Introduction

- Tries to get rid of the necessary **negative samples** when doing **contrastive loss** for self-supervised learning
- They combine **momentum contrast** and ___ and then remove **negative samples**

Image Representation Learning - taking an image and feeding it through a function (e.g. NN like RN50) and make it give you a **representation** vector **h** that can be used to **solve many tasks**

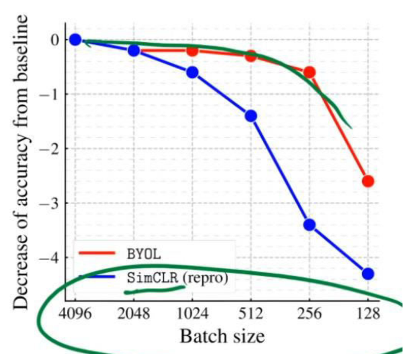
- Fine tune the model to give you good performance
- It means you can learn on a lot of data for something, then finetune to something where you have less data
- Similar approach to the ones of transformers like BERT

Self-Supervision - you'll take an image and make variants of that image using **augmentation**

- You exploit the fact that the different variants are the **same image** which should have **similar representations**

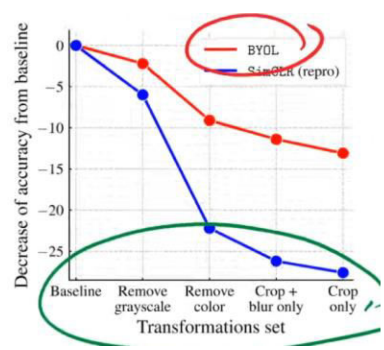
BYOL - has two neural networks, the **online network** and the **target network**

- Each of the two networks gets a **different augmentation** of the same image and the **online network** tries to **predict the target network representation**
- Once you do it one way, you then do it with the **same images but insert them in the opposite** networks
- The **target network** is trained with an **exponentially-moving average** of the online network
- When it comes to blurring, i.e. **gaussian blurring**, we **always** blur the online network, but **p=0.1** for target
- The batch size is 4096 which requires **tons of computation**
  - Decreasing the batch size reduces accuracy but not as much as solutions with **negative samples**
  - **Negative samples are taken from the minibatch**
  - If you have less samples in your minibatch, you have less samples as a representation of the distribution your dataset so you have and you have **less negative samples** = worse performance



(a) Impact of batch size

- More robust for **removal of certain augmentations**
- As **augmentations are important**, if you want to use other **modalities** (audio, text, video, …) you need to find **similarly useful augmentations**

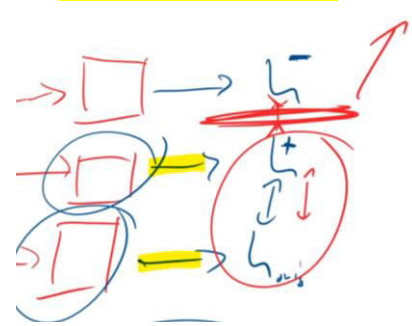


Problem - usually if you'd do this, you would get **collapse**

- The networks would cheat and make a constant function, e.g. **h = 0** to always have the perfect loss

Old Solutions

- **Negative Sampling -** you take a different image and create a **third input** by augmenting it
  - The task now becomes to make the **first two** inputs have a **similar h** that's **different to the third**
  - The network can't map everything to the same function anymore
  - Used as a combination with augmentations
  - The negative sampling prevents **collapse**, but still allows the **augmentations** to help create **good representations**
    - **The augmentation choice is very important**

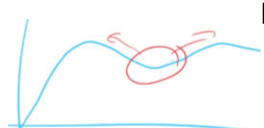


Idea - get rid of negative samples

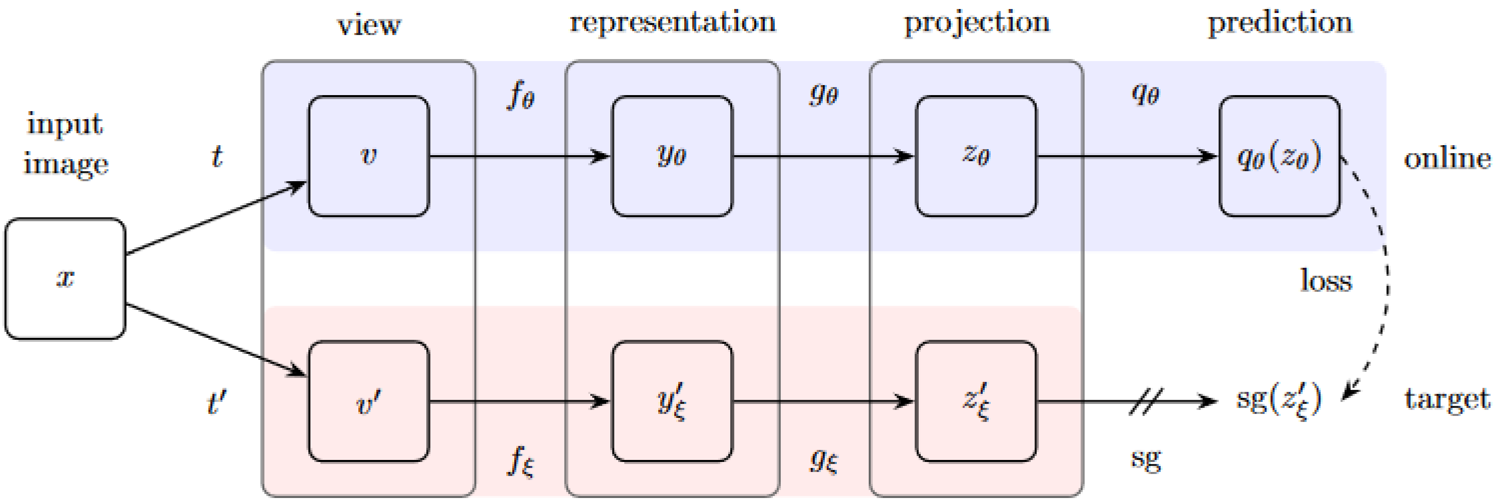
- There are many issues with **negative samples**
- Where do we get them? How do we sample them?

Many hyperparameters

- This means that nothing is preventing **h = constant c**
- *Is this a super delicate balance?*



- *Is it just that you start in a local minimum?*



BYOL Solution

- **t** and **t'** are two different **random augmentations** that create different **variants v** and **v'** that should be close
  - The **augmentations are responsible** for creating variants that are close but different and are responsible for creating really good **h representations**
- The online and target networks have the **same encoder with different parameters**
- We only **learn the online** parameters **θ**, the **target** parameters are a **copy** of the online parameters
  - **Momentum contrast** principle **-** Target parameters are a **lagging average** of the online parameters
    - You need to have a **stable representation of the past for a target**
      - *We don't quite know why this works so well*
- **y** and **y'** are two different **representations** created with our **encoders**
- $\mathbf{f_\theta}$ is our **representer**, the output of this process
  - We discard everything else
- **Projections** are done to **reduce dimensionality** by making the **representation smaller**
  - *We could get rid of it if we wanted to, there is no reason it's there apart from that it works*
  - The first FF layer pumps up the dimensionality (say from 2048 -> 4096), the second FF goes down to 256
- We put the **representations z, z'** through the **predictor** $\mathbf{q_\theta}$ which tries to take one input and predict the representation of the other input
  - We want $\mathbf{q_\theta(z) = z'}$
  - Expanding that, we get $\mathbf{q_\theta(g_\theta(f_\theta(t(x)))) = g_\xi(f_\xi(t'(x')))}$
  - $\mathbf{q_\theta}$ tries to nullify the difference in augmentations and representers
    - **q** tries to predict the **average** $\mathbf{f_\xi(t(x))}$ for **all t** (independent of t)
- This means the **augmentations destroy all non-semantic information**
  - Therefore, our representations should contain only semantic info

et

$q(f_\theta(A(x)) - f_\xi(A'(x)))$

$E_A[f(A(x))]$

$${}^{\text{BYOL}}_{\theta} \triangleq \left\|\overline{q_\theta}(z_\theta) - \bar{z}'_\xi\right\|_2^2 = 2 - 2 \cdot \frac{\langle q_\theta(z_\theta), z'_\xi \rangle}{\left\|q_\theta(z_\theta)\right\|_2 \cdot \left\|z'_\xi\right\|_2}$$

- You want the **l2** norm of the **z'** representation to be close to the norm of the $\mathbf{q_\theta(z)}$ **l2** representation