2️⃣

Annotation data efficient learning

강의 소개

컴퓨터 비전 문제를 푸는 딥러닝 모델은 supervised learning으로 학습하는 것이 유리하다는 사실은 알려져 있습니다.하지만, 딥러닝 모델을 학습할 수 있을 만큼 고품질의 데이터를 많이 확보하는 것은 보통 불가능하거나 그 비용이 매우 큽니다.

2강에서는 Data Augmentation, Knowledge Distillation, Transfer learning, Learning without Forgetting, Semi-supervised learning 및 Self-training 등 **주어진 데이터셋의 분포를 실제 데이터 분포와 최대한 유사하게 만들거나, 이미 학습된 정보를 이용해 새 데이터셋에 대해 보다 잘 학습하거나, label이 없는 데이터셋까지 이용해 학습**하는 등 주어진 데이터셋을 최대한 효율적으로 이용해 딥러닝 모델을 학습하는 방법을 소개합니다.

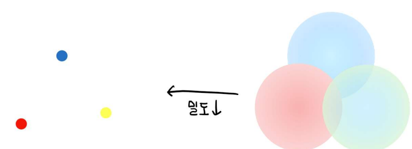
Further Reading

- CutMix : https://arxiv.org/abs/1905.04899

1. Data augmentation

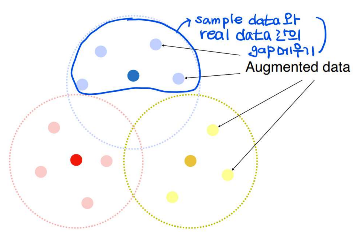
Learning representation of dataset

- Dataset is always **biased**
  - **Images taken by camera (training data) ≠ real data**
- The training dataset is **sparse samples** of real data
  - The training dataset contains **only fractional part of real data**



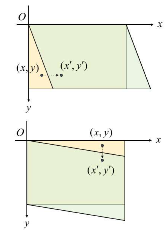
Samples in the training set   Real data distribution

- The training dataset and real data always have **a gap**
  - Datasets do not fully represent **real data distribution**
- Augmenting data to **fill more space and to close the gap**
  - Augmentations to make a dataset **denser**



Data augmentation

- 목적 → **training dataset's distribution** ≒ **real data distribution** 하도록 만들기
- image transformation
  - Crop / Shear / Brightness / Perspective / Rotate
- ▼ 📎 Shear transformation (전단 변환)
  - 영상의 전단 변환은 **x,y 축 방향으로 영상이 밀리는 것 처럼 보이는 변환**을 뜻합니다. (아래 그림 참조) 따라서 **축에 따라서 픽셀의 이동 비율이 달라**집니다.
  - 따라서 전단 변환의 결과로 **한쪽으로 기울어진 영상**을 만들어 낼 수 있습니다. 따라서 각 방향으로 기울어진 변환을 적용하기 위하여 x축과 y축 각각에 대하여 변환을 적용하면 됩니다. 아래와 같습니다.



$$\begin{cases} x' = x + my \\ y' = y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & m & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{cases} x' = x \\ y' = mx + y \end{cases} \quad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  - 참고 : 이미지 Geometric Transformation 알아보기 - gaussian37

Various data augmentation methods

Brightness adjustment

- by `NumPy`

```
def brightness_augmentation(img):
    # numpy array img has RGB value(0~255) for each pixel
    img[:,:,0] = img[:,:,0] + 100 # add 100 to R value
    img[:,:,1] = img[:,:,1] + 100 # add 100 to G value
    img[:,:,2] = img[:,:,2] + 100 # add 100 to B value

    img[:,:,0][img[:,:,0]>255] = 255  # clip R values over 255
    img[:,:,1][img[:,:,1]>255] = 255  # clip G values over 255
    img[:,:,2][img[:,:,2]>255] = 255  # clip B values over 255
    return img
```

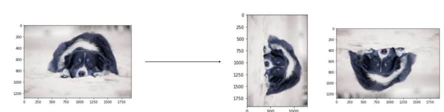
↳특정 숫자 더하기

↳pixel의 상한선 255를 넘지 않도록

Rotate / Flip

- by `OpenCV`

```
img_rotated = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)
img_flipped = cv2.rotate(image, cv2.ROTATE_180)
```



Crop

- by `NumPy`

```
y_start = 500     # y pixel to start cropping
crop_y_size = 400 # cropped image's height
x_start = 300     # x pixel to start cropping
crop_x_size = 800 # cropped image's width
img_cropped = image[y_start : y_start + crop_y_size, x_start : x_start + crop_x_size, :]
```

↳y 시작 ↳y 끝 ↳x 시작 ↳x 끝

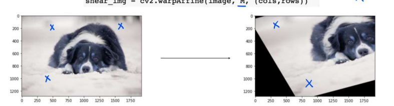
Affine transformation (= Shear transformation)

- preserves `line`, `length ratio`, `parallelism` in image
- ex, transforming `rectangle` → `parallelogram`

- by `OpenCV`

변환 전/후 점

```
rows, cols, ch = image.shape
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv2.getAffineTransform(pts1,pts2)
shear_img = cv2.warpAffine(image, M, (cols,rows))
```



▼ getAffineTransform()

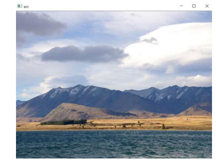
- OpenCV에서 **점 3개의 이동 전, 이동 후 좌표를 입력**하면 **어파인 변환 행렬을 반환**하는 함수를 제공합니다.

```
cv2.getAffineTransform(src, dst) -> retval

• src: 3개의 원본 좌표점. numpy.ndarray. shape=(3, 2)
  e.g) np.array([[x1 , y1 ], [x2 , y2 ], [x3 , y3 ]], np.float32)
• dst: 3개의 결과 좌표점. numpy.ndarray. shape=(3, 2)

# 주의할 점은 넘파이 행렬로 입력해줘야 합니다.
# 3 X 2 행렬을 반환합니다.
```

- 참고 : [파이썬 OpenCV] 어파인 변환과 투시 변환 - cv2.getAffineTransform, cv2.getPerspectiveTransform, cv2.warpPerspectivee (tistory.com)

▼ warpAffine()

```
cv2.warpAffine(src, M, dsize, dst=None, flags=None, borderMode=None, borderValue=None) -> dst

• src: 입력 영상
• M: 2x3 어파인 변환 행렬. 실수형.
• dsize: 결과 영상 크기. (w, h) 튜플. (0, 0)이면 src와 같은 크기로 설정.
• dst: 출력 영상
• flags: 보간법. 기본값은 cv2.INTER_LINEAR.
• borderMode: 가장자리 픽셀 확장 방식. 기본값은 cv2.BORDER_CONSTANT.
• borderValue: cv2.BORDER_CONSTANT일 때 사용할 상수 값. 기본값은 0(검정색).
```

입력 영상

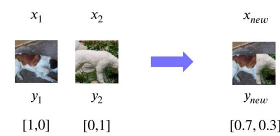
출력 영상

- 참고 : [파이썬 OpenCV] 영상의 기하학적 변환 - 전단 변환 - cv2.warpAffine (tistory.com)

Modern augmentation techniques

CutMix

- Mixing both **images** and **labels**



RandAugment

- Automatically finding **the best sequence of augmentations** to apply
  - Random sample + apply + evaluate augmentations

• identity	• autoContrast	• equalize
• rotate	• solarize	• color
• posterize	• contrast	• brightness
• sharpness	• shear-x	• shear-y
• translate-x	• translate-y	

- Augmentaiton policy has two parameters
  - **Which** augmentation to apply
  - Magnitude of augmentation to apply (**how much** to augment)
- Randomly testing Augmentation policies
  - 1️⃣ Sample a policy : Policy = {**N augmentations** to apply} by random sampling
  - 2️⃣ Train with a sampled policy
  - 3️⃣ evaluate the accuracy

▼ pytorch randaugment

- `torchvision.transforms.RandAugment`
- paramters
  - **num_ops (*int*)** – Number of augmentation transformations to apply sequentially

- **magnitude** (*int*) – Magnitude for all the transformations
- num_magnitude_bins (*int*) – The number of different magnitude values
- interpolation (*InterpolationMode*) – Desired interpolation enum defined by `torchvision.transforms.InterpolationMode`. Default is `InterpolationMode.NEAREST`. If input is Tensor, only `InterpolationMode.NEAREST`, `InterpolationMode.BILINEAR` are supported
- fill (*sequence or number, optional*) – Pixel fill value for the area outside the transformed image. If given a number, the value is used for all bands respectively.

- 참고 : RandAugment — Torchvision main documentation (pytorch.org)

2. Leveraging pre-trained information

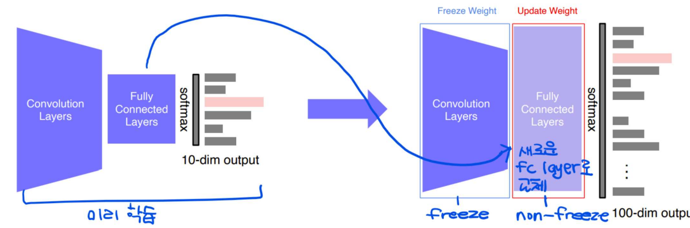
Transfer learning

- The high-quality dataset is expensive and hard to obtain
  - → **Transfer learning** : A practical training method with a small dataset
- `Motivational observation` : Similar datasets share common information
  - Knowledge learned from one dataset can be applied to other datasets

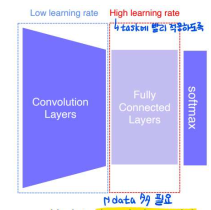
(a) ImageNet Synset: One sample image from each category (b) Corel-1000 Dataset: Sample images from each category

(c) Caltech-256 Dataset: One sample image from each category (d) Caltech-101 Dataset: One sample image from each category

- Approach
  - 1️⃣ **Transfer** knowledge from a pre-trained task to a new task
    - 데이터 개수 ↓ (+ 유사도 ↑)일 때 사용
    - Chop off the final layer of the pre-trained model, and **only re-train a new FC layer**
    - **Extracted features preserve** all the knowledge from pre-training



  - 2️⃣ Fine-tuning **the whole model**
    - 데이터 개수 ↑ (+ 유사도 ↓)일 때 사용
    - Replace the final layer of the pre-trained model to a new one, and **re-train the whole model**
    - **Set learning rates differently**
      - ▼ 📎 Set learning rates differently
        - 데이터 셋이 유사하기 때문에, 시간 비용을 감안할 때, 전체 layer에 대해서 Fine-tuning을 진행할 필요는 없습니다.
        - 즉, 전체 layer의 약간만 기존의 learning rate의 1/10 정도의 값으로 학습을 진행합니다.
        - 참고 : [신경망] 17. Transfer Learning (tistory.com)



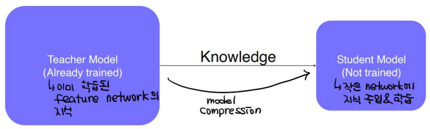
Update the whole model

- ▼ 📎 데이터 셋 크기 + 유사도와 transfer learning
  - 참고 : [딥러닝 알아가기] Transfer Learning과 Fine Tuning — 글쓰는공대생의 IT블로그 (tistory.com)
  - 참고 : https://www.notion.so/dudskrla/Model-20c1237b15c4474991df33a515770c1e#646a76c76db146d480aa17d6bc82bcc3

Knowledge distillation

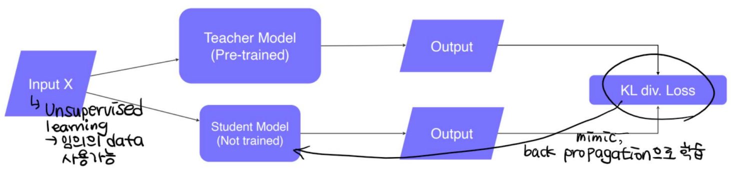
Teacher-student Learning

- `distillate` = knowledge of a **trained model** into another **smaller model** 하는 과정
- Used for `model compression` **(Mimicking what a larger model knows)**
- Used for `pseudo-labeling` **(Generating pseudo-labels for an unlabeled dataset)**
  - → 더 많은 데이터를 만듦 ⇒ regularizer 역할 (overfitting 방지)

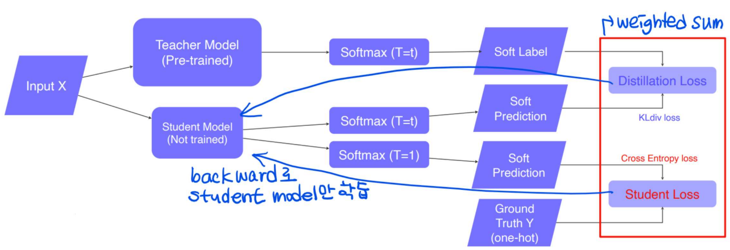


Teacher-student network structure

- **The student** network learns what **the teacher** network knows
- The student network **mimics** outputs of the teacher network
- **Unsupervised learning**, since training can be done only **with unlabeled data**



Knowledge distillation



- `Student loss` → When labeled data is available, can leverage data for training
- `Distillation loss` → to predict similar outputs with the teacher model (mimicking)
- Semantic information is not considered in distillation (?) → mimicking이 중요 (의미를 모두 따라할 필요 X)
- ▼ Student loss VS. Distillation loss ?
  - `Student Loss` 는 일반적인 **cross_entropy**, 혹은 사용하고 싶은 Loss로 **Ground Truth와 비교**하여 계산한다.
  - `Distillation Loss` 의 경우 **KLDivLoss**를 사용하여 **Student와 Teacher의 logits**을 **비교**하여 계산한다.
  - `weighted sum` → 최종적으로 alpha의 값으로 **student_loss와 distillation_loss의 비율을 조절**하여 합한다.
  - cf. 다양한 커뮤니티에서는 alpha: 0.1, Temperature: 10이 일반적으로 성능이 잘나온다고 한다.

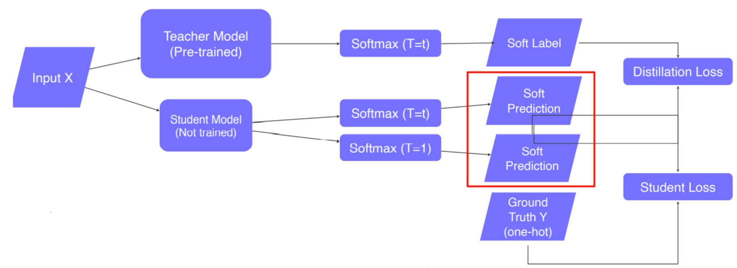
```
def knowledge_distillation_loss(self, logits, labels, teacher_logits):
  alpha = 0.1
  T = 10

  student_loss = F.cross_entropy(input=logits, target=labels)
  distillation_loss = nn.KLDivLoss(reduction='batchmean')(F.log_softmax(logits/T, dim=1), F.softmax(teacher_logits/T, dim=1)) * (T
  total_loss = alpha*student_loss + (1-alpha)*distillation_loss return total_loss
```


- 참고 : Knowledge Distillation 구현 — re-code-cord (tistory.com)
- 읽어보기 : NLP 논문리뷰 - Distilling the Knowledge in a Neural Network - 데이터 사이언스 사용 설명서 (tistory.com)

Intuition about distillation loss and student loss

- `Distillation loss`
  - KLdiv (Soft label, Soft prediction)
  - loss = difference **between the teacher and student network's inference**
  - learn what teacher network knows by `mimicking`
- `Student loss`
  - CrossEntropy (Hard label, Soft prediction) (단, hard label from `ground truth Y`)
  - loss = difference **between the student network's inference and true label**
  - learn the `right answer`

Hard label VS. Soft label



- Hard label (one-hot vector) (0/1)
  - typically obtained from the **dataset**
  - indicates whether a class is `true answer` or not
- Soft label (0~1)
  - typically **output of the model** (= inference result)
  - regard it as `knowledge` → useful to observe **how the model thinks**

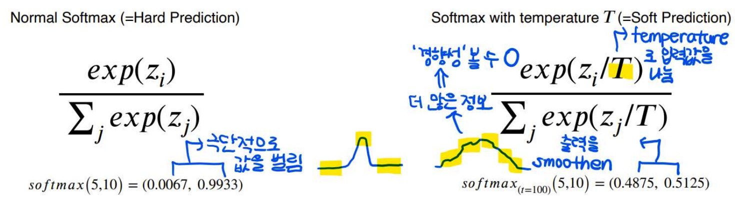
$$\begin{pmatrix} Bear \\ Cat \\ Dog \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} Bear \\ Cat \\ Dog \end{pmatrix} = \begin{pmatrix} 0.14 \\ 0.8 \\ 0.06 \end{pmatrix}$$

Hard label Soft label

Softmax with temperature (T)

- `Softmax with temperature` → controls difference in output between small & large input values
  - `A large T` **smoothens** large input value **differences**
- useful to **synchronize the student and teacher models' outputs** (mimicking)
  - ▼ temperature ↔ synchronize the student and teacher models' outputs 관계
    - 가열하는 온도(temperature)를 잘 조절 해 주면 증류(distillation)가 더 잘 될 수 있겠죠?
    - 일부 클래스들에 대한 probability는 거의 0에 가까워서 학습 시에 정보가 잘 전달되지 않을 수 있으므로 이를 좀 더 soft하게 만들어 **학습에 잘 반영**될 수 있도록 만드는 역할을 합니다.

- 참고 : Distilling the Knowledge in a Neural Network (NIPS 2014 Workshop) – Lunit Tech Blog

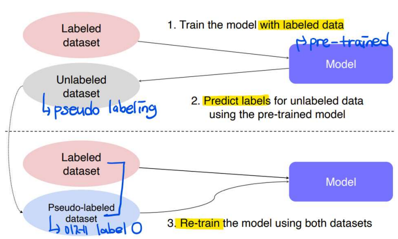


- ▼ softmax temperature T
  - 기존 softmax function에서 temperature라는 파라미터를 추가하여, T가 높을수록 기존보다 더 soft한 probability distribution을 얻을 수 있도록 만들었습니다.
  - 참고 : Distilling the Knowledge in a Neural Network (NIPS 2014 Workshop) – Lunit Tech Blog

3. Leveraging unlabeled dataset for training

Semi-supervised learning

- There are lots of unlabeled data
  - Is there any way to learn from unlabeled data ?
  - `Semi-supervised learning` → **Unsupervised** (No label) + **Fully Supervised** (fully labeled)
  - ▼ semi-supervised learning
    - Semi-supervised learning (준지도학습)은 소량의 labeled data에는 supervised learning을 적용하고 대용량 unlabeled data에는 unsupervised learning을 적용해 추가적인 성능향상을 목표로 하는 방법론이다.
    - 참고 + 읽어보기 : Semi-supervised learning (준지도학습): 개념과 방법론 톺아보기 (tistory.com)
- Semi-supervised learning with pseudo labeling
  - **Pseudo-labeling unlabeled data** using a pre-trained model, then use for training



Self-training

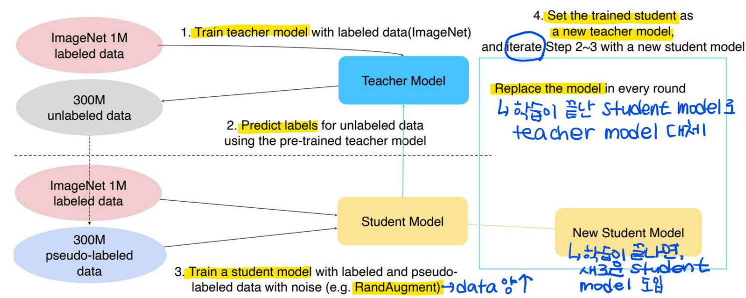
Data efficient learning methods

- `Data Augmentation`
  - augment a dataset to make the dataset **closer to real data distribution**
- `Knowledge distillation`
  - train a **student network** to imitate a teacher network
  - transfer the teacher network's knowledge to the student network
  - (생각) data를 직접 augmentation 하는 것은 아니지만, self-training에서 unlabeled data를 labeling 해주는 용도로 사용하는 듯
- `Semi-supervised learning` (Pseudo label-based method)
  - pseudo-label → an unlabeled dataset using a pre-trained model, then use for training
  - leveraging an unlabeled dataset for training

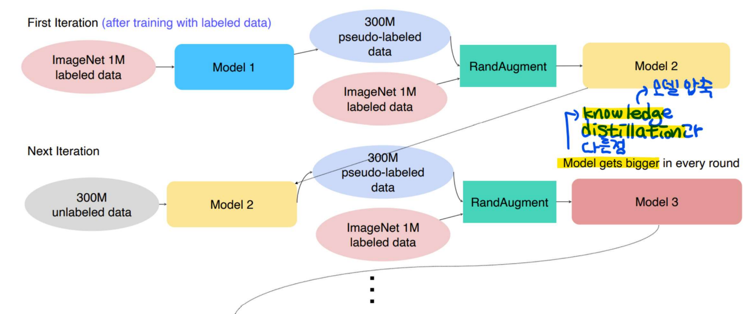
Self-training

- Augmentation + Teacher-Student networks + Semi-supervised learning

Self-training with noisy student



Iteratively training noisy student network using teacher network



Brief overview of self-training algorithm

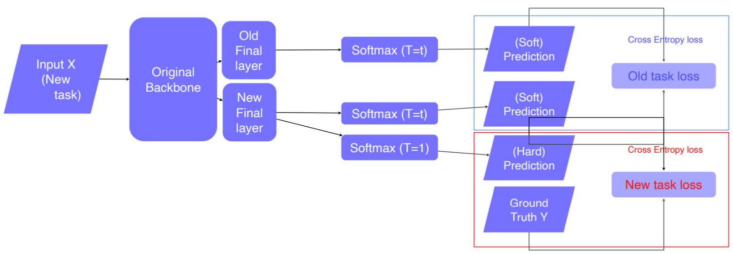
1. Train initial **teacher model** with **labeled data**
2. **Pseudo-label unlabeled data** using teacher model
3. Train **student model** with **both lableled and unlabeled data** with augmentation
4. Set the **student model** as a new teacher, and **set new model (bigger)** as a new student
5. **Repeat** 2~4 with new teacher/student models

Reference

1. Data augmentation
   - Yun et al., CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features, ICCV 2019
   - Cubuk et al., Randaugment: Practical automated data augmentation with a reduced search space, CVPRW 2020
2. Leveraging pre-trained information
   - Ahmed et al., Fusion of local and global features for effective image extraction, Applied Intelligence 2017
   - Oquab et al., Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks, CVPR 2015
   - Hinton et al., Distilling the Knowledge in a Neural Network, NIPS deep learning workshop 2015
   - Li & Hoiem, Learning without Forgetting, TPAMI 2018
3. Leveraging unlabeled dataset for training
   - Lee, Pseudo-label : The simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks, ICML Workshop 2013
   - Xie et al., Self-training with Noisy Student improves ImageNet classification, CVPR 2020

cf. Learning without Forgetting

- weighted sum of `old task loss` and `new task loss`



Fine-tuning a model for both old and new tasks

- Fine-tuning a model that performs well in both old and new tasks

Fine-tuning forgets old tasks

- When fine-tuning, the whole model is updated
  - the feature extractor is fitted for the new task
  - the updated feature extractor is not compatible to the old tasks' classifiers **(Forgetting)**
- Naive approach → train using **both old and new datasets**

Learning without Forgetting

- Fine-tuning a model to perform well on both 'old' and 'new' tasks **without the old dataset**
  - Standard training for **the new task**
  - Training **the old task** branches to follow the pre-trained model's output with the new task data (?)
- The intuition is similar to `(knowledge) distillation` → learning what other models know