.listdir(path) if not filename.startswith('.')]
    for cnt_file, f in enumerate(files):
        file_path = path + '/' + str(f)
        img1 = cv2.imread(file_path)  # 画像の読み込み
        img = cv2.resize(img1, dsize=(224, 224))  # リサイズ,ここでパラメータ数を決定
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # グレイスケール変換，RGB はグレイスケールのパラメータ数の3倍になる
        size = img.shape  # (height, weight, channel)
        img = np.ravel(img)  # 画像のベクトル化(np.ravelは多次元配列を1次元配列に変える関数)
        img = (img - np.min(img)) / (np.max(img) - np.min(img))  # ベクトルを正規化
        if cnt_path == 0 and cnt_file == 0:
            train = img
            labels = np.insert(label, cnt_path, 1)
        else:
            train = np.vstack((train, img))  # trainにベクトル化した画像データをスタックしていく
            labels = np.vstack((labels, np.insert(label, cnt_path, 1)))  # 学習ラベル付け
np.savez('./TrainData.npz', trainimg=train, trainlabel=labels, imgsize=size)  # npz形式ファイルで保存
```

学習（1/4）

コード：検証のため学習データと評価データを8:2で分ける

```
data = np.load("TrainData.npz")

# cross-validation
l = list(zip(data["trainimg"],data["trainlabel"])) # zip関数を使って配列trainimg とtrainlabel の対応を崩さずにシャッフル
np.random.shuffle(l)
shuffled_trainimg, shuffled_trainlabel = zip(*l) # 出力がtuple になってるので注意

tr_num = math.floor(len(shuffled_trainimg)*0.8) # 訓練データ:検証データを0.8:0.2 で分ける

trainimg = np.asarray(shuffled_trainimg[:tr_num])
trainlabel = np.asarray(shuffled_trainlabel[:tr_num])
valimg = np.asarray(shuffled_trainimg[tr_num:])
vallabel = np.asarray(shuffled_trainlabel[tr_num:])
```

学習（2/4）

コード：モデルの構築

```
# モデルの構築
from keras.models import Sequential, Model, model_from_json
from keras.layers import Dense, Flatten, BatchNormalization
from keras.layers import GlobalAveragePooling2D,Input
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.resnet50 import ResNet50

# 全結合層の構築(CNNと同じ)
model = Sequential()  # モデルを生成するためのモジュール

model.add(Conv2D(6, kernel_size=(10, 10), activation="relu", padding='same', input_shape=(64,64,1)))  #
32*32の畳み込み層activation:活性化関数, input_shape:入力画像のサイズ
model.add(AveragePooling2D((2, 2), strides=(2, 2)))  # プーリング層
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), padding='valid', activation='tanh'))
model.add(AveragePooling2D((2, 2), strides=(2, 2)))
model.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), padding='valid', activation='tanh'))
model.add(AveragePooling2D((2, 2), strides=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(48, activation='relu'))
model.add(Dense(10, activation='sigmoid'))  # 分類数=出力素子数
model.summary()
```

学習（3/4）

コード：モデルのコンパイル～画像の水増し～学習

```
# モデルのコンパイル
from keras.optimizers import SGD
model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
              loss='categorical_crossentropy',
              metrics=['acc’])

# 画像の水増し
datagen = ImageDataGenerator(
    rotation_range=45,  # 回転
    shear_range=0.3, #せん断
    zoom_range=0.4, #拡大
    horizontal_flip=True #水平反転)

trainimg = tf.reshape(trainimg, [trainimg.shape[0] 64, 64, 3])  # 一次元の配列データをリシェイプ
valimg = tf.reshape(valimg, [trainimg.shape[0], 64, 64, 3])  # 一次元の配列データをリシェイプ

history = model.fit_generator( # ここでモデルを学習している
            datagen.flow(trainimg, trainlabel, batch_size=170),
            epochs=50,
            validation_data=datagen.flow(valimg, vallabel, batch_size=170)
        )
```

学習（4/4）

コード：モデルの保存と学習結果の検証

```
model_json_str = model.to_json()
open("CNNonly.json", "w").write(model_json_str)

model.save('CNNonly.h5’)

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Case2: Use ResNet50 as Feature Extractor Training and validation accuracy')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Case2: Use ResNet50 as Feature Extractor Training and validation loss')
plt.legend()

plt.show()
```

テスト画像を使って汎化性能を検証

コード：学習結果の検証

```
import numpy as np
from keras.models import load_model
import cv2

data = np.load(“gray64_dataset/TestData.npz”) # テストデータの読み込み
model = load_model(‘FeatureEx_.h5’) # 保存したモデルの読み込み

testimg = data["testimg"]
testlabel = data["testlabel"]

testimg = np.reshape(testimg, (testimg.shape[0], 64, 64, 1))  # 一次元の配列データをリシェイプ

features = model.predict(testimg)

labels_predict = np.argmax(features, axis=1)
labels_test = np.argmax(testlabel, axis=1)

miss_indexes = np.where(labels_predict != labels_test)[0]
print((1-len(miss_indexes)/len(testlabel))*100)  #正解率
```

課題：汎化性能の向上と計算時間の短縮

- 構築するCNN モデルを変える
  - 畳み込み層の数，カーネルの大きさ，活性化関数 etc…
- 画像を水増しする
  - テスト画像を学習データに混ぜるのはズルなのでやめましょう
- 転移学習で他のモデルパラメータを使ってみる
- CNN モデル以外で学習する
  - RNN（Recurrent Neural Network）など
  - EfficientNetV2や Vision Transformer という最新のモデルも提案されています
- 他のパッケージを使ってみる
  - PyTorch もよく使われています