💚

1. iCaRL: Incremental Classifier and Representation Learning

0. Abstract

- 인공지능의 발전 과정에서 주요한 개방형 문제 중 하나는 데이터 스트림으로부터 **시간이 지남에 따라 점차적으로 더 많은 개념을 학습**하는 학습 시스템을 개발하는 것

  ⇒ 클래스 점진적 학습을 가능하게 하는 새로운 교육 전략인 **iCaRL**을 소개

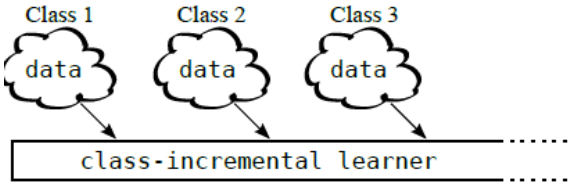
- **iCaRL의 특징(추구하는 방향?)**
  - 처음에는 작은 수의 클래스에 대한 훈련 데이터만 존재

    → 새로운 클래스를 **점진적**으로 추가할 수 있도록 함
  - 분류와 데이터 표현을 동시에 학습

    ⇒ 이전 연구들은 **고정된 데이터 표현**에 근본적으로 제한되어 있었음 → 딥러닝 아키텍처와 호환 x

1. Introduction

- 자연스러운(→ 생물체라면..?) 시각 시스템은 본래 점진적임
  - 새로운 시각 정보가 지속적으로 통합되면서 기존의 지식이 보존되는 형태
- 하지만 대부분의 인공(→ 인공지능) 물체 인식 시스템은 모든 물체 클래스가 **미리 알려진 상태**에서 **일괄적**으로 교육될 수 있음
- 컴퓨터 비전 분야가 인공 지능에 가까워짐에 따라 현실 세계에서의 물체 분류 상황의 대규모 및 동적 특성을 다루기 위해 더 유연한 전략이 필요
  - 시각적 물체 분류 시스템은 새로운 클래스에 대한 교육 데이터가 이용 가능해질 때 증분적으로 배울 수 있어야 함 ⇒ **클래스 증분 학습(class-incremental learning)** 개념 등장



- 알고리즘이 클래스 증분(class-incremental)이 가능하려면 다음 세 가지 특성을 충족해야 함
  - 다른 클래스의 예제가 다른 시간에 발생하는 데이터 스트림에서 훈련 가능해야 함
  - 언제든지 지금까지 관찰된(→ 학습된) 클래스에 대한 경쟁력 있는 다중 클래스 분류기를 제공할 수 있어야 함
  - 계산 요구 사항 및 메모리 사용량은 관찰된 클래스의 수에 따라 제한되거나 적어도 매우 느리게라도 증가해야 함
- 대부분의 기존 다중 클래스 기술은 단일 시점에 **고정된 클래스 수만 처리**할 수 있거나 모든 훈련 데이터를 **동시에** 사용할 수 있어야 함
  - i) 또는 ii)를 위반
  - 단순하게는 클래스 증분 데이터 스트림에서 분류기를 훈련하는 것으로 문제를 해결할 수 있을 것이라 생각됨
    - 이로 인해 분류 정확도가 빠르게 저하되는 문제 발생

      ⇒ "카타스트로피 피치" 또는 "카타스트로피 간섭"
- **iCaRL (증분 분류자 및 표현 학습)**
  - 클래스 증분 환경에서 분류기와 피처 표현을 동시에 학습
  - 세 가지 주요 구성 요소
    - 예제의 평균 근처에서의 분류
    - 허딩을 기반으로 하는 우선 순위 예제 선택
    - 지식 증류 및 프로토타입 재연습을 사용한 표현 학습

2. Method

2-1. 클래스 증분 분류기 학습

- 클래스 증분 형태의 데이터 스트림에서 동시에 분류기와 피처 표현을 학습
- **분류 작업**

- 데이터 스트림에서 동적으로 선택된 예제 이미지 집합인 $P1, ..., Pt$에 의존
  - 현재까지 관찰된 각 클래스에 대한 예제 집합이 하나씩 있으며, iCaRL은 총 예제 이미지 수가 고정된 매개변수인 K를 초과하지 않도록 보장
- 이미지를 현재까지 관찰된 클래스 집합으로 분류하는 데 평균 분류자를 사용

Algorithm 1 iCaRL CLASSIFY

input x // image to be classified
require $\mathcal{P} = (P_1, \ldots, P_t)$ // class exemplar sets
require $\varphi : \mathcal{X} \to \mathbb{R}^d$ // feature map 특성 추출
  for $y = 1, \ldots, t$ **do**
    $\mu_y \leftarrow \frac{1}{|P_y|} \sum_{p \in P_y} \varphi(p)$ // mean-of-exemplars
  end for
  $y^* \leftarrow \underset{y=1,\ldots,t}{\operatorname{argmin}} \|\varphi(x) - \mu_y\|$ // nearest prototype
output class label y^*

- **훈련 작업**
  - 점진적 학습 전략을 사용하여 클래스 배치를 처리
  - 새로운 클래스의 존재에 대한 학습 방법
    - 새로운 클래스에 대한 데이터가 사용 가능할 때마다 업데이트 루틴을 호출

      ⇒ iCaRL의 내부 지식(네트워크 매개변수 및 예제)을 새 관측 데이터(현재 훈련 데이터)에서 제공되는 추가 정보를 기반으로 조정

Algorithm 2 iCaRL INCREMENTALTRAIN

input $X^s, \ldots, X^t$ // training examples in per-class sets
input K // memory size
require Θ // current model parameters
require $\mathcal{P} = (P_1, \ldots, P_{s-1})$ // current exemplar sets
  $\Theta \leftarrow$ UPDATEREPRESENTATION$(X^s, \ldots, X^t; \mathcal{P}, \Theta)$
  $m \leftarrow K/t$ // number of exemplars per class
  for $y = 1, \ldots, s-1$ **do**
    $P_y \leftarrow$ REDUCEEXEMPLARSET(P_y, m)
  end for
  for $y = s, \ldots, t$ **do**
    $P_y \leftarrow$ CONSTRUCTEXEMPLARSET(X_y, m, Θ)
  end for
  $\mathcal{P} \leftarrow (P_1, \ldots, P_t)$ // new exemplar sets

- **구조**
  - iCaRL은 **컨볼루션 신경망(CNN)**을 사용
    - 해당 네트워크를 학습 가능한 피처 추출기인 $\varphi : \chi \to \mathbb{R}^d$로 해석

        - 뒤에 현재까지 관찰된 클래스 수와 동일한 시그모이드 출력 노드를 가진 단일 분류 레이어가 따름
        - 모든 피처 벡터는 L2 정규화 됨
        - 출력 결과

$$g_y(x) = \frac{1}{1 + \exp(-a_y(x))} \quad \text{with } a_y(x) = w_y^\top \varphi(x).$$

▲ 최종 분류 단계 아님, 표현 학습만을 위해 사용된 거임

- **자원 활용**
    - 증분적인 특성 때문에 iCaRL은 미리 알고 있는 클래스와 그 수에 대한 정보가 필요하지 않으며 이론적으로는 무제한으로 실행될 수 있음
    - 런타임 동안의 메모리 요구 사항
        - 피처 추출 매개변수의 크기
        - K개의 예제 이미지의 저장
        - 관찰된 클래스 수만큼의 가중치 벡터 크기
    - 학습해야 하는 클래스마다 적어도 하나의 예제 이미지와 가중치 벡터가 필요
        - 메모리 제한에 걸리지 않는 유한한 수의 클래스만 학습할 수 있음
        - 리소스를 추가로 할당하는 경우 재훈련 후에 실행 가능
    - iCaRL은 리소스를 추가로 할당하더라도 재훈련 없이 실행 가능

2-2. 최근접 평균 예제 분류

- 새 이미지 x에 대한 레이블 y^*를 예측하기 위해 iCaRL은 현재까지 관찰된 모든 클래스에 대해 프로토타입 벡터($\mu_1, \cdots, \mu_t$)를 계산
    - 클래스 y에 대한 모든 예제의 평균 피처 벡터
- 분류해야 할 이미지의 피처 벡터를 계산하여 가장 유사한 프로토타입을 가진 클래스 레이블을 할당

$$y^* = argmin \|\varphi(x) - \mu_y\|$$

- 최근접 평균 예제 분류 규칙은 점진적 학습 환경의 **두 가지 주요 문제**를 극복
    1. 일반적인 신경망의 분류 규칙은 선형의 분류기를 비선형의 특성 맵과 가중치 벡터와 함께 사용함

- 특성 맵(φ)이 변경될 때마다 가중치 벡터(w_y) 또한 업데이트 되어야 함
  - 하지만 가중치 벡터가 피처 추출 루틴에서 분리되어 있기에 클래스 증분 설정에서 문제가 발생
- 최근접 평균 예제 분류 방식은 분리된 가중치 벡터를 갖고 있지 않음
  - 클래스 프로토타입은 피처 표현이 변경될 때 자동으로 변경됨

2. 클래스 증분 설정에서는 실제 클래스 평균을 사용할 수 없음
   - 표현 변경 후 클래스 평균을 다시 계산하기 위해 모든 훈련 데이터를 저장해야 함
   - 대신 클래스 평균을 근사하는 방식으로 선택된 유연한 수의 예제를 평균으로 사용
   - 정규화된 피처 벡터를 사용
     ⇒ 분류 단계를 데이터 표현과 분리되지 않고 일관되게 변경되는 가중치 벡터로 해석할 수 있음

2-3. 표현 학습

- iCaRL은 새로운 클래스 $s, ..., t$에 대한 데이터 $X^s, ..., X^t$를 획득할 때마다 피처 추출 루틴과 예제 집합을 업데이트

Algorithm 3 iCaRL UPDATEREPRESENTATION

input $X^s, \dots, X^t$ // training images of classes $s, \dots, t$
require $\mathcal{P} = (P_1, \dots, P_{s-1})$ // exemplar sets
require Θ // current model parameters
// form combined training set:

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

// store network outputs with pre-update parameters:
for $y = 1, \dots, s-1$ **do**
  $q_i^y \leftarrow g_y(x_i)$ for all $(x_i, \cdot) \in \mathcal{D}$
end for
run network training (*e.g.* BackProp) with loss function

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

새로운 class 기존 정보들
that consists of *classification* and *distillation* terms.

1. 현재 사용 가능한 훈련 예제와 저장된 예제와 함께 구성된 증가된 훈련 세트를 생성
2. 각 예제에 대해 현재 네트워크를 평가하고 모든 이전 클래스에 대한 결과 네트워크 출력을 저장

   ⇒ 새 클래스에 대해서는 아직 훈련되지 않았으므로 해당되지 않음
3. 네트워크 매개변수는 새 이미지마다 네트워크가 올바른 클래스 식별자를 출력하도록 하는 손실 함수를 최소화하여 업데이트 됨(분류 손실)
   - 이전 단계에 저장된 점수를 재생산하도록 이전 클래스에 대해서는 증류 손실을 적용

     ▼ 증류 손실(distillation loss)

     - 모델 학습 과정에서 사용되는 손실 함수 중 하나로, 이전에 훈련된 모델의 예측 결과를 현재 훈련 중인 모델의 출력과 비교하여 새로운 모델을 학습시키는 데 사용

       ⇒ 이전 모델의 지식을 현재 모델로 전달하고 지식 전달을 통해 성능을 향상시키는데 도움을 주는 손실 함수

- **배경(Background)**
  - 일반적인 네트워크 fine-tuning과 유사
    - 이전에 학습한 네트워크 가중치를 시작으로 훈련 세트에서 손실 함수를 최소화
    - back propagation과 같은 표준 end-to-end 학습 방법을 사용할 수 있지만, 미니 배치/드롭아웃/적응형 단계 크기 선택/배치 정규화와 같은 사항들도 활용 가능
  - 카타스트로피 피치 방지/완화를 위한 fine-tuning 방식

    ▼ 카타스트로피 피치

    - 모델이 새로운 데이터를 학습하려고 할 때 이전에 학습한 데이터를 잊어버리는 현상
      - 모델이 새로운 작업 또는 클래스를 학습하기 위해 기존의 학습 내용을 지우거나 덮어쓰는 것을 의미

    1. 훈련 세트 확장
       - 새로운 훈련 예제 뿐만 아니라 이미 저장된 예제도 훈련 세트에 포함
         - 모든 이전 클래스의 데이터 분포에 관한 정보가 훈련 과정에 들어감
       - 예제가 시간이 지남에 따라 더 이상 사용되지 않는 피처 표현이 아닌 이미지로 저장되어야 한다는 점이 중요

2. 손실 함수 확장
    - 표준 분류 손실 외에도 새로 관찰된 클래스를 잘 분류할 수 있도록 피처 표현의 개선을 장려하는 것 외에 증류 손실도 포함
        - 이전에 학습한 판별 정보가 새로운 학습 단계 동안 손실되지 않도록 보장

2-4. 예제 관리

- iCaRL은 새로운 클래스를 만나면 예제 집합을 조정
    - 모든 클래스는 동등하게 처리됨
    - 현재까지 관찰된 클래스 수가 t이고 저장할 수 있는 총 exemplar 수가 K인 경우, iCaRL은 각 클래스당 $m = K/t$ 개의 exemplar를 사용(반올림 포함)
- **exemplar 관리를 담당하는 두 가지 루틴**
    1. 새 클래스를 위한 exemplar

Algorithm 4 iCaRL CONSTRUCTEXEMPLARSET

input image set $X = \{x_1, \dots, x_n\}$ of class y
input m target number of exemplars // $m = \frac{K}{t}$ ← 총 예제 개수 / ← # class
require current feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$
  $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$ // current class mean
  for $k = 1, \dots, m$ **do** // 각 클래스 별 예제가 동등한 개수가 될 때까지
    $p_k \leftarrow \underset{x \in X}{\operatorname{argmin}} \left\| \mu - \frac{1}{k} [\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j)] \right\|$
  end for
  $P \leftarrow (p_1, \dots, p_m)$
output exemplar set P

        - 반복의 각 단계에서 현재 훈련 세트의 한 예제가 exemplar 집합에 추가
            - 해당 예제는 모든 exemplar의 평균 피처 벡터가 모든 훈련 예제의 평균 피처 벡터를 가장 잘 근사화하는 예제
        - exemplar는 우선 순위가 있는 목록
            - 목록의 요소 순서가 중요하며, 목록 앞쪽에 있는 exemplar가 더 중요
    2. 기존 클래스의 exemplar 집합 크기를 줄이기(exemplar 제거)
        - exemplar $p_{m+1}, ..., p_{m'}$을 폐기하고 $p_1, ..., p_m$만 유지
- **배경**
    - exemplar 관리 루틴은 **두 가지 목표**를 감안하여 설계됨

1. 초기 exemplar 집합은 클래스 평균 벡터를 잘 근사화해야 함
2. 알고리즘 실행 중에 언제든지 exemplar를 제거할 수 있어야 함

- 제거 프로시저가 호출될 때 알고리즘에 실제 클래스 평균 벡터가 더 이상 사용 불가능하기 때문에 어려운 문제

  ⇒ 데이터에 독립적인 제거 전략을 채택

  ⇒ 우선순위 생성

  - 어떤 exemplar 하위 집합의 평균 피처 벡터가 처음부터 시작됨 → 평균 벡터의 좋은 근사화
  - herding에서도 분포에서 대표적인 샘플 세트를 만들기 위해 사용됨
    - 반복적인 선택이 랜덤 샘플링과 같은 더 적은 샘플로 높은 근사화 품질을 달성하는 데 더 적은 샘플이 필요함을 보여줌
- 전반적으로 iCaRL의 exemplar 선택 및 축소 단계는 점진적 학습 환경에 정확하게 맞아 떨어짐
  - **선택 단계**는 각 클래스마다 처음 관찰될 때 한 번만 필요

    → 해당 클래스의 훈련 데이터가 처음 사용 가능한 경우
  - 나중에는 예전 훈련 데이터에 액세스하지 않아도 되는 **축소 단계**만 호출됨

3. 관련 연구

3-1. 고정된 데이터 표현과 함께 학습

클래스 증분 학습의 주요 과제

모든 훈련 데이터에 대한 액세스 없이 **언제든지** 훈련 과정 중에 새로운 클래스를 수용할 수 있는 분류기 아키텍처 설계

- **클래스 평균 분류기(NCM)**
  - 장점
    - 각 클래스를 해당 클래스에 대해 지금까지 관찰된 모든 예제의 평균 피처 벡터로 나타냄
      - 데이터 스트림에서 점진적으로 계산할 수 있으므로 모든 훈련 예제를 저장할 필요가 없음

        - 새로운 예제는 해당 예제의 피처 벡터에 대한 메트릭을 기반으로 예제의 피처 벡터와 가장 유사한 프로토타입을 가진 클래스 레이블을 할당함으로써 분류
    - 단점
        - 선형의 분류기와 함께 비선형의 데이터 표현을 학습해야 함
            - 클래스 평균 벡터가 점진적으로 계산되지 못함
              → 쉽게 확장할 수 없음
    - iCaRL에의 시도
        - 모든 예제의 평균 피처 벡터가 아닌 특별히 선택된 하위 집합에 대한 프로토타입을 활용

          → 작은 메모리를 유지하고 모든 필요한 업데이트를 일정한 계산 노력으로 수행할 수 있도록 함
- 여러 시도가 있었지만, 클래스 증분 학습에서 발생할 수 있는 모든 문제를 해결하진 못함

3-2. 표현 학습

- 최근의 심층 신경망의 성공은 분류기 뿐만 아니라 적절한 데이터 표현도 학습할 수 있는 능력에 크게 기인함
    - 고전적인 작업은 주로 연결주의 메모리 네트워크와 관련되어 있었고, 현대 표준에 비해 작고 얕은 네트워크를 사용했음
- 초기 연결주의 작업의 주요 성과는 잊혀짐 현상을 어떻게 다룰 수 있는지에 대한 두 가지 주요 전략을 식별한 것임
    1. **동결/확장 전략**
       ⇒ 일부 네트워크 가중치를 고정하면서 네트워크를 확장하여 학습 능력을 보존
    2. **리허설**
       ⇒ 네트워크를 가장 최근 데이터뿐만 아니라 이전 데이터로도 지속적으로 자극
- 최근 연구에서는 주로 동결/확장 전략을 따르는 반면, iCaRL은 리허설의 원칙을 채택
    - 표현을 학습하기 위한 모델 매개변수를 업데이트하기 위해 현재 사용 가능한 클래스의 훈련 데이터뿐만 아니라 이전 클래스의 예제도 사용
    - 정보가 시간이 지남에 따라 네트워크에서 너무 많이 저하되지 않도록 하기 위해 증류(increment)도 사용

4. 실험 설계

- **평가 프로토콜**
  - 주어진 다중 클래스 분류 데이터 집합에 대해 클래스를 고정된 무작위 순서로 배치
    - 각각의 집합은 사용 가능한 훈련 데이터에 대해 증분 방식으로 훈련
  - 각 클래스 배치 이후 결과 분류기가 데이터 집합의 테스트 부분 데이터에 대해 평가
    - 이미 훈련된 클래스만 고려
    - 테스트 결과는 알고리즘에 공개되지 않음 → 테스트 데이터가 한 번 이상 사용되더라도 알고리즘에는 overfitting이 발생하지 않음
  - **평균 증분 정확도**로 평가
    - 각 클래스 배치 후의 분류 정확도 곡선
    - 하나의 숫자가 선호되는 경우 이러한 정확도의 평균을 보고
- **이미지 분류 작업**

  1. iCIFAR-100
     - CIFAR-100 데이터를 사용
     - 모든 100개 클래스를 2, 5, 10, 20 또는 50개씩 배치로 훈련
     - 평가 메트릭: 테스트 집합의 표준 다중 클래스 정확도
     - 데이터 집합 크기를 관리할 수 있음
       → 클래스 순서가 다른 다양한 실행에서 10번 실행하고 결과의 평균과 표준 편차를 보고
  2. iILSVRC
     - ImageNet ILSVRC 2012 데이터 집합을 두 가지 설정에서 사용

       a. 100개 클래스의 하위 집합을 사용(iILSVRC-small)
       - 하위 집합은 10개씩 배치로 훈련

       b. 1000개 클래스 사용(iILSVRC-full)
       - 100개씩 배치로 처리
- **iCaRL 구현**

  1. iCIFAR-100
     - theano 패키지를 사용
     - 32층 ResNet을 훈련

- 최대 예제 수: K = 2000
- 각 훈련 단계는 70 에포크로 구성
- 학습률
  - 처음: 2.0
  - 49번 epoch(전체 epoch의 7/10) → 0.4
  - 63번 epoch(전체 epoch의 9/10) → 0.08

2. iILSVRC
   - tensorflow 프레임워크를 사용
   - 18층 ResNet을 훈련
   - 최대 예제 수: K = 20000
   - 각 훈련 단계는 60 에포크로 구성
   - 학습률
     - 처음: 2.0
     - 20번 epoch: 0.4
     - 30번 epoch: 0.08
     - 40번 epoch: 0.016
     - 50번 epoch: 0.0032
   - 두 방법 모두 back-propagation을 활용
     - 네트워크를 128 크기의 미니배치로 훈련
     - 가중치 감소 매개변수: 0.00001

4-1. 결과

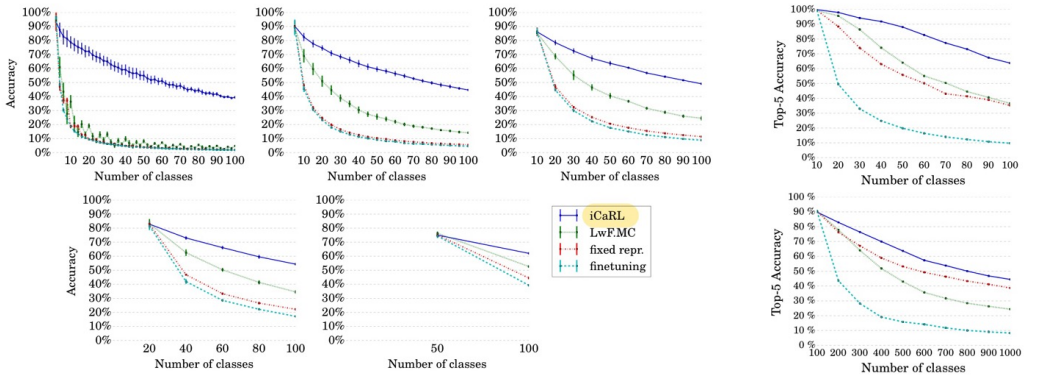
- iCaRL 외에도 세 가지 대안적인 클래스 증분 방법을 구현/ 테스트

1. Fine-tuning
   - 이전에 학습한 다중 클래스 분류 네트워크를 세밀하게 조정하여 새로운 들어오는 클래스에 대한 다중 클래스 분류기를 학습
2. 고정된 표현(Fixed representation)
   - 치명적인 잊혀짐을 방지하는 방식

- 클래스 배치의 첫 번째 일관성을 처리하고 해당 클래스의 가중치를 처리한 후 특징 표현을 고정
- 이후 클래스 배치에 대해서는 새로운 클래스의 가중치 벡터만 훈련

3. Learning without Forgetting(LwF.MC)
    - 학습 중에 iCaRL과 같이 distillation 손실을 사용하여 치명적인 잊혀짐을 방지
    - 예제 세트를 활용하는 것이 아닌 그들만의 네트워크 출력값을 활용

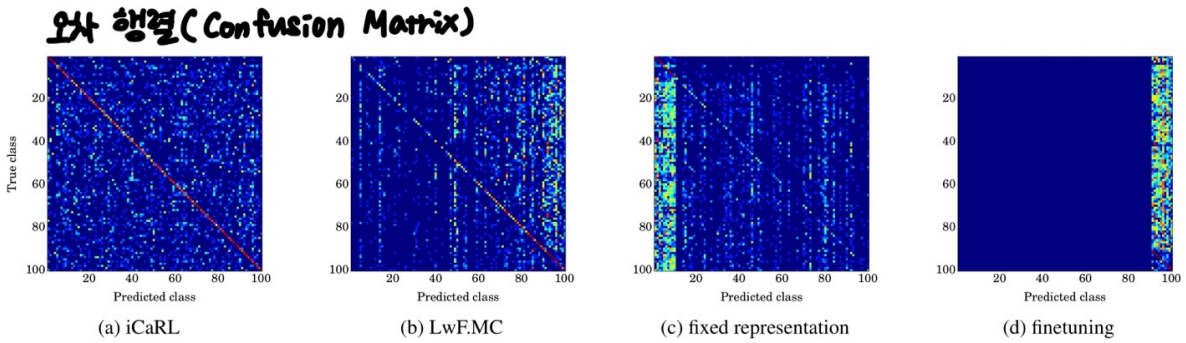
- **결과 정리**



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

(b) Top-5 accuracy on iILSVRC-small (top) and iILSVRC-full (bottom).

    - **iCaRL**이 다른 방법들보다 훨씬 더 우수한 성능을 보임
        - 특히 점진적인 환경(= 동시에 처리할 수 있는 클래스가 적을수록)에서 우수
    - 다른 방법 중에서도 distillation 기반 네트워크 훈련(LwF.MC)이 항상 두 번째로 좋음
        - iILSVRC-full의 경우 처음 100개 클래스 배치 후에 표현을 고정하는 것이 더 나은 결과를 가져옴
    - Fine-tuning은 항상 최악의 결과를 보임
        - 치명적인 잊혀짐이 다중 클래스 증분 학습에서 실제로 주요 문제임을 확인할 수 있음



▲ **iCIFAR-100**을 사용하여 10개 클래스씩 배치로 훈련한 후의 100개 클래스 분류기의 confusion matrix, iILSVRC에 대한 혼동 행렬도 동일한 패턴을 보임

- iCaRL
  - 모든 클래스에 대해 균질한 모습을 보이며, 대각선 항목(즉, 올바른 예측)과 비대각선 항목(즉, 실수) 모두에 대해 마찬가지
  - iCaRL이 학습 중에 일찍 또는 나중에 마주치는 클래스에 대한 본질적인 편견이 없음을 보여줌

    ⇒ 치명적인 잊혀짐에 영향을 받지 않음
- distillation 기반 훈련 (LwF.MC)
  - 오른쪽(= 최근에 학습한 클래스)으로 가면서 더 많은 비대각 항목을 가지고 있음
- Fine-tuning
  - 모든 예측 클래스 레이블은 네트워크가 훈련된 마지막 클래스 배치에서 도출됨
  - 훈련된 네트워크는 이전 클래스조차 존재하지 않음 → 단순히 잊어버림
- Fixed representation
  - 데이터 표현을 얻을 때 사용한 첫 번째 클래스 배치의 클래스를 출력하기를 선호

4-2. 미분 분석(Differential Analysis)

- iCaRL의 작동 메커니즘을 더 자세히 이해하기 위해 iCIFAR-100에서 개별 측면을 분리한 추가 실험을 수행
  - iCaRL이 일반적인 미세 조정 기반 교육과 어떻게 다른지를 분석
    1. 평균-예제자 분류 규칙의 사용
    2. 표현 학습 중 예제자의 사용
    3. 증류 손실의 사용
       - 세 가지 하이브리드 설정을 추가

1. iCaRL과 동일한 방식으로 표현을 학습하지만, 분류 시 평균-예제 분류자 대신 네트워크 출력을 직접 사용하는 방법
2. 분류를 위해 exempler를 사용하지만 훈련 중 증류 손실을 사용하지 않는 방법
3. 분류 또는 증류 손실을 사용하지 않지만 표현 학습 중에 exempler를 사용

비교를 위해 증류를 사용하지만 exempler를 전혀 사용하지 않는 LwF.MC도 포함

(a) Switching off different components of iCaRL (*hybrid1*, *hybrid2*, *hybrid3*, see text for details) leads to results mostly inbetween iCaRL and LwF.MC, showing that all of iCaRL's new components contribute to its performance.

batch size	iCaRL	*hybrid1*	*hybrid2*	*hybrid3*	LwF.MC
2 classes	57.0	36.6	57.6	57.0	11.7
5 classes	61.2	50.9	57.9	56.7	32.6
10 classes	64.1	59.3	59.9	58.1	44.4
20 classes	67.2	65.6	63.2	60.5	54.4
50 classes	68.6	68.2	65.3	61.5	64.5

높은 성능 중간 정도 성능 낮은 성능

▲ 표 1-a. 점진적 교육의 모든 단계에서 분류 정확도의 평균을 요약한 결과

- 대부분의 경우 하이브리드 설정이 iCaRL과 LwF.MC 사이의 결과를 달성
  - 실제로 iCaRL의 모든 새로운 구성 요소가 그 성능에 상당히 기여한다는 점을 시사
- iCaRLvs hybrid 1
  - 평균-예제자 분류자가 작은 배치 크기, 즉 표현 업데이트가 더 많이 수행될 때 특히 유리함
- iCaRL vs hybrid 2
  - 매우 작은 클래스 배치 크기에서는 표준 프로토타입만 사용하는 것에 비해 증류를 사용하면 오히려 분류 정확도가 감소할 수 있음을 시사
  - 더 큰 배치 크기(= 더 적은 업데이트)에 대해서는 증류 손실의 사용이 분명히 유리
- hybrid3 vs LwF.MC
  - exempler가 잊힘을 방지하는 데 효과적임

- 평균-예제자를 사용하는 대신 최근접 클래스 평균(NCM) 규칙을 사용하는 경우 얼마나 많은 정확도가 손실되는지를 연구

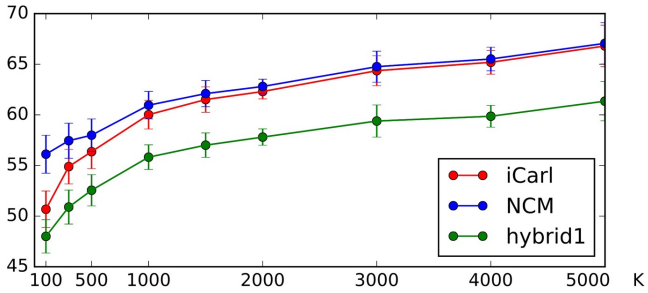
- NCM의 경우 iCaRL의 데이터 표현을 학습할 때 네트워크의 출력을 사용하지만 현재의 특징 추출기를 사용하여 각 표현 업데이트 후에 현재 클래스 평균을 다시 계산하여 이미지를 분류'

  ⇒ 모든 훈련 데이터를 저장해야 하므로 클래스 증가 방법으로는 적합하지 않음

batch size	iCaRL	NCM
2 classes	57.0	59.3
5 classes	61.2	62.1
10 classes	64.1	64.5
20 classes	67.2	67.5
50 classes	68.6	68.7

차이가 미미함

▲ 표 1-b. 오히려 NCM이 성능이 더 좋은데..?



- 표 1a의 하이브리드1 분류기와 표 1b의 NCM 분류기를 비교
  - 모든 방법이 더 큰 메모리를 활용 시 성능이 좋음
  - iCaRL의 평균-예제자 분류기는 충분한 프로토타입이 있는 경우 NCM 분류기와 유사한 성능을 발휘하지만 네트워크 출력으로 분류하는 것은 경쟁력이 없음

5. 결론

- **iCaRL**이라는 클래스 증분 학습을 위한 전략을 소개
- iCaRL은 분류기와 특징 표현을 동시에 학습
- **구성 요소**
  - 데이터 표현의 변화에 견고하면서도 각 클래스 당 소량의 예제자(exemplar)만 저장해야 하는 최근접 평균 예제자 분류기
  - 우선순위를 부여하는 예제자 선택을 위한 herding 기반 단계

    - 치명적인 잊혀짐(catastrophic forgetting)을 피하기 위해 예제자를 사용하며 증류(distillation)를 결합하는 표현 학습 단계
- iCaRL의 강력한 분류 결과의 주요 원인: 예제 이미지(exempler)의 사용
- iCaRL의 성능은 여전히 모든 클래스의 모든 훈련 예제를 동시에 사용하는 배cl 설정에서 시스템이 달성하는 것보다 낮음 → 개선 필요