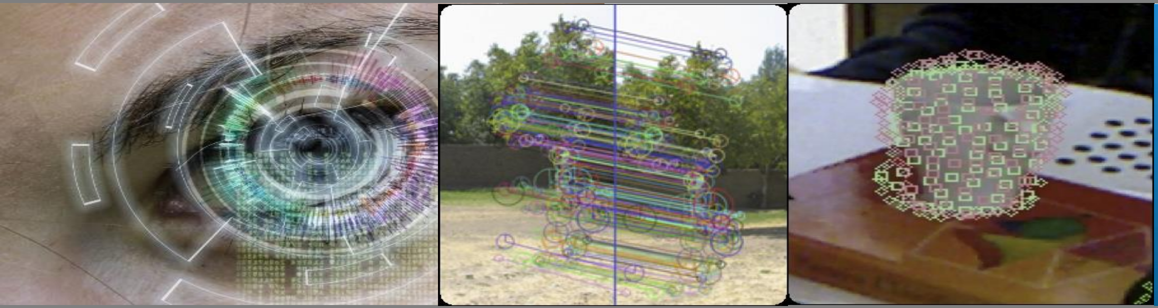
Deep Learning Basics
(#xx: Keras-based Convolutional Neural Network Practice-Part 7)

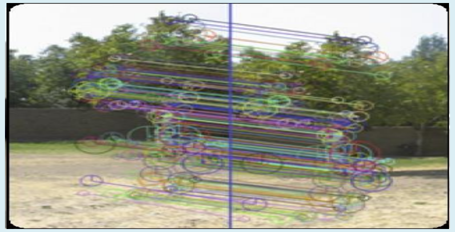


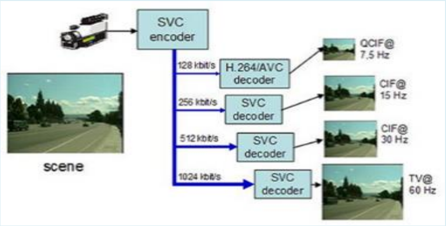
2023 Autumn

Prof. Byung-Gyu Kim
Intelligent Vision Processing Lab. (IVPL)
http://ivpl.sookmyung.ac.kr
Dept. of IT Engineering, Sookmyung Women's University
E-mail: bg.kim@ivpl.sookmyung.ac.kr

Goal of this lecture

- Understanding what is the transfer learning
  - Transfer learning
  - How to implement the transfer learning
  - Actual practice





Contents

- Transfer Learning

Transfer Learning (1)

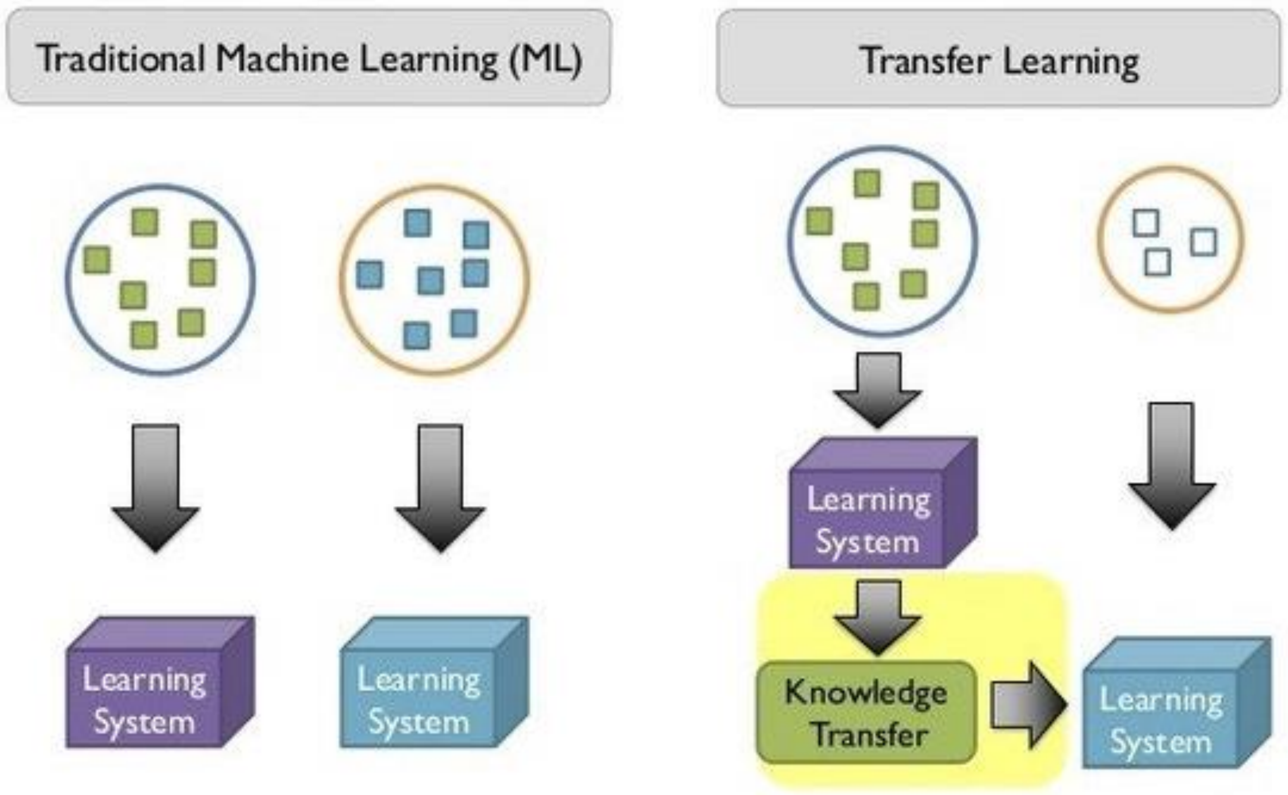
❖ What is "**Transfer Learning**"?

- When a new object recognition or classification is required using the previously learned (trained) object identification model.

EX) How to create an automated computer vision application that can distinguish between "food" and "not food". Which way is the best????

Transfer Learning (2)

- Two ways:
  1) **New model generation** (New training)
  2) **Utilize the pre-trained model** to get some results



Transfer Learning (3) : using Keras

❖ **Transfer Learning** is composed of:

1) Taking a network *pre-trained* on a dataset.
   - Utilize the robust, discriminative filters learned by state-of-the-art networks on challenging datasets (such as ImageNet or COCO).
2) And utilizing it to recognize image/object categories it was not trained on.
   - then apply these networks to recognize objects the model was *never trained* on.

Transfer Learning (4) : using Keras

❖ **Two types of transfer learning** in the context of deep learning:

1) Transfer learning via **feature extraction**
2) Transfer learning via **fine-tuning**

In *feature extraction*, we treat the pre-trained network as an arbitrary feature extractor, **allowing the input image to propagate forward, stopping at pre-specified layer, and taking the *outputs* of that layer** as your features.

Fine-tuning, on the other hand, requires that we update the model architecture itself **by removing the previous fully-connected layer heads, providing new, freshly initialized ones, and then training the new FC layers** to predict our input classes.

❖ Feature Extraction Approach

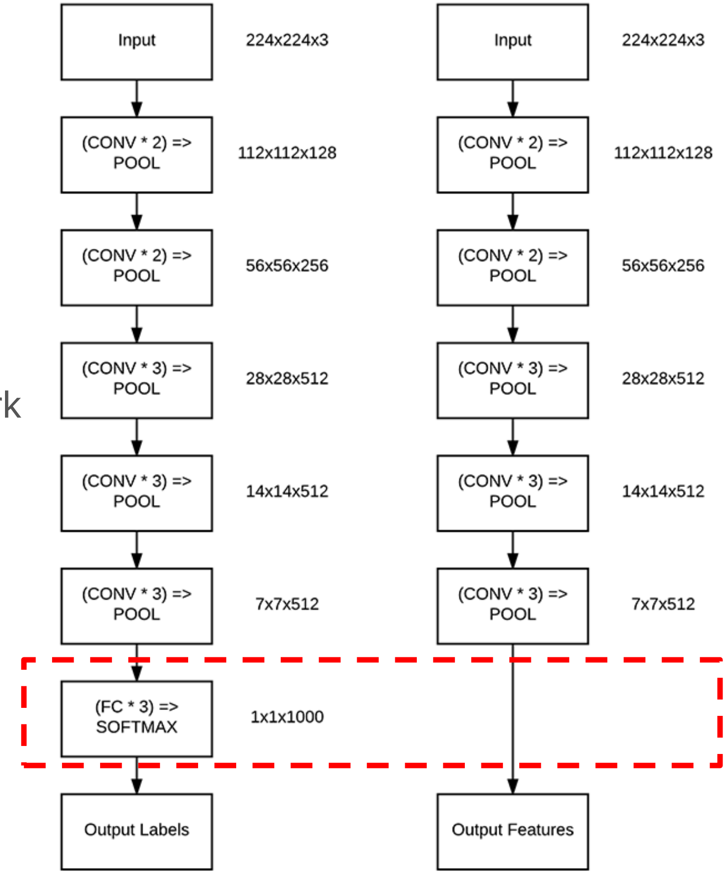
- 1) Datasets
  - Here, Food-5k dataset, a dataset containing 5,000 images falling into two classes: "food" and "not-food" (https://mmspg.epfl.ch/downloads/food-image-datasets/) curated by the Multimedia Signal Processing Group (MSPG) of the Swiss Federal Institute of Technology.

    (You can use FTP client program to download Food-5K dataset.)

[the Foods-5K dataset]

- 2) Train the CNN, first..!!!
  - Deep neural networks trained on large-scale datasets such as **ImageNet** and **COCO** have proven to be *excellent* at the task of transfer learning.
  - These networks learn a set of rich, discriminative features capable of recognizing 100s to 1,000s of object classes — it only makes sense that these filters can be reused for tasks other than what the CNN was originally trained on.

The original VGG16 network architecture

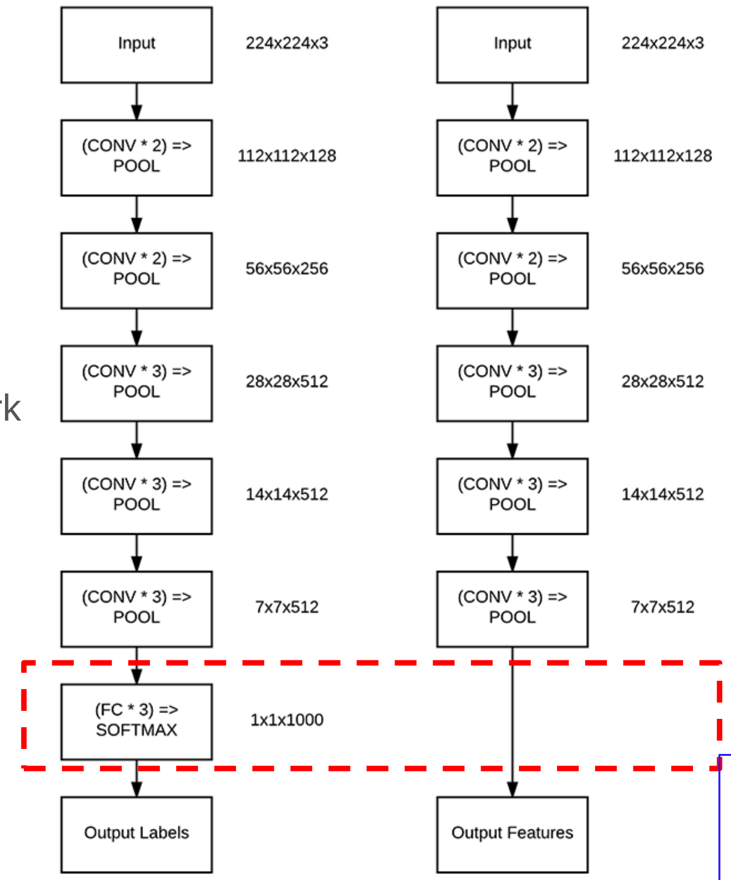


Removing the FC layers from VGG16 and instead of returning the final POOL layer→ Feature extractor

Transfer Learning (7) : using Keras

- 3) The input image to **forward propagate** through the *entire* network.
  - Stop propagation at an arbitrary, but pre-specified layer (such as an activation or pooling layer).
  - **Extract the values from the specified layer (**typically prior to the fully-connected layers, but it really depends on your particular dataset)**.**
  - **Treat the values as a feature vector.**

The original VGG16 network architecture



Removing the FC layers from VGG16 and instead of returning the final POOL layer→ Feature extractor

***7 x 7 x 512 = 25,088* values**

Given a total of *N* images in our network, our dataset would now be represented as a list of *N* vectors, each of 25,088-dim.

IVPL Intelligent Vision Processing Lab

Transfer Learning (9) : using Keras

- 4) **Train off-the-shelf machine learning models**
  - **Linear SVM, Logistic Regression, Decision Trees, or Random Forests** on top of these features to obtain a classifier that can recognize new classes of images.

I want you to keep in mind that the CNN *itself* is *not* capable of recognizing these new classes. Instead, we are using the CNN as an *intermediary feature extractor*.

❖ Project structure

```
(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning>tree /f
폴더 PATH의 목록입니다.
볼륨 일련 번호는 5417-ADDA입니다.
C:.
│  build_dataset.py
│  extract_features.py
│  train.py
│
├─dataset
├─output
└─pyimagesearch
        config.py
        __init__.py


(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning>
```

dataset/ directory, while empty now, will soon contain the Food-5K images in a more organized form.
output/ directory will house our extracted features (stored in three separate .csv files).

- **pyimagesearch/config.py** : Our custom configuration file will help us manage our dataset, class names, and paths. It is written in Python directly so that we can use os.path to build OS-specific formatted file paths directly in the script.
- **build_dataset.py** : Using the configuration, this script will create an organized dataset on disk, making it easy to extract features from.
- **extract_features.py** : The transfer learning magic begins here. This Python script will use a pre-trained CNN to extract raw features, storing the results in a .csv file. The label encoder .cpickle file will also be output via this script.
- **train.py** : Our training script will train a Logistic Regression model on top of the previously computed features. We will evaluate and save the resulting model as a .cpickle .

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (2)

- config.py

```
# import the necessary packages
import os

# initialize the path to the *original* input directory of images
ORIG_INPUT_DATASET = "Food-5K"

# initialize the base path to the *new* directory that will contain
# our images after computing the training and testing split
BASE_PATH = "dataset“


# define the names of the training, testing, and validation
# directories
TRAIN = "training"
TEST = "evaluation"
VAL = "validation"

# initialize the list of class label names
CLASSES = ["non_food", "food"]

# set the batch size
BATCH_SIZE = 32
```

(continue)

```
# initialize the label encoder file path and the output directory to
# where the extracted features (in CSV file format) will be stored
LE_PATH = os.path.sep.join(["output", "le.cpickle"])
BASE_CSV_PATH = "output"

# set the path to the serialized model after training
MODEL_PATH = os.path.sep.join(["output", "model.cpickle"])
```

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (2)

- build_dataset.py

```
# import the necessary packages
from pyimagesearch import config
from imutils import paths
import shutil
import os

# loop over the data splits
for split in (config.TRAIN, config.TEST, config.VAL):
# grab all image paths in the current split
print("[INFO] processing '{} split'...".format(split))
p = os.path.sep.join([config.ORIG_INPUT_DATASET, split])
imagePaths = list(paths.list_images(p))
```

(continue)

```
# loop over the image paths
for imagePath in imagePaths:
# extract class label from the filename
filename = imagePath.split(os.path.sep)[-1]
label = config.CLASSES[int(filename.split("_")[0])]

# construct the path to the output directory
dirPath = os.path.sep.join([config.BASE_PATH, split, label])

# if the output directory does not exist, create it
if not os.path.exists(dirPath):
os.makedirs(dirPath)

# construct the path to the output image file and copy it
p = os.path.sep.join([dirPath, filename])
shutil.copy2(imagePath, p)
```

→ reconstructing "dataset_name/split_name/class_label/example_of_class_label.jpg"

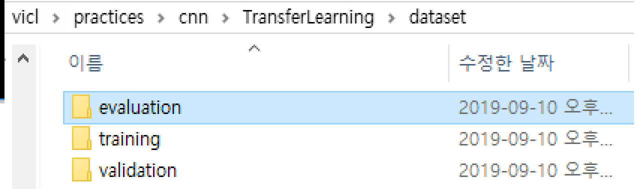
Transfer Learning (8) : Actual Practice – Food/Non-Food classification (3)

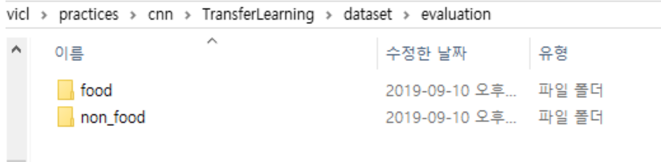
- build_dataset.py

```
# import the necessary packages
from pyimagesearch import config
from imutils import paths
import shutil
import os

# loop over the data splits
for split in (config.TRAIN, config.TEST, config.VAL):
    # grab all image paths in the current split
    print("[INFO] processing '{} split'...".format(split))
    p = os.path.sep.join([config.ORIG_INPUT_DATASET, split])
    imagePaths = list(paths.list_images(p))

        (continue)
```

```
    # loop over the image paths
    for imagePath in imagePaths:
        # extract class label from the filename
        filename = imagePath.split(os.path.sep)[-1]
        label = config.CLASSES[int(filename.split("_")[0])]

        # construct the path to the output directory
        dirPath = os.path.sep.join([config.BASE_PATH, split, label])

        # if the output directory does not exist, create it
        if not os.path.exists(dirPath):
        os.makedirs(dirPath)

        # construct the path to the output image file and copy it
        p = os.path.sep.join([dirPath, filename])
        shutil.copy2(imagePath, p)
```

```
(BGKim) C:\Users\vicl\practices\cnn\TransferLearning>python build_dataset.py
[INFO] processing 'training split'...
[INFO] processing 'evaluation split'...
[INFO] processing 'validation split'...

(BGKim) C:\Users\vicl\practices\cnn\TransferLearning>
```





- extract_features.py(1)

```
# import the necessary packages
from sklearn.preprocessing import LabelEncoder
from keras.applications import VGG16
from keras.applications import imagenet_utils
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import load_img
from pyimagesearch import config
from imutils import paths
import numpy as np
import pickle
import random
import os

# load the VGG16 network and initialize the label encoder
print("[INFO] loading network...")
model = VGG16(weights="imagenet", include_top=False)
le = None
```

```
# loop over the data splits
for split in (config.TRAIN, config.TEST, config.VAL):
	# grab all image paths in the current split
	print("[INFO] processing '{} split'...".format(split))
	p = os.path.sep.join([config.BASE_PATH, split])
	imagePaths = list(paths.list_images(p))

	# randomly shuffle the image paths and then extract the class
	# labels from the file paths
	random.shuffle(imagePaths)
	labels = [p.split(os.path.sep)[-2] for p in imagePaths]

	# if the label encoder is None, create it
	if le is None:
		le = LabelEncoder()
		le.fit(labels)

	# open the output CSV file for writing
	csvPath = os.path.sep.join([config.BASE_CSV_PATH,
	"{}.csv".format(split)])
	csv = open(csvPath, "w")
```

Load VGG16 model without Fully Connected Layers

IVPL
Intelligent Vision Processing Lab

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (5)

- extract_features.py (2)

```
# loop over the images in batches
for (b, i) in enumerate(range(0, len(imagePaths), config.BATCH_SIZE)):
    # extract the batch of images and labels, then initialize the
    # list of actual images that will be passed through the network
    # for feature extraction
    print("[INFO] processing batch {}/{}".format(b + 1,
    int(np.ceil(len(imagePaths) / float(config.BATCH_SIZE)))))
    batchPaths = imagePaths[i:i + config.BATCH_SIZE]
    batchLabels = le.transform(labels[i:i + config.BATCH_SIZE])
    batchImages = []

    # loop over the images and labels in the current batch
    for imagePath in batchPaths:
        # load the input image using the Keras helper utility
        # while ensuring the image is resized to 224x224 pixels
        image = load_img(imagePath, target_size=(224, 224))
        image = img_to_array(image)

        # preprocess the image by (1) expanding the dimensions and
        # (2) subtracting the mean RGB pixel intensity from the
        # ImageNet dataset
        image = np.expand_dims(image, axis=0)
        image = imagenet_utils.preprocess_input(image)

        # add the image to the batch
        batchImages.append(image)
```

IVPL Intelligent Vision Processing Lab

- extract_features.py (3)

```
# pass the images through the network and use the outputs as
# our actual features, then reshape the features into a
# flattened volume
batchImages = np.vstack(batchImages)
features = model.predict(batchImages, batch_size=config.BATCH_SIZE)
features = features.reshape((features.shape[0], 7 * 7 * 512))
# loop over the class labels and extracted features
for (label, vec) in zip(batchLabels, features):
        # construct a row that exists of the class label and
        # extracted features
        vec = ",".join([str(v) for v in vec])
        csv.write("{},{}\n".format(label, vec))

# close the CSV file
csv.close()

# serialize the label encoder to disk
f = open(config.LE_PATH, "wb")
f.write(pickle.dumps(le))
f.close()
```

the output of the CNN as a feature vector.

- Execute result of "extract_features.py":

```
2019-09-17 11:56:44.030495: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1187]      0
2019-09-17 11:56:44.033300: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1200] 0:   N
2019-09-17 11:56:44.043098: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1326] Created TensorFlow device (/job:localhost/replica:0/
task:0/device:GPU:0 with 6357 MB memory) -> physical GPU (device: 0, name: GeForce GTX 1070 Ti, pci bus id: 0000:01:00.0, compute capabil
ity: 6.1)
[INFO] processing 'training split'...
[INFO] processing batch 1/94
[INFO] processing batch 2/94
[INFO] processing batch 3/94
[INFO] processing batch 4/94
[INFO] processing batch 5/94
[INFO] processing batch 6/94
[INFO] processing batch 7/94
[INFO] processing batch 8/94
[INFO] processing batch 9/94
[INFO] processing batch 10/94
[INFO] processing batch 11/94
[INFO] processing batch 12/94
[INFO] processing batch 13/94
[INFO] processing batch 14/94
[INFO] processing batch 15/94
[INFO] processing batch 16/94
[INFO] processing batch 17/94
[INFO] processing batch 18/94
[INFO] processing batch 19/94
[INFO] processing batch 20/94
[INFO] processing batch 21/94
[INFO] processing batch 22/94
[INFO] processing batch 23/94
[INFO] processing batch 24/94
[INFO] processing batch 25/94
[INFO] processing batch 26/94
[INFO] processing batch 27/94
[INFO] processing batch 28/94
```

```
[INFO] processing batch 3/32
[INFO] processing batch 4/32
[INFO] processing batch 5/32
[INFO] processing batch 6/32
[INFO] processing batch 7/32
[INFO] processing batch 8/32
[INFO] processing batch 9/32
[INFO] processing batch 10/32
[INFO] processing batch 11/32
[INFO] processing batch 12/32
[INFO] processing batch 13/32
[INFO] processing batch 14/32
[INFO] processing batch 15/32
[INFO] processing batch 16/32
[INFO] processing batch 17/32
[INFO] processing batch 18/32
[INFO] processing batch 19/32
[INFO] processing batch 20/32
[INFO] processing batch 21/32
[INFO] processing batch 22/32
[INFO] processing batch 23/32
[INFO] processing batch 24/32
[INFO] processing batch 25/32
[INFO] processing batch 26/32
[INFO] processing batch 27/32
[INFO] processing batch 28/32
[INFO] processing batch 29/32
[INFO] processing batch 30/32
[INFO] processing batch 31/32
[INFO] processing batch 32/32

(BGKim) C:\Users\vicl\practices\cnn\TransferLearning>
```

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (8)

- Implementing our training module (train.py) (1)

```
# import the necessary packages
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from pyimagesearch import config
import numpy as np
import pickle
import os

def load_data_split(splitPath):
    # initialize the data and labels
    data = []
    labels = []

    # loop over the rows in the data split file
    for row in open(splitPath):
        # extract the class label and features from the row
        row = row.strip().split(",")
        label = row[0]
        features = np.array(row[1:], dtype="float")

        # update the data and label lists
        data.append(features)
        labels.append(label)

    # convert the data and labels to NumPy arrays
    data = np.array(data)
    labels = np.array(labels)

    # return a tuple of the data and labels
    return (data, labels)
```

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (9)

- Implementing our training module (train.py) (2)

```
# derive the paths to the training and testing CSV files
trainingPath = os.path.sep.join([config.BASE_CSV_PATH, "{}.csv".format(config.TRAIN)])
testingPath = os.path.sep.join([config.BASE_CSV_PATH, "{}.csv".format(config.TEST)])

# load the data from disk
print("[INFO] loading data...")
(trainX, trainY) = load_data_split(trainingPath)
(testX, testY) = load_data_split(testingPath)

# load the label encoder from disk
le = pickle.loads(open(config.LE_PATH, "rb").read())

# train the model
print("[INFO] training model...")
model = LogisticRegression(solver="lbfgs", multi_class="auto")
model.fit(trainX, trainY)

# evaluate the model
print("[INFO] evaluating...")
preds = model.predict(testX)
print(classification_report(testY, preds, target_names=le.classes_))

# serialize the model to disk
print("[INFO] saving model...")
f = open(config.MODEL_PATH, "wb")
f.write(pickle.dumps(model))
f.close()
```

로지스틱(Logistic) 회귀분석은 그 명칭과 달리 회귀분석 문제와 분류문제 모두에 사용할 수 있다. 로지스틱 회귀분석 모형에서는 종속 변수가 **이항 분포**를 따르고 그 모수 μ가 독립 변수 x에 의존한다고 가정한다.

```
Model = Sequential()
model.add(Dense(2, # output dim is 2, one score per each class
    activation='softmax',
    kernel_regularizer=L1L2(l1=0.0, l2=0.1),
    input_dim=len(feature_vector)) # input dimension = number of featu
ur data has
model.compile(optimizer='sgd', loss='categorical_crossentropy',
      metrics=['accuracy'])

model.fit(x_train, y_train, epochs=100, validation_data=(x_val, y_val))
```

IVPL
Intelligent Vision Processing Lab

Transfer Learning (8) : Actual Practice – Food/Non-Food classification (10)

- Let's run train.py…!!!! And check on "output" folder…!!!!

```
(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning>python train.py
[INFO] loading data...
[INFO] training model...
C:₩ProgramData₩Anaconda3₩envs₩BGKim₩lib₩site-packages₩sklearn₩linear_model₩logistic.py:947: ConvergenceWarning: l
 Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
[INFO] evaluating...
              precision    recall  f1-score   support

        food       0.99      0.98      0.98       500
    non_food       0.98      0.99      0.99       500

    accuracy                           0.98      1000
   macro avg       0.99      0.98      0.98      1000
weighted avg       0.99      0.98      0.98      1000

[INFO] saving model...
```

```
(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning>cd output

(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning₩output>dir
 C 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 5417-ADDA

 C:₩Users₩vicl₩practices₩cnn₩TransferLearning₩output 디렉터리

2019-09-17  오후 12:12    <DIR>          .
2019-09-17  오후 12:12    <DIR>          ..
2019-09-17  오전 11:58       117,179,296 evaluation.csv
2019-09-17  오전 11:58               318 le.cpickle
2019-09-17  오후 12:12           201,522 model.cpickle
2019-09-17  오전 11:57       352,327,512 training.csv
2019-09-17  오전 11:58       117,343,088 validation.csv
               5개 파일         587,051,736 바이트
               2개 디렉터리  197,541,494,784 바이트 남음

(BGKim) C:₩Users₩vicl₩practices₩cnn₩TransferLearning₩output>
```

Saved model

Thank you for your attention.!!!
QnA

http://ivpl.sookmyung.ac.kr