iCaRL (Incremental Classifier & Representation Learning)

- Class-incremental learning: 필수 구성 요소 두가지

CL의 3가지 시나리오

1. 연속성 있는 데이터로부터(*지속적으로* 클래스가 추가되는, 데이터 스트림) 모델이 항상 훈련 가능 해야함 **(Update Routine Algorithm)**
2. 모델은 여태까지 관찰한 모든 클래스에 대해 multi-class classifacation 해야 함

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, 네트워크를 *trainable feature extractor라고 생각해*

하나의 classification 레이어가 있으며, 여러 클래스에 대한 sigmoid output 노드가 있음.

모든 피쳐 벡터는 L2-normalized, 되어 있으며, 네트워크의 파라미터들은 Θ로 정의

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

t는 여태까지 관찰한 클래스의 수, 따라서 모델의 아웃풋은 y ∈ {1, . . . , t}, 이걸로 실제 classifiacation하는 것은 아님

⇒ 여기에서 CNN 모델은 Representation Learning을 위한 feature 생성을 위해 쓰이는 거

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

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

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

- 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 변경하는 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.
- 앞서 말했듯이 모든 feature vector들은 l2-normalized 되어 있기 때문에 새로운 x에 대한 라벨 y*은 다음과 같이 쓰일 수도 있음(이해 X)

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

Representation Learning

- ICaRL은 augemtned된 훈련 데이터셋을 가짐, 현재 이용 가능한 훈련 샘플과 저장된 exemplar로 구성됨.

// form combined training set:

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

- 새로운 클래스가 들어오기 전에, 이전 클래스로 학습한 네트워크의 아웃풋(추론 결과)은 모두 저장됨, 왜냐면 이후에 쓰이는 knowledge distillation에서 distallation loss랑 섞어 쓸 거야, 기존 네트워크에 대한 정보도 잊지 않기 위해

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

- 최종적으로 각각의 새로운 이미지에 대해서 새로운 클래스를 출력하게 하는 로스(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 클래스인지 아닌지를 **확률**로서 결정한다.

- 따라서 평범한 파인 튜닝과 다른 점 두 가지

1. train 데이터셋이 augment 되어있음. 새로운 훈련 샘플뿐만 아니라 이전에 저장된 exemplar들을 포함함. 이를 통해 이전에 학습한 클래스의 데이터 분포에 대한 정보를 얻을 수 있음, 여기서 저장된 exemplar들은 feature representation이 아닌 image로써 저장되어있다는 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

- iCaRL에서 새로운 클래스가 들어올 때마다, exemplar set들은 스스로 조정됨

- 여태까지 t 클래스가 관찰되었고, 총 K개의 exemplar들이 저장되어 있다면 각 클래스 별로 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 사실 우선순위가 있는 리스트다. 가장 먼저 있는 exemplar가 더 중요하다. 그래서 class가 늘어날 수록 exemplar를 삭제하게 되면 뒤에서 부터 한다

- 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. 반면에 다른 방법을 좋은 근사 퀄리티를 보장하지 않았음.

Experiments

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)

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

→ batchsize 128, weight decay 0.00001

네트워크 계층 에서 binary cross-entropy 사용

적은 러닝 레이드는 멀티 클래스 소프트맥스에서 필요할 것,

- Fine-tuning: catastrophic forgetting에 대한 어떤 방안도 없는
- Fixed representation: 클래스의 첫번째 배치 이후로 feature representation을 얼림, 해당하는 클래스가 진행되면 그리고 classification layer의 weights도 얼림 (보존)

  → 두번째 배치부터는 새로운 클래스의 weight vector만 학습된
- LwF.MC: iCaRL처럼 distillation loss를 사용, 하자만 exemplar set을 사용하지 않음
- iCaRL:

1. 분류 시에 mean-of-exemplar 사용,
2. representation learning에서 exemplar를 사용,
3. distillation loss를 사용

iCaRL의 3가지 요소에 대한 차분 분석

hybrid1: 2,3번은 사용하되, 1번은 사용 안함

hybrid2: 1,2번은 사용하되, 3번은 사용 안함

hybrid3: 2번은 사용하되, 1,3번은 사용 안

LwF.MC : 3번은 사용하되, 1,2번은 사용 안함