Supporting information

ディープラーニング等高線 HPLC 法を用いた
食用きのこ識別に関する研究

北尾修平 [a)]、森山祐羽 [a)]、高山卓大 [a)]、井之上浩一 [a)*]

a) 立命館大学薬学部

**Corresponding author at: Laboratory of Clinical & Analytical Chemistry, College of Pharmaceutical Sciences, Ritsumeikan University, 1-1-1 Nojihigashi, Kusatsu, Shiga 525-8577, Japan*
E-mail address: kinoue@fc.ritsumei.ac.jp (K. Inoue)

きのこ試料の情報

きのこ種	記号	産地	メーカー
ヒラタケ	A	長野県	ホクト株式会社
クサウラベニタケ	B		
ニガクリタケ	C		
シイタケ	D	徳島県	農事組合法人櫛渕椎茸組合
スギヒラタケ	E		
ツキヨタケ	F		
ウラベニホテイシメジ	G		
ブナシメジ	H	長野県	信州青果生産グループ　生産者番号01
マイタケ	I	新潟県	一正蒲鉾株式会社
エノキ	J	長野県	有限会社小池青果問屋
エリンギ	K	長野県	信州青果生産グループ　生産者番号01
ナメコ	L	長野県	JA全農長野
ツクリタケ	M	兵庫県	扇港興産株式会社

Python スクリプト

```
import keras
import os,glob,random
import cv2
import numpy as np

outfile="image/photos_chromAI_add.npz"
max_photo=9
photo_size=128
sheet=0

photo_data=[]
label_data=[]
photo_list={}

def main():
    glob_files("./train/hira",0)
    glob_files("./train/kusa",1)
    glob_files("./train/niga",2)
    glob_files("./train/sii",3)
    glob_files("./train/sugi",4)
    glob_files("./train/tuki",5)
    glob_files("./train/ura",6)
    glob_files("./train/sime",7)
    glob_files("./train/mai",8)
    glob_files("./train/eno",9)
    glob_files("./train/erin",10)
    glob_files("./train/name",11)
    glob_files("./train/mush",12)

    np.savez(outfile,photo_data=photo_data,label_data=label_data)
    sheet = len(photo_data)

def glob_files(path,label):
    files=glob.glob(path+"/*.png")
    random.shuffle(files)
```

```
    num=0
    for f in files:
        if num >=max_photo:break
        num+=1
        img=cv2.imread(f)
        img=cv2.resize(img, (photo_size,photo_size))
        img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

        img=np.asarray(img)

        photo_data.append(img)
        label_data.append(label)

        file_name=files[num-1]
        file_name=file_name.replace(path,"").replace(".png","")
        file_name=file_name[1:]
        file_name=path+"/"+file_name
        img=img.tolist()
        photo_list[file_name]=img


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

```
import keras
from keras.models import    Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.layers import Conv2D,MaxPooling2D
from keras.optimizers import    RMSprop, Adam, Adamax

optimize = "Adamax"
activefun = "relu"

def def_model(in_shape,nb_classes):
    model=Sequential()
    model.add(Conv2D(32,
                     kernel_size=(3,3),
                     activation=activefun,
                     input_shape=in_shape))
    model.add(Conv2D(32,(3,3),activation=activefun))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(64,(3,3),activation=activefun))
    model.add(Conv2D(64,(3,3),activation=activefun))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(512,activation=activefun))
    model.add(Dropout(0.5))
    model.add(Dense(nb_classes,activation="softmax"))

    return model

def get_model(in_shape,nb_classes):
    model=def_model(in_shape,nb_classes)
    model.compile(
        loss="categorical_crossentropy",
        optimizer=optimize,
```

```
        metrics=["accuracy"])
    return model
```

```
import cnn_model
import matplotlib.pyplot as plt
import numpy as np
import os
import datetime
from sklearn.model_selection import train_test_split, StratifiedKFold
from keras.preprocessing.image import ImageDataGenerator, array_to_img,
img_to_array, load_img
import pandas as pd
from keras.callbacks import CSVLogger
import csv
import seaborn as sns
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support

size = photo_size
photo_data_train_key=[]
photo_data_val_key=[]
models=[]

epoch=20
classes=13
true_label=["hira","kusaura","nigakuri","sii","sugihira","tukiyo","urabeni","simeji","mai
take","enoki","eringi","nametake","mush"]

datagen = ImageDataGenerator(channel_shift_range=10,fill_mode='nearest')

def images_gen(photo_data_list,label_data_list):
    photo_data_list_add=[]
    label_data_list_add=[]
    for photo_data_num ,label_data_num in zip(photo_data_list,label_data_list):
        photo_data_num = photo_data_num.reshape((1,) + photo_data_num.shape)
        batch_list=[]
        i = 0
        for batch in datagen.flow(photo_data_num, batch_size=1):
            batch=batch.astype(np.uint8)
```

```
            batch=batch.reshape((size, size, 3))
            photo_data_list_add.append(batch)
            label_data_list_add.append(label_data_num)
            i += 1
            if i > 9:
                break
    photo_data_np_add=np.array(photo_data_list_add)
    label_data_np_add=np.array(label_data_list_add)

    return photo_data_np_add,label_data_np_add

im_rows=size
im_cols=size
im_color=3
in_shape=(im_rows,im_cols,im_color)
nb_classes=classes

photos=np.load("image/photos_chromAI_add.npz")
photo_data=photos["photo_data"]
label_data=photos["label_data"]

photo_data=photo_data.reshape(-1,im_rows,im_cols,im_color)
now=datetime.datetime.now()
rand=now.second

photo_data_train,photo_data_val,label_data_train,label_data_val=train_test_split(phot
o_data,label_data,train_size=0.8,stratify=label_data)

K_fold = StratifiedKFold(n_splits=5, shuffle=True)
for fld,(train_cv_no, eval_cv_no) in enumerate(K_fold.split(photo_data,label_data)):
    photo_data_arr=np.array(photo_data)
    label_data_arr=np.array(label_data)

    photo_data_train = photo_data_arr[train_cv_no]
    label_data_train = label_data_arr[train_cv_no]
    photo_data_val = photo_data_arr[eval_cv_no]
```

```
    label_data_val = label_data_arr[eval_cv_no]

    photo_data_val_sub = photo_data_val

photo_data_train_add,label_data_train_add=images_gen(photo_data_train,label_data_
train)
    photo_data_train_add=photo_data_train_add.astype("float32")/255
    photo_data_val=photo_data_val.astype("float32")/255

label_data_train_add=keras.utils.np_utils.to_categorical(label_data_train_add.astype("
int32"),nb_classes)

label_data_val=keras.utils.np_utils.to_categorical(label_data_val.astype("int32"),nb_cla
sses)
    true_classes = np.argmax(label_data_val,1)

    data_time=str(now.month)+"-"+str(now.day)+" "+str(now.hour)+"-
"+str(now.minute)+"-"+str(now.second)+" FOLD-"+str(fld)
    os.makedirs("./result/"+data_time+"/")

    fname1="./result/"+data_time+"/"+"Accuracy "+str(now.month)+"月
"+str(now.day)+"日"+str(now.hour)+"時"+str(now.minute)+"分"+str(now.second)+"
秒.png"
    fname2="./result/"+data_time+"/"+"loss "+str(now.month)+"月"+str(now.day)+"日
"+str(now.hour)+"時"+str(now.minute)+"分"+str(now.second)+"秒.png"
    fname3="./result/"+data_time+"/"+"prosess "+str(now.month)+"月"+str(now.day)+"
日"+str(now.hour)+"時"+str(now.minute)+"分"+str(now.second)+"秒.csv"
    fname4="./result/"+data_time+"/"+"Matrix "+str(now.month)+"月"+str(now.day)+"
日"+str(now.hour)+"時"+str(now.minute)+"分"+str(now.second)+"秒.png"
    fname5="./result/"+data_time+"/train_data "+data_time+".txt"
    fname6="./result/"+data_time+"/val_data "+data_time+".txt"

    def get_keys_from_value(d, value):
        return [k for k, v in d.items() if v == value]
    for k in range(len(photo_data_train)):
        train_photo=photo_data_train[k].tolist()
```

```
        keys = get_keys_from_value(photo_list,train_photo)
        photo_data_train_key.append(keys)

    for k in range(len(photo_data_val_sub)):
        val_photo=photo_data_val_sub[k].tolist()
        keys = get_keys_from_value(photo_list,val_photo)
        photo_data_val_key.append(keys)

    with open(fname5, 'w') as f:
        for d in photo_data_train_key:
            f.write("%s¥n" % d)

    with open(fname6, 'w') as f:
        for d in photo_data_val_key:
            f.write("%s¥n" % d)

    model=cnn_model.get_model(in_shape,nb_classes)
    csv_logger=CSVLogger(fname3)

    hist=model.fit(photo_data_train_add,label_data_train_add,
                   batch_size=32,
                   epochs=epoch,
                   verbose=1,

validation_data=(photo_data_val,label_data_val),callbacks=[csv_logger])

    res=hist.history
    result=[res["loss"][epoch-1],res["accuracy"][epoch-1],res["val_loss"][epoch-
1],res["val_accuracy"][epoch-1]]

    score=model.evaluate(photo_data_val,label_data_val,verbose=1)

    plt.plot(hist.history["accuracy"])
    plt.plot(hist.history["val_accuracy"])
    plt.title("Accuracy")
    plt.legend(["train","val"],loc="upper left")
```

```
plt.savefig(fname1)
plt.show()

plt.plot(hist.history["loss"])
plt.plot(hist.history["val_loss"])
plt.title("Loss")
plt.legend(["train","val"],loc="upper left")
plt.savefig(fname2)
plt.show()

def print_cmx(label_data_true, label_data_pred):

    labels = sorted(list(set(label_data_true)))
    cmx_data = confusion_matrix(label_data_true, label_data_pred, labels=labels)
    labels=true_label
    df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cmx, annot=True)
    plt.xlabel("Predict-labels")
    plt.ylabel("True-labels")
    plt.savefig(fname4)
    plt.show()

predict_classes = model.predict_classes(photo_data_val, batch_size=32)
print_cmx(true_classes, predict_classes)
raw_resutl=precision_recall_fscore_support(true_classes,predict_classes)

data = {"適合率": list(raw_resutl[0]),
        "再現率":list(raw_resutl[1]),
        "F 値":list(raw_resutl[2]),
       "枚数":list(raw_resutl[3])}
df = pd.DataFrame(data)
df.index=true_label
df

model.save_weights("./weight/photos-model-light_add_"+data_time+".hdf5")
```

```
models.append(model)
```

```
import os,glob,random,csv
import cv2
import numpy as np
import cnn_model
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support

outfile="image/photos_test_add.npz"

max_photo=9
data_time="X-XX XX-XX-XX"
classes=13
photo_size=128
photo_list={}

fname7="./result/"+data_time+"/test_data "+data_time+".txt"
fname8="./result/"+data_time+"/"+"Test_Matrix "+data_time+".png"
fname9="./result/"+data_time+"/"+"Test_result "+data_time+".csv"


photo_data=[]
label_data=[]
path_data=[]
test_a=[]
test_b=[]
file_names=[]
csv_t_label=[data_time,"教師ラベル"]
csv_p_label=["","予測ラベル"]
csv_percent=["","%"]

def t_main():
```

```
    glob_files("./test/hira",0)
    glob_files("./test/kusa",1)
    glob_files("./test/niga",2)
    glob_files("./test/sii",3)
    glob_files("./test/sugi",4)
    glob_files("./test/tuki",5)
    glob_files("./test/ura",6)
    glob_files("./test/sime",7)
    glob_files("./test/mai",8)
    glob_files("./test/eno",9)
    glob_files("./test/erin",10)
    glob_files("./test/name",11)
    glob_files("./test/b-mush",12)


np.savez(outfile,photo_data=photo_data,label_data=label_data,path_data=path_data)
    print("保存しました："+outfile,len(photo_data))

    count=len(photo_data)
    return count

def glob_files(path,label):
    files=glob.glob(path+"/*.png")
    random.shuffle(files)
    num=0

    for f in files:
        if num >=max_photo:break
        num+=1
        img=cv2.imread(f)
        img=cv2.resize(img, (photo_size,photo_size ))
        img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
        img_2=np.asarray(img)
        photo_data.append(img_2)
        label_data.append(label)
        path_data.append(f)
```

```
        file_name=files[num-1]
        file_name=file_name.replace(path,"").replace(".png","")
        file_name=file_name[1:]
        file_name=path+"/"+file_name
        img=img.tolist()
        photo_list[file_name]=img
        file_names.append(file_name)
        with open(fname7, 'w') as f:
            for d in file_names:
                f.write("%s¥n" %d)

def classify(count):


    size=photo_size
    labeldata = classes

    photos=np.load("image/photos_test_add.npz")
    photo_data=photos["photo_data"]
    label_data=photos["label_data"]
    path_data=photos["path_data"]


label=["hira","kusaura","nigakuri","sii","sugihira","tukiyo","urabeni","simeji","maitake",
"enoki","eringi","nametake","b-mush"]

    acc=0
    i=0
    for i in range(count):
        label_data2=label_data[i]
        path_data2=path_data[i]
        photo_data2=photo_data[i]

        photo_data3=photo_data2.reshape(-1,size,size,3)
        photo_data3=photo_data3/255
```

```
l_models=len(models)
test_pred=np.zeros((labeldata, l_models))

for fold_,model in enumerate(models):

    pre=model.predict([photo_data3])[0]
    test_pred[:,fold_]=pre


label_data_preds = np.zeros(labeldata)
for label_ in range(labeldata):
    label_data_preds[label_] = np.mean(test_pred[label_, :], axis=0)

idx = np.argmax(label_data_preds, axis=0)

per=int(label_data_preds[idx]*100)

if per>10:

        csv_t_label.append(label[label_data2])
    csv_p_label.append(label[idx])
    csv_percent.append(per)

    print("ラベル→"+label[label_data2]+"　パス→"+path_data2)
    print("これは"+str(per)+"%の確率で"+label[idx]+"です！！")
    print(label_data_preds[idx])


    test_a.append(label_data2)
    test_b.append(idx)
    if label_data2==idx:
        acc+=1
else:
    print("わかりません")
```

```
def print_cmx(label_data_true, label_data_pred):
    labels = sorted(list(set(label_data_true)))
    cmx_data = confusion_matrix(label_data_true, label_data_pred, labels=labels)
    labels= label
    df_cmx = pd.DataFrame(cmx_data, index=labels, columns=labels)

    plt.figure(figsize = (10,7))
    sns.heatmap(df_cmx, annot=True)
    plt.xlabel("Predict-labels")
    plt.ylabel("True-labels")
    plt.savefig(fname8)

    plt.show()

predict_classes = test_b

true_classes = test_a
print_cmx(true_classes, predict_classes)

with open(fname9,"w",newline="") as f:
    writer=csv.writer(f)
    writer.writerow(csv_t_label)
    writer.writerow(csv_p_label)
    writer.writerow(csv_percent)


raw_resutl=precision_recall_fscore_support(true_classes,predict_classes)

data = {"適合率": list(raw_resutl[0]),
        "再現率":list(raw_resutl[1]),
        "F 値":list(raw_resutl[2]),
       "枚数":list(raw_resutl[3])}
print(data)
df = pd.DataFrame(data)
df.index=label
df
```

```
if __name__=="__main__":
    count=t_main()
    classify(count)
```