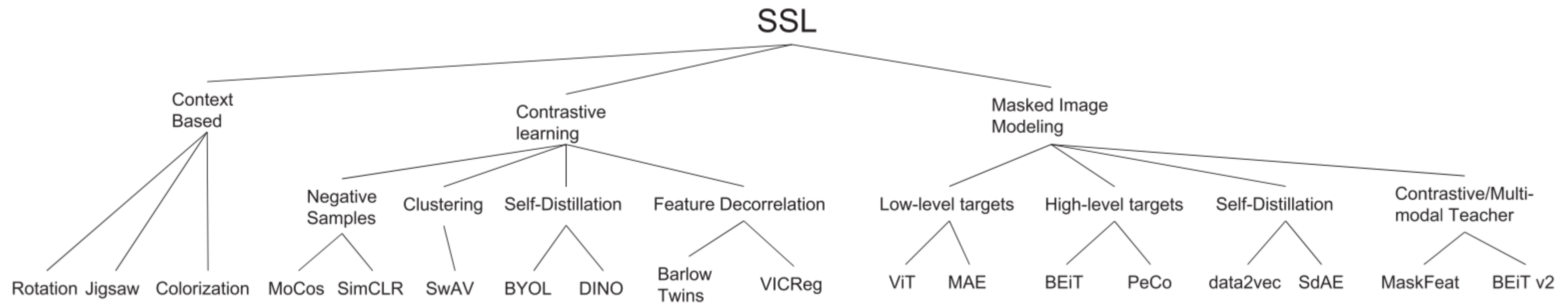


Sesión 5.0

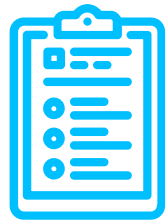
Self-supervised learning I

Contrastive learning

Self-Supervised *Learning*



1.



Context-Based *Methods*

Rotation

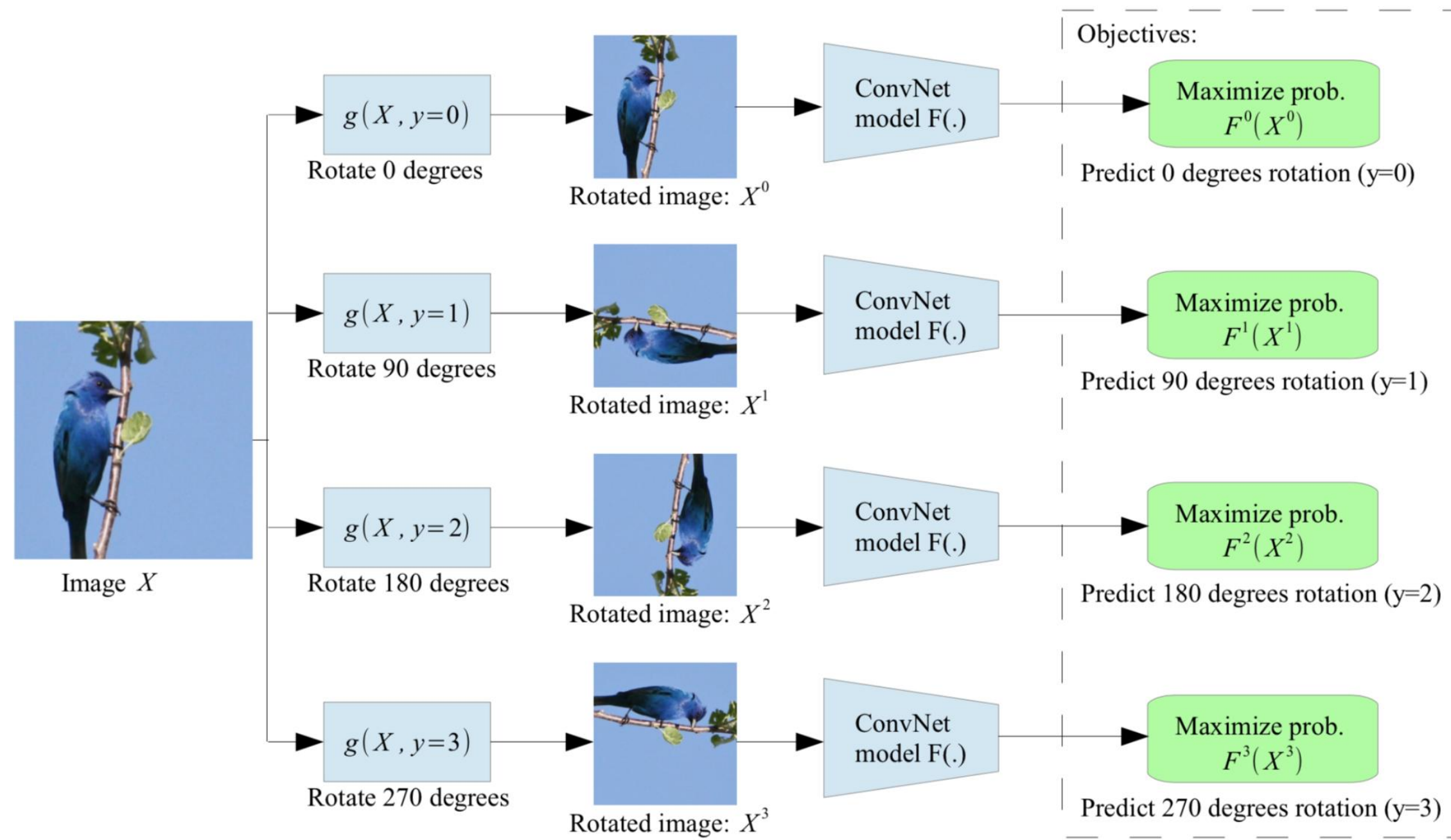
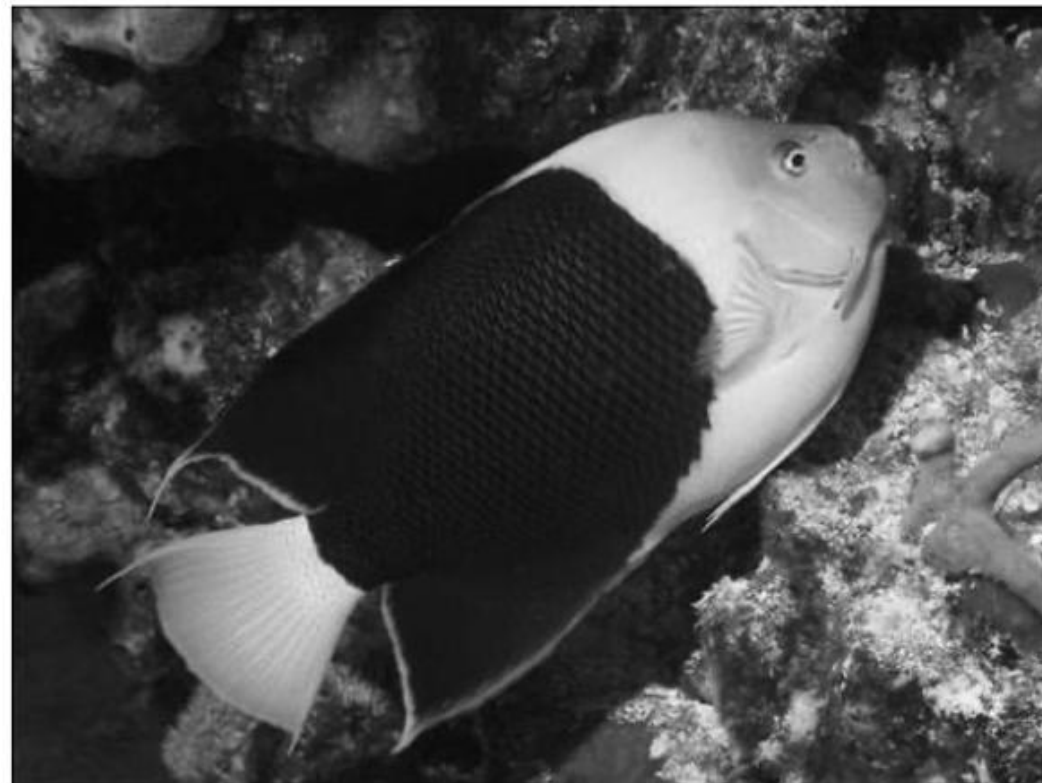


Image *Colorization*



Grayscale image: L channel

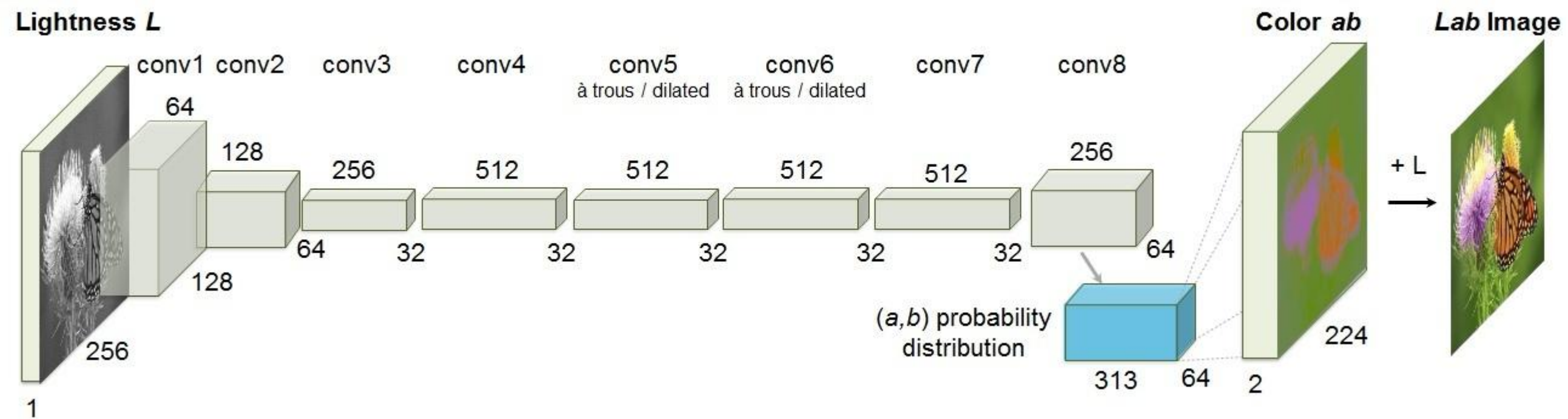
$$\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$$

Concatenate (L,ab)

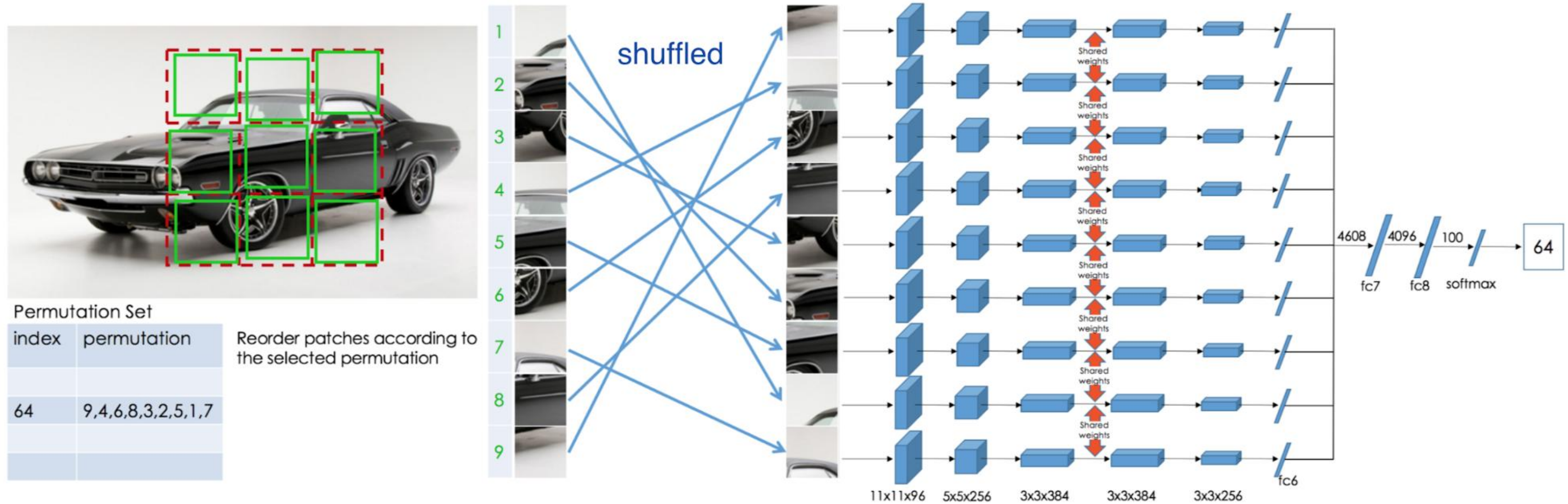
$$(\mathbf{X}, \hat{\mathbf{Y}})$$



Image *Colorization*

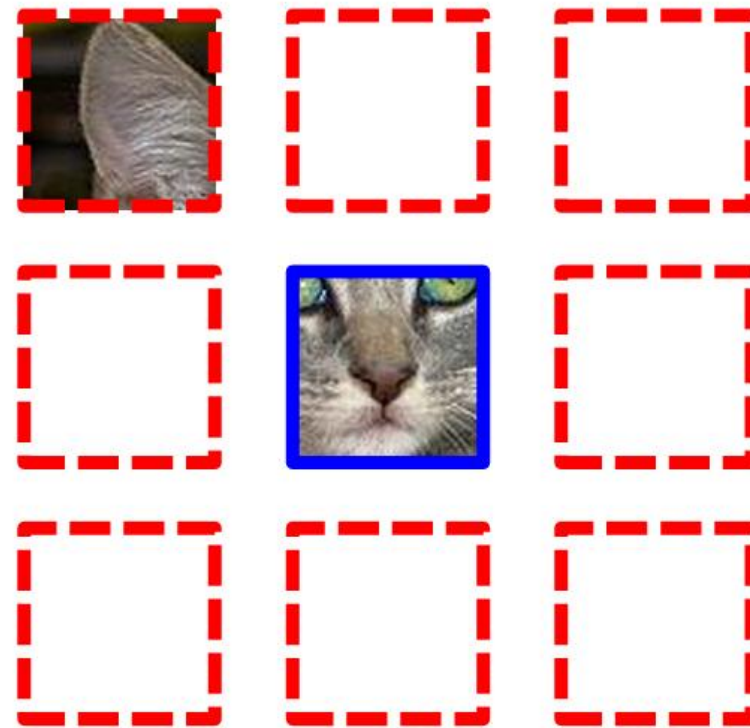


Jigsaw *Puzzles*



Context *Prediction*

Example:



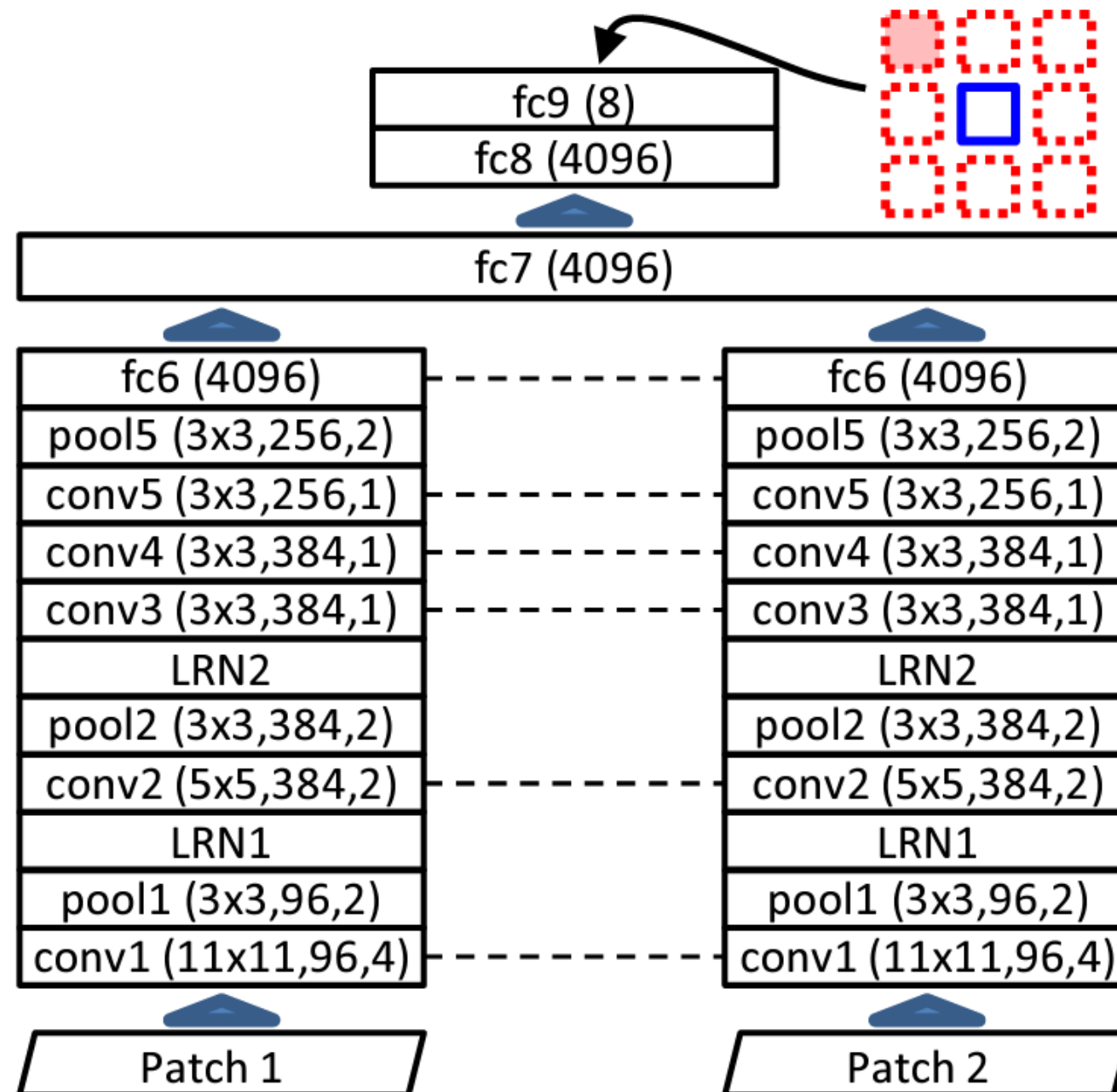
Question 1:



Question 2:



Context *Prediction*



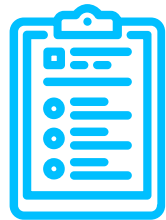
Question 1:



Question 2:

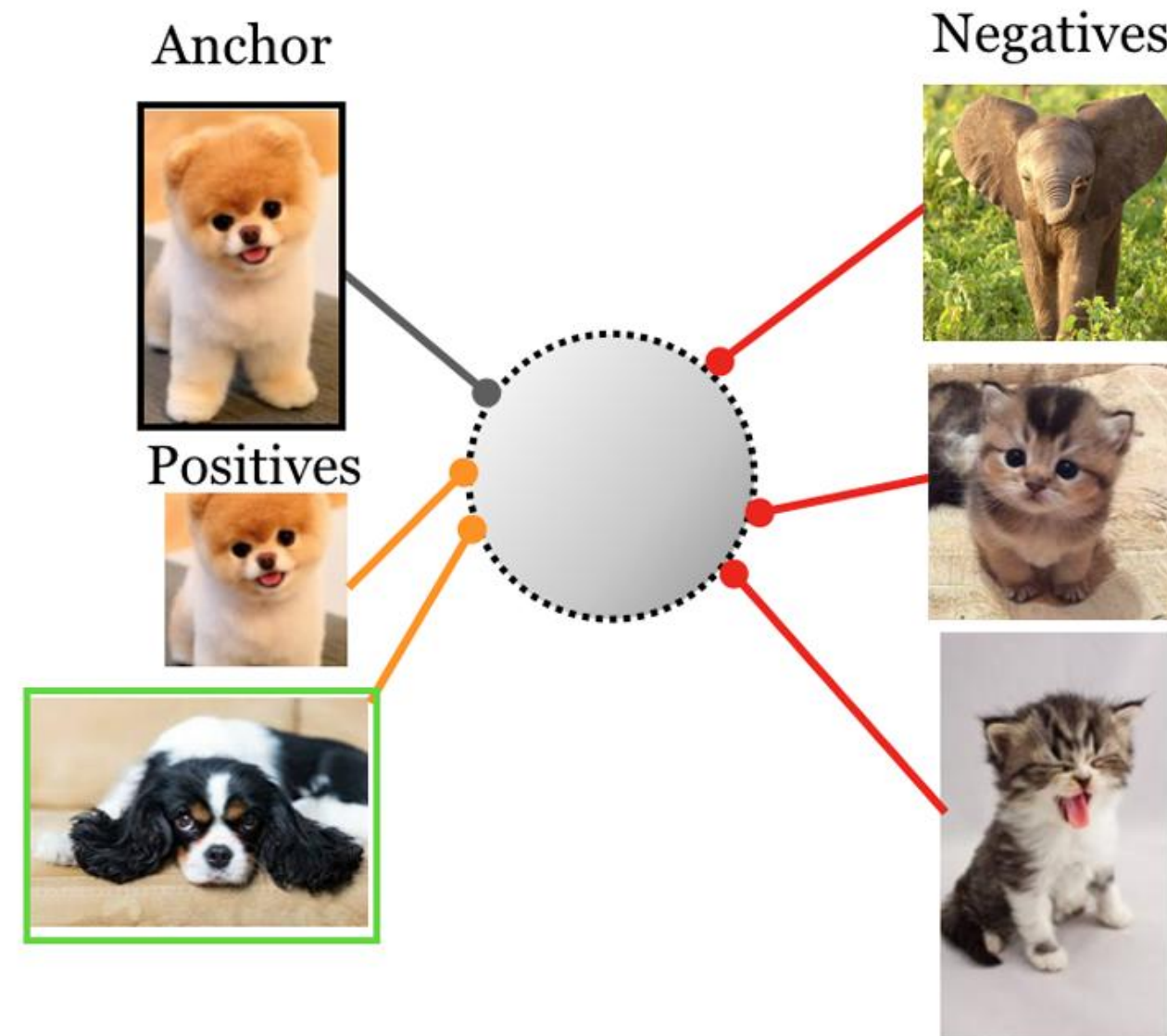


2.



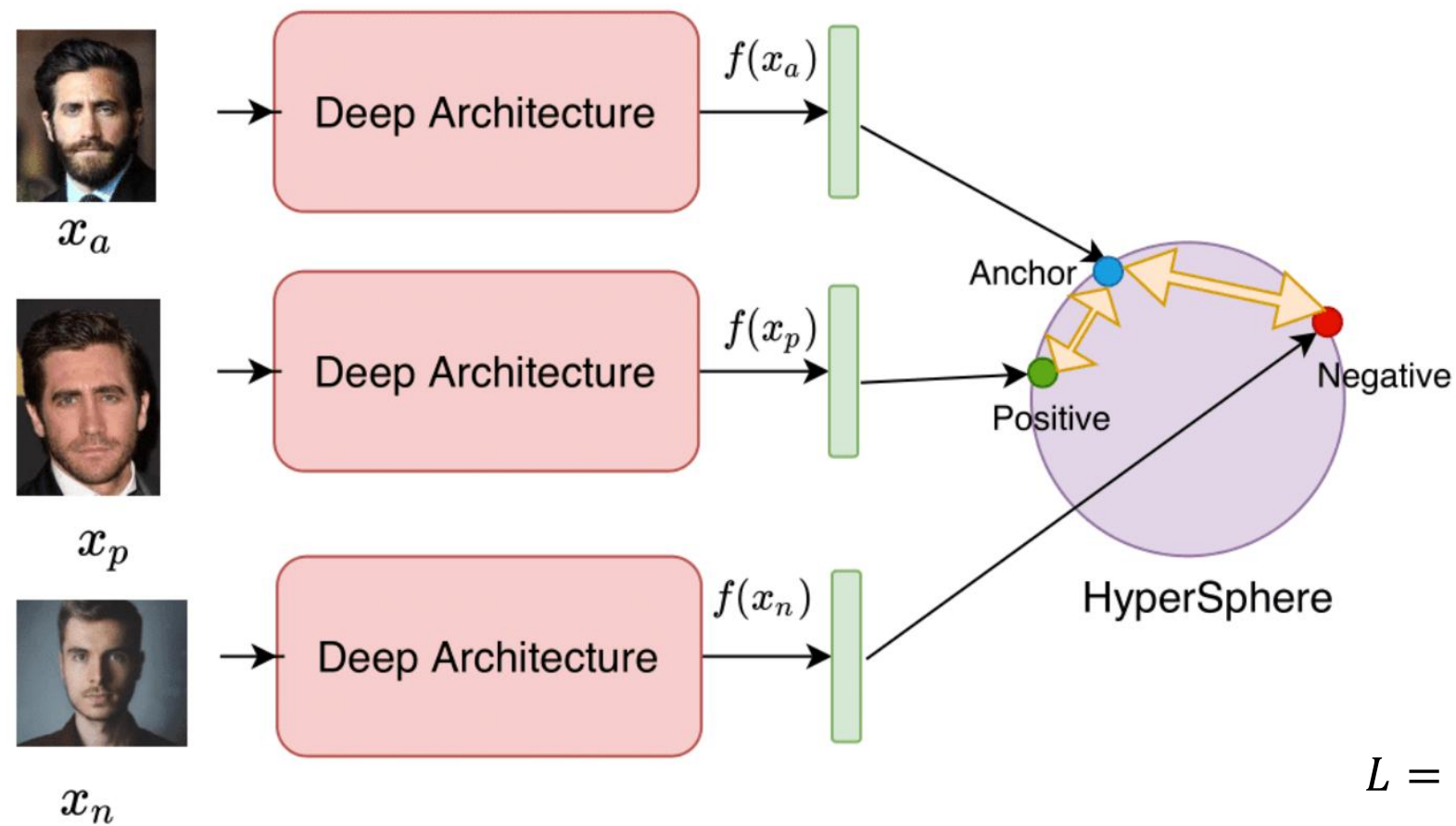
Contrastive *Learning*

Self-Supervised *Learning*



Triplet Loss

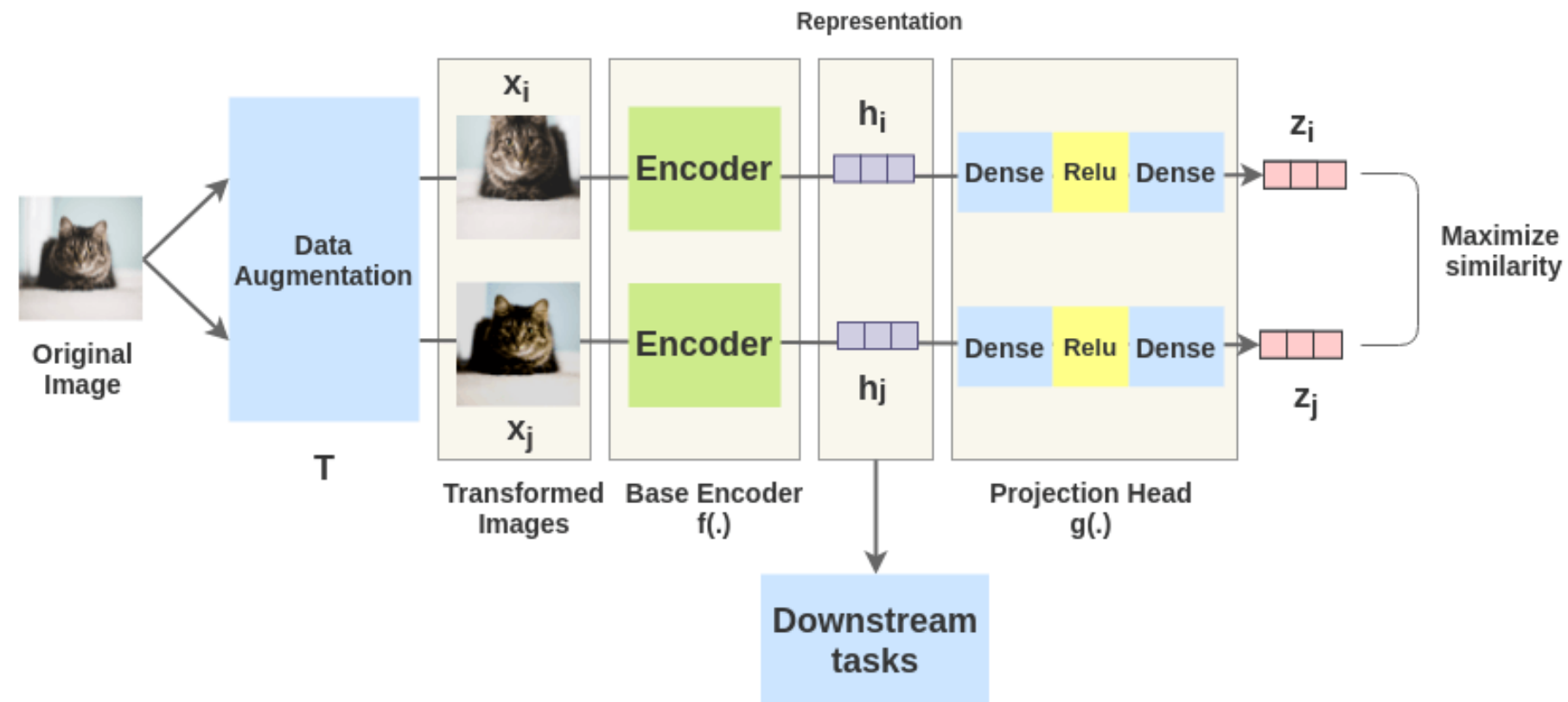
Es una técnica de **aprendizaje supervisado**, el cual es predecesor de propuestas en self-supervised.



$$L = \sum_i^N \left[|f(x_i^a) - f(x_i^p)|_2^2 - |f(x_i^a) - f(x_i^n)|_2^2 + \alpha \right]$$



SimCLR



Normalized Temperature-scaled Cross-Entropy (NT-Xent):

$$\mathcal{L}_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} 1[k \neq i] \exp(\text{sim}(z_i, z_k)/\tau)}$$

donde $\text{sim}(z_i, z_j) = \frac{z_i \cdot z_j}{|z_i| |z_j|}$

SimCLR



(a) Original



(b) Crop and resize



(c) Crop, resize (and flip)



(d) Color distort. (drop)



(e) Color distort. (jitter)



(f) Rotate $\{90^\circ, 180^\circ, 270^\circ\}$



(g) Cutout



(h) Gaussian noise



(i) Gaussian blur



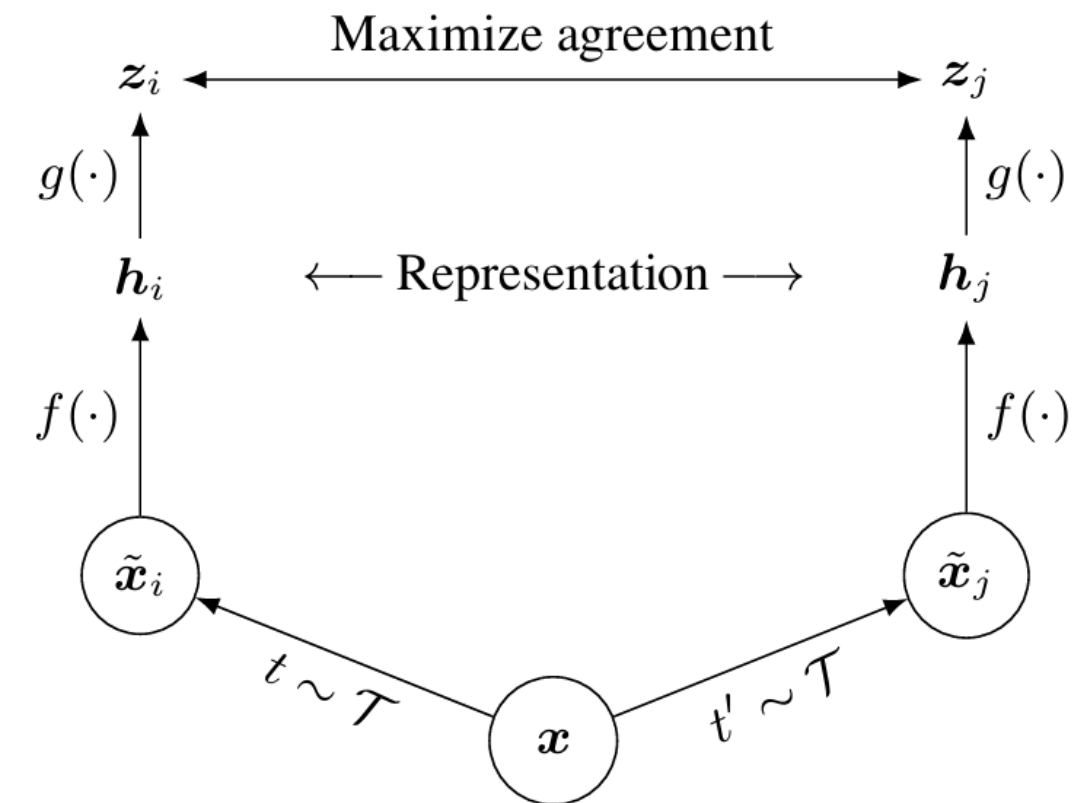
(j) Sobel filtering



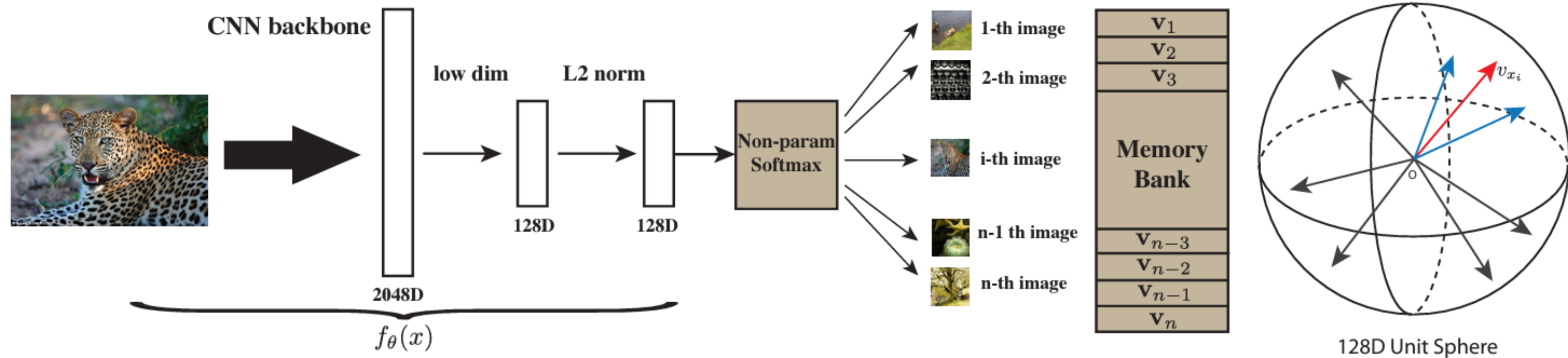
SimCLR

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N , constant τ , structure of f, g, \mathcal{T} .
for sampled minibatch $\{\mathbf{x}_k\}_{k=1}^N$ **do**
 for all $k \in \{1, \dots, N\}$ **do**
 draw two augmentation functions $t \sim \mathcal{T}, t' \sim \mathcal{T}$
 # the first augmentation
 $\tilde{\mathbf{x}}_{2k-1} = t(\mathbf{x}_k)$
 $\mathbf{h}_{2k-1} = f(\tilde{\mathbf{x}}_{2k-1})$ # representation
 $\mathbf{z}_{2k-1} = g(\mathbf{h}_{2k-1})$ # projection
 # the second augmentation
 $\tilde{\mathbf{x}}_{2k} = t'(\mathbf{x}_k)$
 $\mathbf{h}_{2k} = f(\tilde{\mathbf{x}}_{2k})$ # representation
 $\mathbf{z}_{2k} = g(\mathbf{h}_{2k})$ # projection
 end for
 for all $i \in \{1, \dots, 2N\}$ and $j \in \{1, \dots, 2N\}$ **do**
 $s_{i,j} = \mathbf{z}_i^\top \mathbf{z}_j / (\|\mathbf{z}_i\| \|\mathbf{z}_j\|)$ # pairwise similarity
 end for
 define $\ell(i, j)$ **as** $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
 $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
 update networks f and g to minimize \mathcal{L}
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$

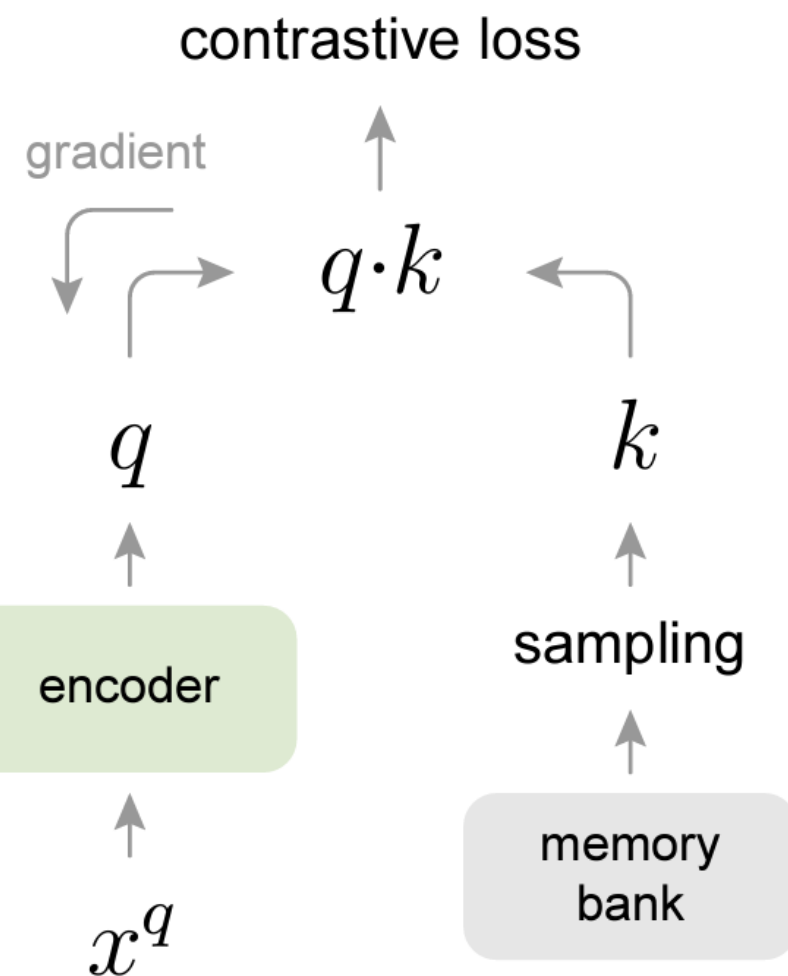


Noise-Contrastive *Estimation (NCE)*



Noise-Contrastive *Estimation (NCE)*

E



La probabilidad de que una imagen v pertenezca a una instancia i está dada por:

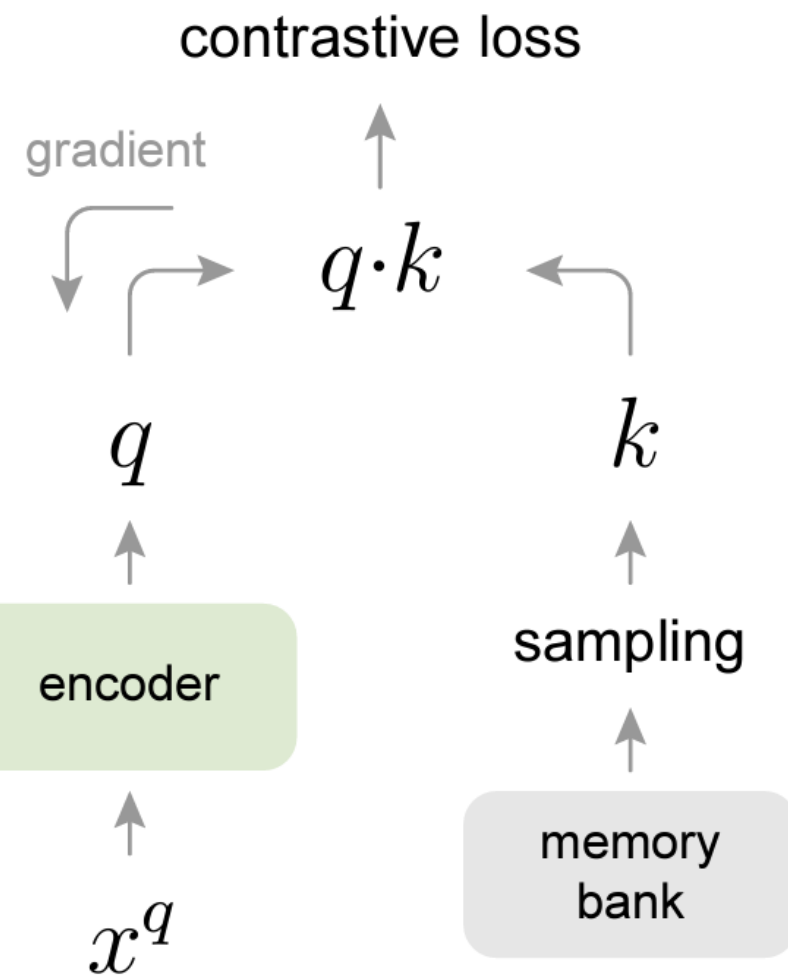
$$p(i|v) = \frac{\exp(v_i^T v / \tau)}{\sum_{j=1}^n \exp(v_j^T v / \tau)}$$

Loss function:

$$J(\theta) = -\sum_{i=1}^n \log p(i|f_{\theta}(x_i))$$



Noise-Contrastive *Estimation (NCE)*



La probabilidad de que una imagen v pertenezca a una instancia i está dada por:

$$p(i|v) = \frac{\exp(v_i^T v / \tau)}{\sum_{j=1}^n \exp(v_j^T v / \tau)}$$

Loss function:

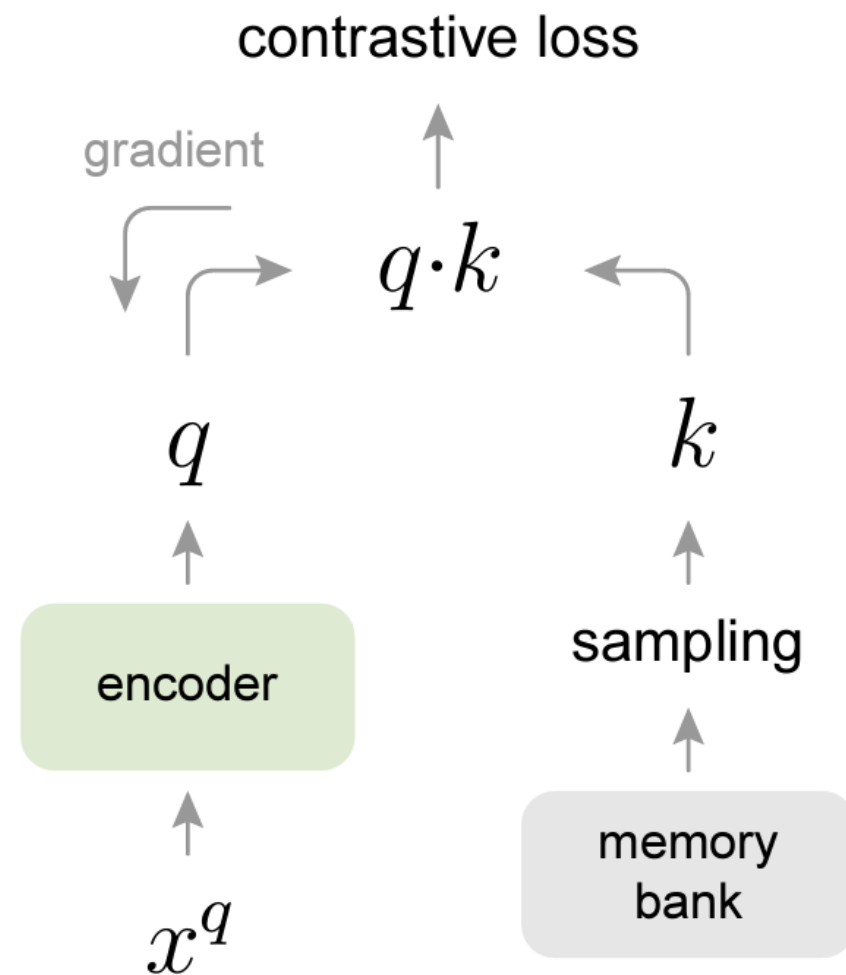
$$J(\theta) = -\sum_{i=1}^n \log p(i|f_{\theta}(x_i))$$

Problemas:

- Muchas clases (una por muestra en la memoria).
- Es difícil calcular $p(i|v)$



Noise-Contrastive *Estimation (NCE)*



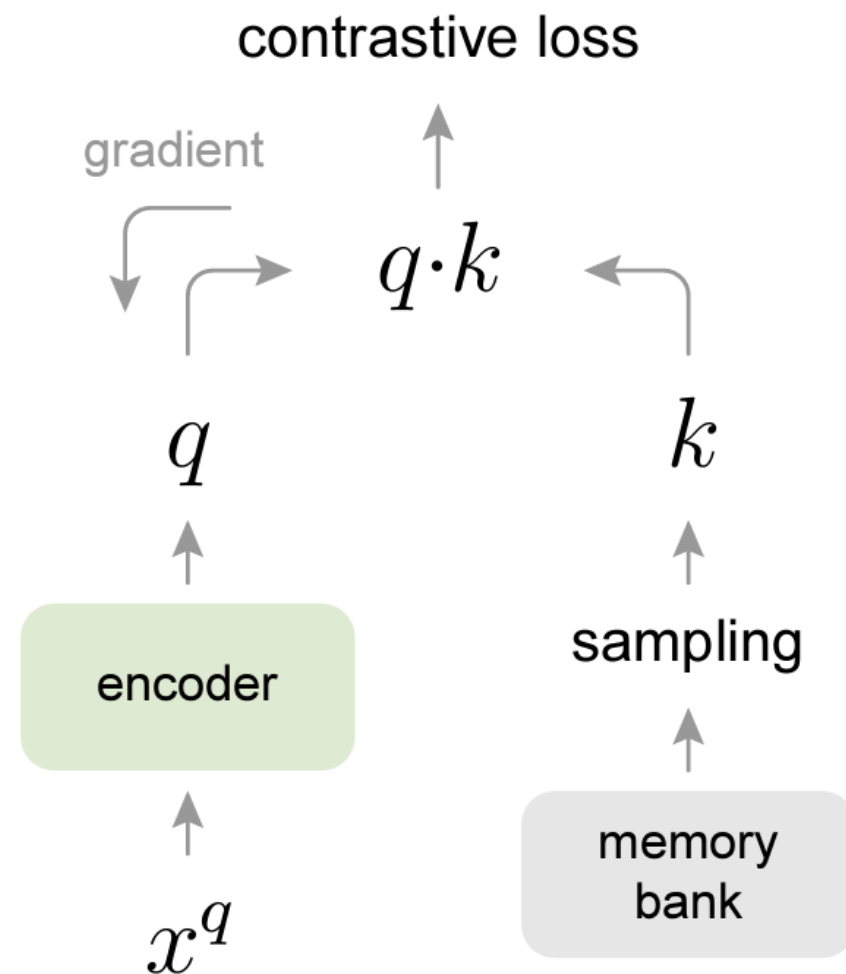
Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.



Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

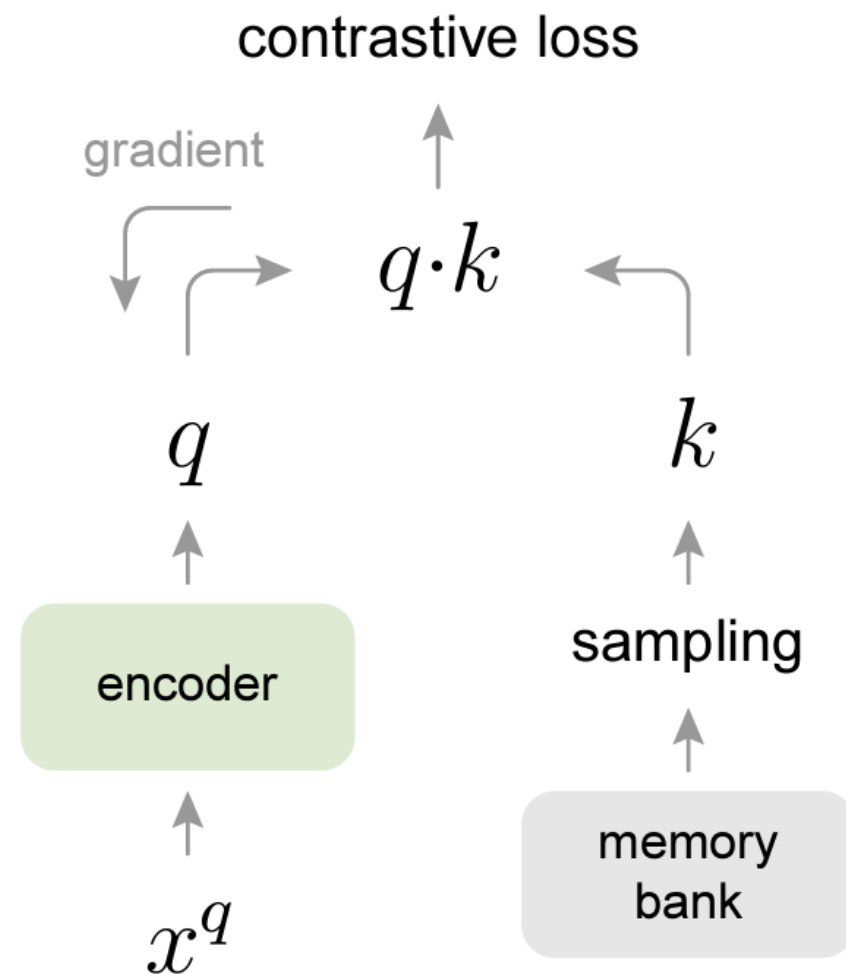
Calculamos la probabilidad posteriori:

$$P(D = 1|i, v) = \frac{P(i|v)P(D = 1)}{P(i|v)P(D = 1) + P_n(i)P(D = 0)}$$

donde:

- $P(D = 1)$ es la probabilidad de que una muestra provenga del conjunto real de datos.
- $P(D = 0)$ es la probabilidad de que una muestra provenga del ruido.
- $P_n(i)$ es la probabilidad de la muestra bajo la distribución de ruido (se asume distribución uniforme).

Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$P(D = 1|i, v) = \frac{P(i|v)P(D = 1)}{P(i|v)P(D = 1) + P_n(i)P(D = 0)}$$

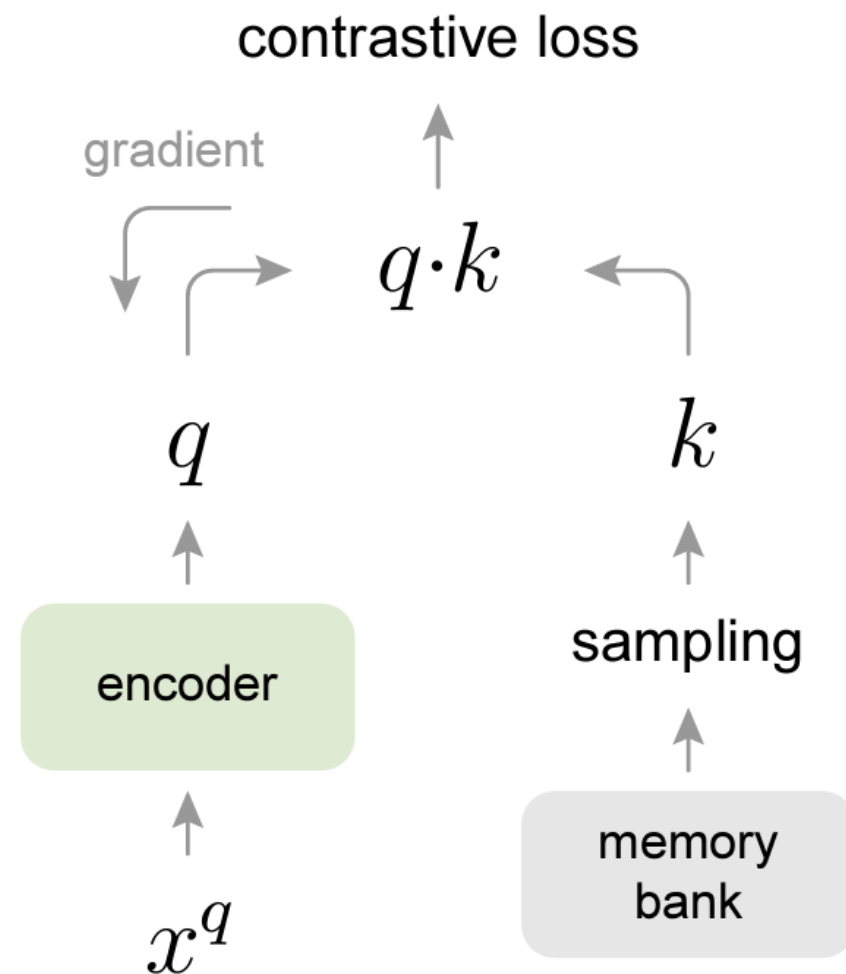
donde:

- $P(D = 1)$ es la probabilidad de que una muestra provenga del conjunto real de datos.
- $P(D = 0)$ es la probabilidad de que una muestra provenga del ruido.
- $P_n(i)$ es la probabilidad de la muestra bajo la distribución de ruido (se asume distribución uniforme).

Hacemos:

- Los datos reales ocurren con probabilidad $\frac{1}{1+m}$.
- El ruido ocurre con probabilidad $\frac{m}{1+m}$.

Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

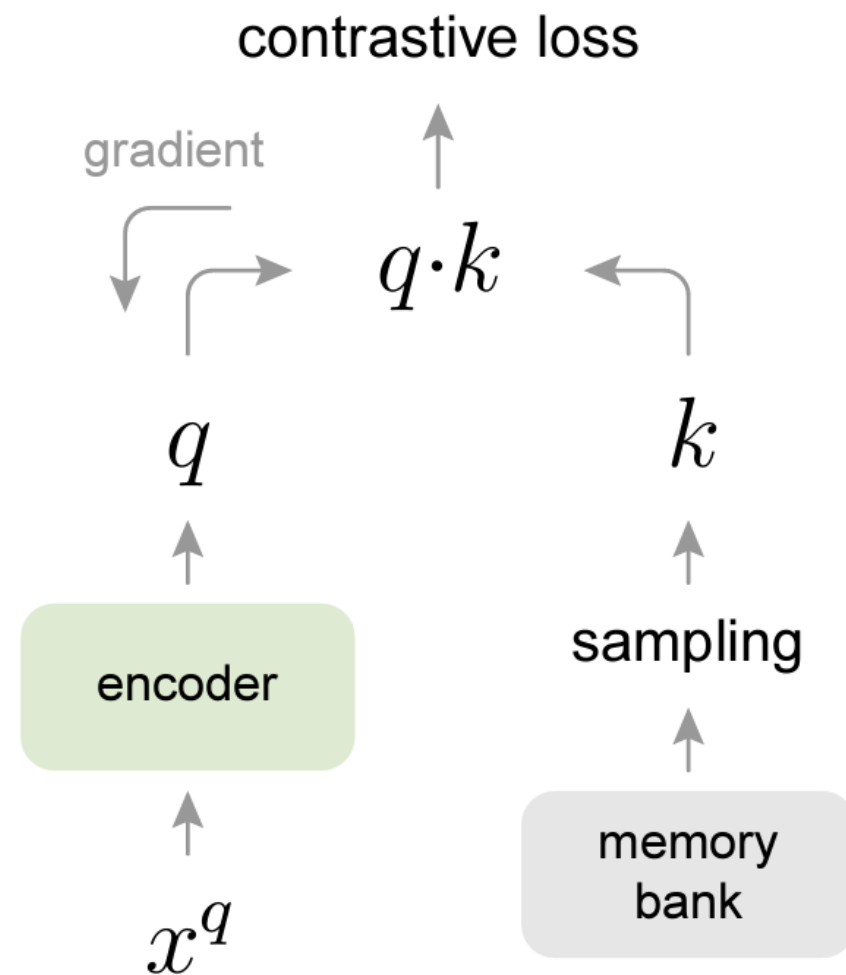
- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$P(D = 1|i, v) = \frac{P(i|v) \cdot \frac{1}{1+m}}{P(i|v) \cdot \frac{1}{1+m} + P_n(i) \cdot \frac{m}{1+m}} = \frac{P(i|v)}{P(i|v) + mP_n(i)}$$



Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

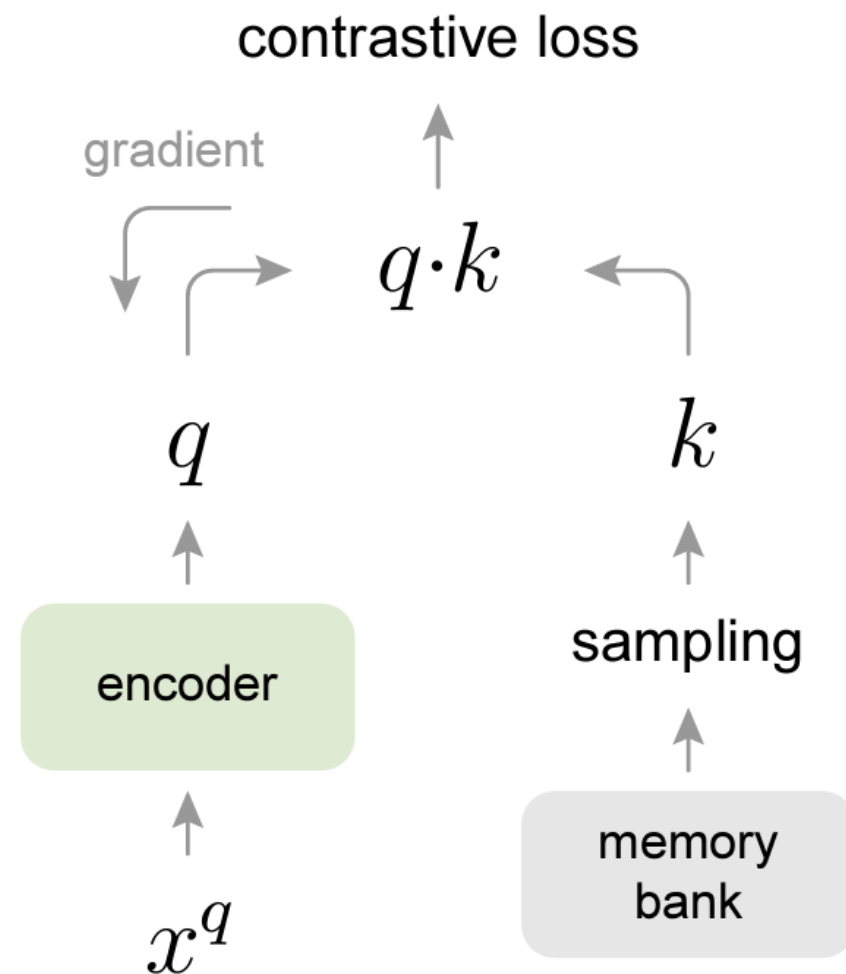
Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{P(i|v)}{P(i|v) + mP_n(i)}$$

Asumimos distribución uniforme para $P_n(i)$, entonces: $P_n(i) = 1/n$



Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

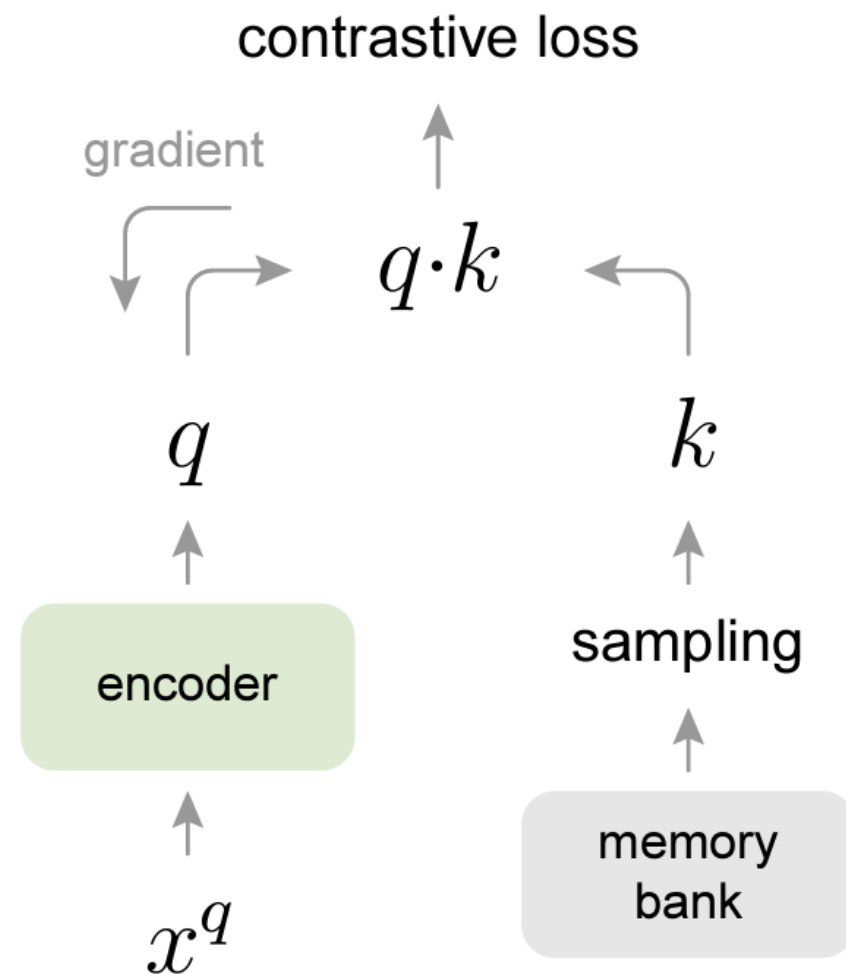
$$h(i, v) = \frac{P(i|v)}{P(i|v) + mP_n(i)}$$

Asumimos distribución uniforme para $P_n(i)$, entonces: $P_n(i) = 1/n$

Aun tenemos a $P(i|v)$



Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{P(i|v)}{P(i|v) + mP_n(i)}$$

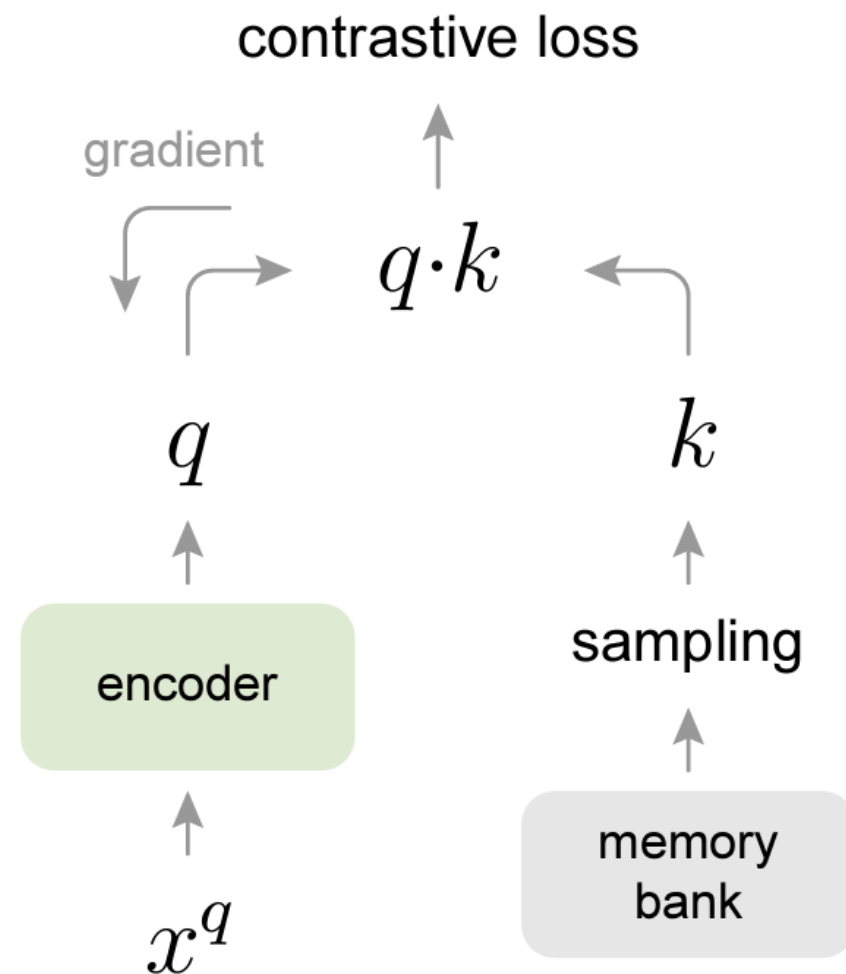
Nos interesa: $P(i|v) = \frac{\exp(v^T v_i / \tau)}{Z}$ donde Z es el término de normalización:

$$Z = \sum_{j=1}^n \exp(v^T v_j / \tau)$$

Aproximamos con muestreo Monte Carlo usando m muestras negativas:

$$Z \approx n \cdot \frac{1}{m} \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)$$

Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{P(i|v)}{P(i|v) + mP_n(i)}$$

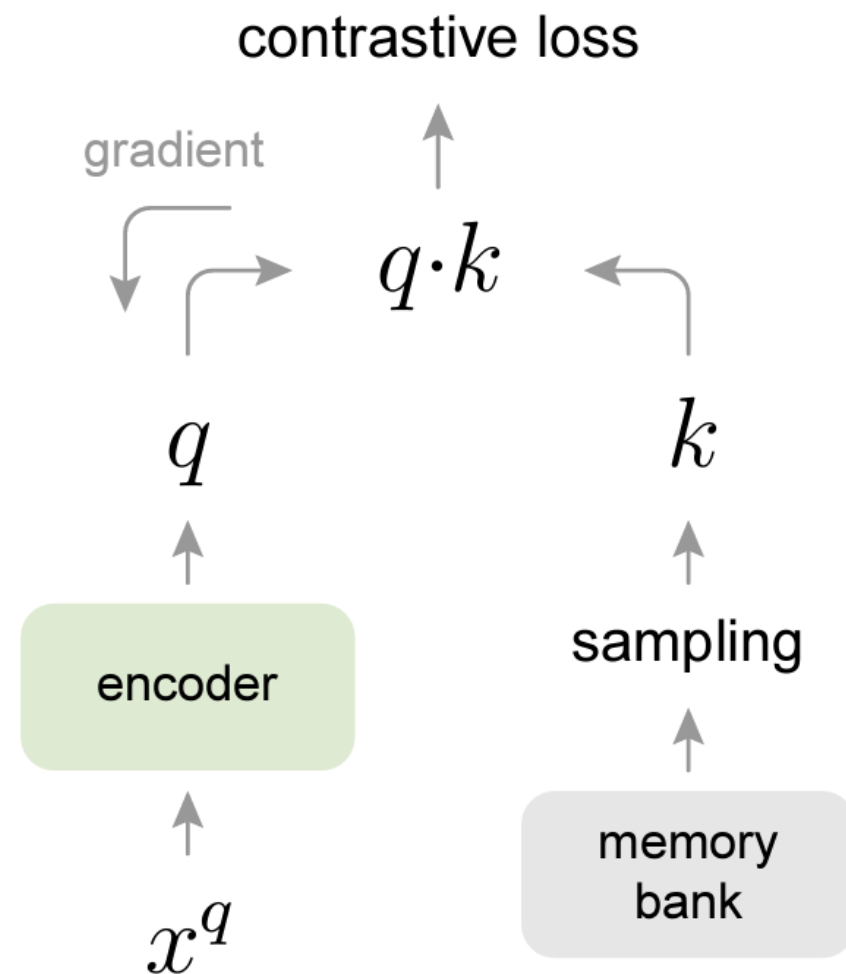
Nos interesa: $P(i|v) = \frac{\exp(v^T v_i / \tau)}{Z}$ donde Z es el término de normalización:

$$Z = \sum_{j=1}^n \exp(v^T v_j / \tau)$$

Aproximamos con muestreo Monte Carlo usando m muestras negativas:

$$Z \approx n \cdot \mathbb{E}_j[\exp(v^T v_j / \tau)] \approx n \cdot \frac{1}{m} \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)$$

Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

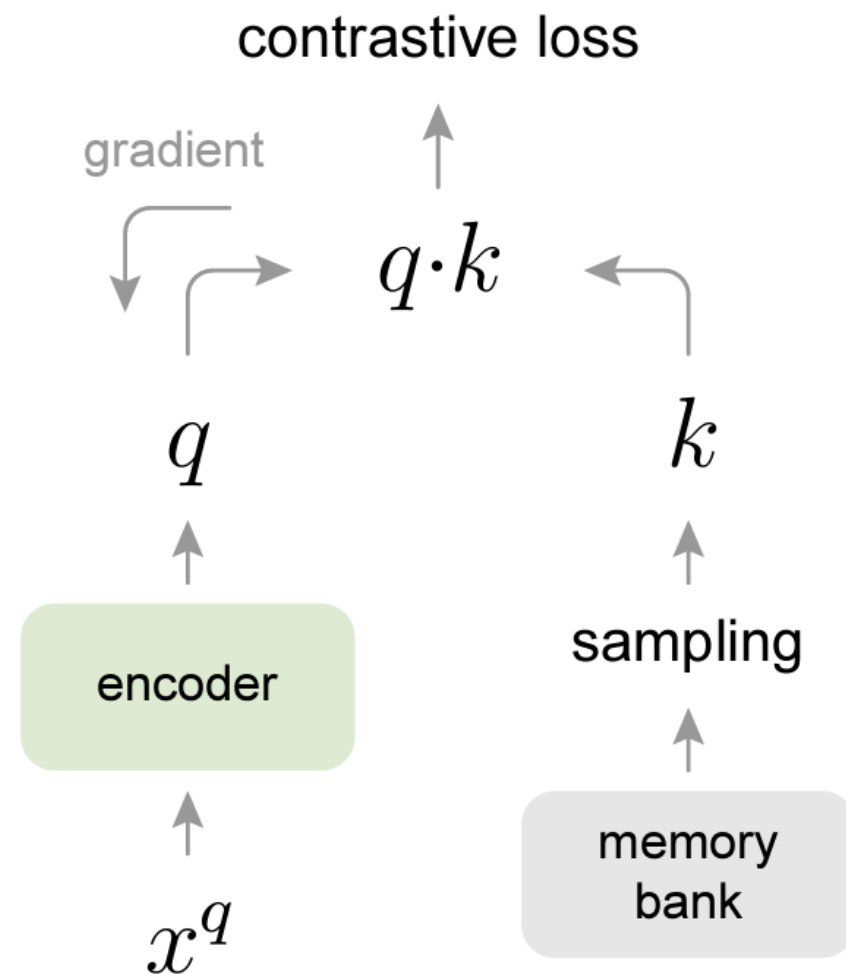
Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{P(i|v)}{P(i|v) + mP_n(i)} = \frac{\frac{\exp(v^T v_i / \tau)}{\frac{n}{m} \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)}}{\frac{\exp(v^T v_i / \tau)}{\frac{n}{m} \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)} + m \cdot \frac{1}{n}}$$

Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

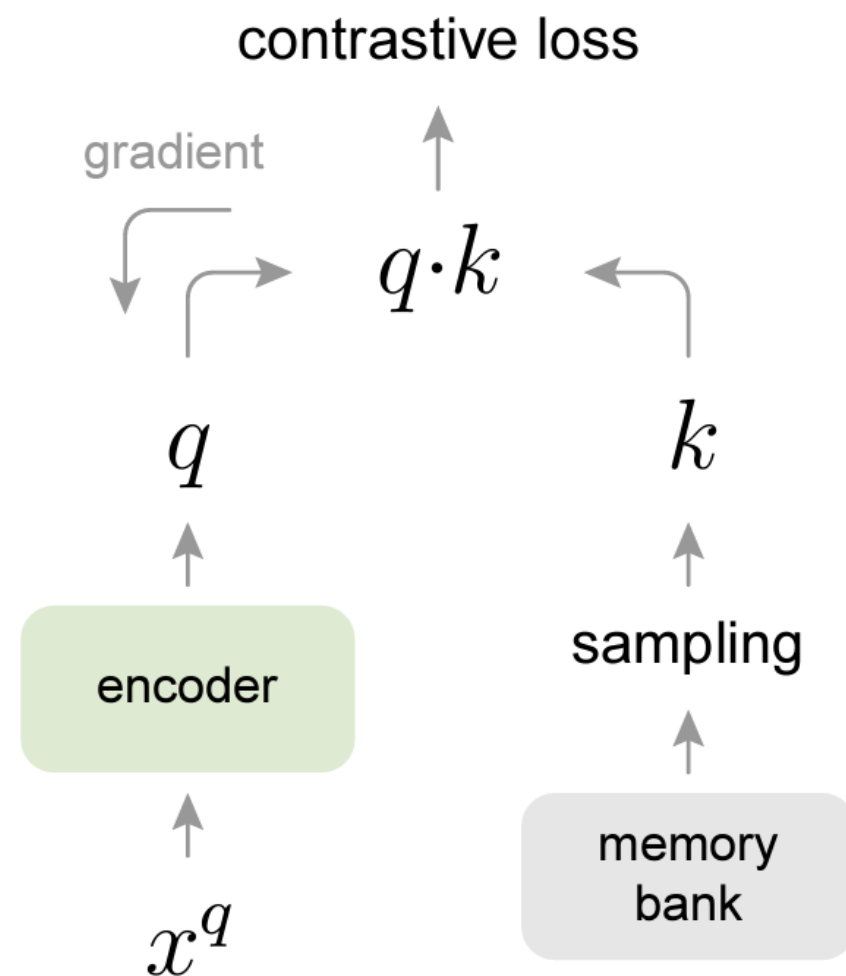
- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{\exp(v^T v_i / \tau)}{\exp(v^T v_i / \tau) + m \cdot \frac{1}{n} \cdot \frac{n}{m} \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)}$$



Noise-Contrastive *Estimation (NCE)*



Cambiamos a un problema de clasificación binaria:

Definimos un evento binario D , donde:

- $D = 1$ significa que la muestra es real.
- $D = 0$ significa que la muestra es ruido.

Calculamos la probabilidad posteriori:

$$h(i, v) = \frac{\exp(v^T v_i / \tau)}{\exp(v^T v_i / \tau) + \sum_{k=1}^m \exp(v^T v_{j_k} / \tau)}$$

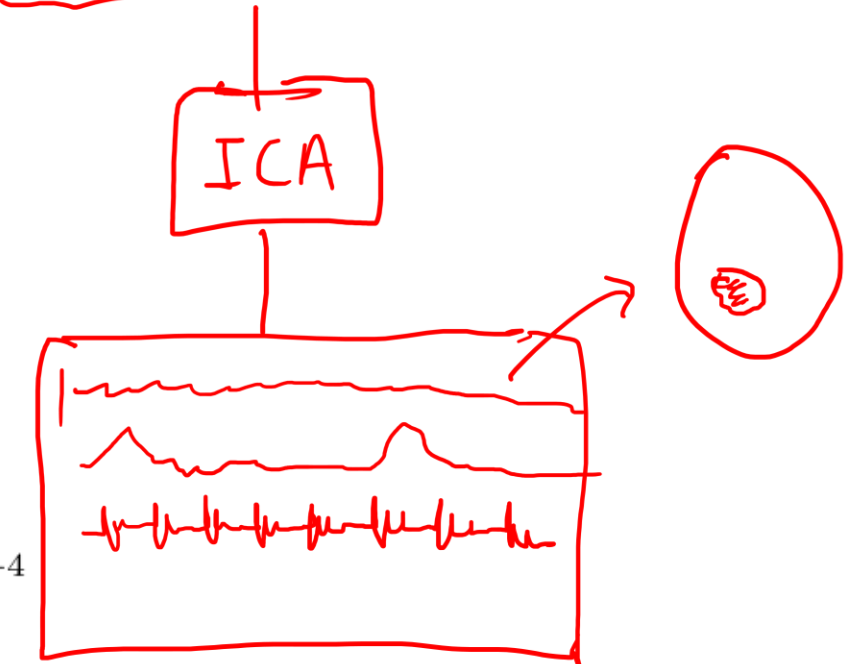
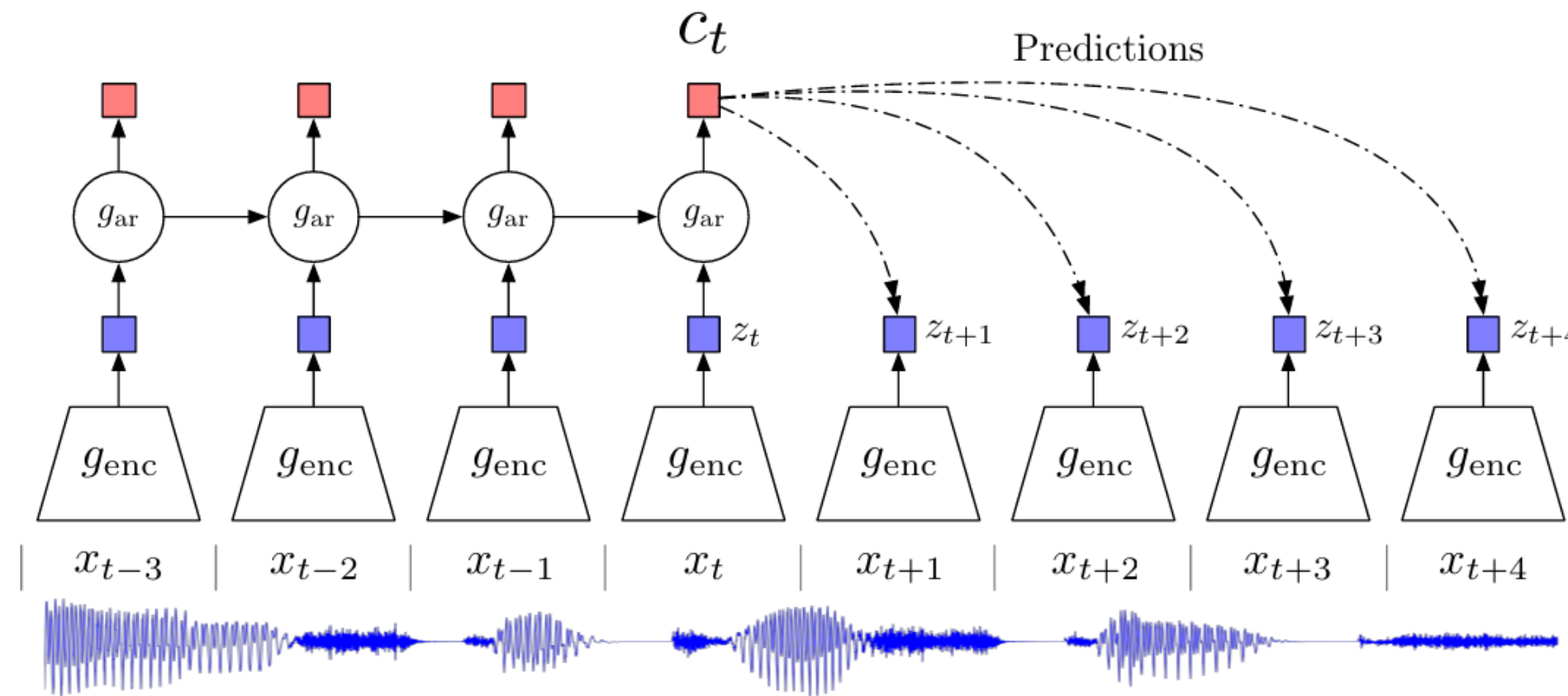
Loss function:

$$J_{NCE}(\theta) = -\mathbb{E}_{P_d}[\log h(i, v)] - m \cdot \mathbb{E}_{P_n}[\log(1 - h(i, v'))]$$

InfoNCE

(Information Noise-Contrastive Estimation)

> Reinventa el mundo <



$$I(x; c) = \sum_{x,c} p(x, c) \log \frac{p(x|c)}{p(x)}$$



$$p(x, y) = p(x)p(y) ?$$

Mutual *Information*

La información mutua es una medida de dependencia entre dos variables aleatorias X y Y .

$$I(X; Y) = D_{KL}(p(x, y) || p(x) \otimes p(y))$$

- donde:
- $p(x, y)$ es la distribución conjunta de X y Y .
 - $p(x)$ y $p(y)$ son las distribuciones marginales de X y Y , respectivamente.



$$P(a|b) = \frac{P(a,b)}{P(b)}$$

Mutual Information

La información mutua es una medida de dependencia entre dos variables aleatorias X y Y .

$$I(X; Y) = D_{KL}(p(x, y) || p(x) \otimes p(y)) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

donde:

- $p(x, y)$ es la distribución conjunta de X y Y .
- $p(x)$ y $p(y)$ son las distribuciones marginales de X y Y , respectivamente.

Handwritten notes and derivations:

$$\sum_x \sum_y p(x, y) \log \frac{p(y|x)}{p(y)}$$

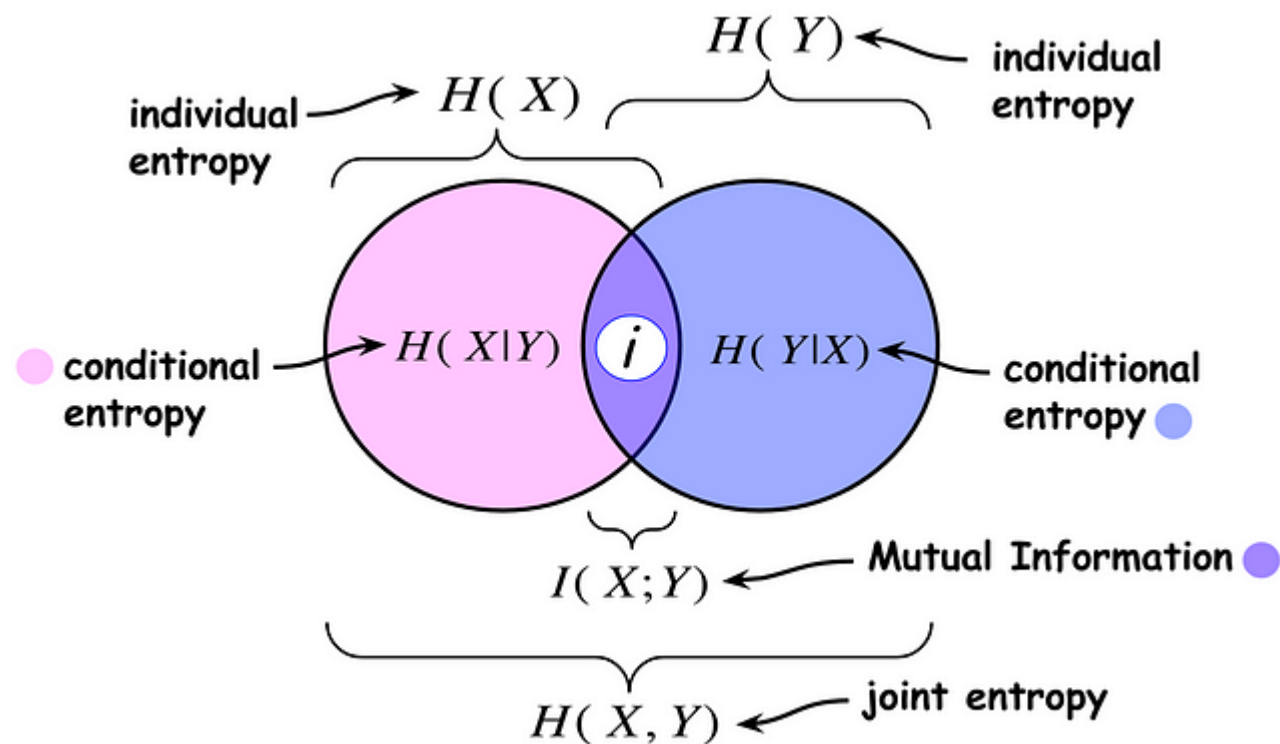
$$p(x, y) \log \frac{p(x|y)}{p(x)}$$

Mutual Information

La información mutua es una medida de dependencia entre dos variables aleatorias X y Y .

$$I(X; Y) = D_{KL}(p(x, y) || p(x) \otimes p(y)) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

- donde:
- $p(x, y)$ es la distribución conjunta de X y Y .
 - $p(x)$ y $p(y)$ son las distribuciones marginales de X y Y , respectivamente.

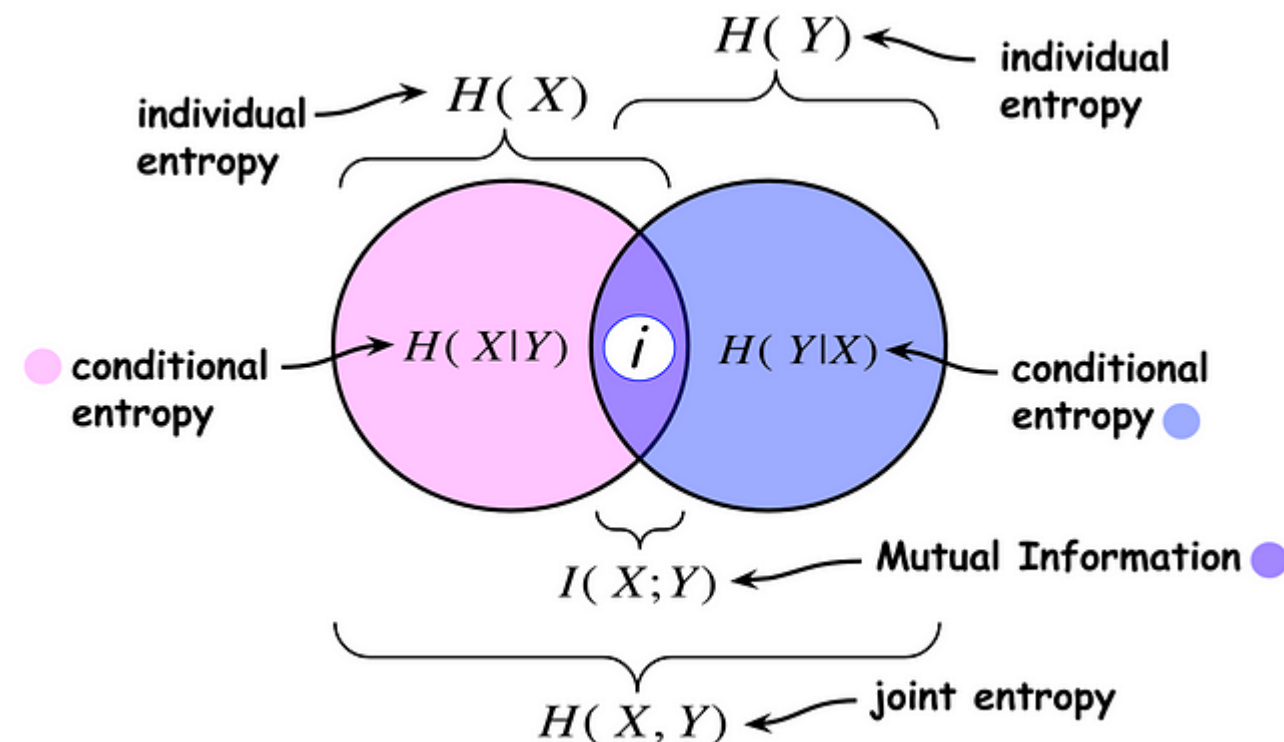


Mutual Information

La información mutua es una medida de dependencia entre dos variables aleatorias X y Y .

$$I(X; Y) = D_{KL}(p(x, y) || p(x) \otimes p(y)) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}$$

- donde:
- $p(x, y)$ es la distribución conjunta de X y Y .
 - $p(x)$ y $p(y)$ son las distribuciones marginales de X y Y , respectivamente.



Interpretación:

- $I(X; Y)$ mide cuánta información sobre X se obtiene conociendo Y (y viceversa).
- Si X e Y son independientes, entonces $p(x, y) = p(x)p(y)$, lo que implica $I(X; Y) = 0$, es decir, no comparten información.
- Si X e Y son completamente dependientes, la información mutua es máxima y coincide con la entropía de cualquiera de las dos variables si son idénticas.

InfoNCE

(Information Noise-Contrastive Estimation)

Partimos de **NCE loss**:

$$p(d = i | X, c_t) = \frac{p(x_i | c_t) \prod_{l \neq i} p(x_l)}{\sum_{j=1}^N p(x_j | c_t) \prod_{l \neq j} p(x_l)} = \frac{\frac{p(x_i | c_t)}{p(x_i)}}{\sum_{j=1}^N \frac{p(x_j | c_t)}{p(x_j)}}$$



InfoNCE

(Information Noise-Contrastive Estimation)

Partimos de **NCE loss**:

$$p(d = i | X, c_t) = \frac{p(x_i | c_t) \prod_{l \neq i} p(x_l)}{\sum_{j=1}^N p(x_j | c_t) \prod_{l \neq j} p(x_l)} = \frac{\frac{p(x_i | c_t)}{p(x_i)}}{\sum_{j=1}^N \frac{p(x_j | c_t)}{p(x_j)}}$$

Definimos **InfoNCE loss**:

$$\mathcal{L}_N = -\mathbb{E}_X \left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \quad \text{donde: } f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k} | c_t)}{p(x_{t+k})}$$



InfoNCE

(Information Noise-Contrastive Estimation)

Partimos de **NCE loss**:

$$p(d = i | X, c_t) = \frac{p(x_i | c_t) \prod_{l \neq i} p(x_l)}{\sum_{j=1}^N p(x_j | c_t) \prod_{l \neq j} p(x_l)} = \frac{\frac{p(x_i | c_t)}{p(x_i)}}{\sum_{j=1}^N \frac{p(x_j | c_t)}{p(x_j)}}$$

Definimos **InfoNCE loss**:

$$\mathcal{L}_N = -\mathbb{E}_X \left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \quad \text{donde: } f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k} | c_t)}{p(x_{t+k})}$$

Lo interesante es que garantiza:

$$I(x_{t+k}, c_t) \geq \log(N) - \mathcal{L}_N$$



InfoNCE

(Information Noise-Contrastive Estimation)

Partimos de **NCE loss**:

$$p(d = i | X, c_t) = \frac{p(x_i | c_t) \prod_{l \neq i} p(x_l)}{\sum_{j=1}^N p(x_j | c_t) \prod_{l \neq j} p(x_l)} = \frac{\frac{p(x_i | c_t)}{p(x_i)}}{\sum_{j=1}^N \frac{p(x_j | c_t)}{p(x_j)}}$$

Definimos **InfoNCE loss**:

$$\mathcal{L}_N = -\mathbb{E}_X \left[\log \frac{f_k(x_{t+k}, c_t)}{\sum_{x_j \in X} f_k(x_j, c_t)} \right] \quad \text{donde: } f_k(x_{t+k}, c_t) \propto \frac{p(x_{t+k} | c_t)}{p(x_{t+k})}$$

Lo interesante es que garantiza: $I(x_{t+k}, c_t) \geq \log(N) - \mathcal{L}_N$

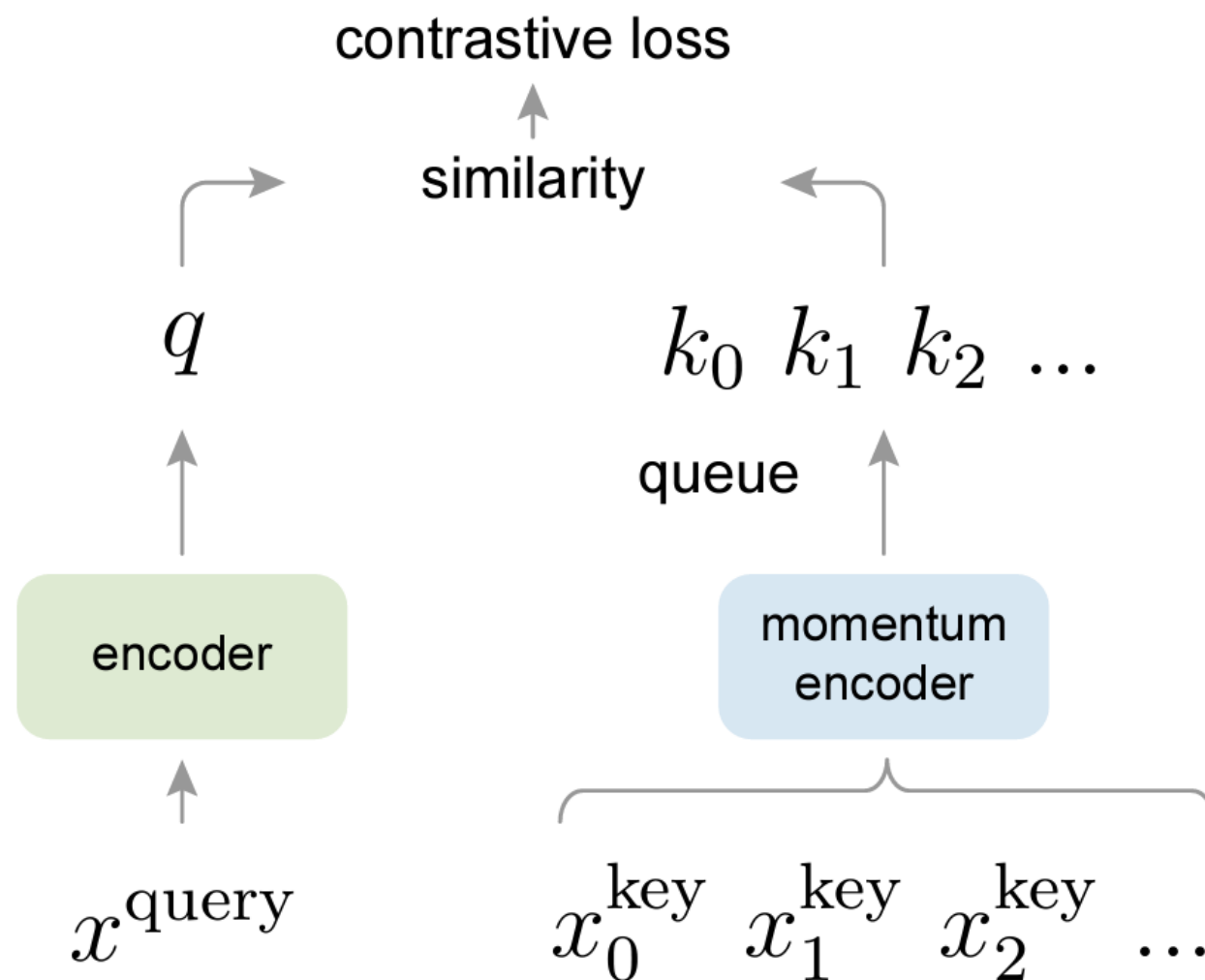
Demostración:

$$\begin{aligned} \mathcal{L}_N^{\text{opt}} &= -\mathbb{E}_X \log \left[\frac{\frac{p(x_{t+k} | c_t)}{p(x_{t+k})}}{\frac{p(x_{t+k} | c_t)}{p(x_{t+k})} + \sum_{x_j \in X_{\text{neg}}} \frac{p(x_j | c_t)}{p(x_j)}} \right] = \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k} | c_t)} \sum_{x_j \in X_{\text{neg}}} \frac{p(x_j | c_t)}{p(x_j)} \right] \\ &\approx \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k} | c_t)} (N-1) \mathbb{E}_{x_j} \frac{p(x_j | c_t)}{p(x_j)} \right] = \mathbb{E}_X \log \left[1 + \frac{p(x_{t+k})}{p(x_{t+k} | c_t)} (N-1) \right] \\ &\geq \mathbb{E}_X \log \left[\frac{p(x_{t+k})}{p(x_{t+k} | c_t)} N \right] = -I(x_{t+k}, c_t) + \log(N) \end{aligned}$$



MoCo (Momentum Contrast)

MoCo entrena un modelo para buscar representaciones similares dentro de un diccionario.



Dado un query q y un conjunto de keys $\{k_0, k_1, \dots\}$, se minimiza el InfoNCE loss:

$$\mathcal{L}_q = -\log \frac{\exp(q \cdot k^+ / \tau)}{\sum_{i=0}^K \exp(q \cdot k_i / \tau)}$$

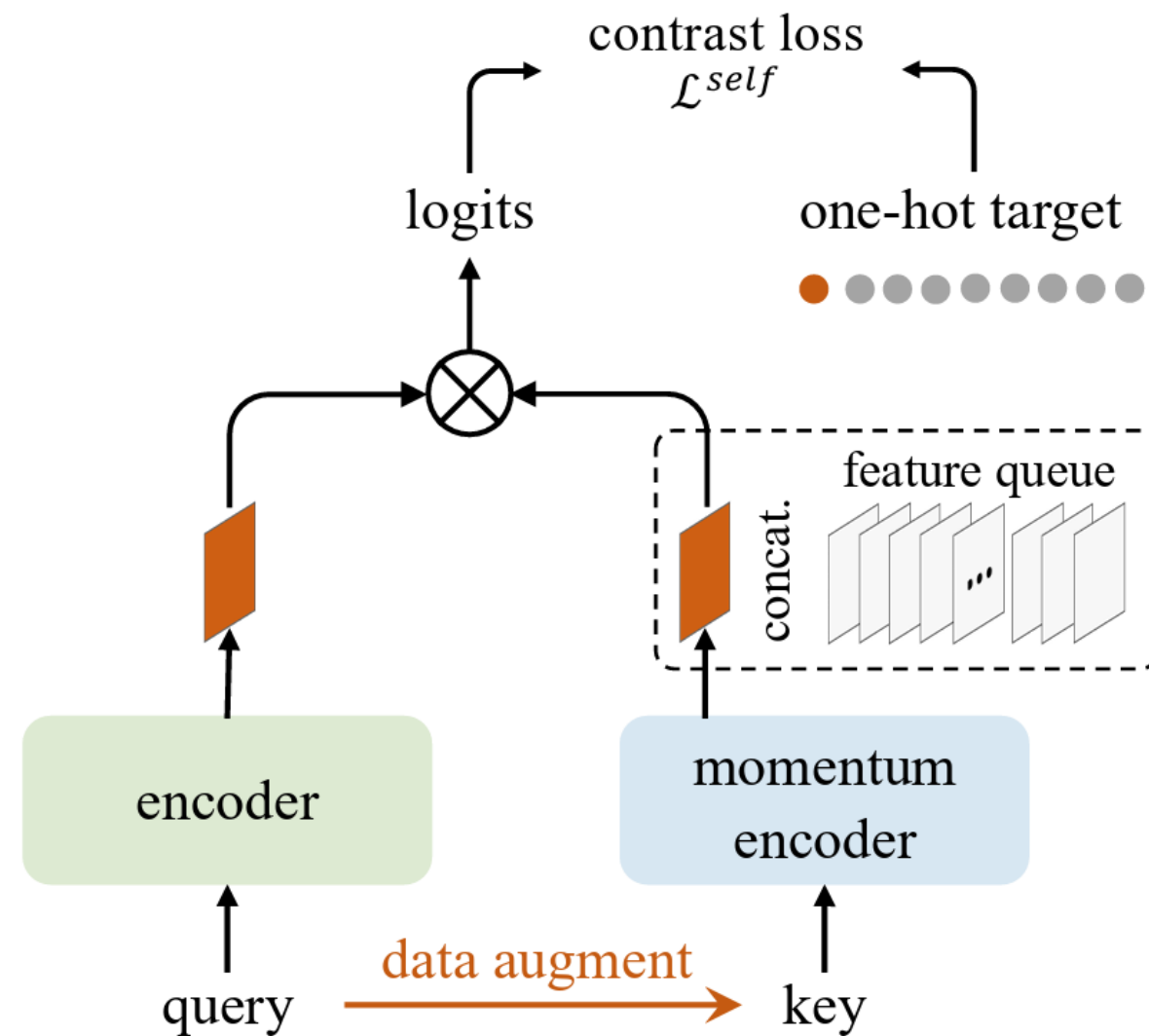
donde:

- k^+ es la clave positiva (matching key).
- k_i son claves negativas.
- τ es un hiperparámetro de temperatura.

MoCo

(Momentum Contrast)

MoCo entrena un modelo para buscar representaciones similares dentro de un diccionario.

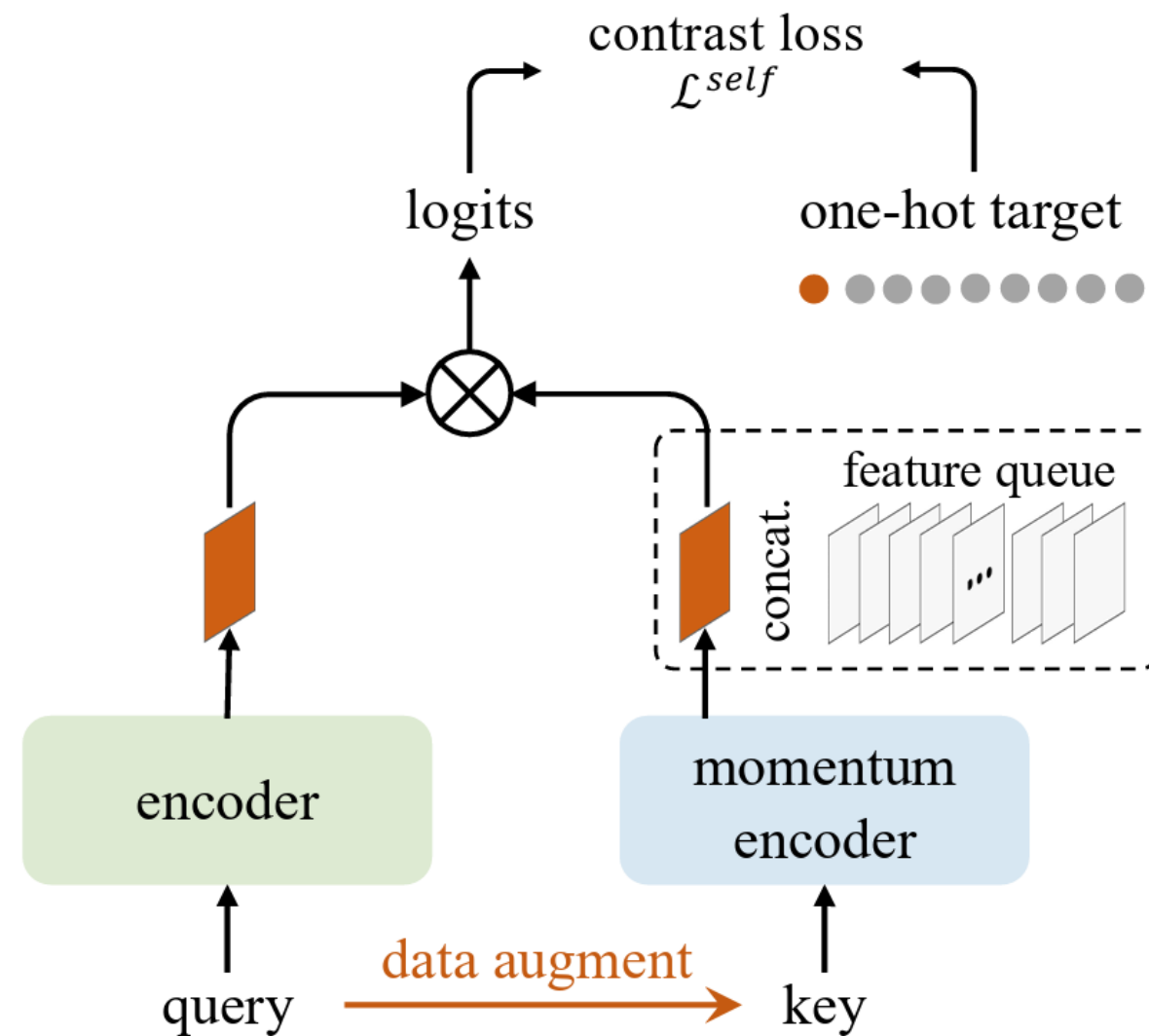


Dictionary queue

Es una cola FIFO (First In, First Out) que almacena representaciones de imágenes (*keys*) de los mini-batches previos en el entrenamiento. Su propósito principal es proporcionar un conjunto grande de ejemplos negativos para el aprendizaje contrastivo, desacoplando el tamaño del mini-batch del tamaño del diccionario.

MoCo (Momentum Contrast)

MoCo entrena un modelo para buscar representaciones similares dentro de un diccionario.



Momentum updates

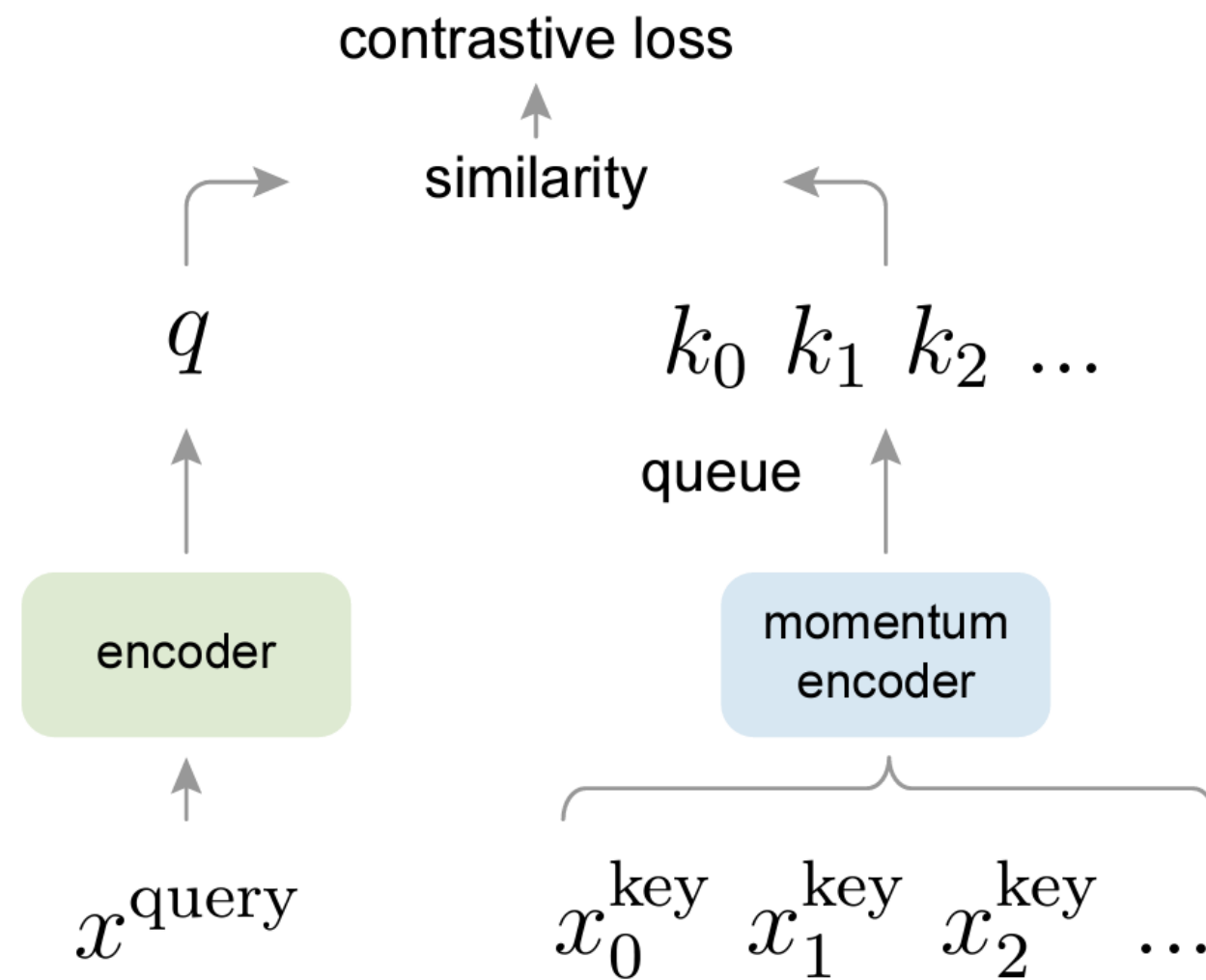
En lugar de copiar el codificador del query al key en cada paso (lo cual introduce inestabilidad), se usa una actualización por momentum:

$$\theta_k \leftarrow m\theta_k + (1 - m)\theta_q$$

esto mantiene la consistencia de las representaciones a lo largo del entrenamiento.

MoCo (Momentum Contrast)

MoCo entrena un modelo para buscar representaciones similares dentro de un diccionario.



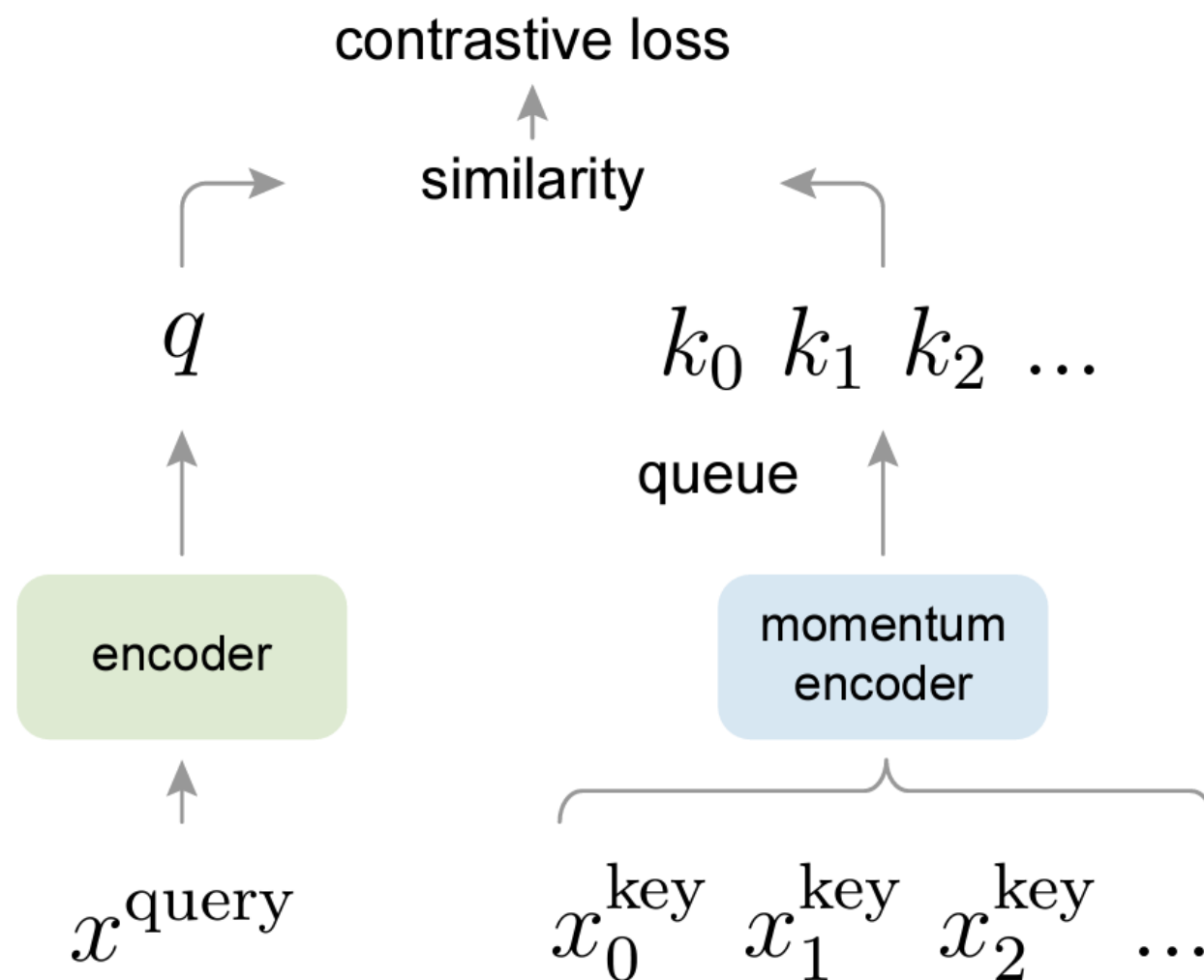
CODE?



MoCo

(Momentum Contrast)

MoCo entrena un modelo para buscar representaciones similares dentro de un diccionario.



Algorithm 1 Pseudocode of MoCo in a PyTorch-like style.

```
# f_q, f_k: encoder networks for query and key
# queue: dictionary as a queue of K keys (CxK)
# m: momentum
# t: temperature

f_k.params = f_q.params # initialize
for x in loader: # load a minibatch x with N samples
    x_q = aug(x) # a randomly augmented version
    x_k = aug(x) # another randomly augmented version

    q = f_q.forward(x_q) # queries: Nx1
    k = f_k.forward(x_k) # keys: Nx1
    k = k.detach() # no gradient to keys

    # positive logits: Nx1
    l_pos = bmm(q.view(N,1,C), k.view(N,C,1))

    # negative logits: NxK
    l_neg = mm(q.view(N,C), queue.view(C,K))

    # logits: Nx(1+K)
    logits = cat([l_pos, l_neg], dim=1)

    # contrastive loss, Eqn. (1)
    labels = zeros(N) # positives are the 0-th
    loss = CrossEntropyLoss(logits/t, labels)

    # SGD update: query network
    loss.backward()
    update(f_q.params)

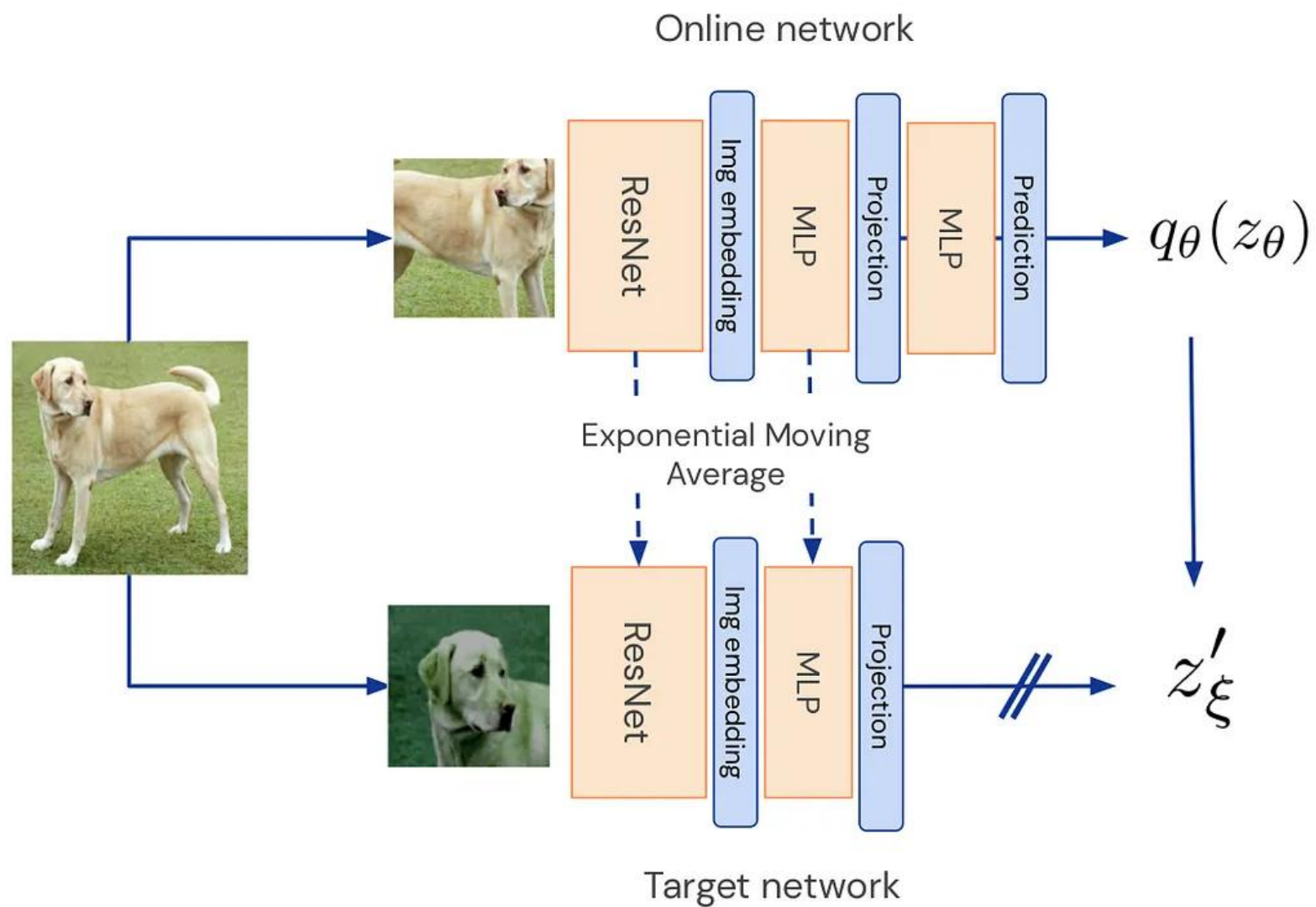
    # momentum update: key network
    f_k.params = m*f_k.params+(1-m)*f_q.params

    # update dictionary
    enqueue(queue, k) # enqueue the current minibatch
    dequeue(queue) # dequeue the earliest minibatch
```

bmm: batch matrix multiplication; mm: matrix multiplication; cat: concatenation.

TRANSFORMATEC

Bootstrap Your Own Latent (BYOL)



Online network:

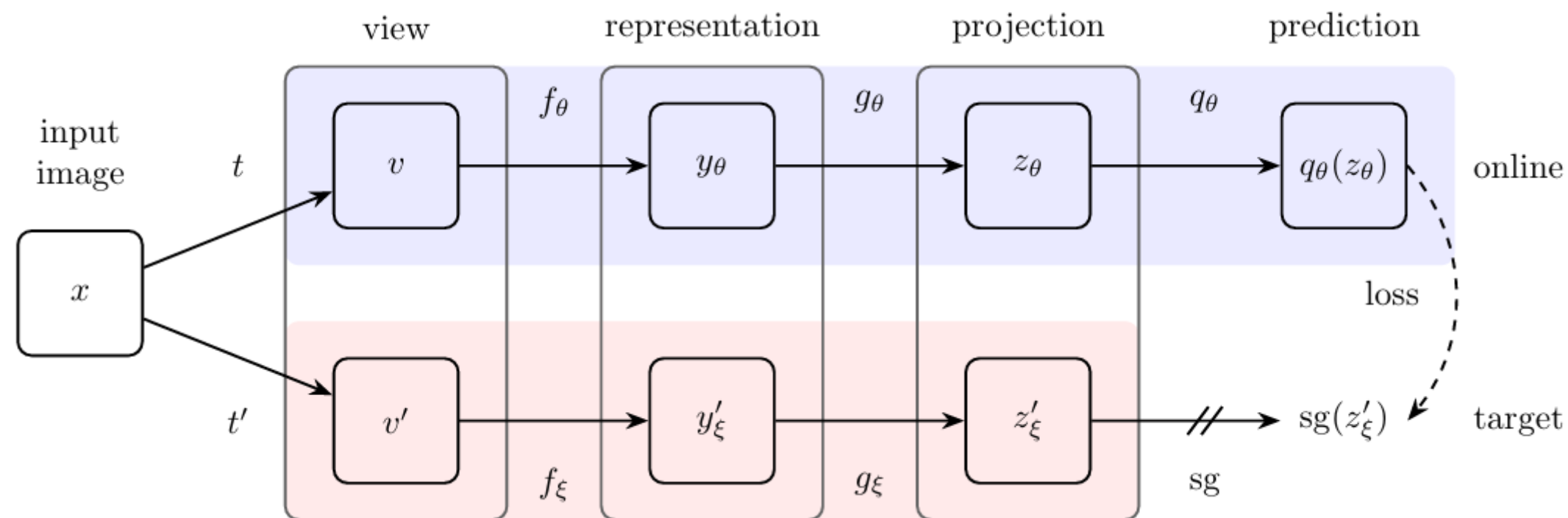
Se entrena activamente en cada paso de optimización.

Target network:

No se entrena directamente, sino que es una versión suavizada de la red online, actualizada con **exponential moving average**:

$$\xi \leftarrow \tau \xi + (1 - \tau) \theta$$

Bootstrap Your *Own Latent* (BYOL)



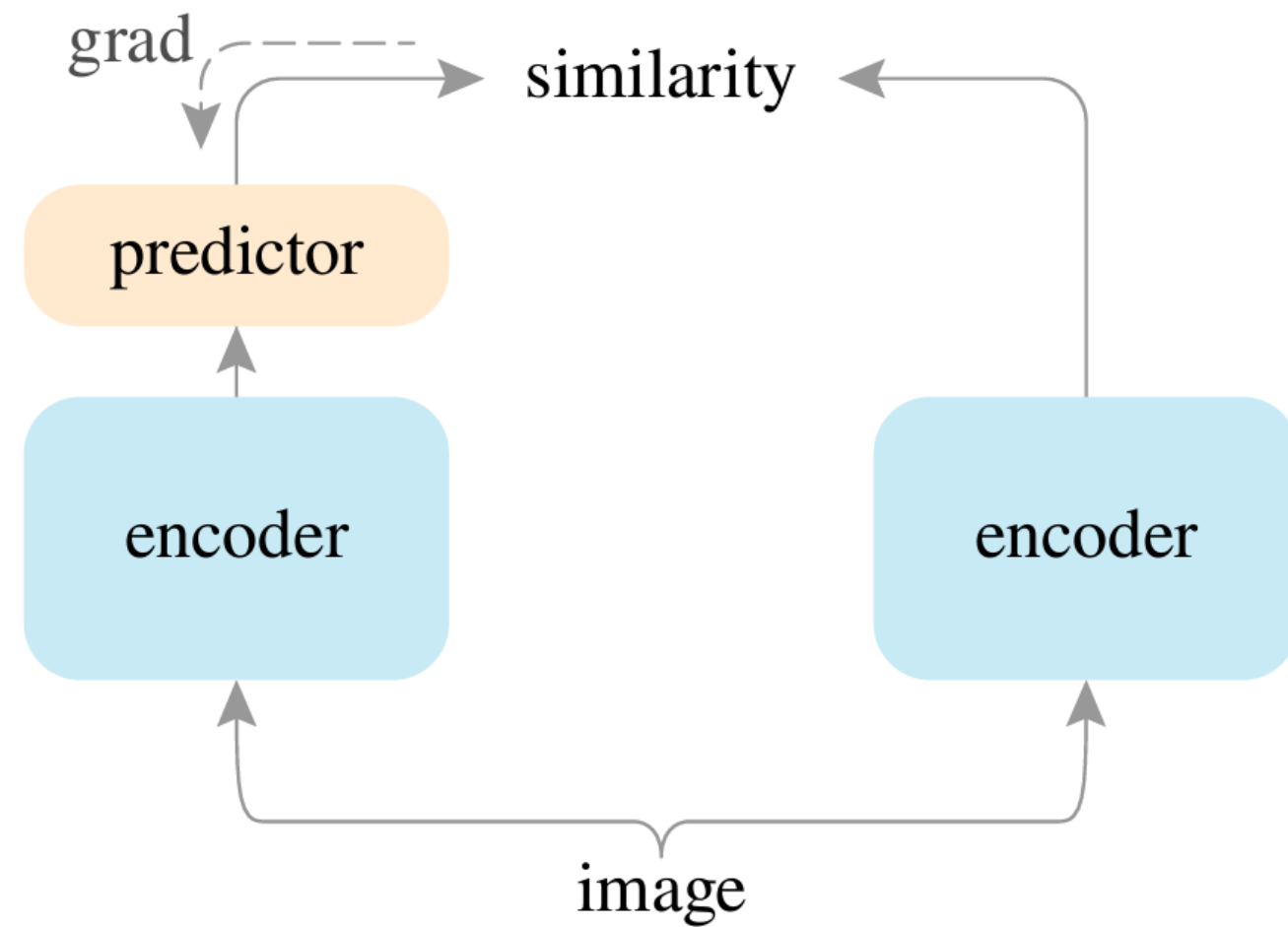
$$\mathcal{L}_{\theta, \xi} \triangleq \|\overline{q_{\theta}}(z_{\theta}) - \overline{z'_{\xi}}\|_2^2 = 2 - 2 \cdot \frac{\langle q_{\theta}(z_{\theta}), z'_{\xi} \rangle}{\|q_{\theta}(z_{\theta})\|_2 \cdot \|z'_{\xi}\|_2}$$



SimSiam



SimSiam



Negative cosine similarity

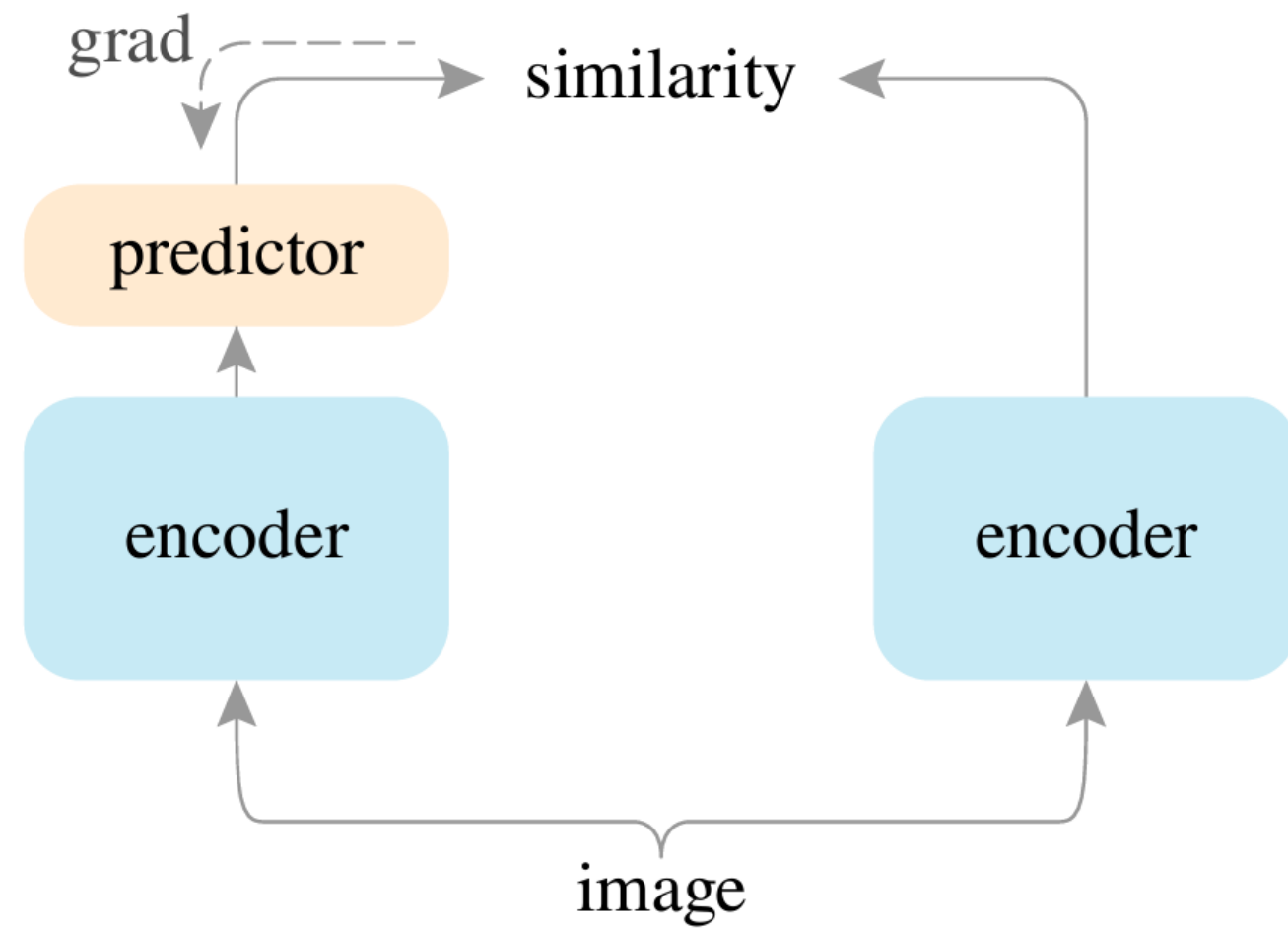
$$\mathcal{D}(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2}$$

Symmetric loss

$$\mathcal{L} = \frac{1}{2}\mathcal{D}(p_1, z_2) + \frac{1}{2}\mathcal{D}(p_2, z_1).$$



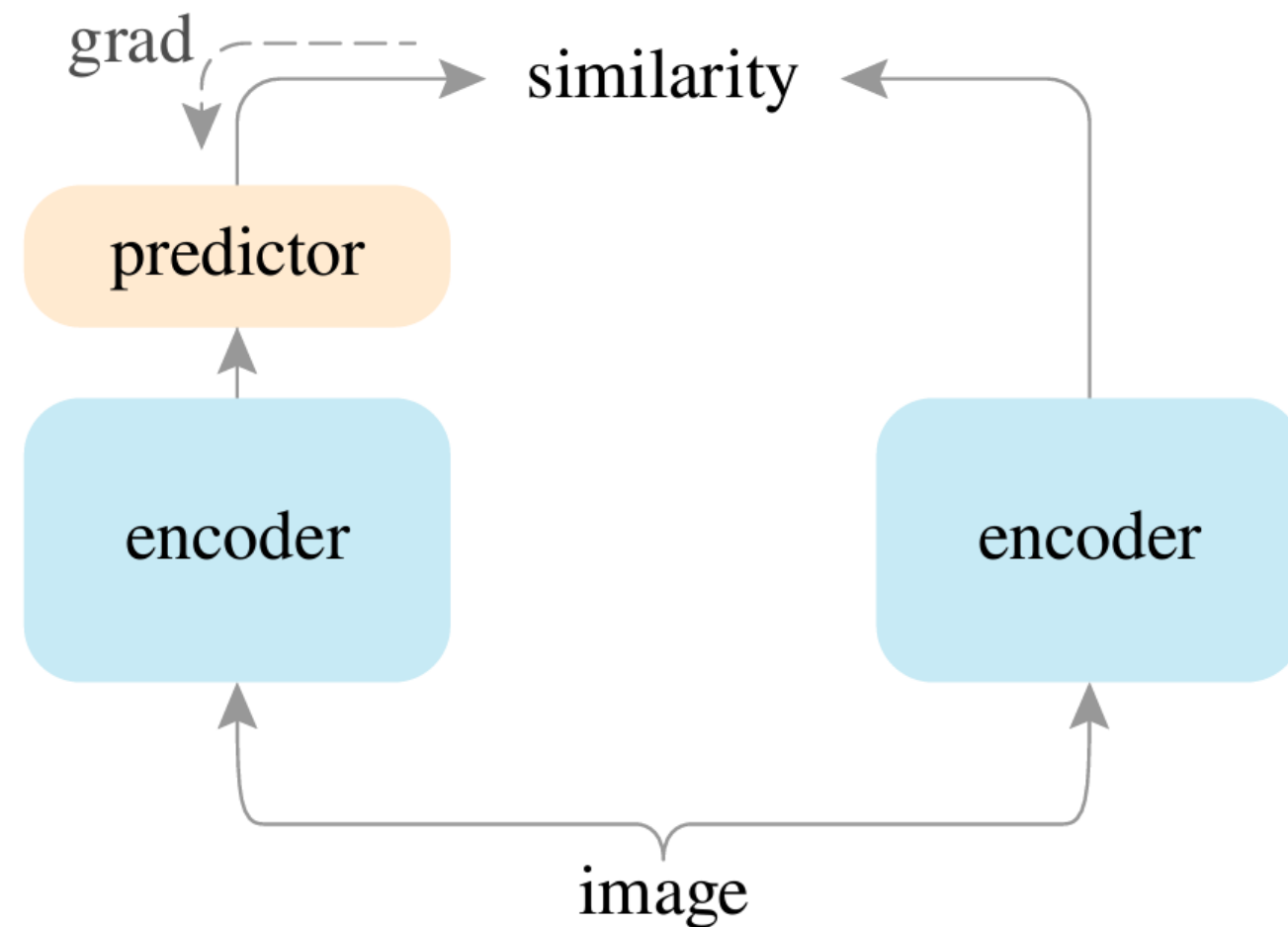
SimSiam



CODE?



SimSiam



Algorithm 1 SimSiam Pseudocode, PyTorch-like

```

# f: backbone + projection mlp
# h: prediction mlp

for x in loader: # load a minibatch x with n samples
    x1, x2 = aug(x), aug(x) # random augmentation
    z1, z2 = f(x1), f(x2) # projections, n-by-d
    p1, p2 = h(z1), h(z2) # predictions, n-by-d

    L = D(p1, z2)/2 + D(p2, z1)/2 # loss

    L.backward() # back-propagate
    update(f, h) # SGD update

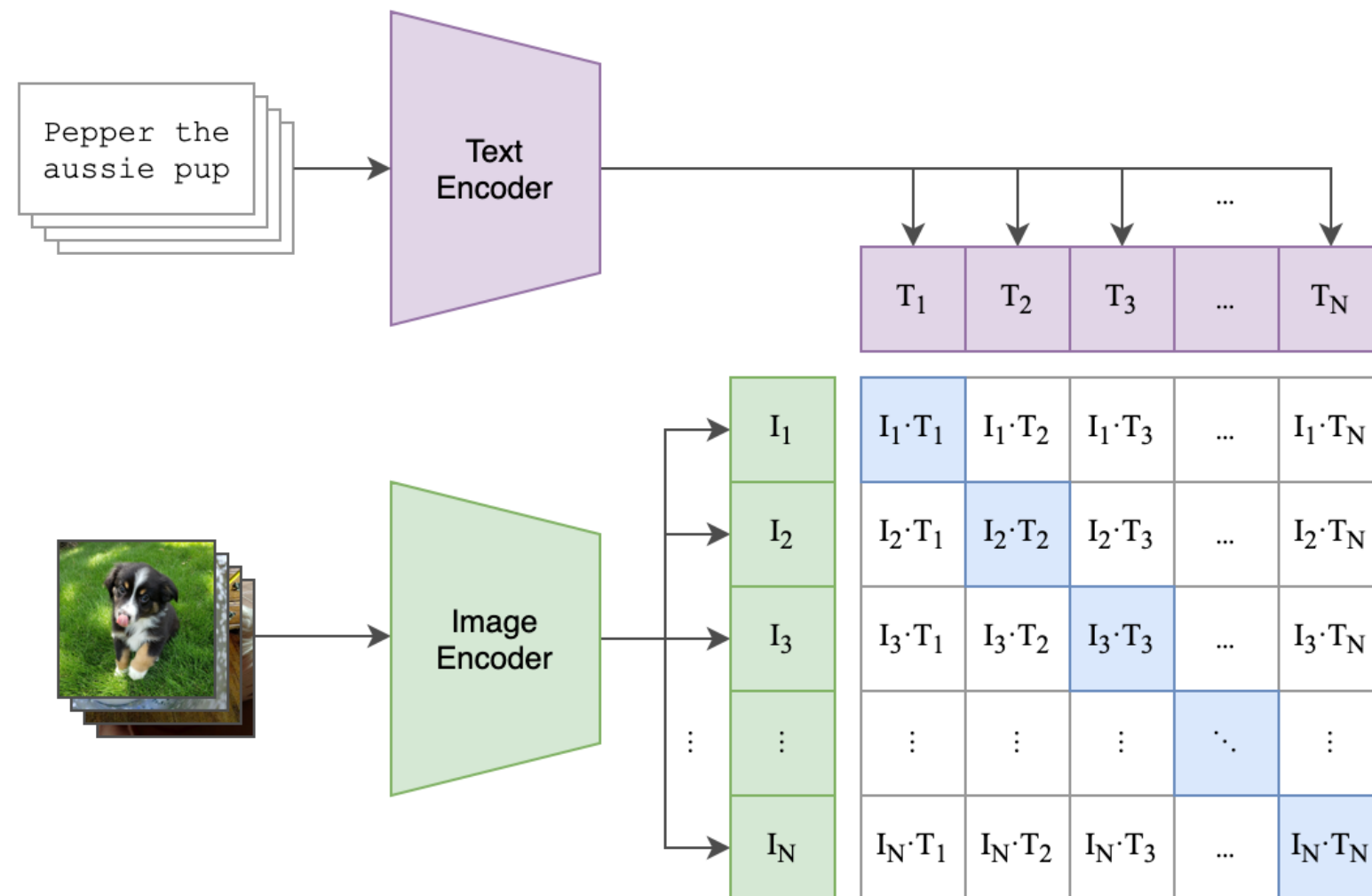
def D(p, z): # negative cosine similarity
    z = z.detach() # stop gradient

    p = normalize(p, dim=1) # l2-normalize
    z = normalize(z, dim=1) # l2-normalize
    return -(p*z).sum(dim=1).mean()
  
```

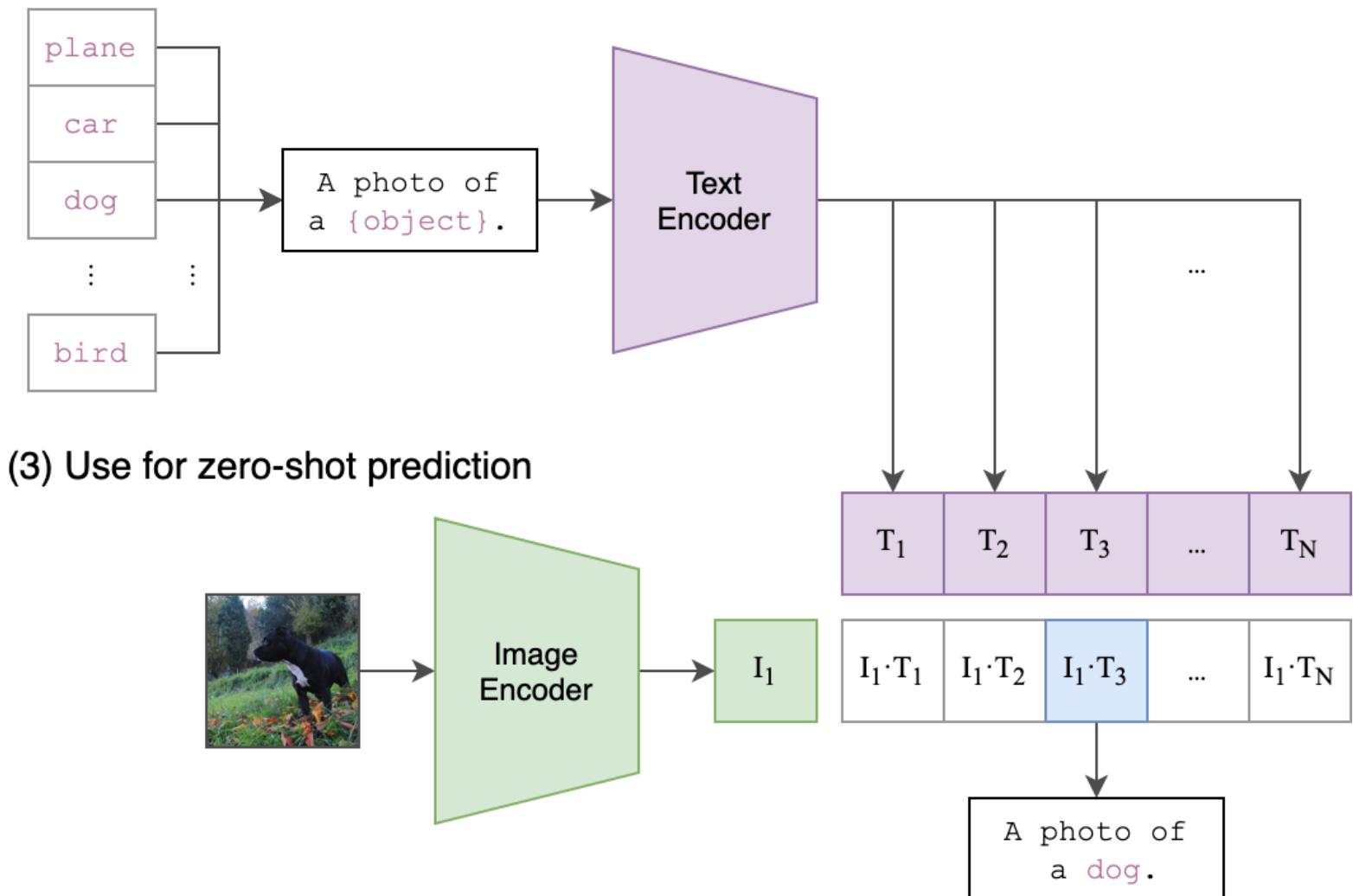
CLIP

(Contrastive Language-Image Pre-training)

(1) Contrastive pre-training



(2) Create dataset classifier from label text



(3) Use for zero-shot prediction



GRACIAS

Victor Flores Benites

