# #1 iCaRL (Incremental Classifier & Representation Learning)

- Class-incremental learning: 필수 구성 요소 두 가지

1. 연속성 있는 데이터로부터(*지속적으로* 클래스가 추가되는, stream of datas) 모델이 항상 훈련 가능 해야 함 **(Update Routine Algorithm)**
2. 모델은 여태까지 관찰한 모든 클래스에 대해 multi-class classifacation 할 수 있어야 함

💡 CL의 3가지 시나리오

- Domain-incremental Learning:
  - 각 experiment에 대한 task indentity t가 필요하지 않다.
  - input 이미지의 분포는 다르지만, 아웃풋 분포는 같다

    → 각 클래스는 모든 task에 대해서 같은 semantic meaning을 가진다
  - 이 학습 방법의 모델은 single-headed output layer를 갖는다.

⇒ 도메인(같은 의미를 가진 이미지, instance)이 늘어나는

- Task-incremental Learning:
  - 각 task identity에 대한 정보를 갖는다., 항상 제공됨 (트레인에서나 테스트에서나)
  - 각 task의 클래스는 disjoint된 meaning을 가진다.
  - 모델은 multi-headed output layer를 갖고 있어 여러 task에 대응하는 output layer에서 클래스를 예측한다.

⇒ Task, 환경이 늘어나는 with multi-headed output layer

- Class-incremental Learning:
  - 여태까지 모두 관찰된 task를 예측하고 그 task 안에서 분류 등의 문제를 해결해야 함
  - single-head 구조, 아웃풋이 모두 같은 분포를 가짐
  - task identity 없이 모든 클래스에 대한 분류를 수행해야 함
  - 가장 실제 상황과 가깝기에 실용적이지만 어려운 방법

# 📄 Method

## Class-Incremental Classifier Learning

- **Exemplar**: 데이터 스트림에서부터 *동적으로* 선택된 이미지들(여태까지 관찰된 클래스에 대한), Exemplar의 전체 개수는 고정된 파라미터 값 K를 넘지 않는다.
- **Update routine: 루틴에 따라** iCaRL의 내부 네트워크 파라미터나 Exemplar를 수정한다.  (현재 학습 데이터를 기반으로)
- iCaRL에서는 CNN 네트워크 이용, 하지만 추론에 사용되진 않는다, 네트워크를 *trainable feature extractor라고 생각하면 됨*

하나의 classification 레이어를 가지며, 여기에 여러 클래스에 대한 sigmoid output 노드가 있음.

고정된 수의 파라미터들은 두 가지 파트로 나뉘는데, 하나는 feature extraction을 위해, 하나는 weight vector들을 저장하기 위해 (weight vector들은 w1,…wt로 표기) 사용된다.

t는 여태까지 관찰한 클래스의 수, 따라서 모델의 아웃풋은 y ∈ {1, . . . , t}

실제 이미지에 대한 classification은 exemplar set의 mean(prototype vector)과 가장 유사한 유클리디안 디스턴스를 계산하여 가장 가까운 prototype vector와의 거리를 클래스로 분류함 **(rehearsal-based)**

⇒ iCaRL에서 CNN 모델은 Representation Learning을 위한 feature 생성을 위해 쓰인다.

---

## Nearest-Mean-of-Exempers Classifiacation

- 여태까지 본 각 클래스의 prototype vector (μ1, . . . , μ_t로 표기)는 클래스의 모든 exemplar들에 대한 average feature vector
- 새로운 이미지 x에 대한 클래스 라벨을 만들 때는 가장 유사한 프로토타입 벡터와 x의 feature vector 간의 유클리디안 디스턴스가 최소가 되는 값으로 할당

$$y^* = \underset{y=1,\ldots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|.$$

- 기존 이미지 분류는 웨이트 벡터와 피쳐맵 파이를 곱하는 방식, 이는 decoupled 되어있다가 소프트맥스 전에 각각 곱해지는 방식. 이는 class-incremental 환경에서는 문제가 될 수 있음. **피쳐맵이 바뀔 때마다 웨이트 벡터들도 업데이트 반드시 되어야 함**
- 하지만 이 means-of-exemplars 방법은 다르다. 만들어진 feature representation이 바뀔 때마다 마지막에 곱해질 각 클래스의 prototype vector가 자동으로 바뀐다.
- 근데 prototype vector를 클래스의 데이터들의 average로 구성하긴 하는데 진짜 각 클래스의 모든 훈련 데이터에 대한 mean으로 할 순 없음. feature representation을 변경

하는 과정(prototype vector가 계속 바뀌는 과정)에서 항상 다시 계산하려고 얘네들을 다 저장하긴 어렵기 때문

- 대신 클래스 mean에 근접하는 몇몇 개의 적당한 수의 exemplar(datas from stream)의 average로 사용함.

$$\operatorname{argmax}_y \quad \mu_y^\top \varphi(x).$$

---

## Representation Learning

- iCaRL Augmented Train Dataset을 가짐 → 현재 이용 가능한 훈련 샘플과 저장된 exemplar로 구성.

// form combined training set:

$$\mathcal{D} \leftarrow \bigcup_{y=s,\ldots,t} \{(x,y) : x \in X^y\} \cup \bigcup_{y=1,\ldots,s-1} \{(x,y) : x \in P^y\}$$

- 새로운 클래스가 들어오기 전에, 이전 클래스로 학습한 네트워크의 아웃풋(추론 결과)은 모두 저장됨 → 기존 네트워크를 통한 정보도 획득하기 위해 이후에 쓰이는 knowledge distillation에서 distallation loss랑 섞어서 사용한다.

// store network outputs with pre-update parameters:
**for** $y = 1, \ldots, s-1$ **do**
  $q_i^y \leftarrow g_y(x_i) \quad$ for all $(x_i, \cdot) \in \mathcal{D}$

- 최종적으로 각각의 새로운 이미지에 대해서 새로운 클래스를 출력하게 하는 Loss(classification loss), *예전 클래스에 대해 score를 만드는 로스(distillation loss)*

  이걸로 각 클래스를 binary classification하도록 학습한다. 분류 결과는 sigmoid로 산출, 새로운 클래스에 대해서는 이 클래스가 기존에 있는 클래스인 지 아닌 지를 **확률**로서 결정한다.

- 따라서 평범한 파인 튜닝과 다른 점 두 가지가 존재

1. train 데이터셋이 augment 되어있음. 새로운 훈련 샘플 뿐만 아니라 이전에 저장된 exemplar들을 포함함. 이를 통해 이전에 학습한 클래스의 데이터 분포에 대한 정보를

얻을 수 있음, 여기서 저장된 exemplar들은 feature representation이 아닌 image로써 저장되어 있다는 것임.

2. loss 함수 또한 augment 되어있음; feature representation의 향상을 이끌어 새로 관찰되는 클래스에 대한 분류를 가능하게 해주는 classification loss, 새로운 학습 과정에서 이전에 배운 데이터인지 판별하는 정보들을 담당하는 distillation loss.

$$\ell(\Theta) = -\sum_{(x_i, y_i) \in \mathcal{D}} \Big[ \sum_{y=s}^{t} \delta_{y=y_i} \log g_y(x_i) + \delta_{y \neq y_i} \log(1 - g_y(x_i)) + \sum_{y=1}^{s-1} q_i^y \log g_y(x_i) + (1 - q_i^y) \log(1 - g_y(x_i)) \Big]$$

that consists of *classification* and *distillation* terms.

---

## Exemplar Management

- iCaRL에 새로운 클래스가 들어올 때마다, Exemplar set들은 스스로 조정됨
- 여태까지 t 클래스가 관찰되었고, 총 K개의 exemplar들이 저장되어 있다면 각 클래스별로 M = K / t (반올림) 개가 exemplar가 쓰인다.
- 이를 통해 exemplar들을 저장하는 메모리 공간이 넘치지 않고 낭비 없이 사용되도록 함
- Exemplar 관리를 위해서 사용되는 두 가지 루틴은 다음과 같음.

1. **Exemplar set 구성하기: 새로운 클래스에 n개의 샘플이 있다고 한다면, 이에 대한 전체 데이터의 평균 μ를 계산한다. 그리고 제한된 크기의 M에 대해서 가장 μ의 값과 근사할 수 있도록 하는 이미지를 선별적으로 선택한다. e.g 모든 샘플에 대해서 반복적으로 평균을 계산하면서 기존 μ와 거리가 가장 적은 이미지 set이 exemplar set이 된다.**
2. exemplar set은 우선순위가 있는 리스트, 가장 먼저 있는 exemplar가 더 중요함. 그래서 class가 늘어날 수록 exemplar를 삭제하게 되면 뒤에서부터 삭제.

- Background: 이 루틴들을 디자인할 때, 두 가지 주요 목표가 있었음. 첫번째는 처음으로 만들어지는 exemplar set들이 클래스의 mean vector를 잘 대표하고 근사할 수 있도록 하는 것, 두번째로는 알고리즘의 런타임 중에 exemplar의 삭제가 가능 할 것.
    - 두번째 목표를 만족하는 게 실제로 어려웠고, 따라서 마지막 elements를 제거하고 계속 그 set가 만족할만한 근사 속성을 가지고 있는지 확인하는 작업을 수행했음.

- *herding*에서 쓰인 것처럼 첫번째 elements가 좋은 대표적과 근사적 속성을 갖는 mean vector이게 끔. 하지만 다른 방법은 좋은 approximation 퀄리티를 보장하지 않았음.

## 🔬 Experiments

for iCIFAR, ResNet-32, a number of exemplars = 2000, each training step 70 epoch,

learning rate starts at 2.0 and is divided by 5 after 49 and 63 epochs (7/10, 9/10 of all epoch)

for iILSVRC, exemplars K = 20000. ResNet-18, 60 epoch for each class batch

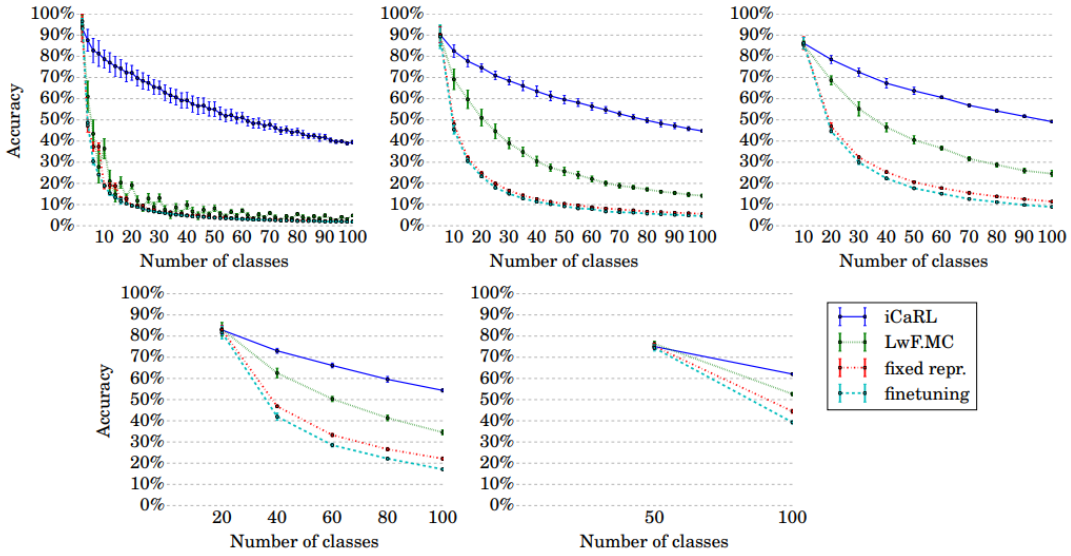
learning rate starts at 2.0 and divided by 5 after 20, 30, 40, 50 epochs (1/3. 1/2, 2/3 and 5/6 of all epochs)

두 실험 모두 standard backpropagation,  mini-batches of 128 and weight decay 0.00001

## Experiment Methods & Results

- Fine-tuning: 아무런 조치 X
- Fixed representation: 클래스의 첫번째 배치 이후로 feature representation을 freeze, 해당하는 클래스가 진행된 이후 classification layer의 weights를 freeze (보존)

  → 두번째 배치부터는 새로운 클래스의 weight vector만 학습되는 것과 같음
- LwF(MC): iCaRL처럼 distillation loss를 사용, 하지만 Exemplar set을 사용하지 않음
- iCaRL:
  1. distillation loss를 사용
  2. representation learning에서 exemplar를 사용,
  3. 분류 시에 mean-of-exemplar 사용,




(a) Multi-class accuracy (averages and standard deviations over 10 repeats) on iCIFAR-100 with 2 (top left), 5 (top middle), 10 (top right), 20 (bottom left) or 50 (bottom right) classes per batch.