AI 기반 영상 데이터 분석 실습
- Day 1 -

Course overview

Course overview

Hands-on Practice in AI-Based Image Data Analysis

- **Instructor**

  Prof. Hwan-ho Cho
  E-mail. hwanho@inu.ac.kr
  Office. Building No.8 room 367

- **Class time**

  2026. 1. 19 – 22 (Mon – Thu), 26 – 27 (Mon – Tue)
  09:00 – 17:00
  (Jan 27: 13:00 – 15:00)

- **Lecture room**

  Building No.8 room 325

Course overview

- **Aim and scope**

  Vision task 소개 및 개인 프로젝트 진행
  데이터 수집, 전처리, 증강, 시각화, 데이터 split, CNN 모델 구조 이해 및 설계, 모델 훈련 및 평가, 프로젝트 발표

  Theory (9:00 – 12:00) + Lab session (13:00 – 15:00)

- **Learning Objectives**

  - 영상 데이터를 활용한 인공지능 모델 설계를 위해 단계별 분석 전략 수립 방법 학습
  - 이미지 분류를 중심으로 한 딥러닝 프로젝트 진행

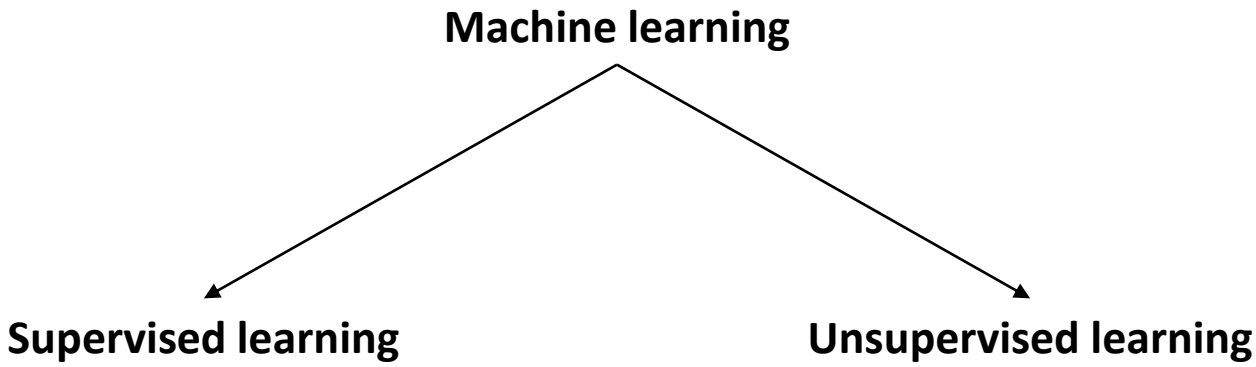
Course overview

Day	Morning session (이론)	Afternoon Lab session (실습)
Jan. 19 (Mon)	• 인공지능 개요 • Vision task 소개 • MNIST Classification review	• GitHub 활용법 • 주제 선정 • 데이터 수집
Jan. 20 (Tue)	• 영상처리 기초 • 이미지 전처리 기법	• 전처리, 증강, 시각화 • Dataset split • Data set class & Data loading
Jan. 21 (Wed)	• Fundamentals on CNN • Basic architectures	• CNN architecture 구현 (Resnet)
Jan. 22 (Thu)	• Weight update (Gradient descent, Optimizers) • Loss monitoring	• Torch model • 아키텍처 선택 및 구현 (Resnet + @) • 모델 학습 및 하이퍼파라미터 튜닝
Jan. 26 (Mon)	• Model evaluation • Grad-CAM	• 모델 평가 • Grad-CAM 실습
Jan. 27 (Tue)	• Github에 프로젝트 업로드 • 프로젝트 발표	

Basic types of machine learning

Basic types of machine learning

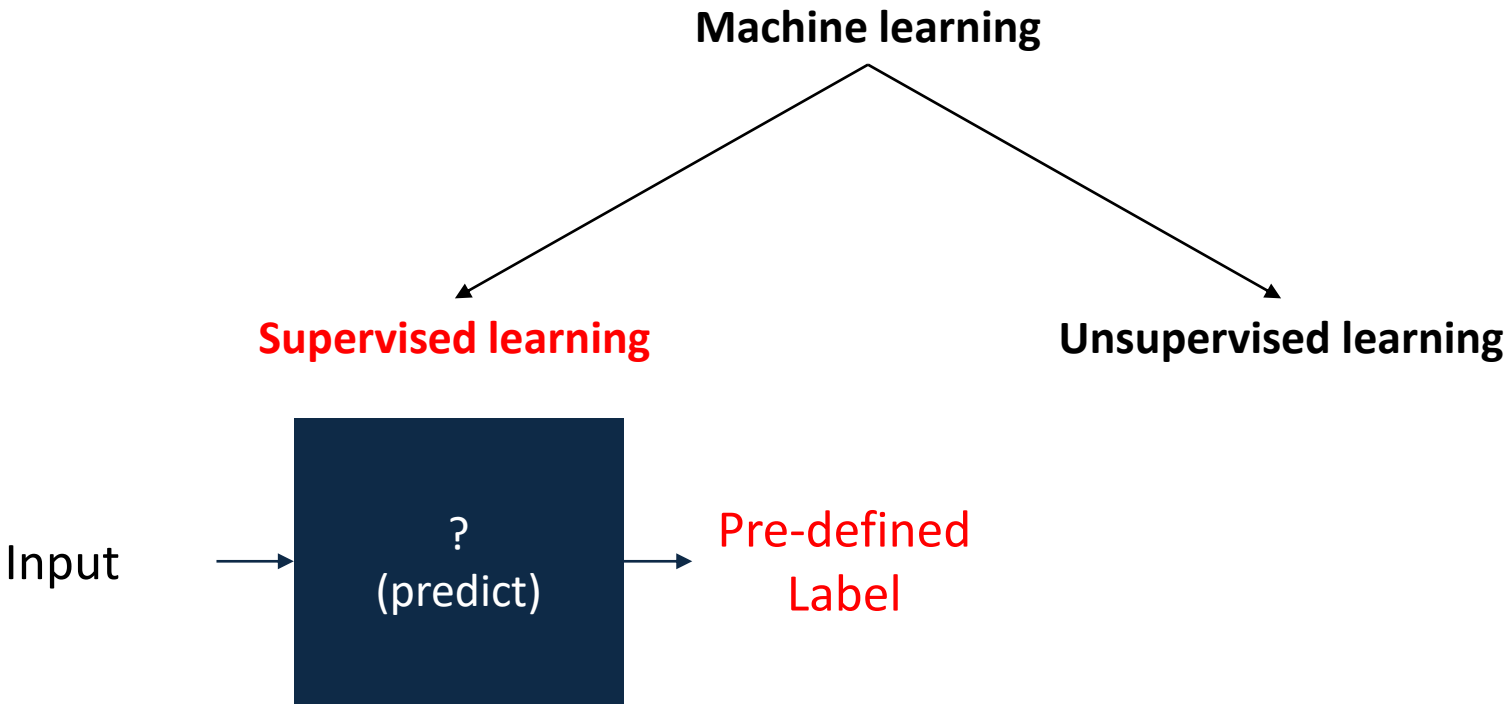
Type of machine learning



- (broadly) Divided into two main categories
  - **Supervised learning**
  - **Unsupervised learning**
- Note. From different perspectives, semi-supervised learning, reinforcement learning, generation and self-supervised learning are also classified into machine learning categories.

Basic types of machine learning

Type of machine learning



ex>

- Classification
- Regression

Basic types of machine learning

Supervised Learning

- Supervised learning is learning from **data and corresponding correct answers**, essentially predicting pre-defined label.
  - Terms synonymous with the correct answer: **label**, **ground truth**, **category**, and **response variable**.

- **Dataset for supervised learning**

$$\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

  - **Labeled samples**

  - $\mathbf{x}$: Feature vector (data)
  - y: Label
  - N: Total number of samples

Basic types of machine learning

Supervised Learning

- There are two main tasks:
  - **Classification**:
    The label for the data → categorical (e.g., apple, watermelon).
    - Two-category problems: Binary classification
    - Problems with more than two-category: Muli-class classification
  - **Regression**:
    The label for the data → continuous variable (e.g., 1, 1.2, 1.3).

Basic types of machine learning

Supervised Learning (ex> Classification)

Input → **Classification** → Label

orange

apple

watermelon

Basic types of machine learning

Supervised Learning (ex> Regression)

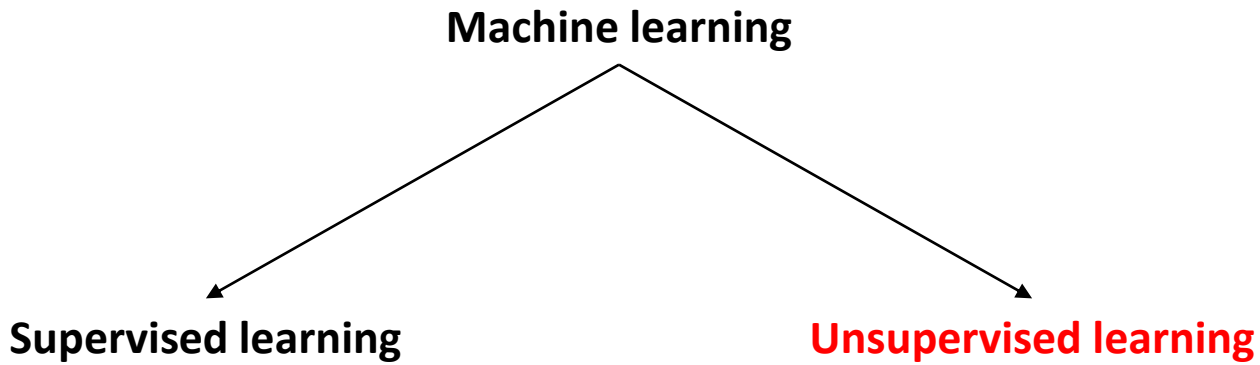
Weight
1 kg
5 kg
3.5 kg
7 kg
2 kg
8kg

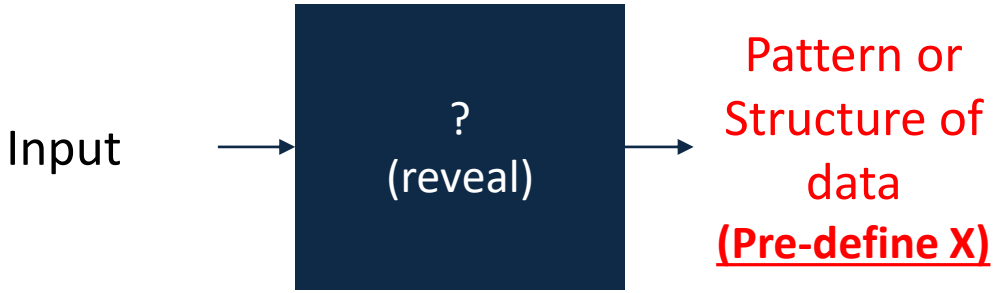
Input → **Regression** → Label

Cost
8,000
39,000
27,000
55,000
26,000
69,000

Basic types of machine learning

Type of machine learning





ex>

- Clustering

Basic types of machine learning

Unsupervised Learning

- Unsupervised learning is used when **there are no desired answer available** (or when not utilizing them) for the data, meaning there are no pre-defined label.
  - It is commonly employed when trying to understand the **structure**, **patterns**, or **distributions** within data.
- **Dataset for unsupervised learning**

$$\{(\mathbf{x}_i)\}_{i=1}^{N}$$

  - **Unlabeled samples**
  - $\mathbf{x}$: Feature vector (data)
  - N: Total number of samples

Basic types of machine learning

Unsupervised Learning

- Well-known unsupervised learning methods
  - **Clustering**: Grouping similar data into clusters.
  - **Dimensional Reduction**: Finding mathematical representations that succinctly express the data.

Basic types of machine learning

Unsupervised Learning (ex> Clustering)

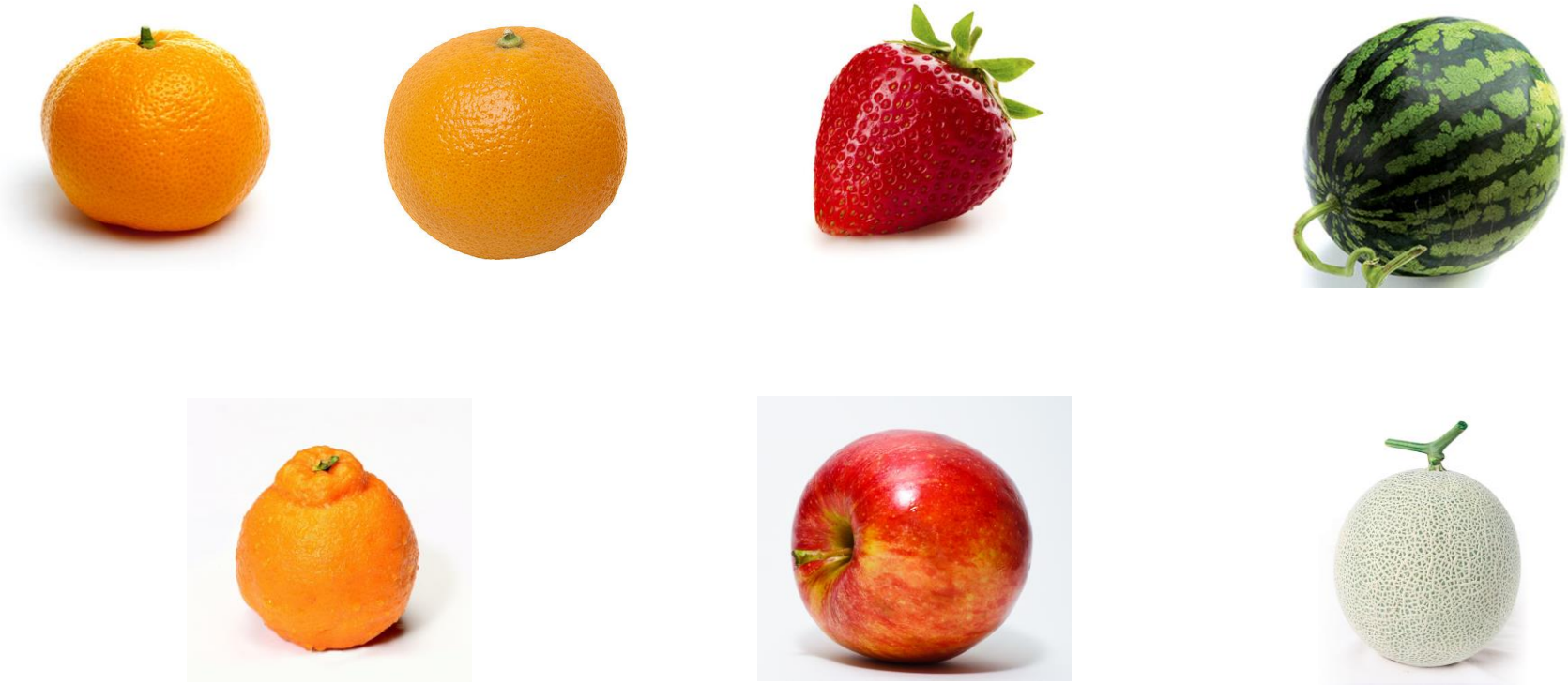






Basic types of machine learning

Unsupervised Learning (ex> Clustering)



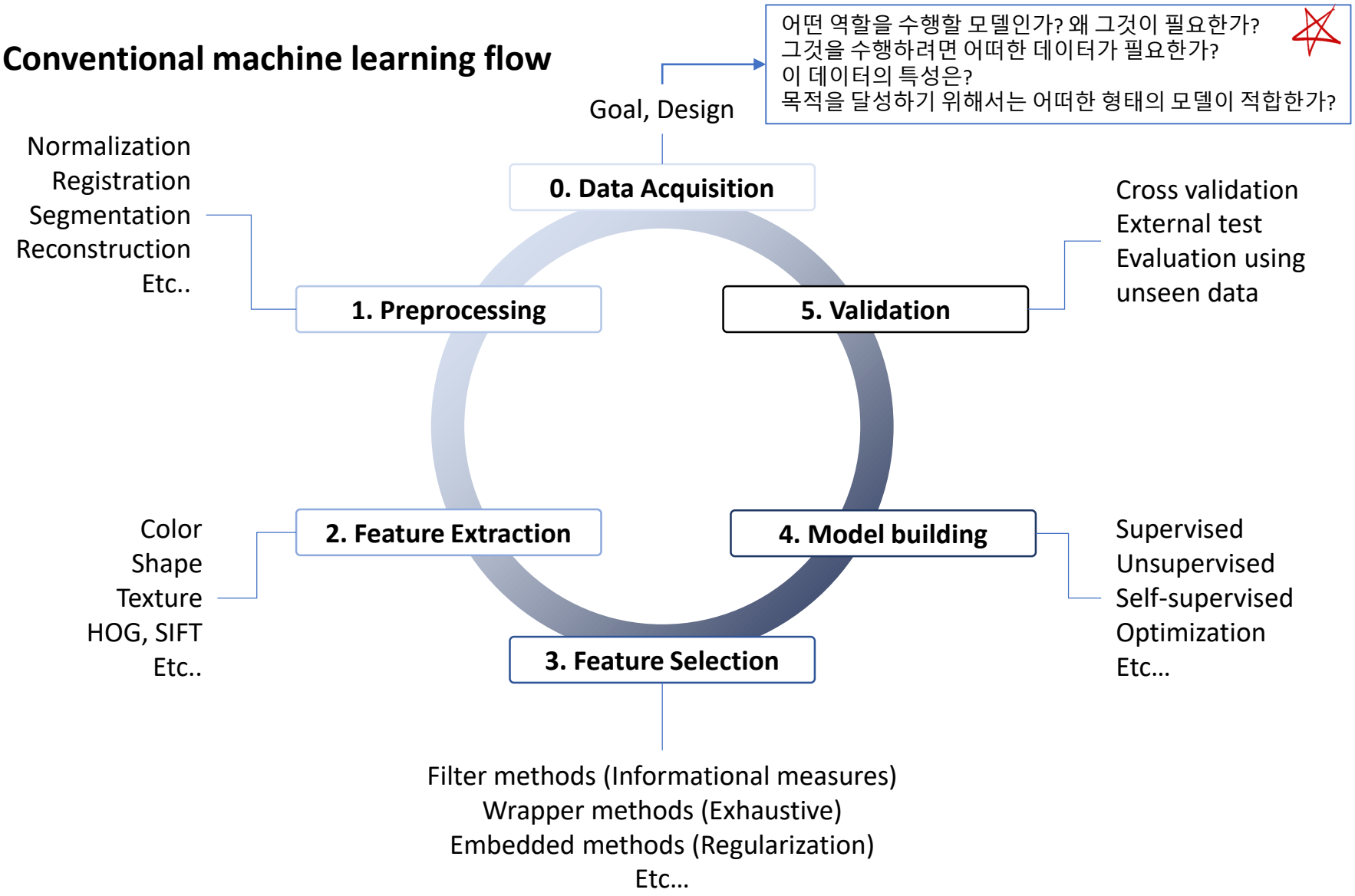
Basic types of machine learning

Types of machine learning

- Difference between Supervised Learning and Unsupervised Learning
  - **Existence (or utilization) of predefined label!!**

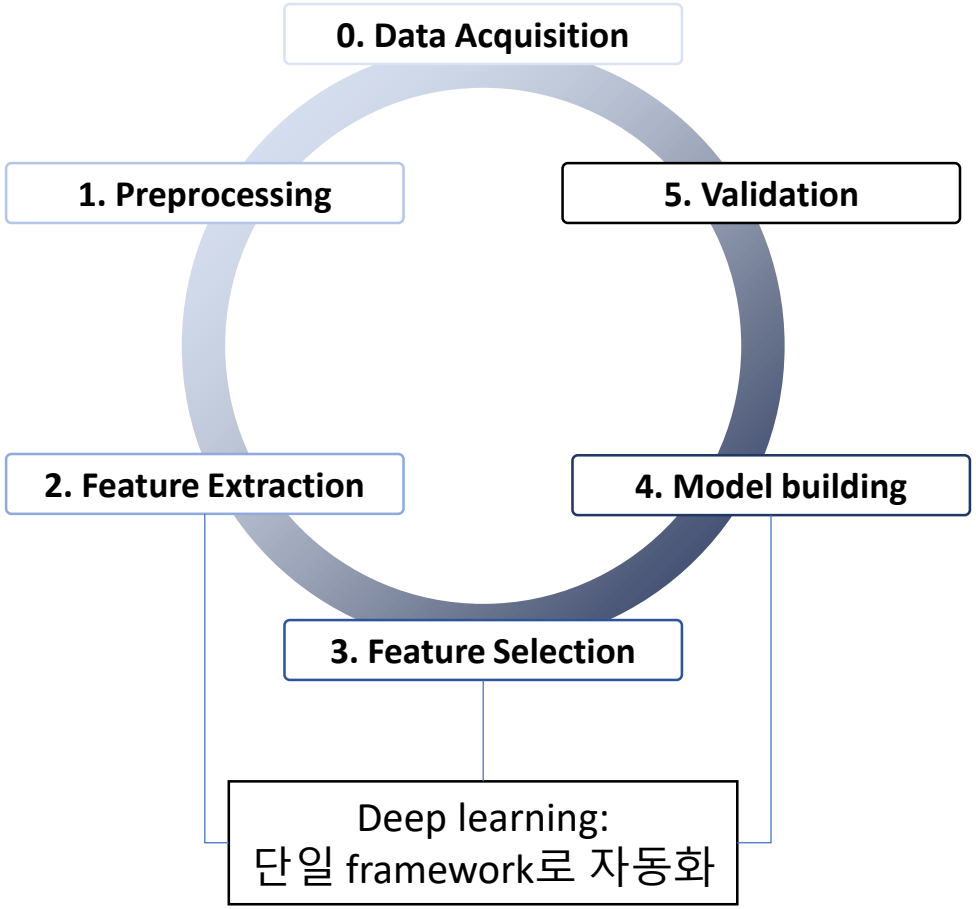
Basic types of machine learning

Conventional machine learning flow



Basic types of machine learning

Conventional machine learning flow



Introduction to vision tasks

Image classification

Quick hands-on experience: Google Teachable Machine

https://teachablemachine.withgoogle.com/

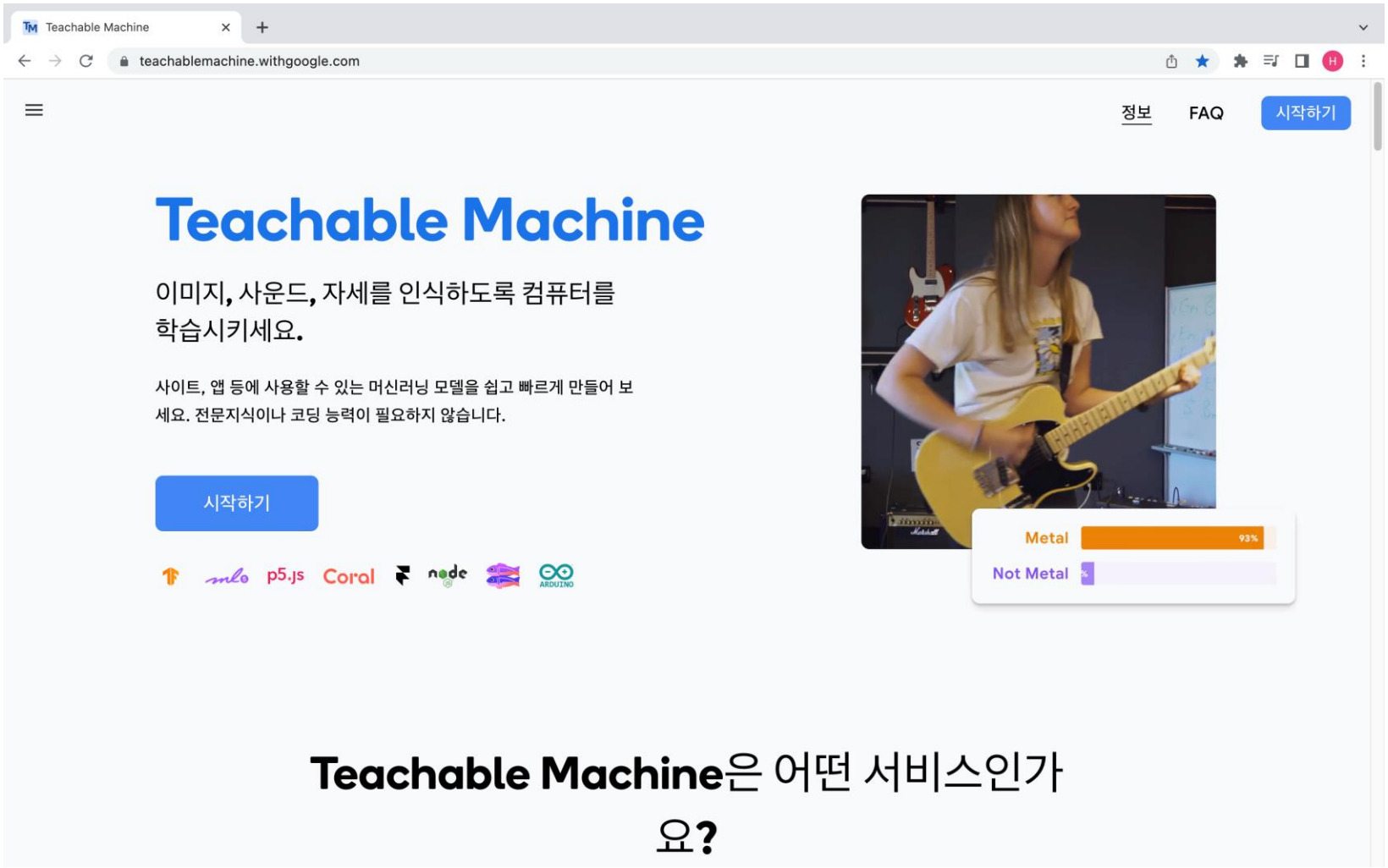
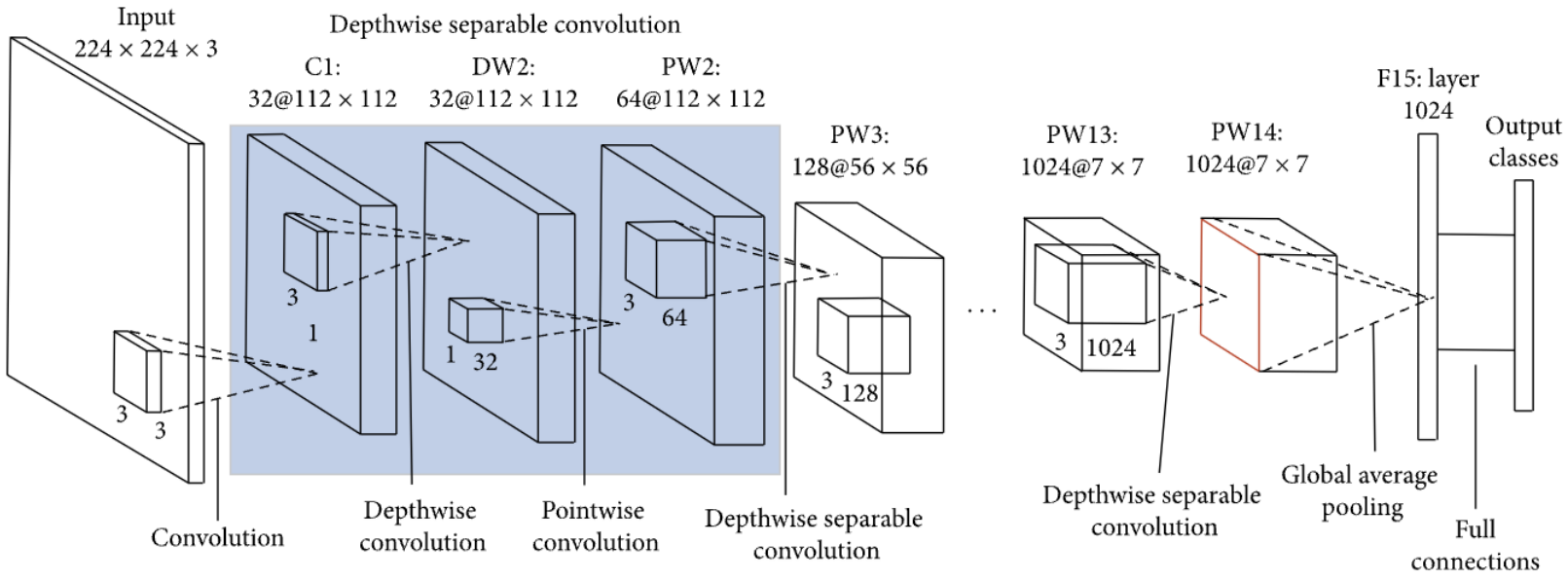


Image classification

Google Teachable Machine

- Model spec.
  - MobileNet backbone



  - Pretrained using ImageNet
    - 1,281,167 training images
    - 1000 classes

Image classification

Google Teachable Machine

- Transfer learning

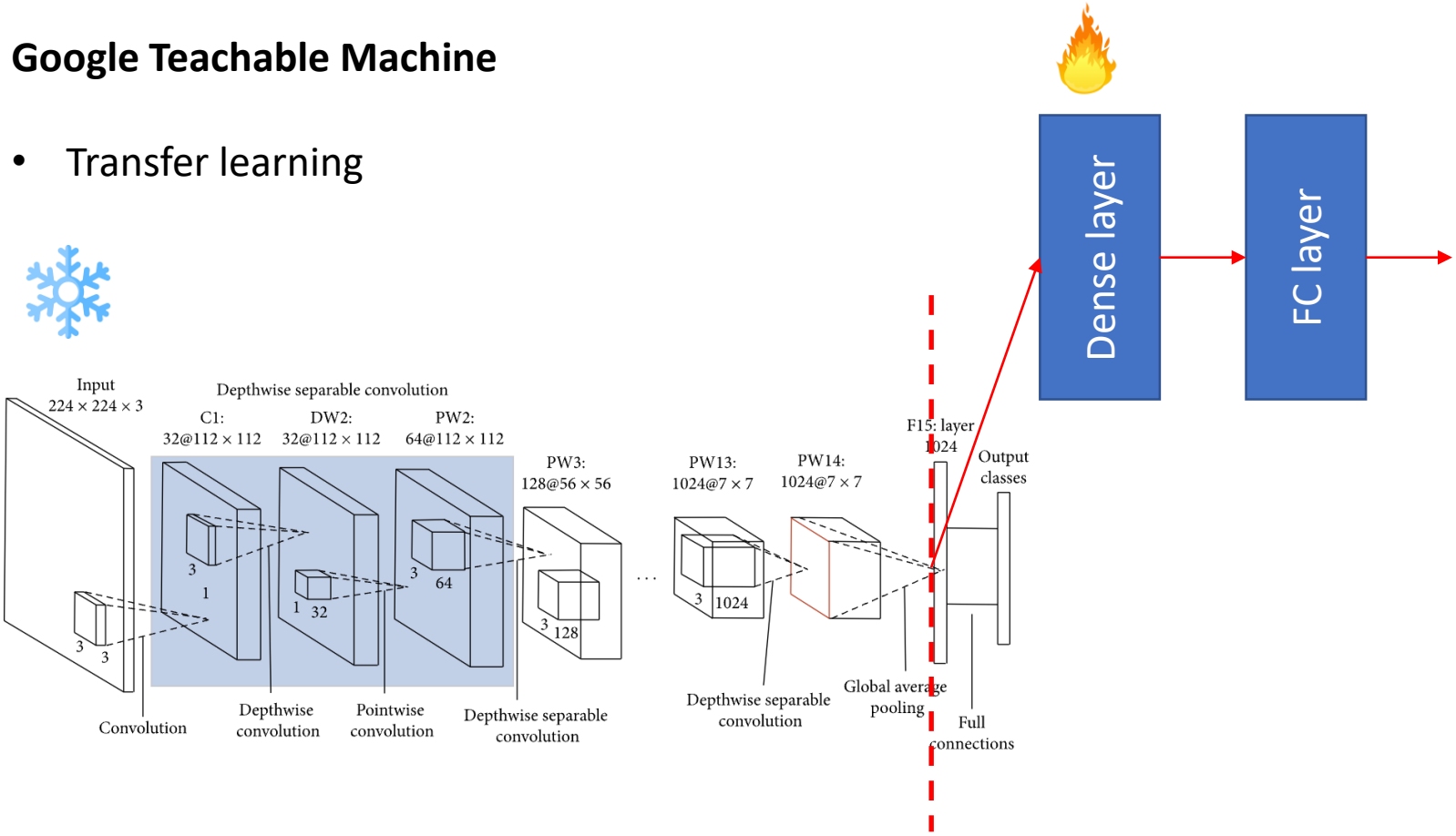


Image classification

Google Teachable Machine

- 이미지 프로젝트 선택 (표준 이미지 모델)

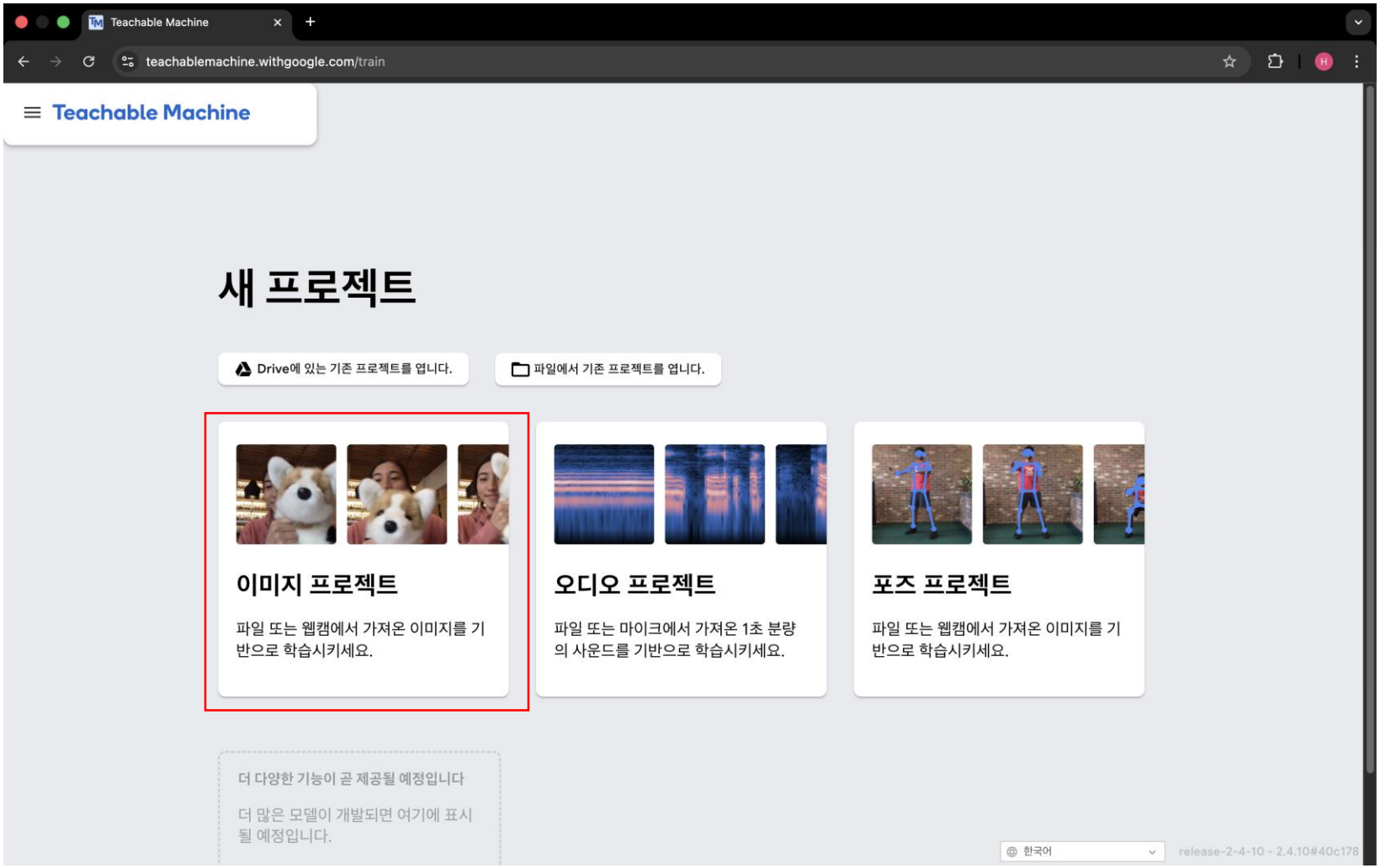


Image classification

Google Teachable Machine

- 이미지 프로젝트 선택 (표준 이미지 모델)

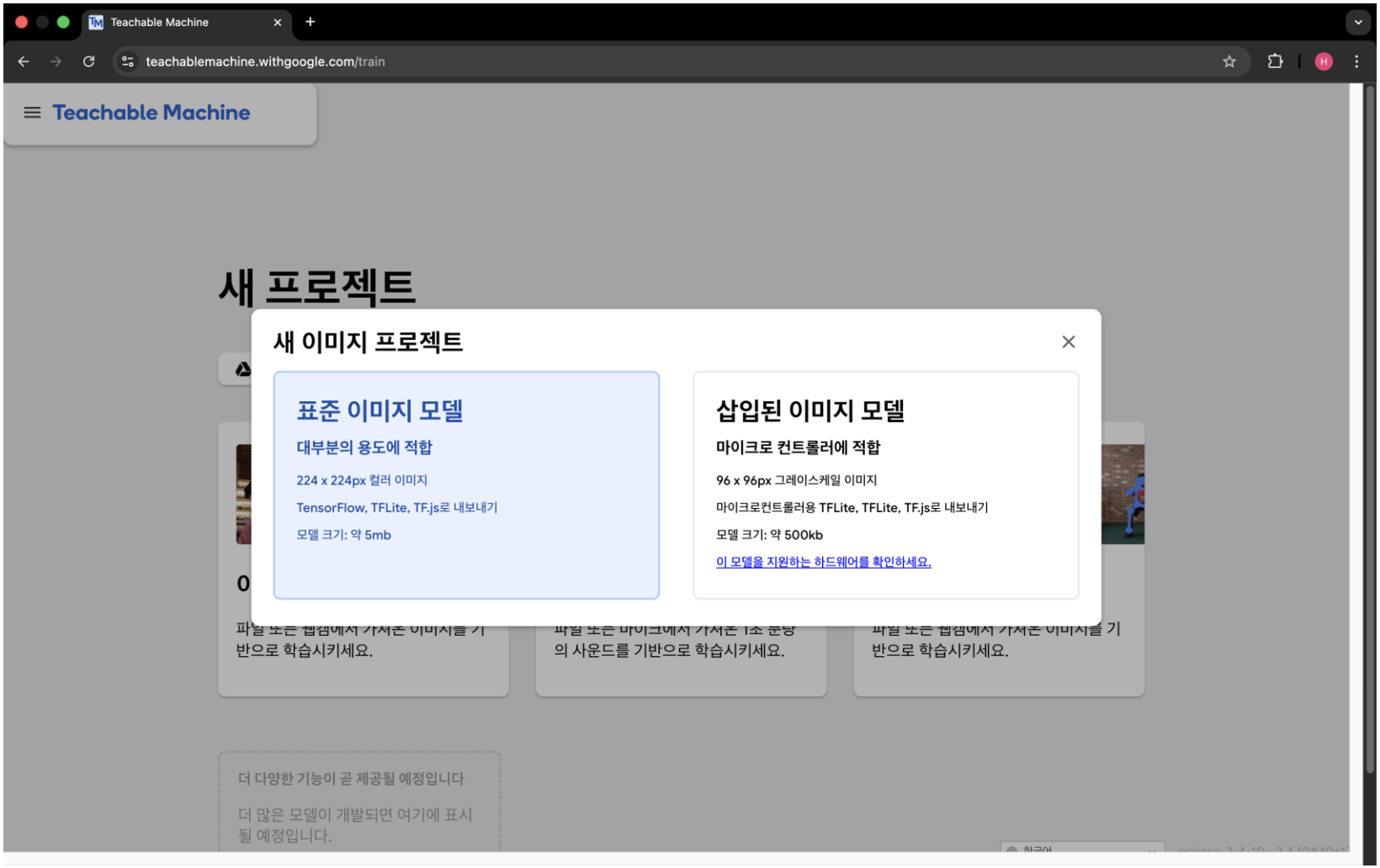


Image classification

Google Teachable Machine

- 웹캠을 이용한 클래스 별 이미지 수집

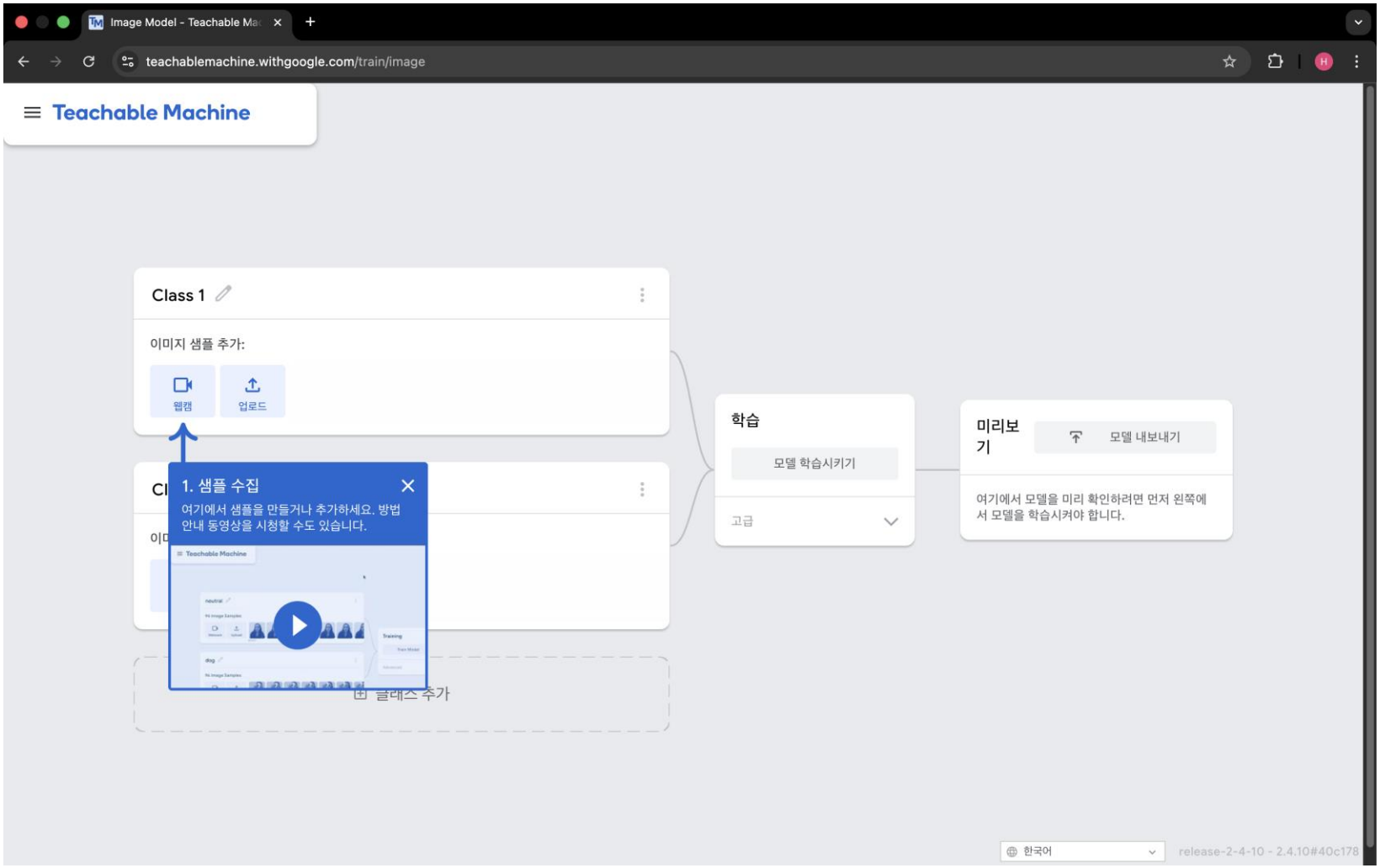


Image classification

Google Teachable Machine

- **고급 탭 – hyperparameter 설정**
  - **Epoch**
  - **Batch size**
  - **Learning rate**

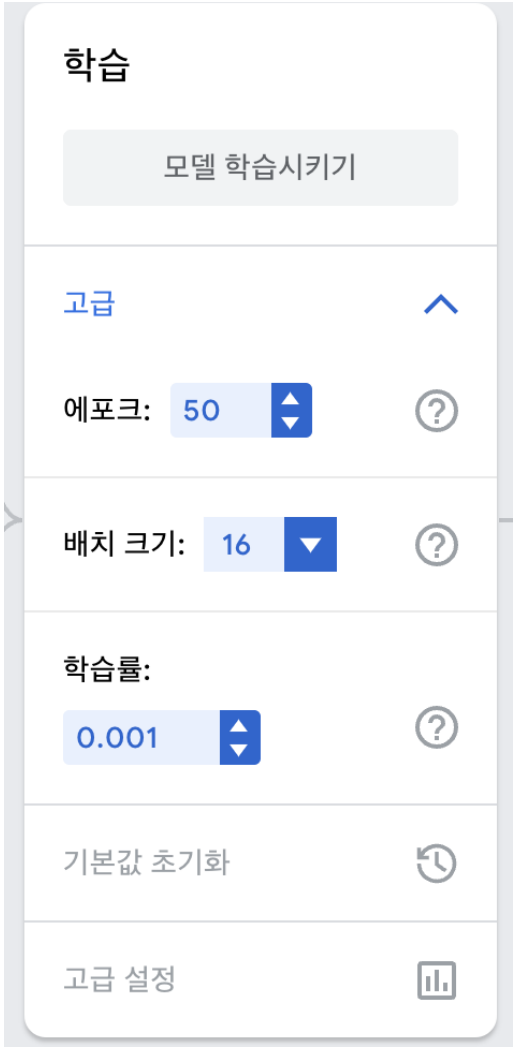


Image classification

Google Teachable Machine

- 훈련 완료 후 실시간 테스트

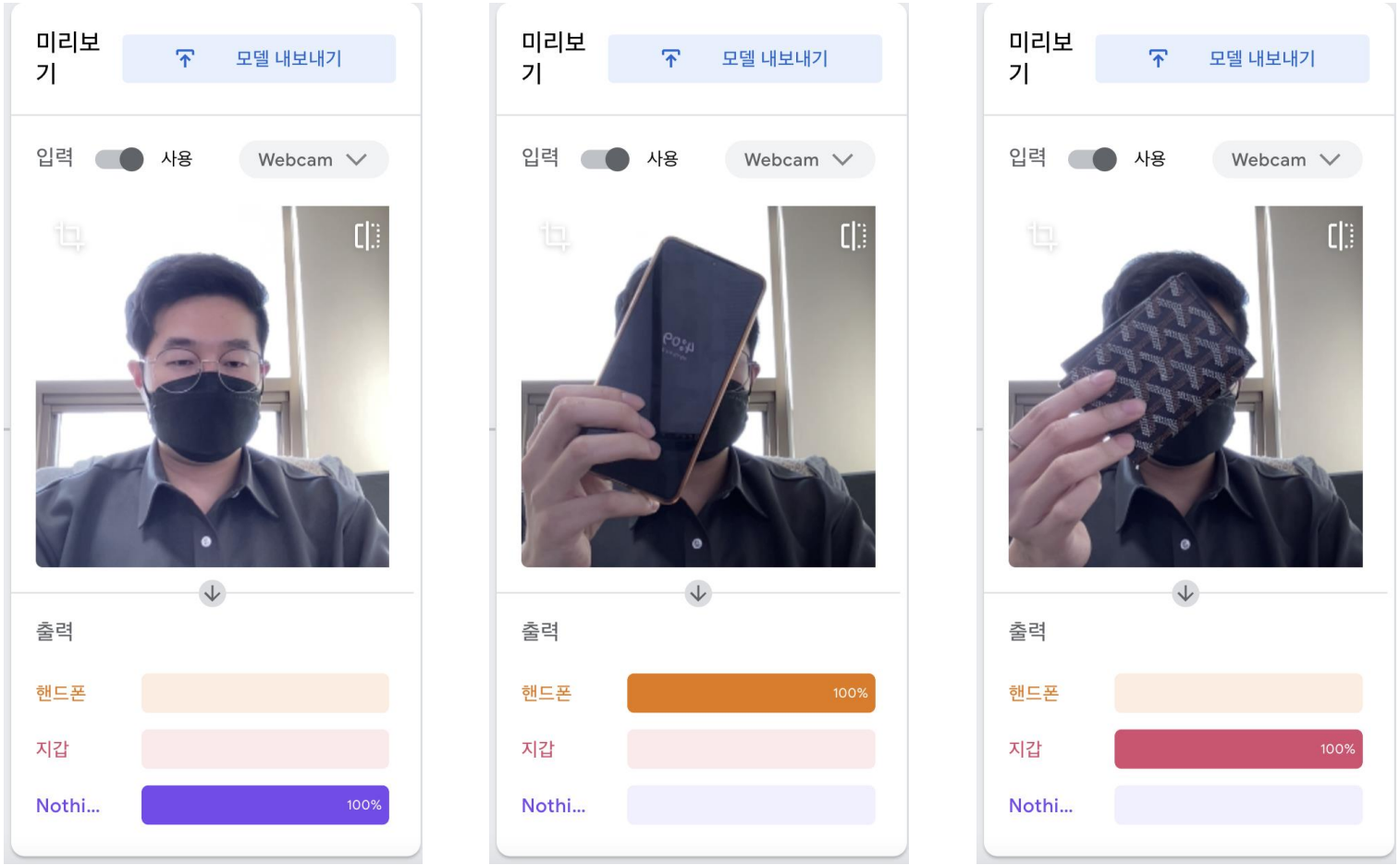


Image classification

Google Teachable Machine

- 훈련된 파라미터 다운 및 활용 가능

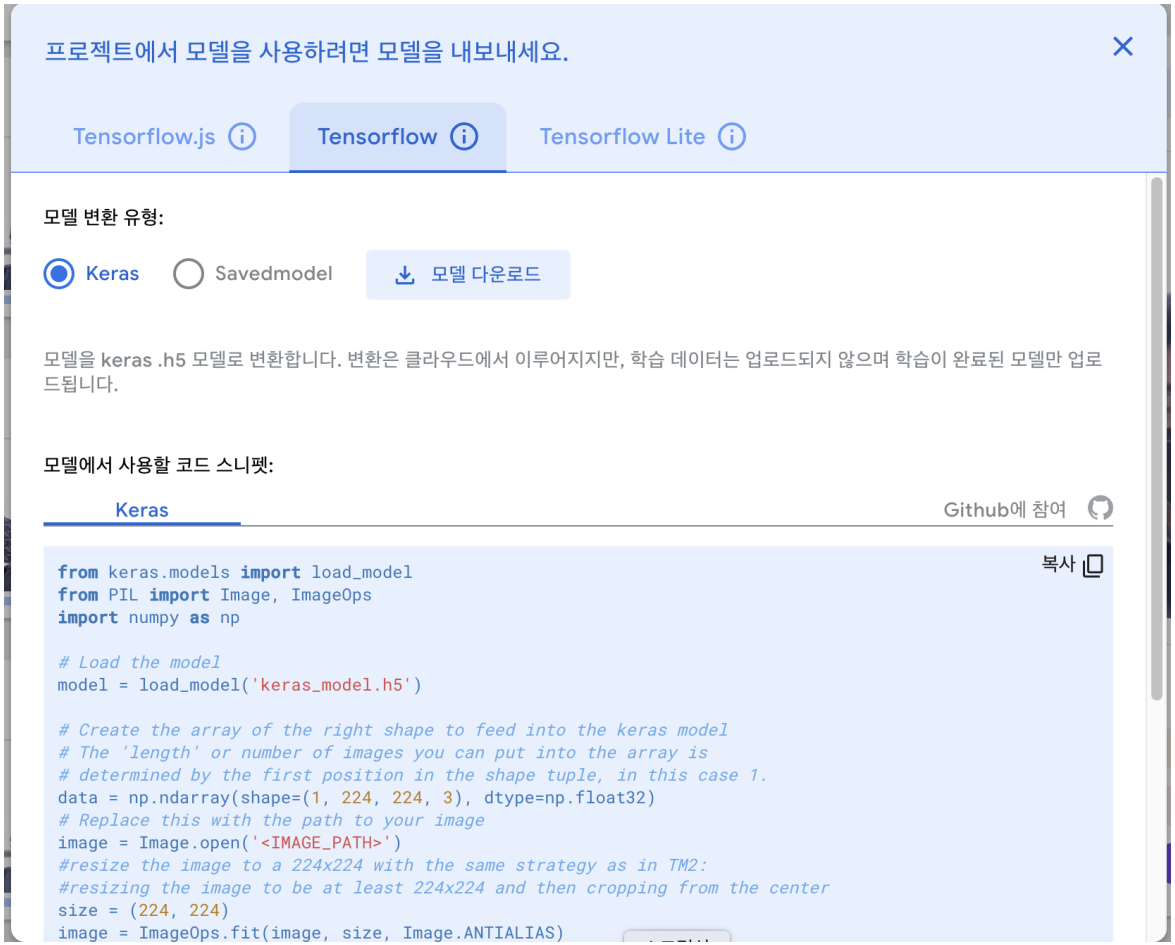


Image classification

Google Teachable Machine

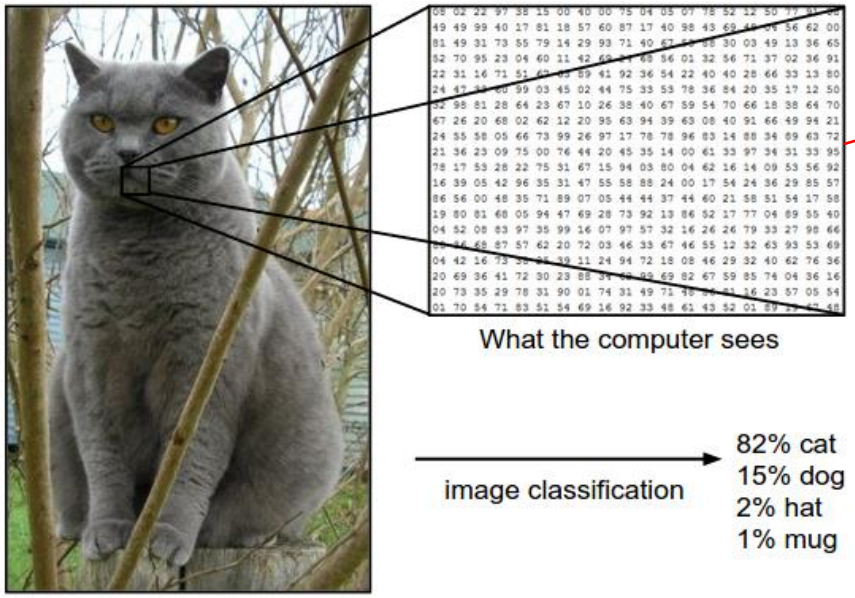
- 코드 스니펫

```
from keras.models import load_model
from PIL import Image, ImageOps
import numpy as np

# Load the model
model = load_model('keras_model.h5')

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1.
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
# Replace this with the path to your image
image = Image.open('<IMAGE_PATH>')
#resize the image to a 224x224 with the same strategy as in TM2:
#resizing the image to be at least 224x224 and then cropping from the center
size = (224, 224)
image = ImageOps.fit(image, size, Image.ANTIALIAS)

#turn the image into a numpy array
image_array = np.asarray(image)
# Normalize the image
normalized_image_array = (image_array.astype(np.float32) / 127.0) - 1
# Load the image into the array
data[0] = normalized_image_array

# run the inference
prediction = model.predict(data)
print(prediction)
```

Image classification

Image classification

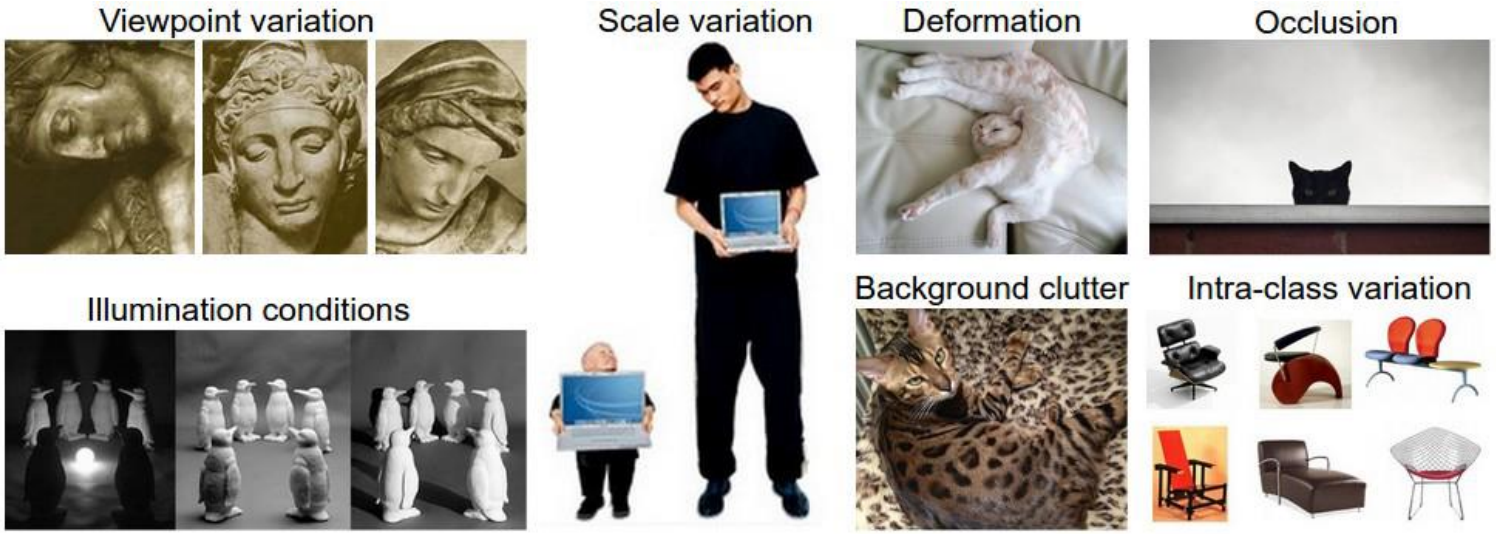
- One of the core problems in Computer Vision
- **Assigning an input image one label from a fixed set of categories**
- Many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification



- Image is 3D matrix containing 2D {R,G,B} channel
- Each element = pixel intensity

Image classification

Challenges in Image classification



- **Viewpoint variation**: Objects can appear in multiple orientations.
- **Scale variation**: Objects vary in real-world size and image extent.
- **Deformation**: Non-rigid objects can change shape significantly.
- **Occlusion**: Objects may be partially visible, sometimes as few pixels.
- **Illumination conditions**: Lighting changes greatly affect pixel values.
- **Background clutter**: Objects may blend into their surroundings, making identification difficult.
- **Intra-class variation**: Broad categories (e.g., chairs) contain diverse appearances.

Image classification

Challenges in Image classification

- Good image classification model → Robust to all these variations

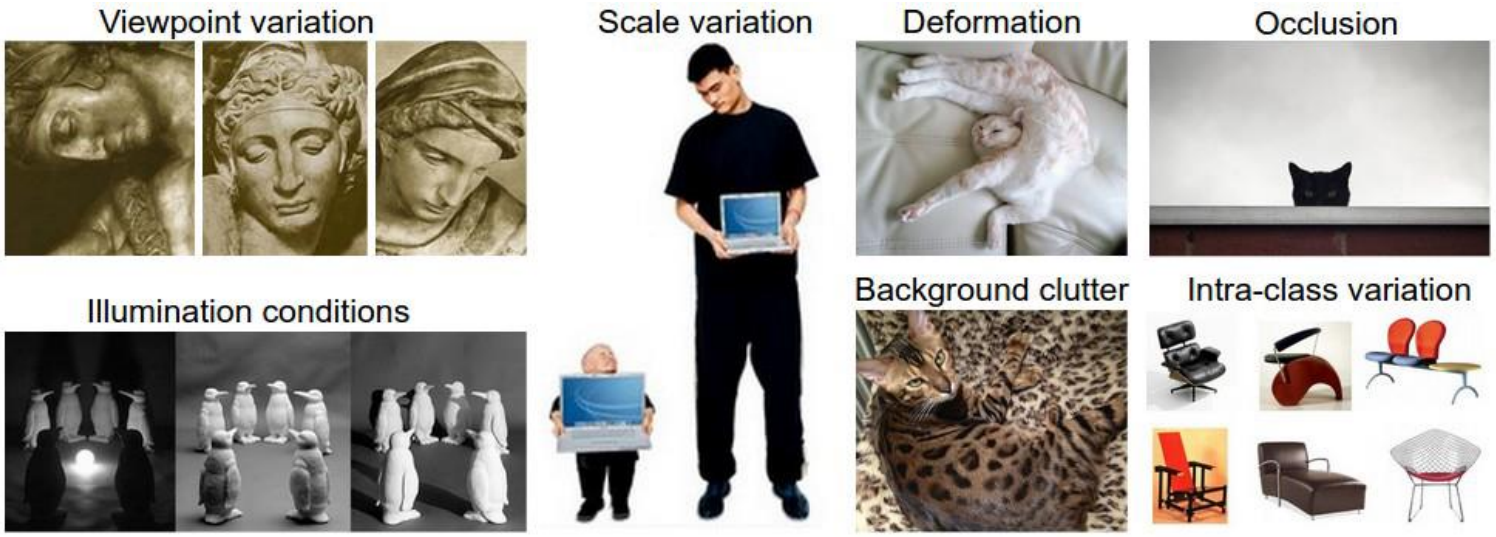


Image classification

Data-driven approach

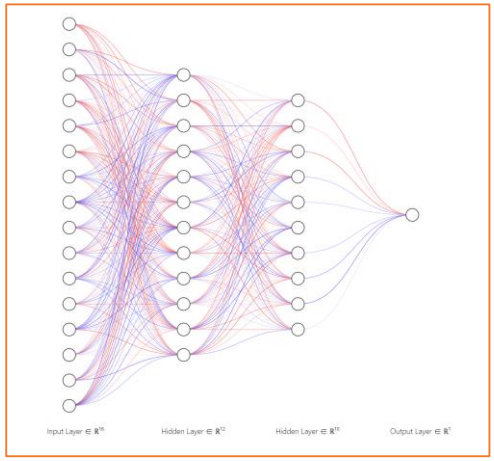
- In a data-driven approach, class characteristics are not explicitly specified in the code.
- Instead,

1. Provide the computer with many example of each class

2. Develop learning algorithm that look at these examples and learn about visual appearance of each class

3. Make prediction on unseen data

<Labeled image dataset>



<Learning algorithm>

?

<Unseen data>

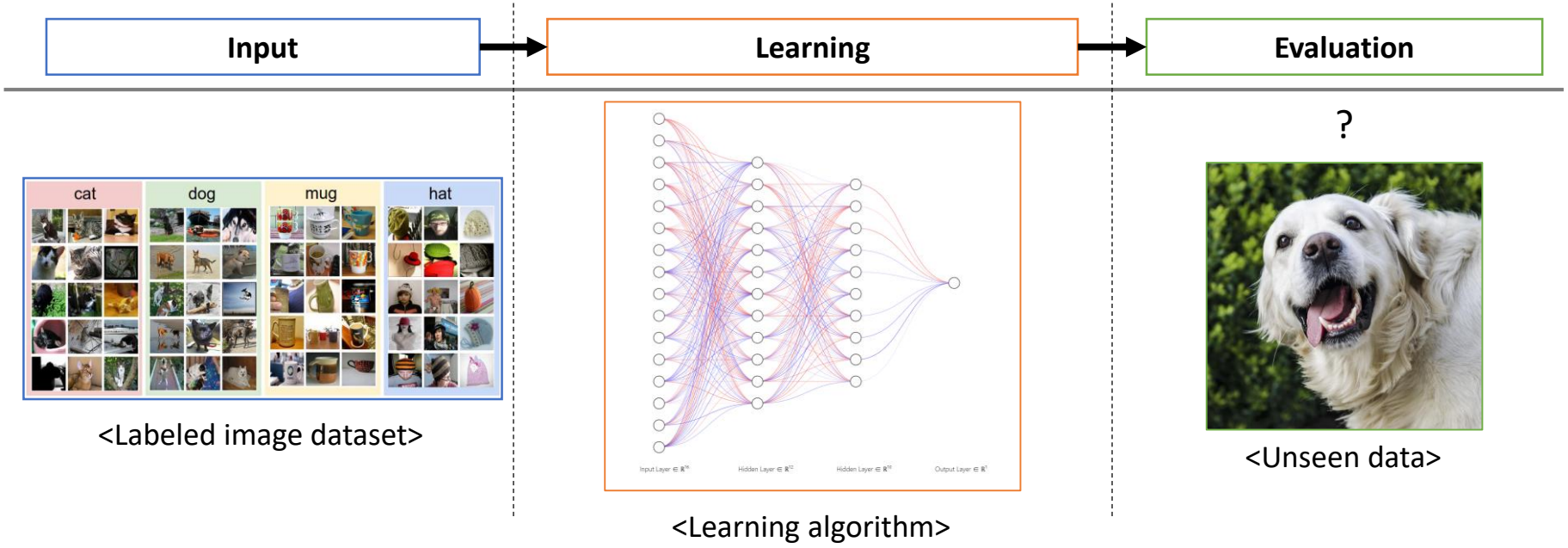
Image classification

The image classification pipeline

1. **Input**: A set of N images, each labeled with one of K different classes. (Training set)

2. **Learning**: To use the training set to learn what every one of the classes looks like. (Training a classifier or learning a model)

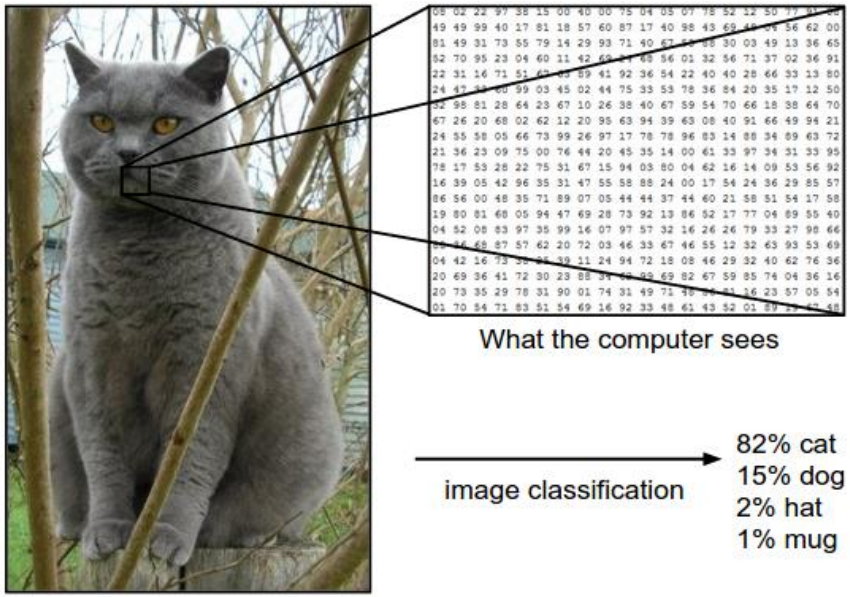
3. **Evaluation**: Evaluating the quality of the classifier by asking it to predict labels for a new set of unseen images.



Segmentation

Recall: Image classification

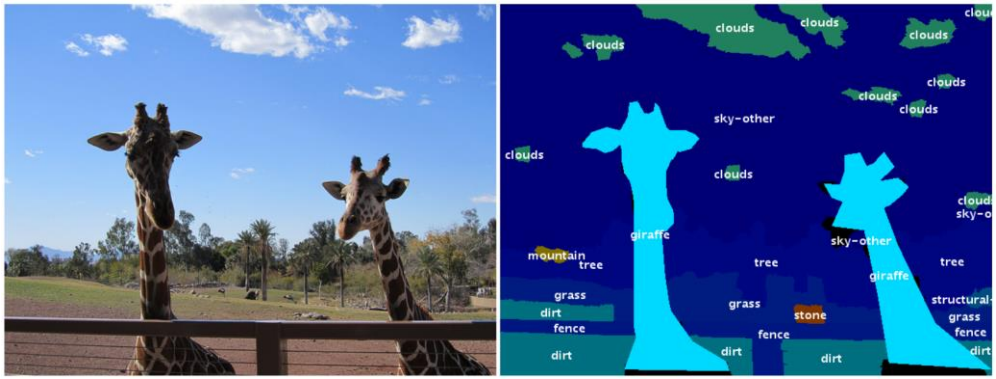
- One of the core problems in Computer Vision
- **Assigning an input image one label from a fixed set of categories**
- Global understanding about an image



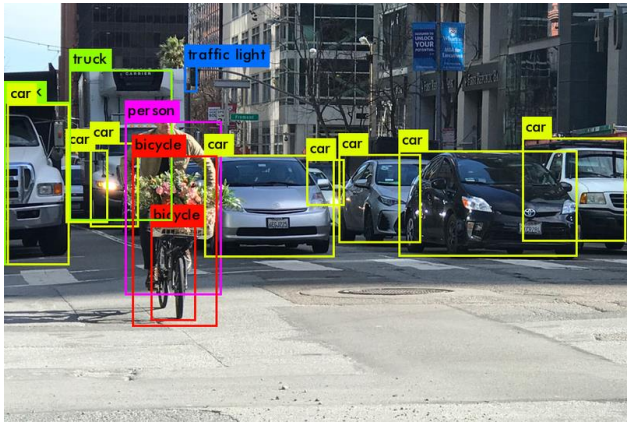
Segmentation and Object Detection

Localized understanding: Segmentation and Detection

- **Image segmentation**
  - Pixel-wise classification for distinguishing object regions
- **Object detection**
  - Localizing and classifying objects in the image



[Image segmentation example]

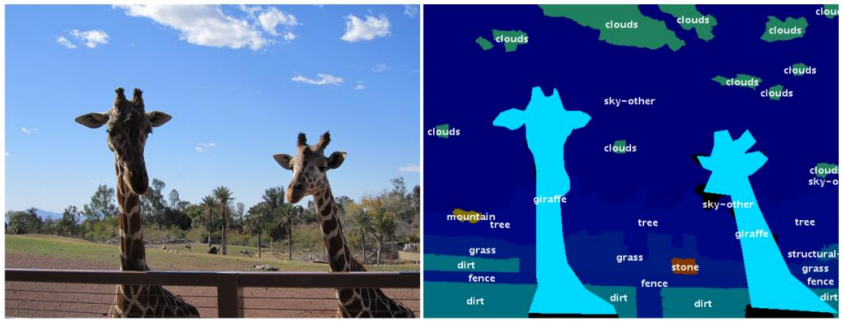


[Object detection example]

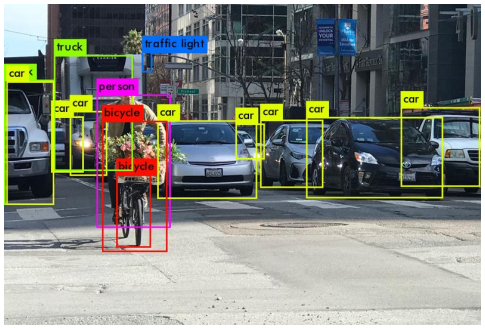
Segmentation and Object Detection

Segmentation vs Detection

- **Segmentation**
  - Output: Pixel-level labeling
  - Pros: detailed shape info
  - Cons: computationally heavier



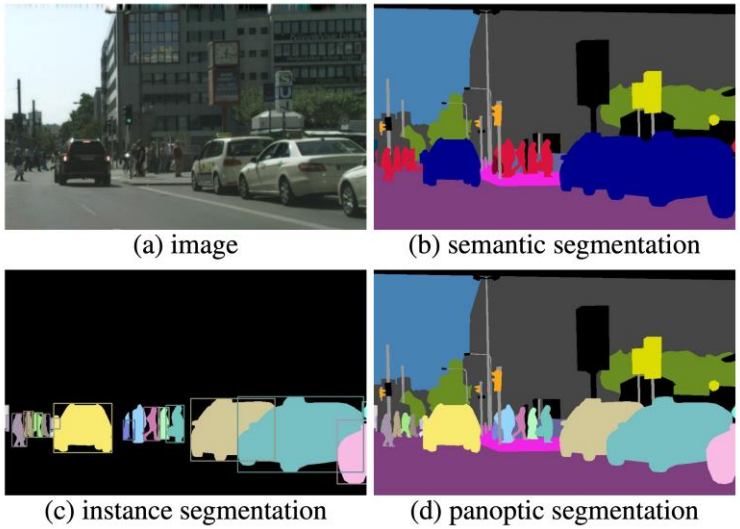
- **Object Detection**
  - Output: Bounding box + class label
  - Pros: fast, compact representation
  - Cons: ignores object shape



Segmentation

Taxonomy of segmentation

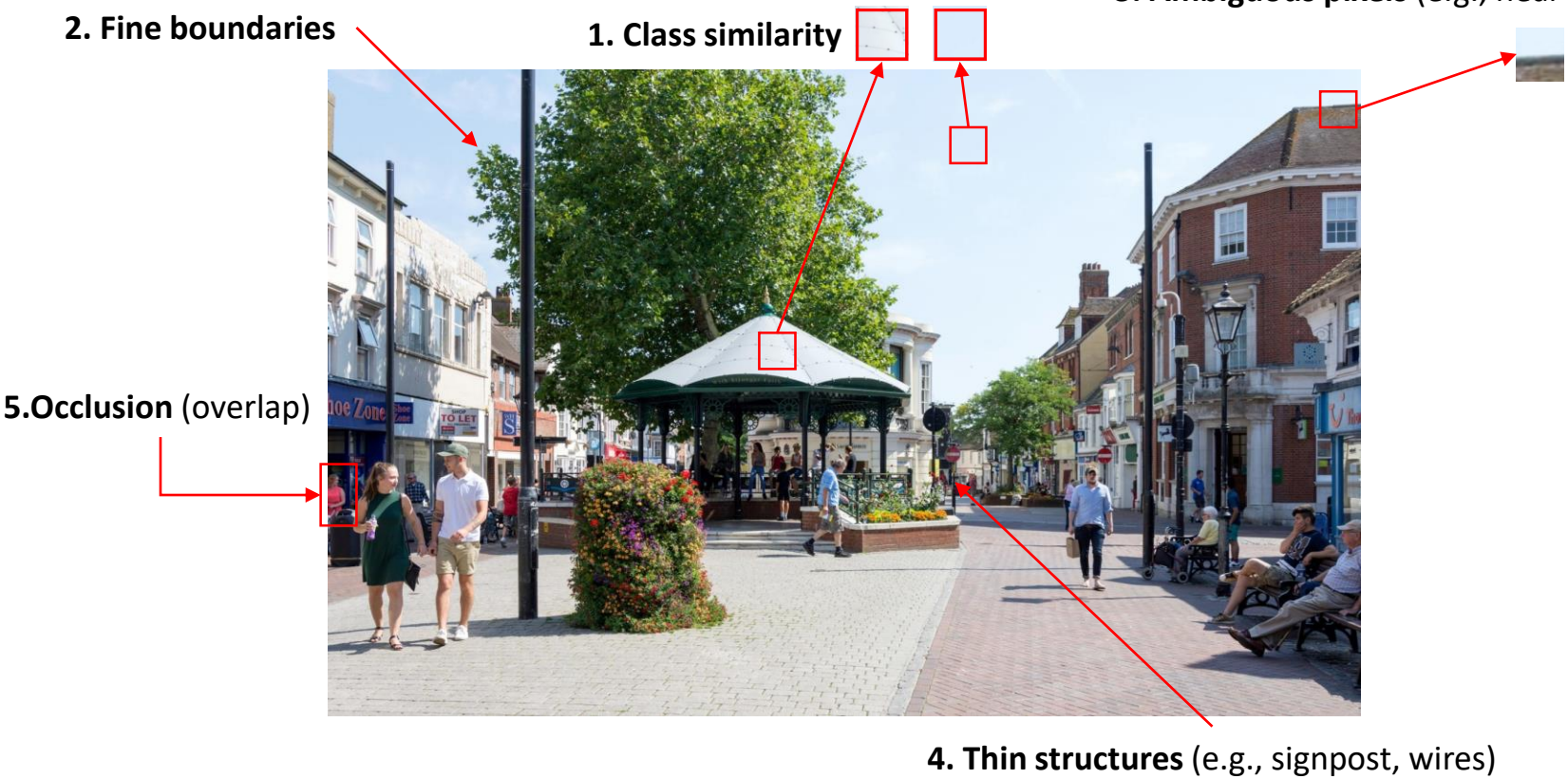
- **Semantic Segmentation**
  - Assigns a class label to each pixel
  - Same class objects indistinguishable
- **Instance Segmentation**
  - Detects and segments each object instance
  - Separates each object of same class
- **Panoptic Segmentation:**
  - Unified task → semantic + instance
  - Gives full scene understanding


(a) image (b) semantic segmentation (c) instance segmentation (d) panoptic segmentation

Kirillov, Alexander, et al. "Panoptic segmentation." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019.

Segmentation

Challenges in image segmentation



Segmentation

Challenges in image segmentation

6. Scale variation

7. Domain shift

(e.g., human in the shadow vs light, stand vs seat, day vs night etc.)

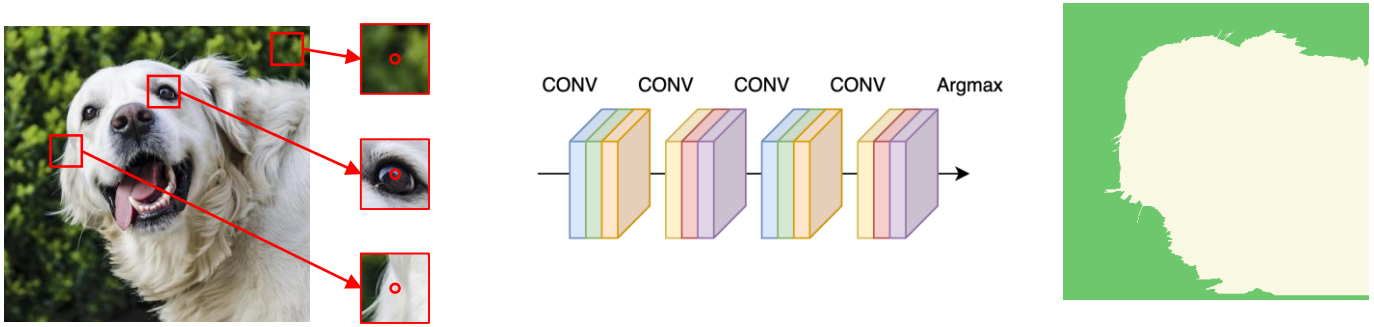
8. Class imbalance

(e.g., in this image, only a very small fraction of the pixels correspond to the human region)

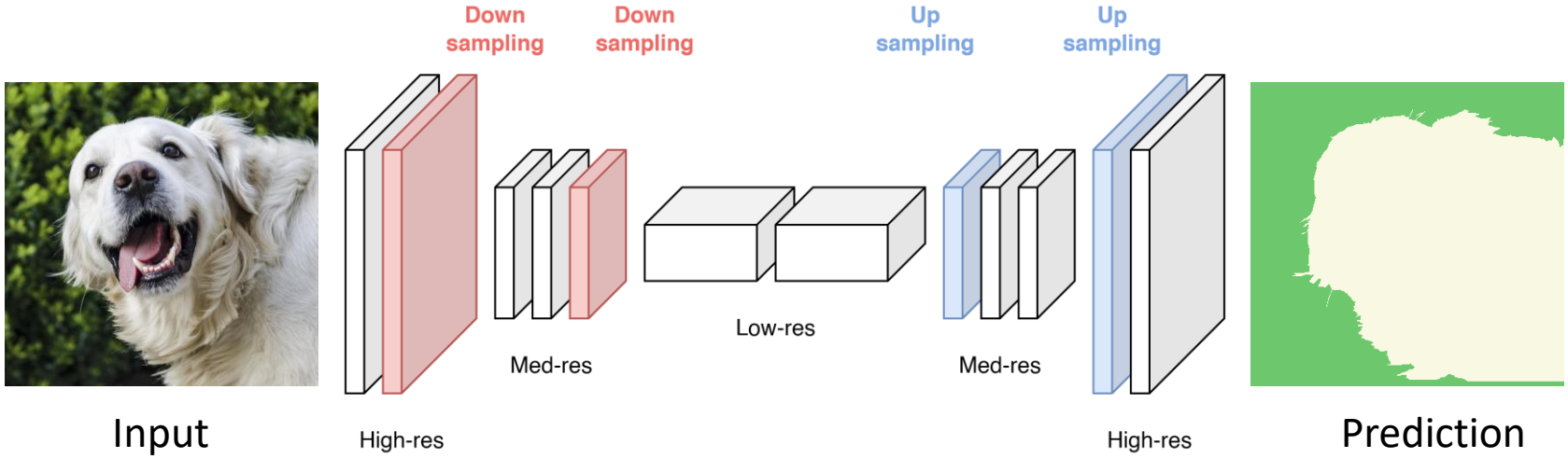
Segmentation

General flow

- **Sliding window + Center pixel classification strategy**



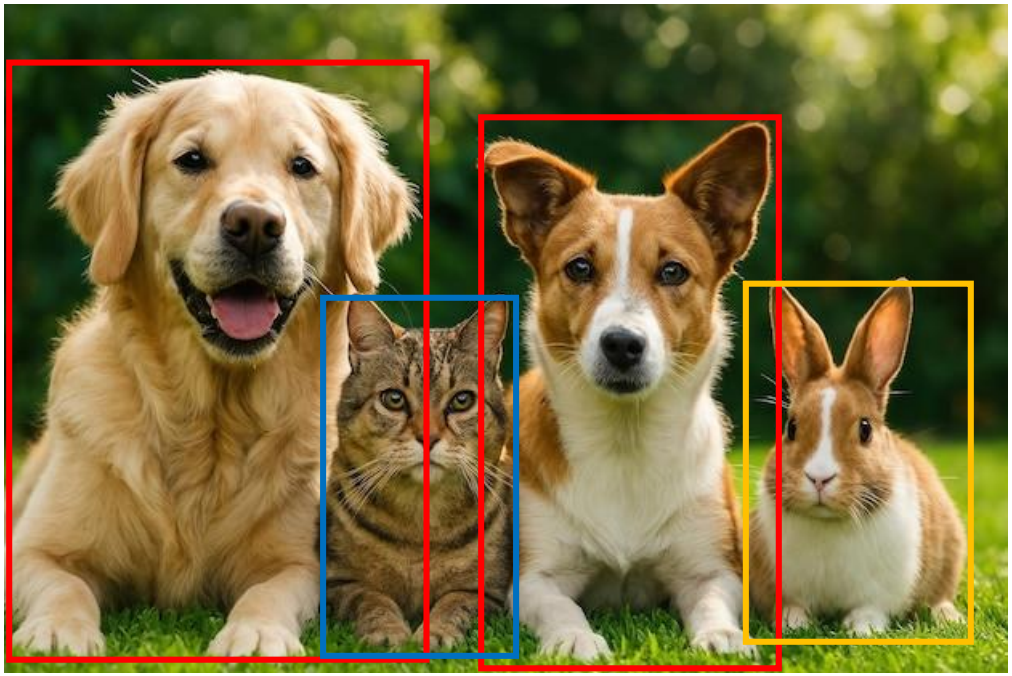
- **Fully convolutional network with both down-sampling and up-sampling**



Object detection

Object detection

- **Goal: identify what objects are present and where they are**
  - Output = bounding boxes + class labels (Localization + classification)
  - Challenges: scale variation, occlusion, dense scenes

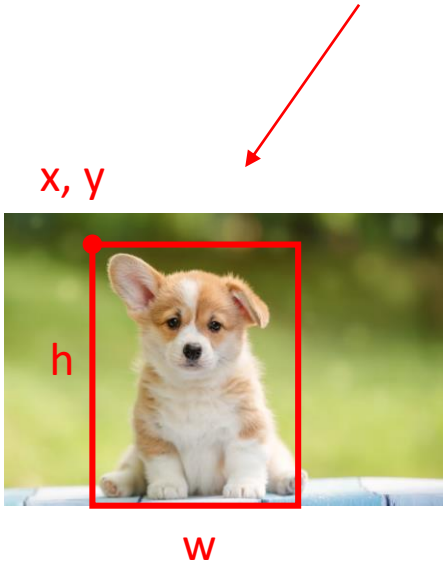


{Dog, Cat, Rabbit}

Object detection

Single object

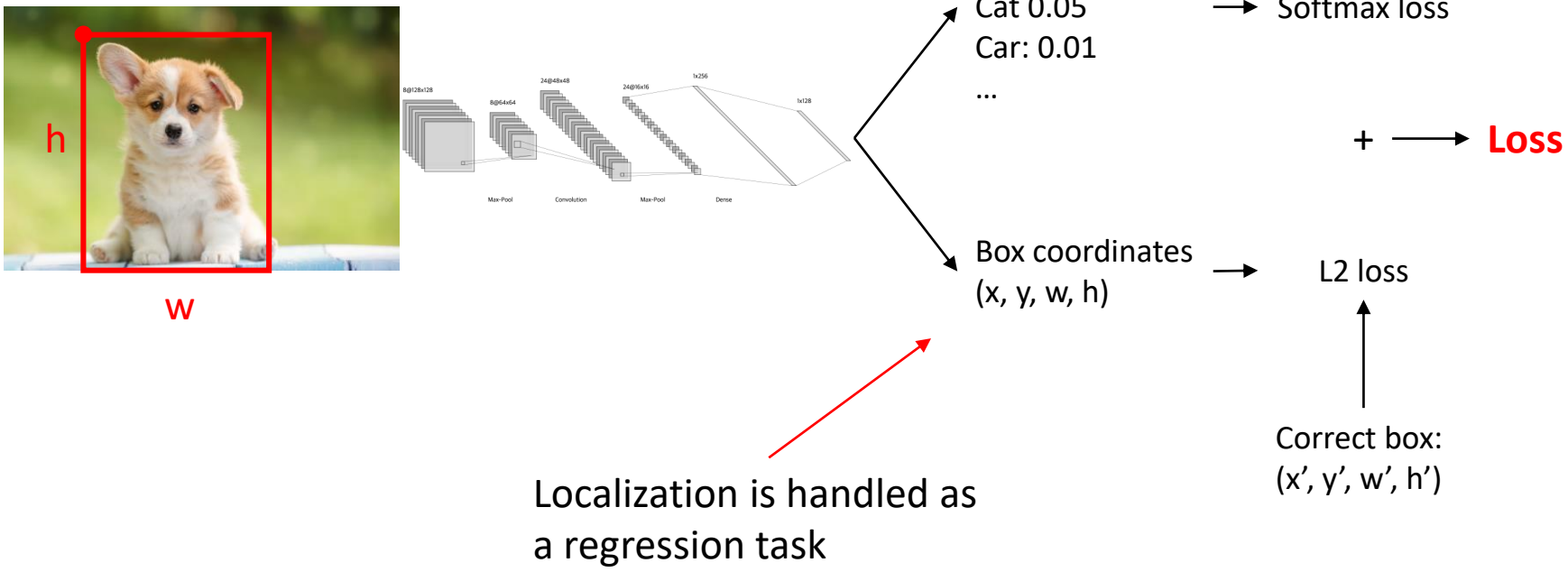
- Bounding box → left upper (x, y) + (height, width)



Object detection

Single object

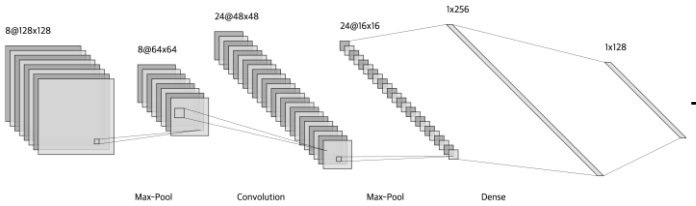
- Bounding box → left upper (x, y) + (height, width)
- Multi-task approach



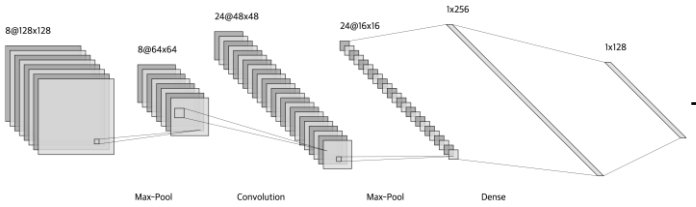
Object detection

Multiple objects

- Each image needs a different number of outputs



DOG: (x,y,w,h)

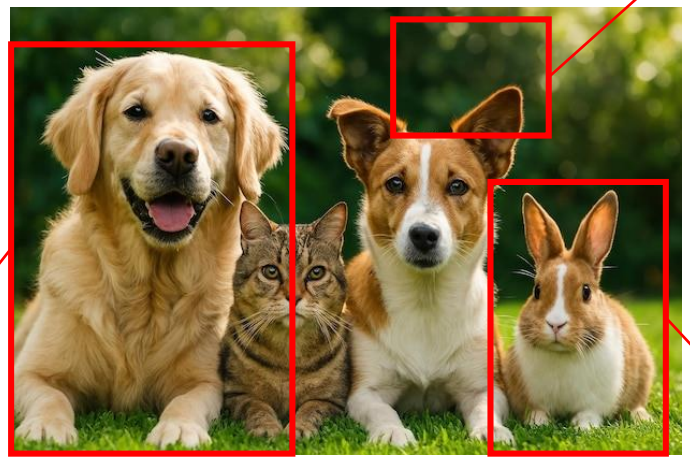


DOG: (x,y,w,h)
DOG: (x,y,w,h)
CAT: (x,y,w,h)
RABBIT: (x,y,w,h)

Object detection

Multiple objects

- If we apply a CNN to many different crops of the image, CNN classifies each crop as object or background
  → We are going to apply CNN to huge number of locations, scales, and aspect ratios → very computationally expensive



Dog? No
Cat? No
Rabbit? No
Background? Yes

Dog? Yes
Cat? No
Rabbit? No
Background? No

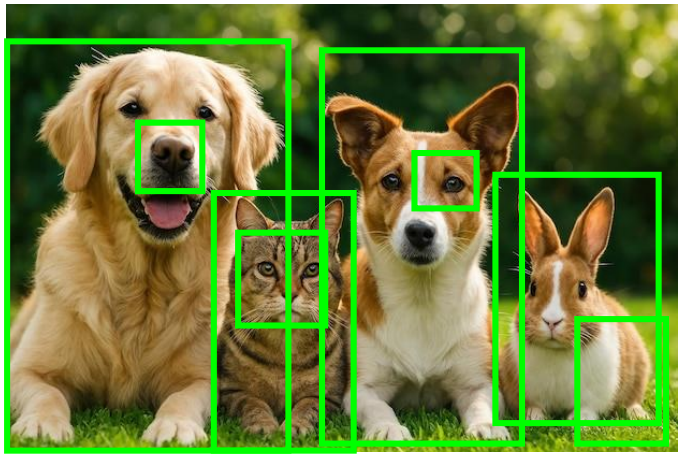
Dog? No
Cat? No
Rabbit? Yes
Background? No

Object detection

Multiple objects

- So we adopt **region proposals** to conduct selective search
  → Finding image regions that are likely to contain object

  - Relatively fast to run

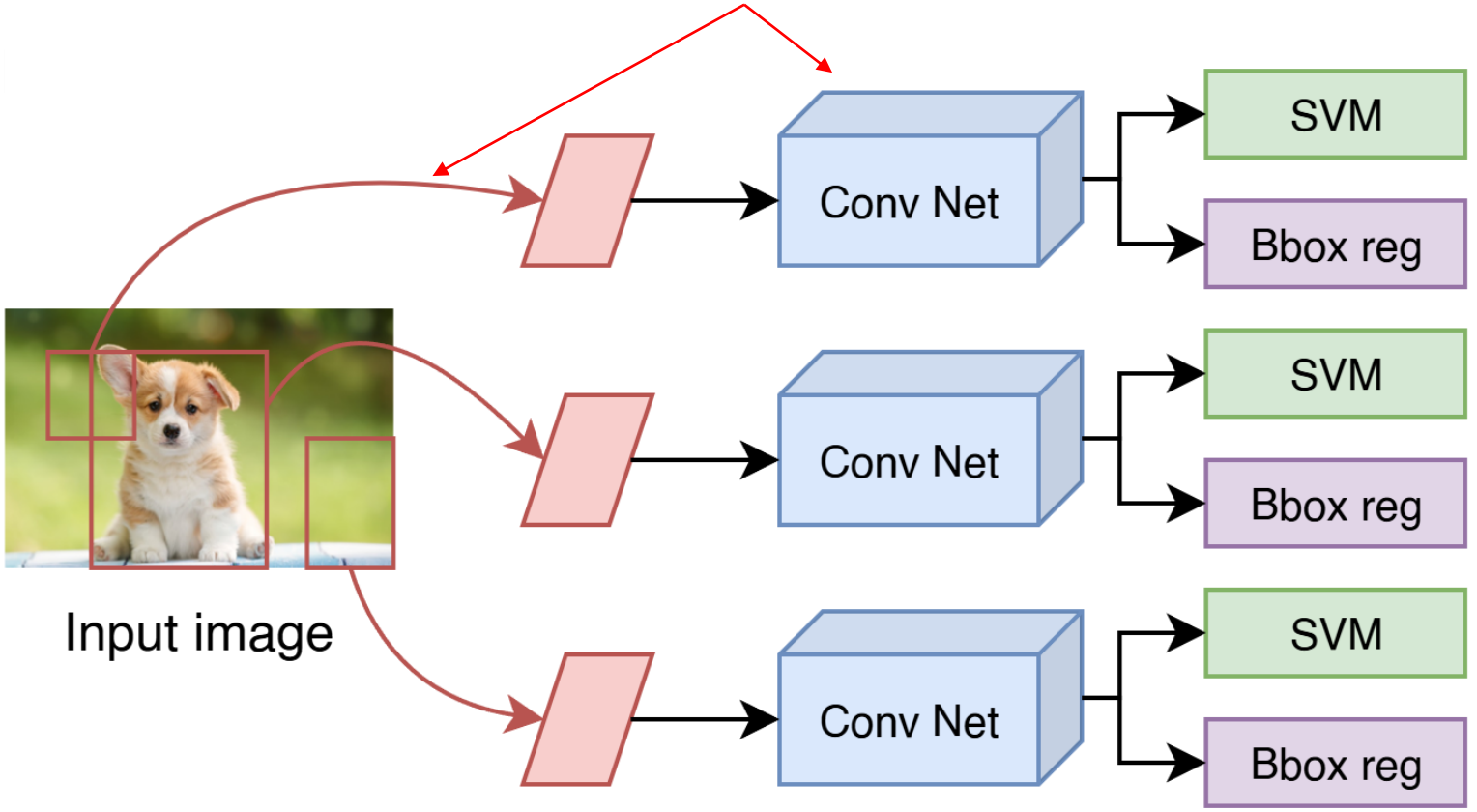


Object detection

Case study

- R-CNN

Two-stage approach
(Region proposal + Object Recognition)

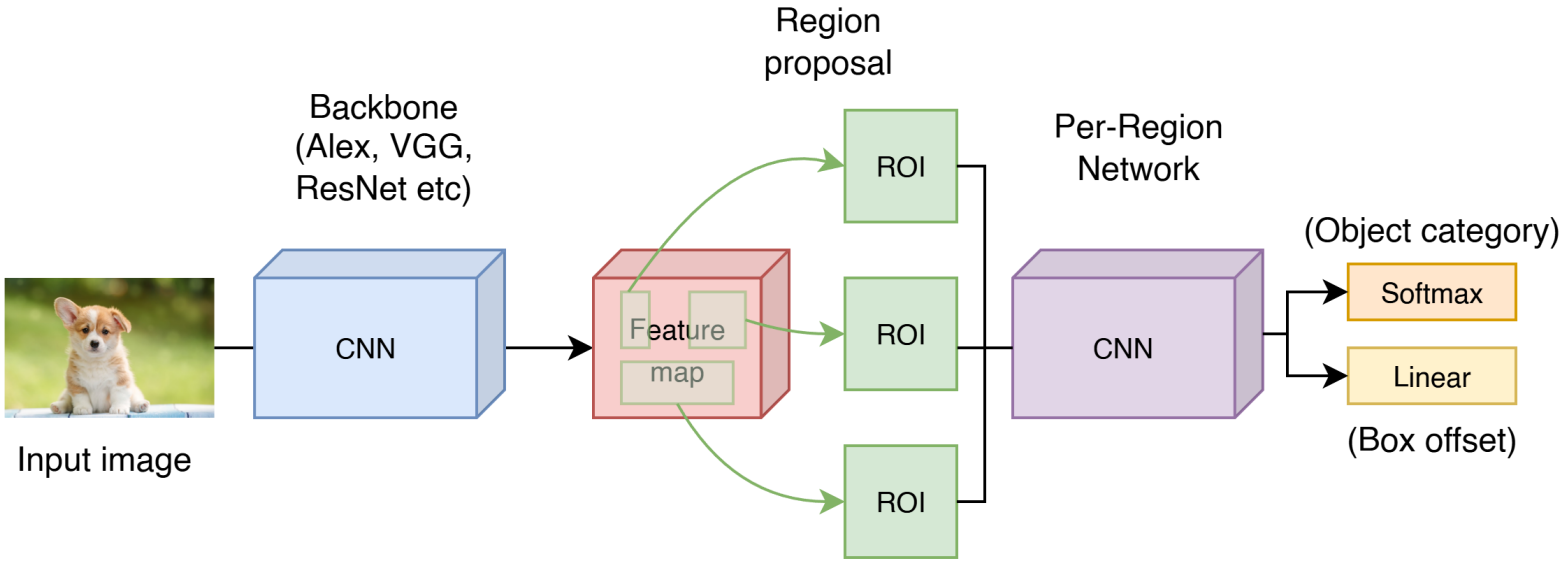


Girshick, Ross, et al. "Rich feature hierarchies for accurate object detection and semantic segmentation." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014.

Object detection

Case study

- Fast R-CNN
- Two-stage approach (Region proposal + Classification & Localization)



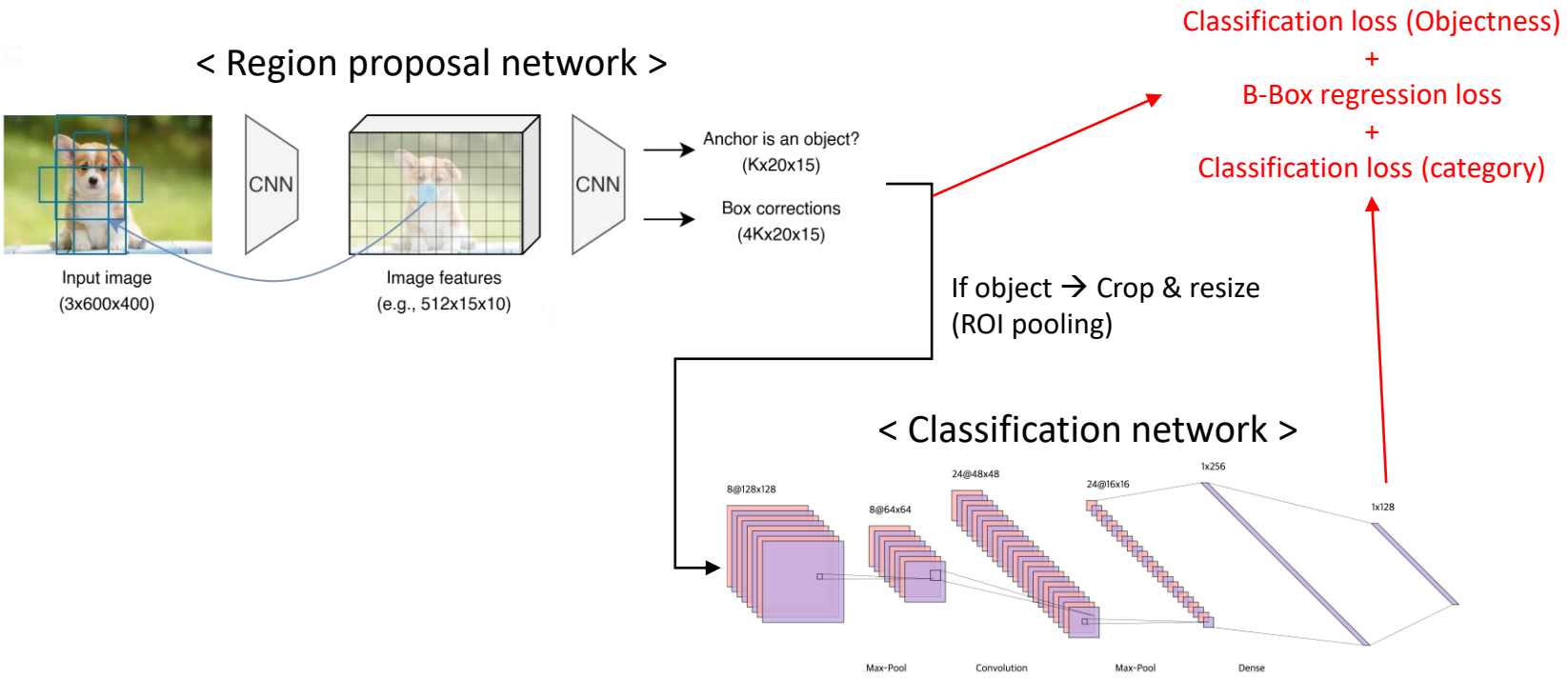
- Limitation: Still using conventional region proposal algorithm

Girshick, Ross. "Fast r-cnn." Proceedings of the IEEE international conference on computer vision. 2015.

Object detection

Faster R-CNN

- Architecture – Two-stage approach



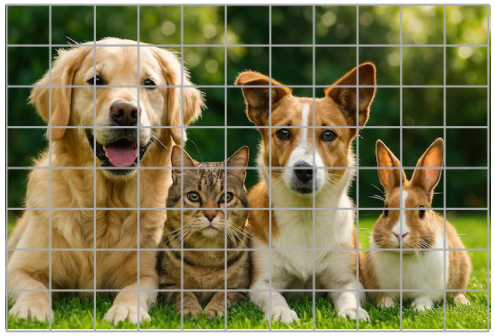
Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).

Object detection

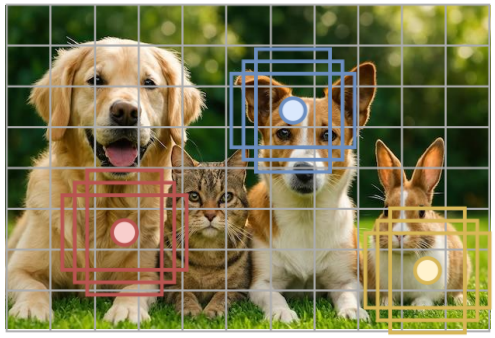
Single-stage object detector

- Exmaple) YOLO, SSD, RetinaNet
  - Predict classes and boxes directly on dense anchors/features.
  - Trade-off: one-stage is generally faster; two-stage (like Faster R-CNN) typically offers higher localization accuracy.

Image (3xHxW)



Divide image into grid



Predict set of base boxes centered at each grid cell Here B = 3

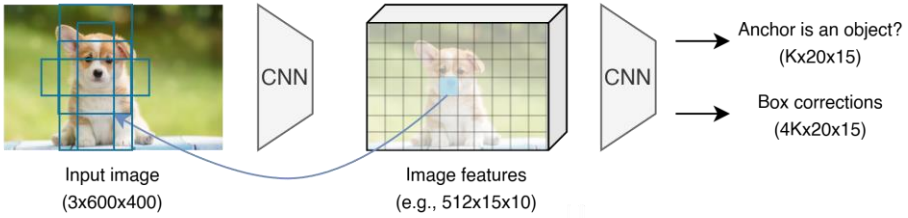
Object detection

Instance segmentation

- Mask R-CNN
- He, Kaiming, et al. "Mask r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.

(Faster R-CNN)

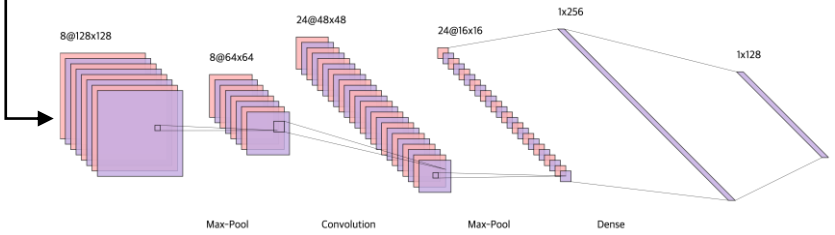
< Region proposal network >



Idea:
Add a small mask network that operates on each ROI and predicts a small binary mask

If object → Crop & resize
(ROI pooling)

< Classification network >



Object detection

Instance segmentation

- Mask R-CNN

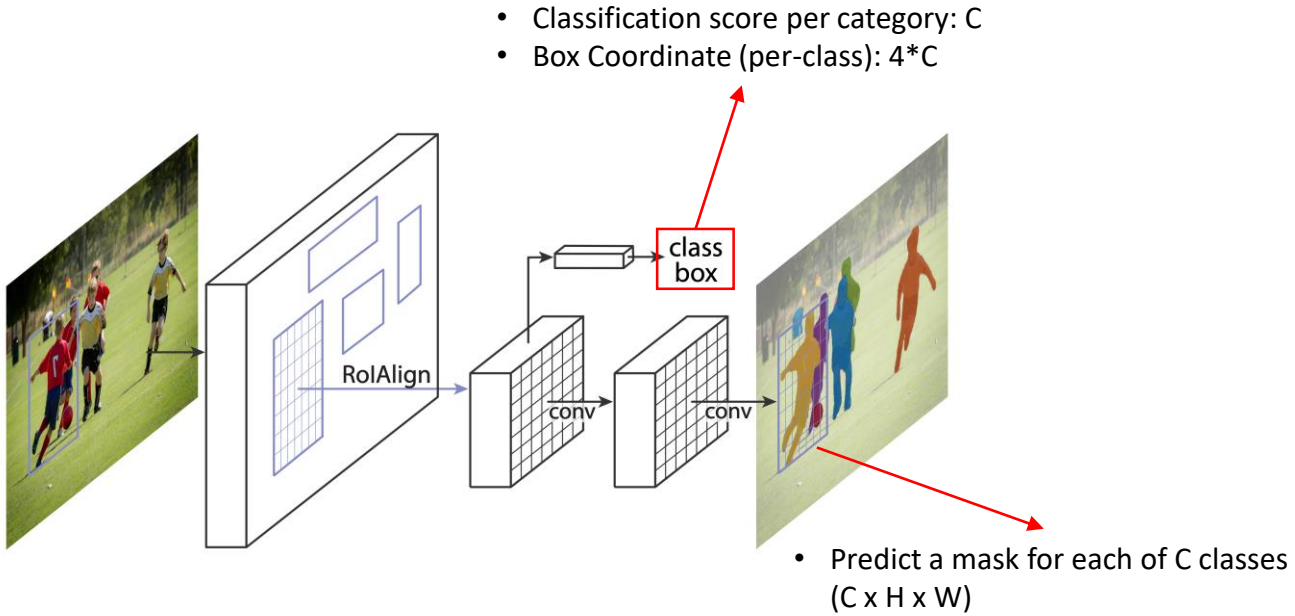


Figure 1. The **Mask R-CNN** framework for instance segmentation.

He, Kaiming, et al. "Mask r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.

Object detection

Instance segmentation

- Mask R-CNN

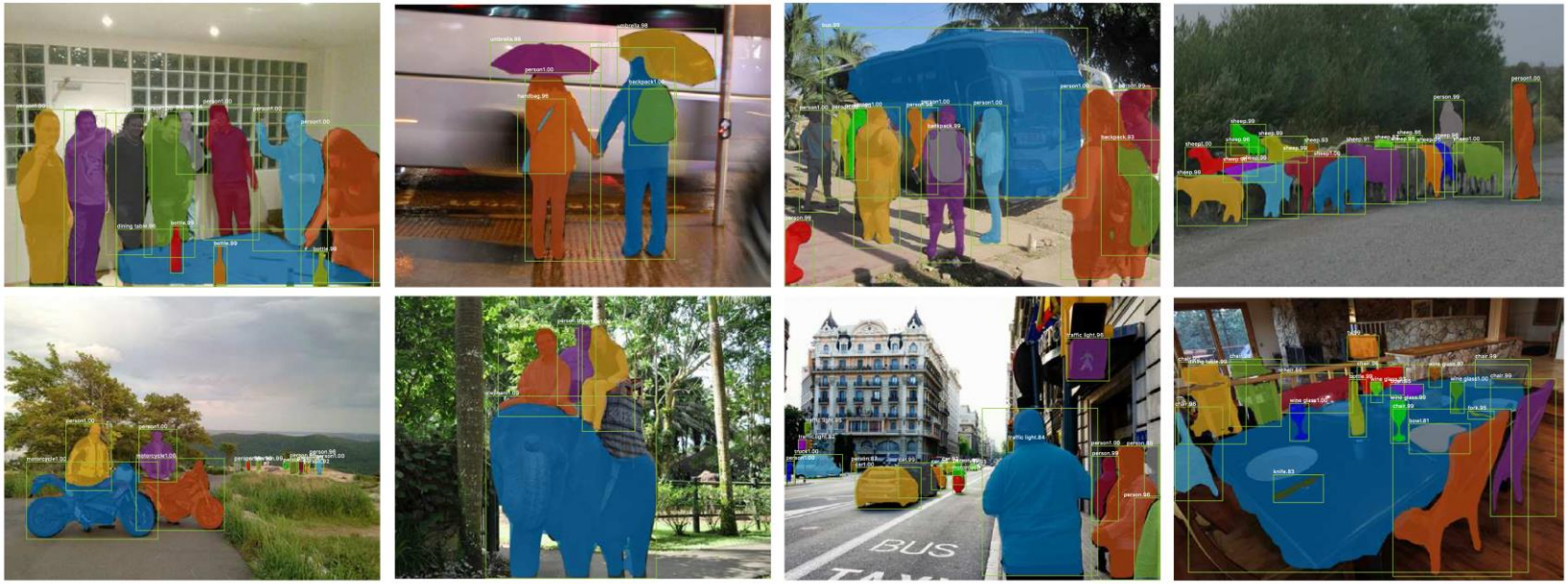


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

He, Kaiming, et al. "Mask r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.

Toy example: MNIST

Nearest Neighbor Classifier (Non-parametric model example)

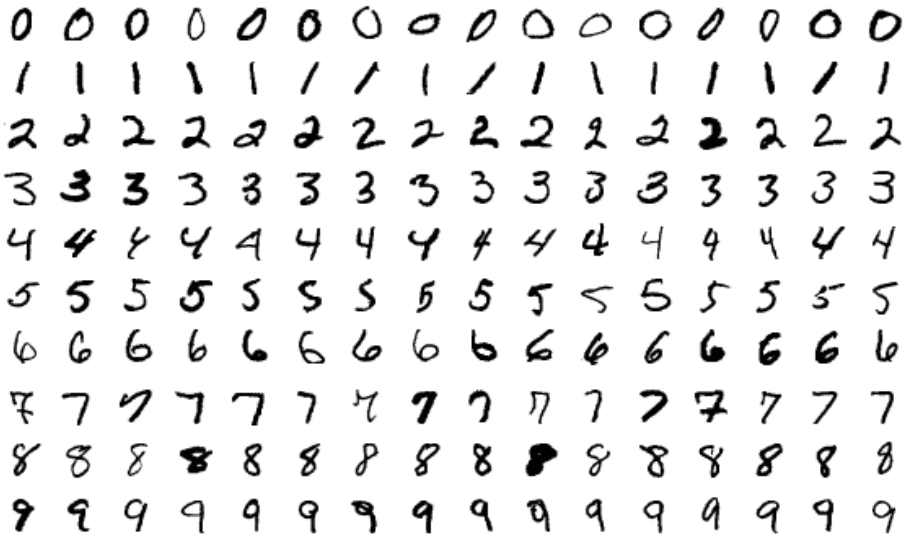
Machine learning 맛보기

Nearest Neighbor Classifier

Nearest Neighbor Classifier

- Example image classification dataset: MNIST
  - Data Included: Handwritten digit images (0-9).
  - Number of Images: 70,000 (60,000 training images and 10,000 test images).
  - Image Size: 28x28 pixels.
  - Number of Classes: 10 (digits 0-9).
  - Difficulty Level: Relatively easy. It is often used as a beginner dataset for deep learning due to its simplicity.

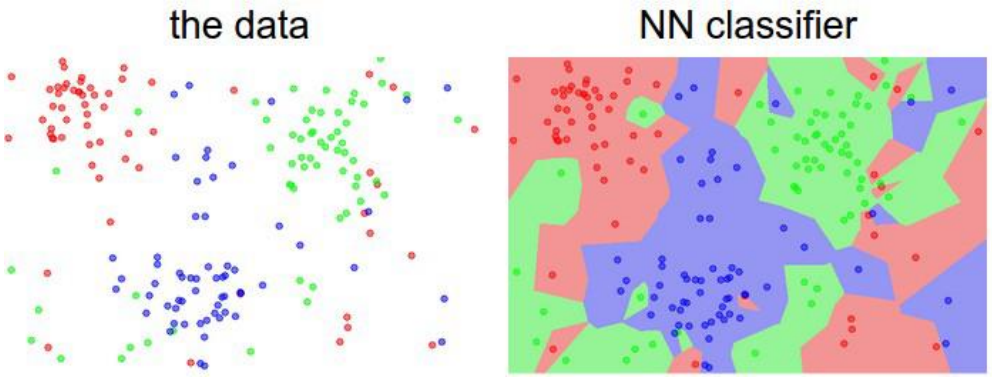
https://keras.io/api/datasets/mnist/



Nearest Neighbor Classifier

Nearest Neighbor Classifier

- The most simple and basic classifier
- It is very rarely used in practice, but it will allow us to get an idea about the basic approach to an classification problem



Nearest Neighbor Classifier

Nearest Neighbor Classifier

- Assumption: Data points that are close to (or similar to) each other will have the same label.

- Classification rule

  1. Collect training images and save in database

  2. Take a test image

  3. Compare it to every single one of the training images

  4. Predict by assigning the label of the "**closest**" training image

Nearest Neighbor Classifier

How can we compare two images?

- Simple metric: Distance
  - An image is numerical matrix → can be represented as a vector
  - L1 distance between two images can be defined:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

where, I - image, p - pixel, d_1 - L1 distance.

- Ex>

test image

56	32	10	18
90	23	128	133
24	26	178	200
2	0	255	220

-

training image

10	20	24	17
8	10	89	100
12	16	178	170
4	32	233	112

=

pixel-wise absolute value differences

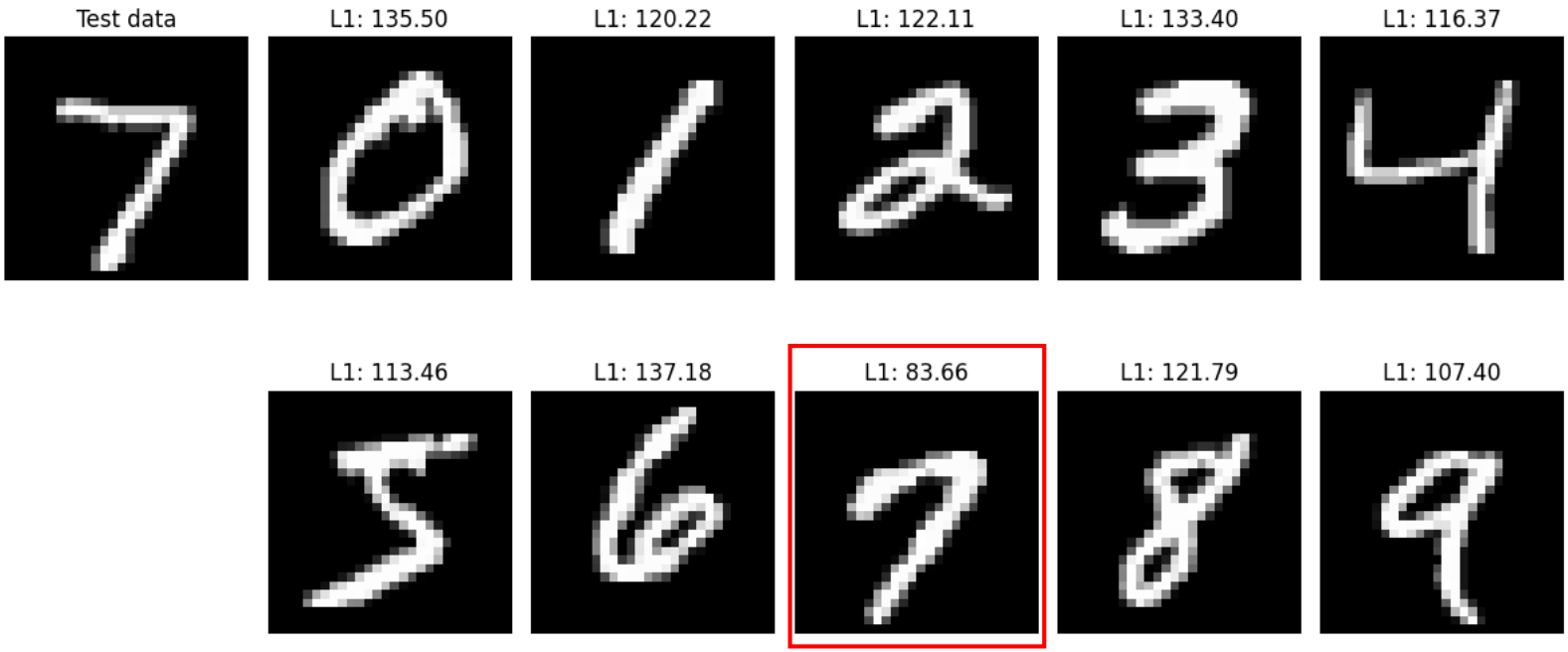
46	12	14	1
82	13	39	33
12	10	0	30
2	32	22	108

→ 456

Nearest Neighbor Classifier

How can we compare two images?

- MNIST Example)



Nearest Neighbor Classifier

Python implementation

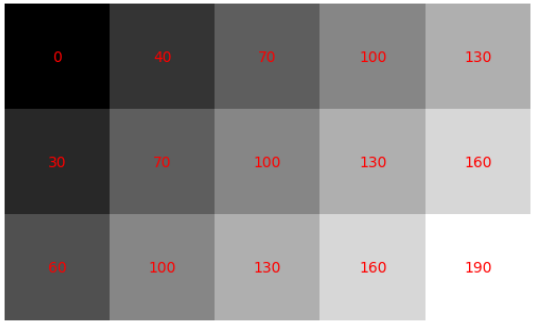
- Nearest Neighbor class

```
class NearestNeighbor():
    def __init__(self):
        pass

    def train(self, X, y): # input: flatten image and label dataset
        self.x_train = X
        self.y_train = y

    def predict(self, test_data): # input: flatten test image
        l1_distance = np.sum(np.abs(self.x_train - test_data), axis=1)
        return self.y_train[np.argmin(l1_distance)]
```

Nearest Neighbor Classifier

Image flattening

- Reshaping 2D image → 1D vector

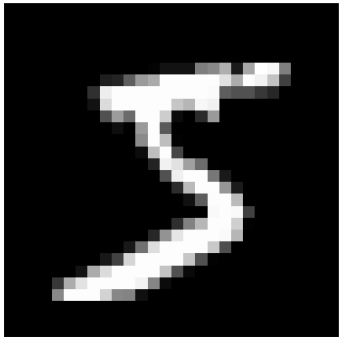


Flatten

Nearest Neighbor Classifier

Image flattening

- Reshaping 2D image → 1D vector



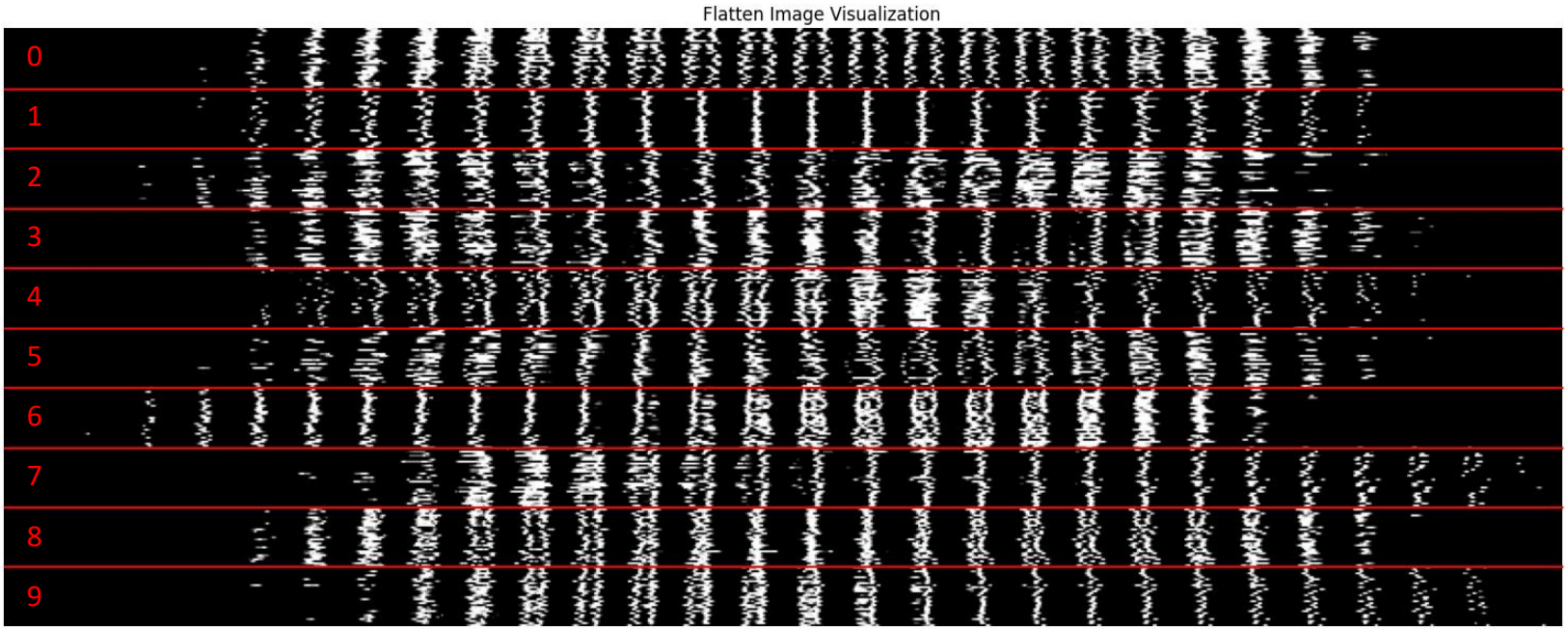
Flatten

Nearest Neighbor Classifier

Flattening result of sample cased in MNIST

We can observe similar patterns within the same digit and distinct patterns between different digits

→ **Flatten handwritten digit image can be stratified**



Nearest Neighbor Classifier

Python implementation

- Utilizing NN classifier

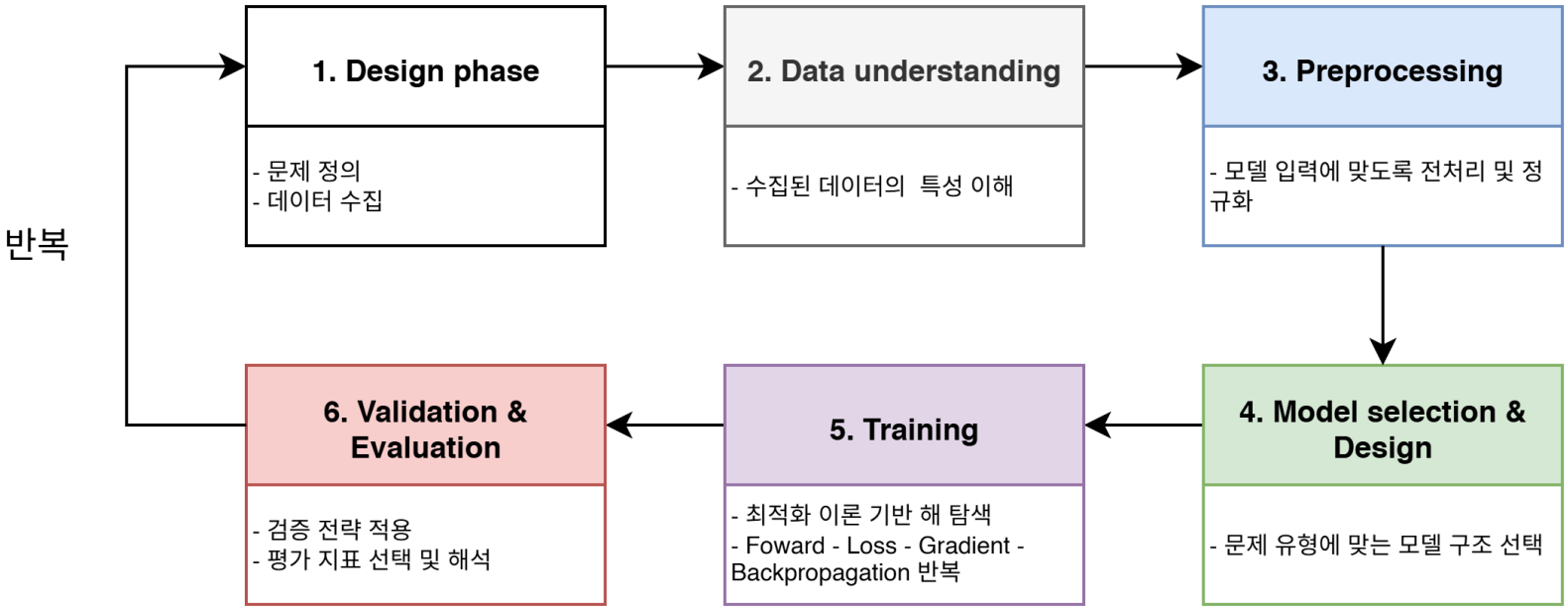
```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt

class NearestNeighbor():
    def __init__(self):
        pass

    def train(self, X, y): # input: flatten image and label dataset
        self.x_train = X
        self.y_train = y

    def predict(self, test_data): # input: a flatten test image
        l1_distance = np.sum(np.abs(self.x_train - test_data), axis=1)
        return self.y_train[np.argmin(l1_distance)]
```

```
# Load MNIST data
train_dataset = torchvision.datasets.MNIST('./', train=True,
download=True)
test_dataset = torchvision.datasets.MNIST('./', train=False,
download=True)

x_train = train_dataset.data.numpy()
y_train = train_dataset.targets.numpy()

x_test = test_dataset.data.numpy()
y_test = test_dataset.targets.numpy()

# Normalize image to be 0~1
x_train = x_train/255.0
x_test = x_test/255.0

# Flatten image
x_train = x_train.reshape(-1, 28*28)
x_test = x_test.reshape(-1, 28*28)

# Train NN classifier
nnClf = NearestNeighbor()
nnClf.train(x_train, y_train)

# Predict test sample
testIdx = 0
y_pred = nnClf.predict(x_test[testIdx])

# Results
print(f'Ground truth: {y_test[testIdx]}, Predicted: {y_pred}')
plt.imshow(x_test[testIdx].reshape(28,28), cmap='gray')
```

Practical guide
(Parametric model example)

Practical guide

Practical workflow for model development



Practical guide

Practical workflow for model development

1. Design phase – 아래 사항 검토 후 조건에 맞는 데이터 수집

- 모델의 역할 정의
  - 이 모델은 무엇을 예측/분류/추정하는가?
  - 기존 방법으로는 왜 부족하며, 모델이 해결해야 할 핵심 문제는 무엇인가?

- 데이터 요구사항 명확화
  - 해당 역할을 수행하기 위해 어떤 데이터가 필요한가?
  - 입력 데이터(X)와 출력(target, Y)은 무엇인가?

- 데이터의 현실성 검토
  - 실제 환경에서 수집 가능한 데이터는 무엇인가?
  - 학습 시점과 실제 사용 시점(inference)의 데이터 분포는 일치하는가?

- 모델 형태와 데이터 구조의 정합성
  - 목적에 적합한 모델 유형은 무엇인가?
  - 해당 모델이 요구하는 데이터 형식(shape, dimension, annotation 수준)은 무엇인가?

Practical guide

Practical workflow for model development

1. Design phase – 아래 사항 검토 후 조건에 맞는 데이터 수집

- 모델의 역할 정의
  - 이 모델은 무엇을 예측/분류/추정하는가? → 손글씨 숫자 분류
  - 기존 방법으로는 왜 부족하며, 모델이 해결해야 할 핵심 문제는 무엇인가?

- 데이터 요구사항 명확화
  - 해당 역할을 수행하기 위해 어떤 데이터가 필요한가? → 손글씨 숫자 이미지
  - 입력 데이터(X)와 출력(target, Y)은 무엇인가? → X: 이미지, Y: 0~9

- 데이터의 현실성 검토
  - 실제 환경에서 수집 가능한 데이터는 무엇인가? → MNIST
  - 학습 시점과 실제 사용 시점(inference)의 데이터 분포는 일치하는가?

- 학습 방식, 모델 형태와 데이터 구조의 정합성
  - 목적에 적합한 모델 유형은 무엇인가? → Multi-class classification model (Sup)
  - 해당 모델이 요구하는 데이터 형식(shape, dimension, annotation 수준)은 무엇인가? → Flatten image, 28x28, 이미지 1개당 1개 label (One-hot encoded)

Practical guide

Practical workflow for model development

1. Design phase – 아래 사항 검토 후 조건에 맞는 데이터 수집

결과:

```
import torch
import torchvision

# Load MNIST data
train_dataset = torchvision.datasets.MNIST('./', train=True, download=True)
test_dataset = torchvision.datasets.MNIST('./', train=False, download=True)
```

Practical guide

Practical workflow for model development

2. Data understanding

- 모달리티 파악
  - 예: RGB 이미지, X-ray, CT, MRI, 센서 데이터, 현미경 영상 등
- 데이터 자체의 물리적·통계적 특성 분석
  - 밝기값 분포, 다이나믹 레인지
  - 노이즈 수준, 아티팩트, 해상도, 콘트라스트
- 시각적 검증
  - 샘플 데이터 시각화
  - 이상치(outlier), 잘못된 라벨, 품질 문제 확인
- 이 단계에서의 이해 부족은 이후 모든 단계의 성능 한계를 결정함

Practical guide

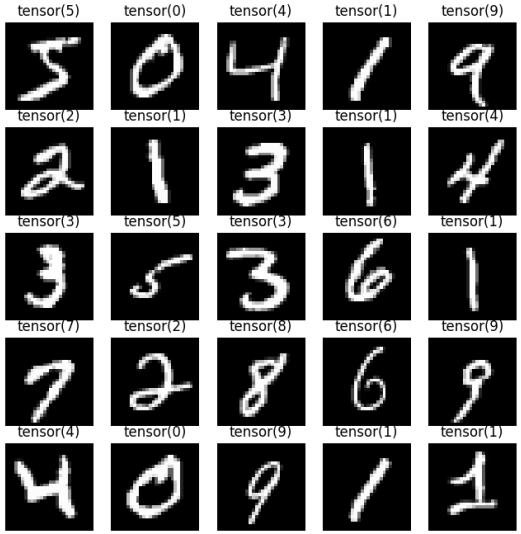
Practical workflow for model development

2. Data understanding

- 모달리티 파악 → 28x28 단채널 흑백 이미지
  - 예: RGB 이미지, X-ray, CT, MRI, 센서 데이터, 현미경 영상 등

- 데이터 자체의 물리적·통계적 특성 분석 → Tr 60,000, Test 10,000, Balanced label
  - 밝기값 분포, 다이나믹 레인지 → 0~255 uint8 Tensor
  - 노이즈 수준, 아티팩트, 해상도, 콘트라스트 → 왜곡 거의 없음

- 시각적 검증
  - 샘플 데이터 시각화
  - 이상치(outlier), 잘못된 라벨, 품질 문제 확인



Practical guide

Practical workflow for model development

2. Data understanding

```
import matplotlib.pyplot as plt
import numpy as np

print(train_dataset.data.shape)
print(train_dataset.data.dtype)
print(train_dataset.data.min())
print(train_dataset.data.max())

plt.close('all')
plt.figure(figsize=(8,8))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(train_dataset.data[i].numpy(), cmap='gray')
    plt.title(train_dataset.targets[i])
    plt.axis('off')
```

Practical guide

Practical workflow for model development

3. Preprocessing

- 모델 입력에 맞는 정규화
  - intensity normalization, scaling, standardization, registration, reconstruction 등
- 의미 보존과 공정성의 균형
  - 각 데이터가 고유한 의미를 유지하면서도
  - 모델 학습에서 특정 데이터가 과도한 영향을 주지 않도록 조정
- 재현 가능한 파이프라인 구성
  - 학습/검증/테스트에 동일한 전처리 적용
  - inference 단계에서도 동일하게 적용 가능해야 함

Practical guide

Practical workflow for model development

3. Preprocessing

- 모델 입력에 맞는 정규화
  - intensity normalization, scaling, standardization, registration, reconstruction 등

  → Numpy로 변환 (필수x), 0~1 min-max norm, image flattening, label one-hot encoding (필요시)
- 의미 보존과 공정성의 균형
  - 각 데이터가 고유한 의미를 유지하면서도
  - 모델 학습에서 특정 데이터가 과도한 영향을 주지 않도록 조정

- 재현 가능한 파이프라인 구성
  - 학습/검증/테스트에 동일한 전처리 적용
  - inference 단계에서도 동일하게 적용 가능해야 함

Practical guide

Practical workflow for model development

3. Preprocessing

```
# change to numpy
train_x = train_dataset.data.numpy()
train_y = train_dataset.targets.numpy()
test_x = test_dataset.data.numpy()
test_y = test_dataset.targets.numpy()

# intensity normalization
train_x = train_x / 255.0
test_x = test_x / 255.0

# flatten images
train_x = train_x.reshape(-1, 28*28)
test_x = test_x.reshape(-1, 28*28)
```

Practical guide

Practical workflow for model development

4. Model selection & Design

- 문제 유형에 맞는 모델 구조 선택
  - Regression
  - Binary / Multi-class Classification
  - Segmentation
  - Detection, Survival analysis 등
- 모델 복잡도와 데이터 규모의 균형
  - 데이터 수 대비 파라미터 수
  - 과적합 가능성 고려
- 도메인 특성 반영
  - 의료 영상, 산업 영상, 자연 영상 등 도메인별 특화 구조 고려

Practical guide

Practical workflow for model development

4. Model selection & Design

- 문제 유형에 맞는 모델 구조 선택 → Multi-class classification 가능한 model
  - Regression
  - Binary / Multi-class Classification
  - Segmentation
  - Detection, Survival analysis 등

- 모델 복잡도와 데이터 규모의 균형 → Linear softmax classifier (28*28+1 param)
  - 데이터 수 대비 파라미터 수
  - 과적합 가능성 고려

- 도메인 특성 반영 → 이번에는 특별히 해당사항 없음
  - 의료 영상, 산업 영상, 자연 영상 등 도메인별 특화 구조 고려

Practical guide

Practical workflow for model development

4. Model selection & Design

모델 선택: 10 개 class에 대한 score 출력가능한 linear model

$$f(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}$$

- $\boldsymbol{X}$ - flatten images (nx784 numbers), $\boldsymbol{W}$ – 784 x 10 weight matrix, $\boldsymbol{b}$ – 1 x 10

```
class myClassifier(nn.Module):
    def __init__(self):
        super().__init__()
        self.w = nn.Parameter(torch.randn(28*28, 10), requires_grad=True)
        self.b = nn.Parameter(torch.randn(10), requires_grad=True)

    def forward(self, X):
        return X @ self.w + self.b

    def __call__(self, X):
        return self.forward(X)
```

Practical guide

Practical workflow for model development

5. Training

- Traditional models
  - 통계적 가정, 최적화 이론 기반 해 탐색
- Modern models
  - Loss 선정 (training process guide)
  - Forward (Inference) → Loss 계산 → Gradient 계산 → Error Backpropagation 반복
- Hyperparameters
  - Epochs, learning rate, batch size, regularization
  - class imbalance, data leakage 점검
  - Validation Loss 추적

Practical guide

Practical workflow for model development

5. Training

- Traditional models
  - 통계적 가정, 최적화 이론 기반 해 탐색

- Modern models
  - Loss 선정 (training process guide) → Cross entropy
  - Forward (Inference) → Loss 계산 → Gradient 계산 → Error Backpropagation 반복 → 구현

- Hyperparameters
  - Epochs, learning rate, batch size, regularization → 100 epochs, lr 1e-3, batch size 128
  - class imbalance, data leakage 점검 → 없음
  - Validation Loss 추적 → overfitting 감지, validation loss 기준으로 최적 epoch 결정

Practical guide

Practical workflow for model development

5. Training

(참고, optimizer 사용 x)

```
myMdl = myClassifier()
crierion = nn.CrossEntropyLoss()
lr = 1e-3 # learing rate
batch_size = 128
EPOCHS = 100

for epoch in range(EPOCHS):
    epoch_loss = 0.0 # for monitoring
    n_samples = 0

    myMdl.train() # training mode, trun on dropout, BN etc
    for i in range(0, len(train_x), batch_size):
        # batch batch data
        batch_img = train_x[i:i + batch_size]
        batch_label = train_y[i:i + batch_size]

        # Forward -> loss calculation -> gradient calculation -> backpropagation
        logits = myMdl(batch_img) # forward
        loss = crierion(logits, batch_label) # loss calculation

        # loss.backward() computes the gradient of loss with respect to every tensor
        # that has requires_grad=True and was involved in producing loss,
        # and stores those gradients in .grad.
        loss.backward()

        # Error backpropagation
        with torch.no_grad(): # without gradient recording
            for p in myMdl.parameters(): # for each parameters
                p -= lr * p.grad # gradient descent
                p.grad.zero_() # reset gradient

        # accumulate training loss
        bs = batch_img.size(0)
        epoch_loss += loss.item() * bs
        n_samples += bs

    # Validation loss
    myMdl.eval() # evaluation mode, trun off dropout, BN etc
    with torch.no_grad():
        epoch_loss_val = 0.0 # for monitoring
        n_samples_val = 0
        for i in range(0, len(test_x), batch_size):
            # batch batch data
            batch_img = test_x[i:i + batch_size]
            batch_label = test_y[i:i + batch_size]

            # Forward -> loss calculation
            logits = myMdl(batch_img) # forward
            loss = crierion(logits, batch_label) # loss calculation

            # accumulate training loss
            bs = batch_img.size(0)
            epoch_loss_val += loss.item() * bs
            n_samples_val += bs

    print(f"Epoch {epoch+1:03d} | loss: {epoch_loss / n_samples:.4f} | val_loss: {epoch_loss_val / n_samples_val:.4f}")
```

Practical guide

Practical workflow for model development

6. Validation & Evaludation

- 검증 전략
  - 전략은 design 단계에서 결정. 이 단계에서는 직접 수행
  - Train/Validation/Test 분리
  - Cross-validation (필요 시 patient-level split 등)

- 평가지표 선택
  - 문제 정의에 부합하는 metric 사용 (AUC, Dice, MAE, C-index 등)
  - 결과 해석

- 단순 성능 수치뿐 아니라
  - 실패 사례, 편향, 일반화 가능성 분석

Practical guide

Practical workflow for model development

6. Validation & Evaluation

- 검증 전략
  - 전략은 design 단계에서 결정. 이 단계에서는 직접 수행
  - Train/Validation/Test 분리 → 샘플에서는 train/validation 만 사용
  - Cross-validation (필요 시 patient-level split 등)

- 평가지표 선택
  - 문제 정의에 부합하는 metric 사용 (AUC, Dice, MAE, C-index 등)
  - 결과 해석 → 샘플에서는 accuracy를 이용해서 평가

- 단순 성능 수치뿐 아니라
  - 실패 사례, 편향, 일반화 가능성 분석

Practical guide

Practical workflow for model development

6. Validation & Evaluation

```
myMdl.eval()  # evaluation mode (turn off dropout / BN)

correct = 0
total = 0
with torch.no_grad():
    for i in range(0, len(train_x), batch_size):
        batch_img = train_x[i:i + batch_size]
        batch_label = train_y[i:i + batch_size]

        logits = myMdl(batch_img)          # (B, C)
        preds = torch.argmax(logits, dim=1)

        correct += (preds == batch_label).sum().item()
        total += batch_label.size(0)

accuracy = correct / total
print(f"Training Accuracy: {accuracy:.4f}")

correct = 0
total = 0
with torch.no_grad():
    for i in range(0, len(test_x), batch_size):
        batch_img = test_x[i:i + batch_size]
        batch_label = test_y[i:i + batch_size]

        logits = myMdl(batch_img)          # (B, C)
        preds = torch.argmax(logits, dim=1)

        correct += (preds == batch_label).sum().item()
        total += batch_label.size(0)

accuracy = correct / total
print(f"Valiidation Accuracy: {accuracy:.4f}")
```

Linear classifier
(Parametric model example)

Formal method

Linear classification

Linear classification

- **Two major components**

1. **Score function: maps the raw data (image) to class scores (i.e., model)**

f

Cat	0.12
Tiger	0.04
Dog	0.81
Elephant	0.03

2. **Loss function: quantifies the agreement between prediction and ground truth**

Cat	0.12
Tiger	0.04
Dog	0.81
Elephant	0.03

L

$$L_{CE} = -\log(0.81)$$

Linear classification

Linear classifier

- A linear mapping:

$$f(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}$$

  - Input $\boldsymbol{x}_i$: flatten image $[D \times 1]$ (or feature vector of image)
  - Model parameters
    - $\boldsymbol{W}$: weight matrix $[K \times D]$
    - $\boldsymbol{b}$: bias vector $[K \times 1]$

    where D dimensionality of images, K distinct categories

  - For example, D for MNIST (28x28) is 784, and K is 10 (the number of categories)

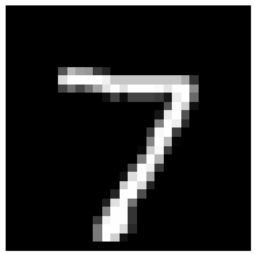
Linear classification

Linear classifier

- A linear mapping:

$$f(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}$$

- Ex>

$$\boldsymbol{W} \cdot$$  $$+\boldsymbol{b} = \boldsymbol{score}$$

$score(0)$
$score(1)$
$score(2)$
$score(3)$
$score(4)$
$score(5)$
$score(6)$
$score(7)$
$score(8)$
$score(9)$

Linear classification

Linear classifier

- A linear mapping:

$$f(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b}$$

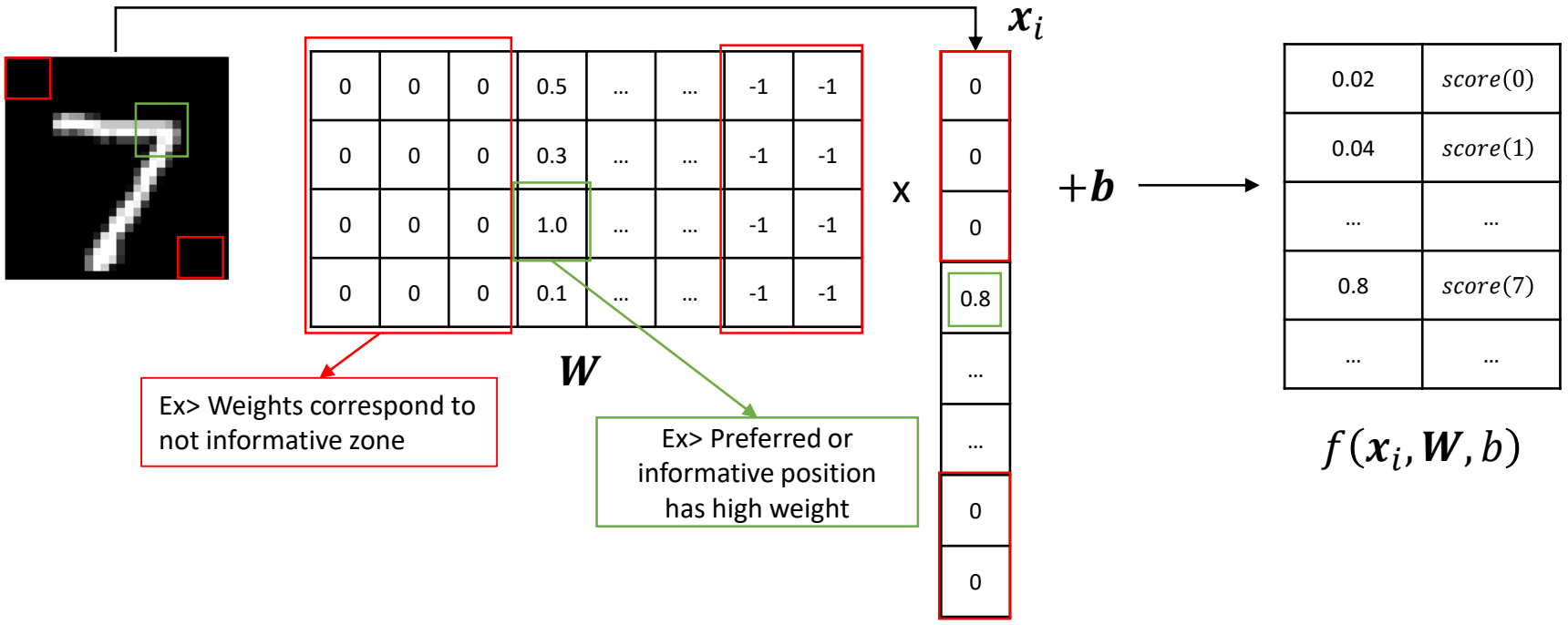
- Characteristics
  - **Parallel Classification**
    - A single matrix multiplication $\boldsymbol{W}\boldsymbol{x}_i$ evaluates K classifiers simultaneously, with each row of $\boldsymbol{W}$ representing a classifier.
  - **Parameter Optimization**
    - The input data is fixed, but $\boldsymbol{W}$ and $\boldsymbol{b}$ are learned to ensure the correct class has the highest score.
  - **Advantage of memory**
    - After the learning is complete, we can discarding the training data.
  - **Fast Classification**
    - A test image is classified using a single matrix multiplication and addition, making it significantly faster than comparing it to all training images.

Linear classification

Interpreting a linear classifier

$$f(x_i, W, b) = Wx_i + b$$

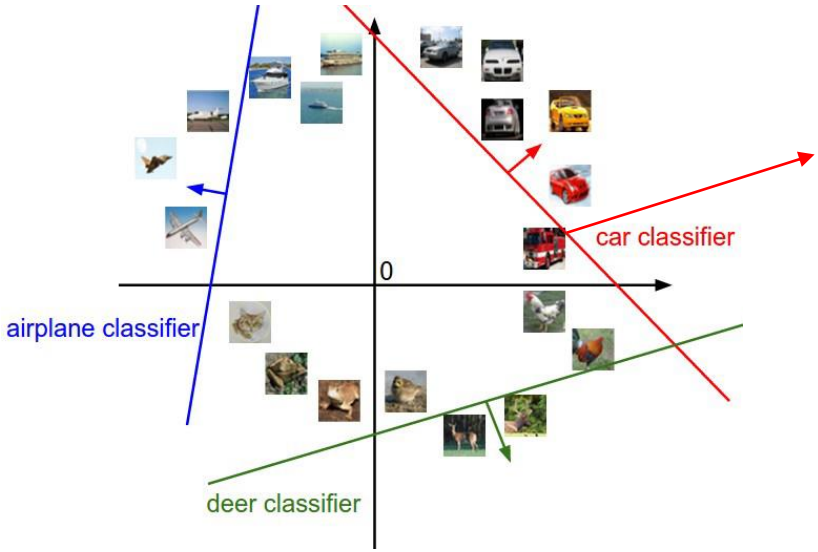
- Notice that a linear classifier computes the score of a class as a weighted sum of all of its pixel values
  - Model has capacity to like or dislike certain colors or certain position in the image
  - Ex>



Linear classification

Analogy of images as high-dimensional points

- We can consider each images (vector) as a single data point in high-dimensional space



Optimal boarder line (or trained ***W*** and ***b***) can linearly discriminate images in the high-dimensional feature space

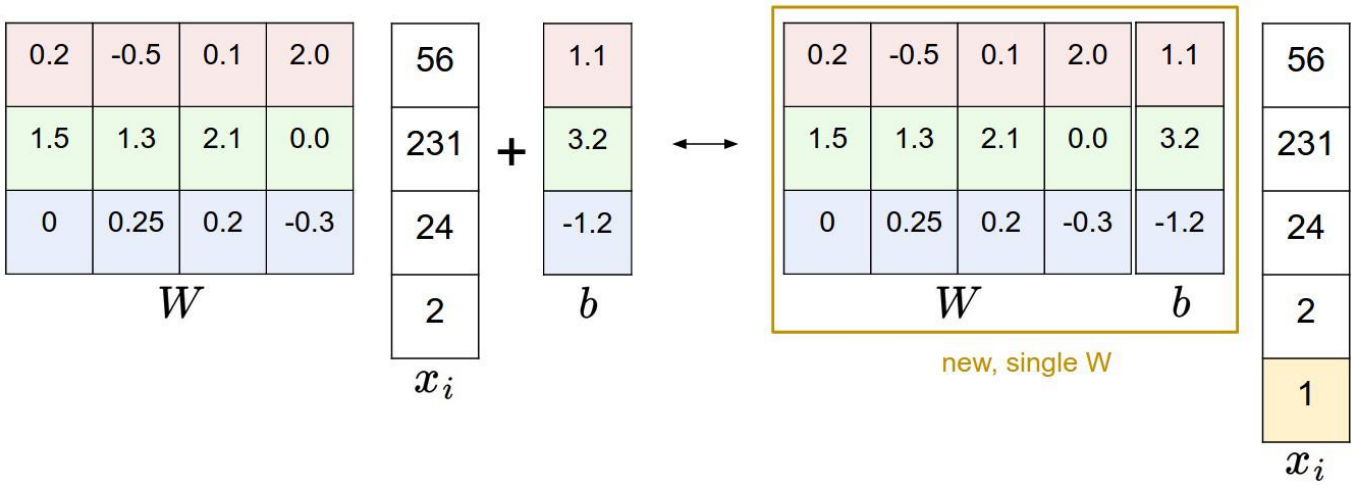
- Since we defined the score of each class as a <u>weighted sum of all image pixels</u>, **each class score is a linear function over this space**
  - $\boldsymbol{W}$: line orientation
  - $\boldsymbol{b}$: line translation
  - Without bias term, $\boldsymbol{x}_i$ of black image always give score zeros, regardless of W.

Linear classification

Bias trick

- Presenting the two parameters $\boldsymbol{W}$, $\boldsymbol{b}$ as one

$$f(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b} \rightarrow f(\boldsymbol{x}_i, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$$



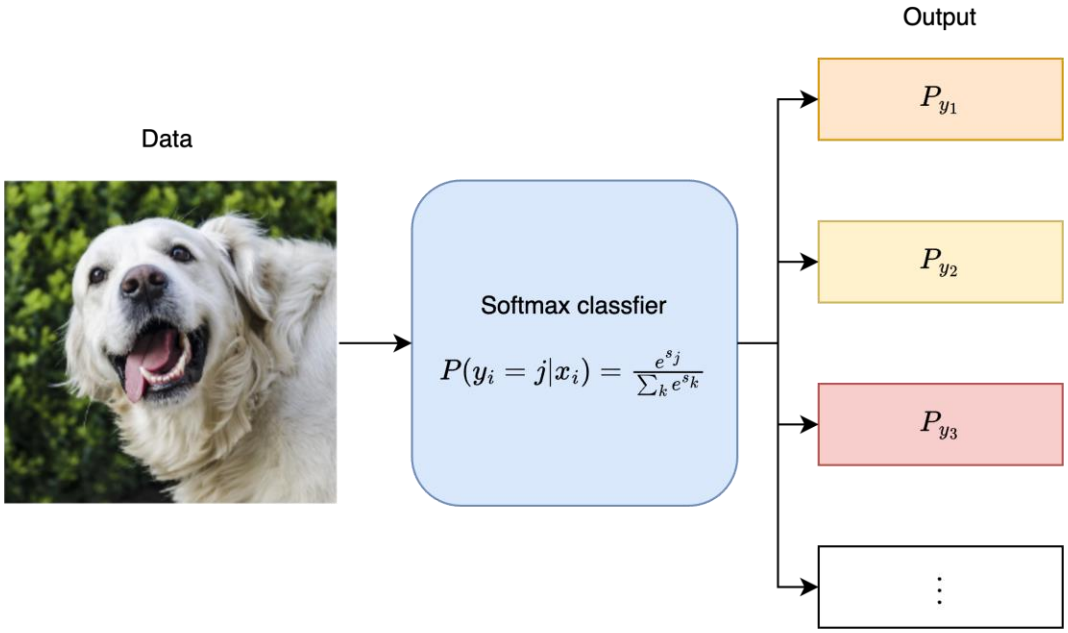
A.k.a., Homogeneous coordinate

- Common trick to combine two sets of parameters into a single matrix
  → It can make linear mapping to single operation

Softmax classifier

Softmax classifier

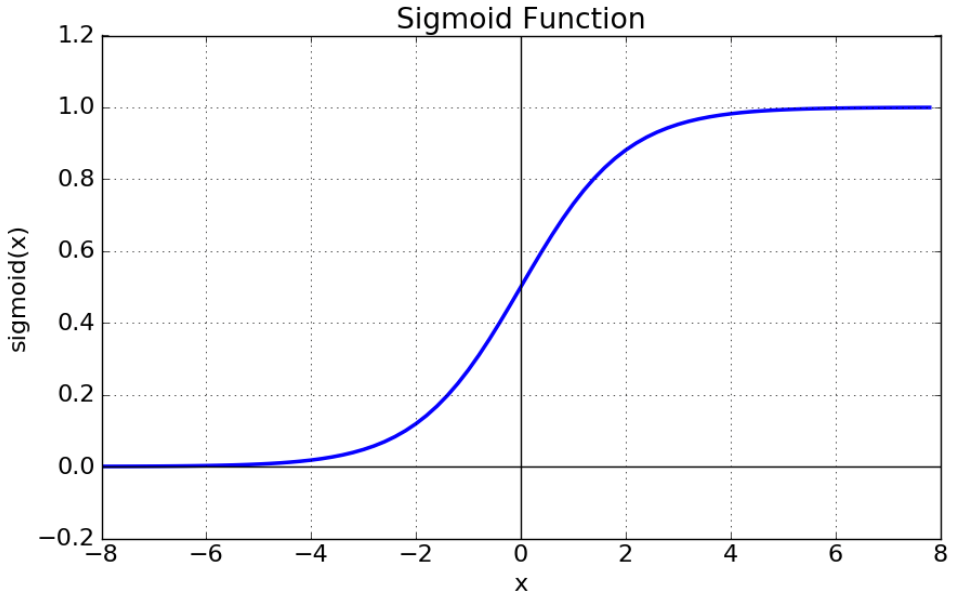
- Another popular choice for linear classifier
- Generalized version of logistic regression



Softmax classifier

Logistic regression

- Statistical method and machine learning algorithm used for binary classification problems
  - Note: Despite its name, logistic regression is used primarily for classification tasks rather than regression tasks
- Core of logistic regression: **Sigmoid function**



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- $0 \leq \sigma(x) \leq 1$
- $\sigma(-x) = 1 - \sigma(x)$

- Map feature → 0~1
- Output of sigmoid can be treated as probability
- $x = 0 (\sigma(x) = 0.5)$ can be (fair) decision boundary

Softmax classifier

Logistic regression

- Model equation

$$\hat{y} = \sigma_{\boldsymbol{w}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}\cdot\boldsymbol{x}}} = \frac{1}{1 + e^{-\sum w_j x_j}}$$

  - It contains linear mapping function $\boldsymbol{w} \cdot \boldsymbol{x}$

Softmax classifier

Softmax classifier

- Linear mapping function

$$f(\boldsymbol{x}_i; \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$$

  - Softmax classifier → Treat $f(\boldsymbol{x}_i; \boldsymbol{W})$ as unnormalized log probability
  - Utilize softmax function to normalize this output
    - Softmax function

$$\sigma(\boldsymbol{z})_i = \frac{e^{\boldsymbol{z}_i}}{\sum_{j=1}^{K} e^{\boldsymbol{z}_j}}$$

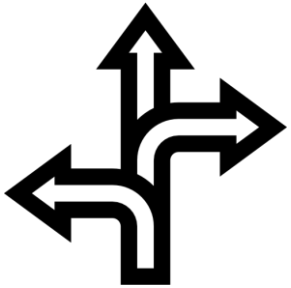
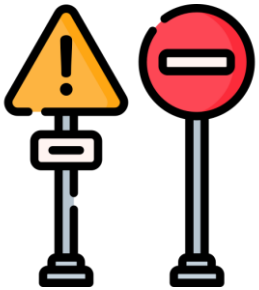
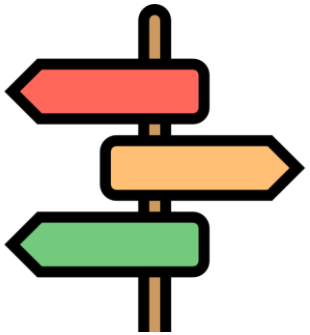
Loss function

Loss function

- Measure of loss incurred in taking any of the available decision or action
  - I.e., a mathematical function that measures the degree of difference between the actual and predicted outputs of a model
  - Also called as cost function – overall measure

Role of loss function

- Guide the optimization (learning) process by providing a measure of how well the model is performing

Loss function

Loss function

- In learning process, our goal is to **minimize the total loss**

  - The loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well

  - Some authors consider a utility function → value to maximize

- It is our choice which loss functions to use based on the purpose
  → important role of AI engineers

- In this chapter, we will focus on how to measure loss, while the optimization process will be discussed later.

Loss function

General classification loss

- Cross Entropy

  - Information: $I(x) = -\log P(x)$
  - Entropy: $H(x) = E_{x\sim P}[I(x)] = -E_{x\sim P}[\log P(x)] = -\sum P(x)\log P(x)$
  - KL-Divergence: $D_{KL}(P||Q) = E_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = E_{x\sim P}[\log P(x) - \log Q(x)]$
    - A.k.a. Relative Entropy, Difference between two distribution
  - **Cross Entropy**:

$$H(P,Q) = H(P) + D_{KL}(P||Q)$$
$$= -E_{x\sim P}[\log Q(x)] = -\sum P(x)\log Q(x)$$

    - The average amount of information we are spending to estimate true distribution P using the distribution Q
      → higher cross-entropy = worse prediction

Loss function

Binary Cross Entropy (BCE)

- Common choices for <u>binary classification</u>
- Recap> Cross entropy: $-\sum P(x) \log Q(x)$
- **Binary cross entropy loss**

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

  - For i^{th} data whose label is 1, $P(x) = y_i = 1, Q(x) = \hat{y}_i$
  - For i^{th} data whose label is 0, $P(x) = y_i = 0, Q(x) = 1 - \hat{y}_i$

Loss function

Categorical Cross Entropy

- Common choices for multi-class classification
- Multi-class label - category
  - Recap> Types of data
    - Numeric data
    - Categorical data
    - Ordinal data

Loss function

Categorical Cross Entropy

- Multi-class label - category
  - Recap> Types of data: Numeric data
    - Data that consists of **numerical values representing quantities or measurements**
    - Clear linear and proportional relationship between values, making it suitable for statistical techniques to be applied.
      - Ex> 1<2<3<4<5, 1+1=2, 2+1=3

Loss function

Categorical Cross Entropy

- Multi-class label - category
  - Recap> Types of data: Categorical data
    - Data that consists of **categories**, also known as nominal data
    - Categorical data cannot be ordered numerically
    - There is no proportional relationship between values
      - Ex> Gender (Male, Female), Color (White, blue, yellow),

Loss function

Categorical Cross Entropy

- Multi-class label - category
  - Recap> Types of data: ordinal data
    - Type of categorical data that has a natural ordering or ranking among its categories
    - However, the intervals between the categories are not necessarily equal.
    - Ordinal data allows for comparison to be made in terms of greater than or less than.
      - Ex> Education level (elementary, middle, high school)

Loss function

Categorical Cross Entropy

- Multi-class label - category
  - Recap> One-hot encoding
    - Data preprocessing for categorical data
    - Encoding categorical data to binary data

Data table	
No.	Color
1	Red
2	Yellow
3	Green

Data table			
No.	Red	Yellow	Green
1	1	0	0
2	0	1	0
3	0	0	1

- Each binary number express whether data is included in specific category

Loss function

Categorical Cross Entropy

- Multi-class label – category (i.e., one-hot encoding)

Data No.	Category
1	2
2	3
3	1
4	2

One-hot encoded label		
0	1	0
0	0	1
1	0	0
0	1	0

Loss function

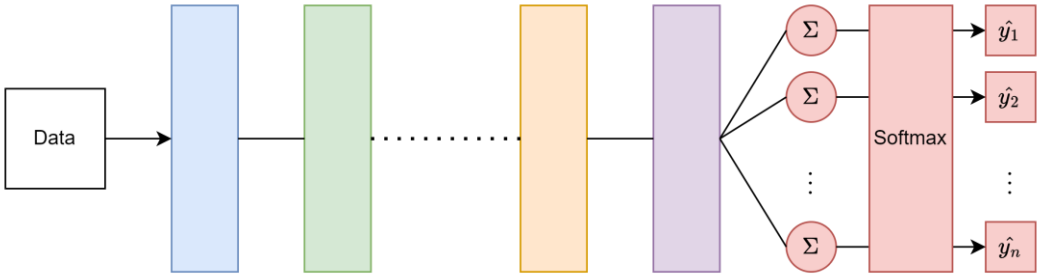
Categorical Cross Entropy

- Multi-class label – category (i.e., one-hot encoding)

Data No.	Category
1	2
2	3
3	1
4	2

One-hot encoded label		
0	1	0
0	0	1
1	0	0
0	1	0

- Multi-class classification model → predicting probabilities for each category



Loss function

Categorical Cross Entropy

- Common choices for multi-class classification: categorical cross entropy
  - Comparison One-hot encoded label vs Predicted probabilities vector
    - Example

Data no.	y			$\hat{y}$		
1	1	0	0	0.8	0.1	0.1
2	0	0	1	0.1	0.0	0.9
3	0	1	0	0.2	0.6	0.2

Loss function

Categorical Cross Entropy

- Common choices for multi-class classification: categorical cross entropy
  - Recap> Cross entropy: $-\sum P(x) \log Q(x)$

Data no.	y			$\hat{y}$		
1	1	0	0	0.8	0.1	0.1
2	0	0	1	0.1	0.0	0.9
3	0	1	0	0.2	0.6	0.2

Ground truth probability for each category = $P(x)$

Predicted probability for each category = $Q(x)$

Loss function

Categorical Cross Entropy

- Common choices for multi-class classification: categorical cross entropy
  - Recap> Cross entropy: $-\sum P(x)\log Q(x)$

Data no.	y			$\hat{y}$		
1	1	0	0	0.8	0.1	0.1
2	0	0	1	0.1	0.0	0.9
3	0	1	0	0.2	0.6	0.2

Cross entropy for data 1

$$CE = -\sum P(x)\log Q(x) = -(1\cdot\log 0.8 + 0\cdot\log 0.1 + 0\cdot\log 0.1)$$

$$= -\log 0.8$$

Loss function

Categorical Cross Entropy

- Common choices for multi-class classification: categorical cross entropy
  - Recap> Cross entropy: $-\sum P(x)\log Q(x)$

Data no.	y			$\hat{y}$		
1	1	~~0~~	~~0~~	0.8	~~0.1~~	~~0.1~~
2	0	0	1	0.1	0.0	0.9
3	0	1	0	0.2	0.6	0.2

Cross entropy for data 1

$$CE = -\sum P(x)\log Q(x) = -(1\cdot\log 0.8 + 0\cdot\log 0.1 + 0\cdot\log 0.1)$$

$$= -\log 0.8$$

We can recognize that we don't need to care about $\hat{y}$ for zeros

Loss function

Categorical Cross Entropy

- Common choices for multi-class classification: categorical cross entropy
  - Recap> Cross entropy: $-\sum P(x)\log Q(x)$

Data no.	y			$\hat{y}$		
1	1	0	0	0.8	0.1	0.1
2	0	0	1	0.1	0.0	0.9
3	0	1	0	0.2	0.6	0.2

Cross entropy loss for all data

$$L_{CE} = -\frac{1}{n}\sum\sum P(x)\log Q(x) = -\frac{1}{3}(\log 0.8 + \log 0.9 + \log 0.6)$$

→ Categorical cross entropy

Loss function

Categorical Cross Entropy

- In formulated form,

$$L_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log\left(\hat{y}_{i,c}\right)$$

  - n – number of samples, C – number of classes
  - $y_{i,c}$ – binary indicator (0 or 1) if class label c is the correct classification for sample i
  - $\hat{y}_{i,c}$ - predicted probability for the class c for sample i

Optimization

Optimization

- The goal of optimization is to find $\boldsymbol{W}$ that minimizes the loss function
  - Strategy #1: Random Search (Bad idea)
  - Strategy #2: Random Local Search
  -  Strategy #3: Following Gradient

Optimization

Strategy #1: A first very bad idea solution: Random search

- To simply try out many different random weights and keep track of what works best
- Pseudocode

Algorithm 1 Random Search for Minimizing Loss

```
bestloss ← ∞
for num ← 1 to 1000 do
    W ← randomly initialize W ~ N(0, 1)
    loss ← L(X_train, Y_train, W)
    if loss < bestloss then
        bestloss ← loss
        bestW ← W
    end if
    print "in attempt num, the loss was loss, best bestloss"
end for
```

Optimization

Strategy #1: A first very bad idea solution: Random search

- To simply try out many different random weights and keep track of what works best
- **Core idea: iterative refinement**
  - Our strategy will be to start with random weights and iteratively refine them over time to get lower loss

Optimization

Strategy #2: Random Local Search

- To apply random perturbation $\delta \boldsymbol{W}$ to the current weight $\boldsymbol{W}$, and if the loss at the perturbed $\boldsymbol{W} + \delta \boldsymbol{W}$ is lower than before, perform update.

- Pseudocode

Algorithm 2 Random Local Search

```
W ← randomly initialize 𝒩(0, 1)
bestloss ← ∞
for i ← 1 to 1000 do
    step_size ← 0.0001
    W_try ← W + 𝒩(0, step_size^2)
    loss ← L(X_train, Y_train, W_try)
    if loss < bestloss then
        W ← W_try
        bestloss ← loss
    end if
    print "iter i, loss is bestloss"
end for
```

- Much better and stable than Strategy #1

Optimization

Strategy #3: Following the Gradient

- In the previous section we tried to find a direction in the weight-space that would improve our weight vector by updating current weight
  - We **can compute the best direction** along which we should change our weight vector that is mathematically guaranteed to be the direction of the steepest descent
- This direction will be related to the **gradient** of the loss function

$$\nabla L = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n}\right]$$

  - A gradient of a function is a vector of partial derivatives.

Optimization

Computing the gradient

- Two ways to compute gradient
  - **Numerical gradient**
    - Approximation (Slow) but easy
  - **Analytic gradient**
    - Using calculus

Optimization

Numerical gradient

- Computing the gradient numerically with finite differences

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Setting h as small value allow us estimate derivatives

Algorithm 3 Numerical Gradient Computation

Input: differentiable function f, input vector $\mathbf{x}$
$h \leftarrow 10^{-5}$ ▷ Step size
$f_x \leftarrow f(\mathbf{x})$ ▷ Evaluate function at original point
$\nabla \leftarrow$ zero vector with same shape as $\mathbf{x}$
for each index i in $\mathbf{x}$ **do**
  $x_{\text{old}} \leftarrow \mathbf{x}[i]$
  $\mathbf{x}[i] \leftarrow x_{\text{old}} + h$ ▷ Perturb the i-th dimension
  $f_{x+h} \leftarrow f(\mathbf{x})$ ▷ Evaluate function at perturbed point
  $\nabla[i] \leftarrow \frac{f_{x+h} - f_x}{h}$ ▷ Approximate $\frac{\partial f}{\partial x_i}$
  $\mathbf{x}[i] \leftarrow x_{\text{old}}$ ▷ Restore original value
end for
return ∇

The updates to all components occur simultaneously

Optimization

Numerical gradient

- After computing the gradients of the loss function, we update the weights in the **opposite direction of the gradient**, since this is the direction that reduces the loss

$$w_{new} = w_{old} - \alpha \nabla L$$

  - The hyperparameter α controls how big each update step is—this is called the **learning rate**. We'll talk more about it later

Optimization

Analytic gradient

- Using calculus
  - Which allows us to derive a direct formula for the gradient (no approximations)
  - Very fast to compute

Optimization

Analytic gradient – Softmax classifier example

- Softmax classifier

$$Z = XW \rightarrow z_{i,c} = x_i^T w_c$$

$$\hat{y}_{i,c} = \frac{e^{z_{i,c}}}{\sum_{k=1}^{C} e^{z_{i,k}}}$$

- Cross-entropy loss

$$L_{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\, log\left(\hat{y}_{i,c}\right)$$

- Goal

$$\frac{\partial L}{\partial W} \leftarrow \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial W}$$

Optimization

Analytic gradient – Softmax classifier example

- For a fixed i,

$$L_i = -\sum_{c=1}^{C} y_{i,c}\, log(\hat{y}_{i,c})$$

- At first,

$$log(\hat{y}_{i,c}) = \log\left(\frac{e^{z_{i,c}}}{\sum_{k=1}^{C} e^{z_{i,k}}}\right) = z_{i,c} - \log\left(\sum_{k=1}^{C} e^{z_{i,k}}\right)$$

- So,

$$L_i = -\sum_{c=1}^{C} y_{i,c}\, log(\hat{y}_{i,c}) = -\sum_{c=1}^{C} y_{i,c} z_{i,c} + \sum_{c=1}^{C} y_{i,c} \log\left(\sum_{k=1}^{C} e^{z_{i,k}}\right)$$

Optimization

Analytic gradient – Softmax classifier example

- Because the second term does not depend on c, and $\sum_{c=1}^{C} y_{i,c} = 1$,

$$L_i = -\sum_{c=1}^{C} y_{i,c}\, log(\hat{y}_{i,c}) = -\sum_{c=1}^{C} y_{i,c} z_{i,c} + \sum_{c=1}^{C} y_{i,c} \log\left(\sum_{k=1}^{C} e^{z_{i,k}}\right)$$

$$= -\sum_{c=1}^{C} y_{i,c} z_{i,c} + \log\left(\sum_{k=1}^{C} e^{z_{i,k}}\right)$$

Optimization

Analytic gradient – Softmax classifier example

- Now, differentiate w.r.t. $z_{i,j}$

$$L_i = -\sum_{c=1}^{C} y_{i,c} z_{i,c} + \log\left(\sum_{k=1}^{C} e^{z_{i,k}}\right)$$

$$\rightarrow \frac{\partial L_i}{\partial z_{i,j}} = -y_{i,j} + \frac{e^{z_{i,j}}}{\sum_{k=1}^{C} e^{z_{i,k}}} = -y_{i,j} + \hat{y}_{i,j}$$

- For entire data,

$$\frac{\partial L}{\partial z_{i,j}} = \frac{1}{n}\left(\hat{y}_{i,j} - y_{i,j}\right)$$

- In matrix form,

$$\frac{\partial L}{\partial Z} = \frac{1}{n}\left(\hat{Y} - Y\right)$$

Optimization

Analytic gradient – Softmax classifier example

- Now, let's derive $\frac{\partial Z}{\partial W}$

$$Z = XW$$

$$\rightarrow \frac{\partial Z}{\partial W} = X$$

- Finally,

$$\frac{\partial L}{\partial Z} = \frac{1}{n}\left(\hat{Y} - Y\right), \qquad \frac{\partial Z}{\partial W} = X$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial W} = \frac{1}{n} X^T \left(\hat{Y} - Y\right)$$

Optimization

Analytic gradient – Softmax classifier example

- Now update,

$$W_{new} \leftarrow W_{old} - \alpha \nabla_W L$$

Softmax Classifier

Python implementation

- Utilizing Softmax classifier

```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt

class SoftmaxClassifier():
    def __init__(self):
        self.w = np.random.normal(0,1,(10, 28*28+1)) # bias trixk

    def forward(self, x_batch):
        return x_batch @ self.w.T

    def predict(self, x_batch):
        return np.argmax(self.forward(x_batch), axis=1)

    def __call__(self, x_batch):
        return self.forward(x_batch)
```

```
# Load MNIST data
train_dataset = torchvision.datasets.MNIST('./', train=True,
download=True)
test_dataset = torchvision.datasets.MNIST('./', train=False,
download=True)

x_train = train_dataset.data.numpy()
y_train = train_dataset.targets.numpy()
y_train_onehot = np.zeros((len(y_train), 10))
y_train_onehot[np.arange(len(y_train)), y_train] = 1

x_test = test_dataset.data.numpy()
y_test = test_dataset.targets.numpy()
y_test_onehot = np.zeros((len(y_test), 10))
y_test_onehot[np.arange(len(y_test)), y_test] = 1

# Normalize image to be 0~1
x_train = x_train/255.0
x_test = x_test/255.0

# Flatten image
x_train = x_train.reshape(-1, 28*28)
x_test = x_test.reshape(-1, 28*28)
```

Softmax Classifier

Python implementation

- Utilizing Softmax classifier

```
# Build model
Clf = SoftmaxClassifier()
alpha = 0.0005 # learning rate
Epochs = 100
batch_size = 64

for epoch in range(Epochs):
    for i in range(0, len(x_train), batch_size):
        batch = x_train[i:i+batch_size]
        batch = np.concat([batch, np.ones(shape=(len(batch),1))], axis=1)

        batch_label = y_train_onehot[i:i+batch_size]
        pred = Clf(batch)

        Clf.w = Clf.w - alpha *  (pred - batch_label).T @ batch # calculate gradient and update

    # Test
    pred_test = Clf(np.concat([x_test, np.ones(shape=(len(x_test),1))], axis=1))
    pred_label_test = np.argmax(pred_test, axis=1)
    print(f"epoch: {epoch}, val_acc: {np.average(pred_label_test==y_test):.4}")
```