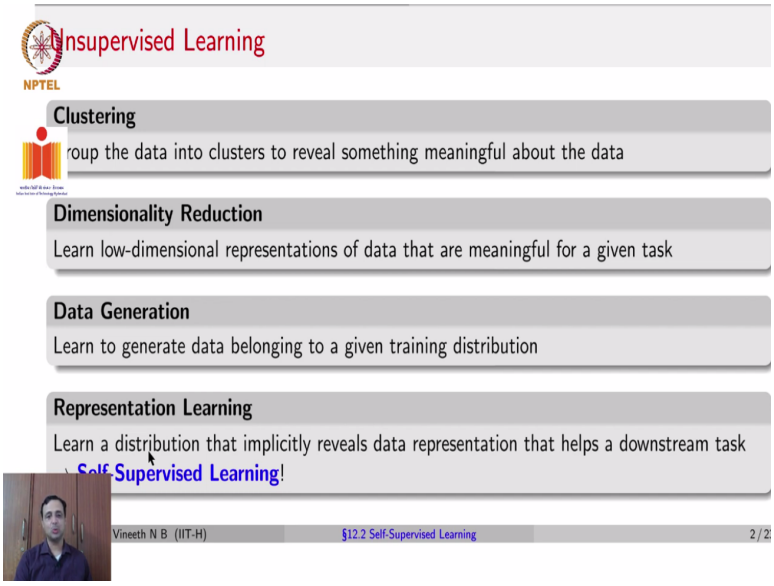
Deep Learning for Computer Vision
Professor. Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
Lecture No. 75
Self-Supervised Learning

The next topic that we will get into for this week is a very popular one these days, called Self Supervised Learning.

(Refer Slide Time: 00:27)

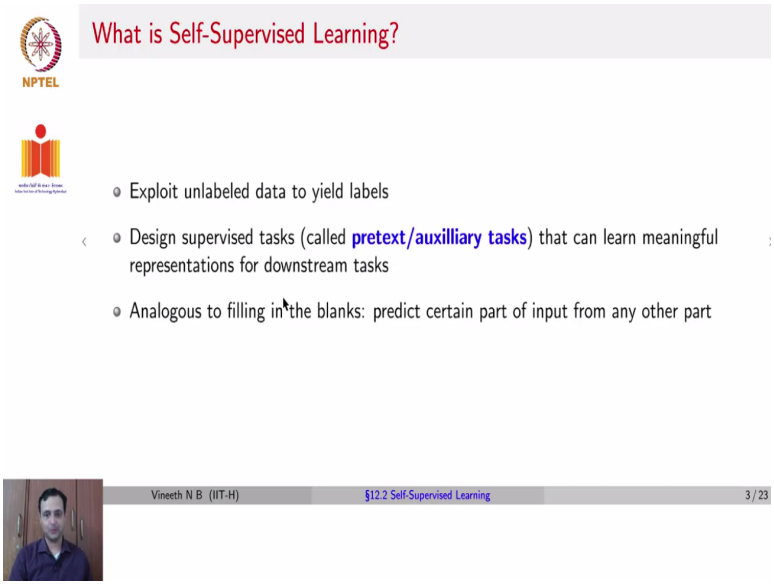


If you recall, Unsupervised Learning the discussion that we had around it, unsupervised learning, the tasks that could be categorized under this topic could be Clustering, which groups data into clusters to reveal something meaningful about the data. Dimensionality Reduction, to learn low dimensional representations of data that could be useful in some tasks. Data Generation, which is where we talked about GANs, and VAEs and other Generative models.

Where the goal is to generate data belonging to a given training distribution. The last one we will include now is the more broader task of Representation Learning. In essence, several of the methods that we have talked about so far, do perform representation learning. The specific context in which we approach representation, representation learning now is with an unsupervised learning, where our goal is to learn a distribution that implicitly reveals data

representation that can eventually help a downstream task. And this leads us to the topic of Self Supervised Learning.

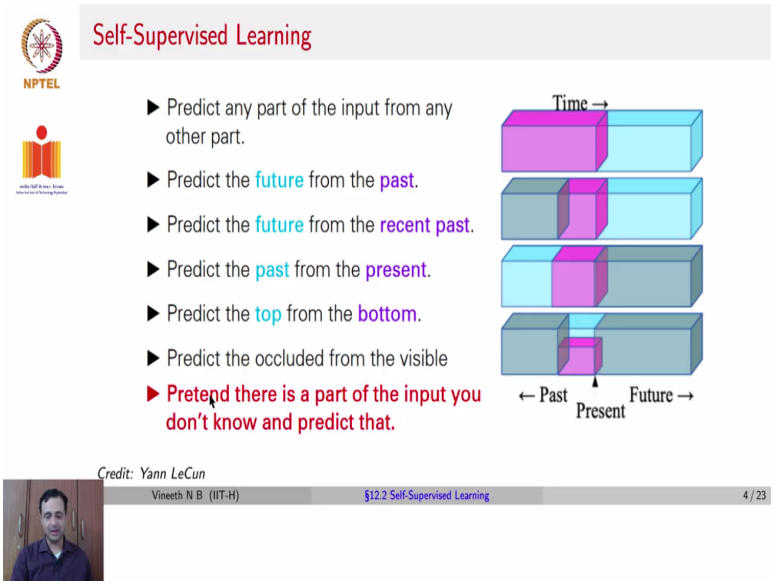
(Refer Slide Time: 01:54)



What is Self Supervised Learning? It is a twist on unsupervised learning, where we exploit unlabeled data to obtain labels. There is no explicit annotation or class labels associated with the data. We exploit unlabeled data itself to get some kind of labels and induce a supervised learning model on unlabeled data. Specifically, we design supervised tasks, which are called pretext or auxiliary tasks, which can learn meaningful representations through which the model becomes more ready to then be able to solve a downstream task, such as a classification or semantic segmentation or any other supervised learning task.

A sample task in this context could be to predict a certain part of the input from another part, somewhat like fill in the blanks of a given input.

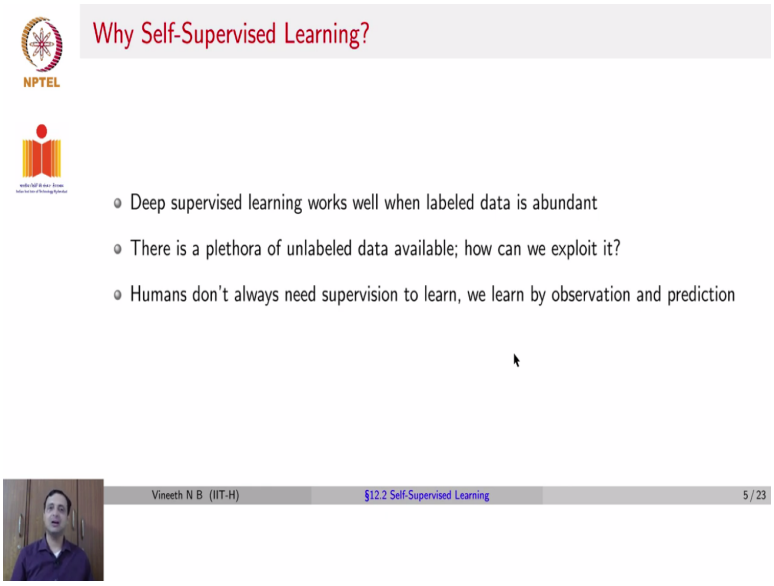
(Refer Slide Time: 03:02)



So, what part of an input can you then predict or what can you learn? You can predict any part of the input from any other part. Between images and videos, you could predict the future from the past, you could predict the future from the recent past, you could try predicting the past from the present, the top from the bottom in case of an image. In a more broader sense, you could predict the occluded from the visible.

In general, you pretend there is a part of the input that you do not know, and try to predict that. That is, those are the different tasks or pretext tasks that you can use in self supervised learning. We will see a more set of more concrete examples or the remainder of this lecture.

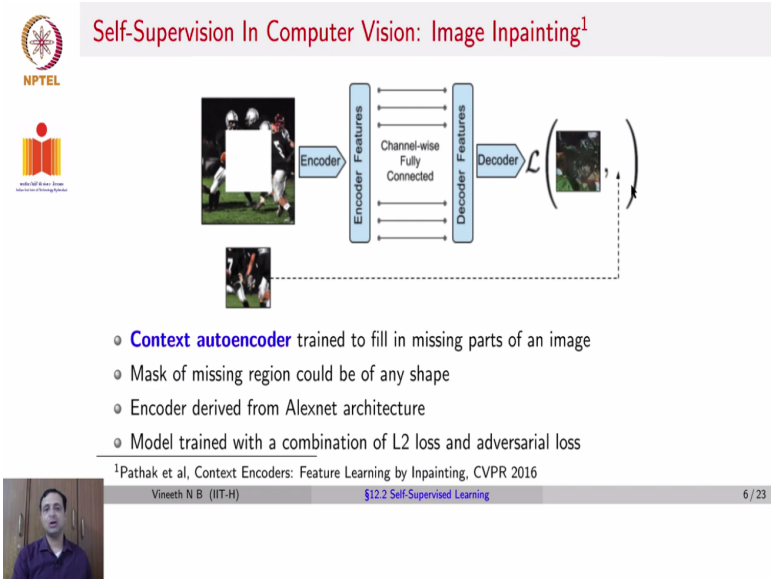
(Refer Slide Time: 03:49)



Why do we do this? Why self supervised learning? We know that deep supervised learning works well, when there is large amounts of labeled data. However, in the real world, we also know that we have large amounts of unlabeled data. How can we exploit this? We take inspiration from humans that humans do not need supervision, to learn everything. They learn, or we rather learn by observation and prediction and by self feedback.

You try something and you see how that boomerangs on you, or how that affects you. And then you keep recalibrating based on a self supervised or a self feedback, paradigm of learning. That is the idea of self supervision.

(Refer Slide Time: 04:45)

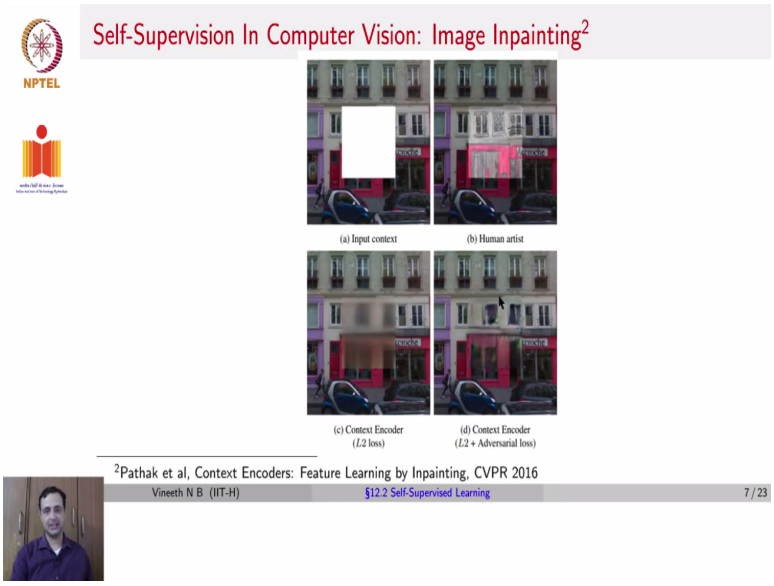


So, how do you do self supervision in computer vision? Let us see a few examples of tasks that are fairly popular. One of such examples, which we already saw in the GAN context, but also becomes relevant in a self supervised context is image In-painting. The goal of this task is to occlude or remove a certain part of the image and ask a network to complete the image. No external labels required, no external annotation required.

In this particular work called context encoders published in CVPR of 2016. There was an encoder-decoder framework learned to perform image In-painting. A context encoder auto-encoder is what it was called, was trained to fill in the missing parts. The mask of the missing region in this example is a square, but it could be of any shape. The encoder in this particular work was derived from an Alexnet architecture and the final model was trained using L_2loss between the final completed image and the original completed ground truth.

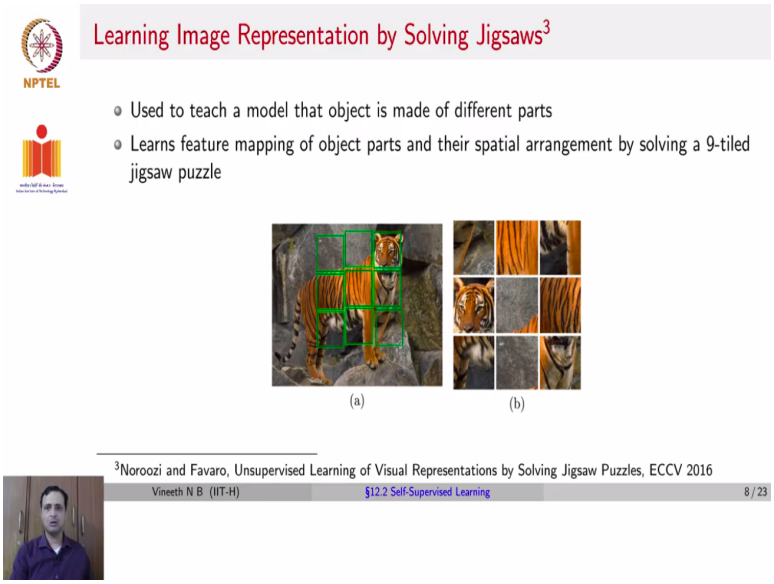
But in addition, this method also introduced an adversarial loss. What is adversarial loss? To look at the model completed image and the original image and have a discriminator say which of them is real and fake?

(Refer Slide Time: 06:31)



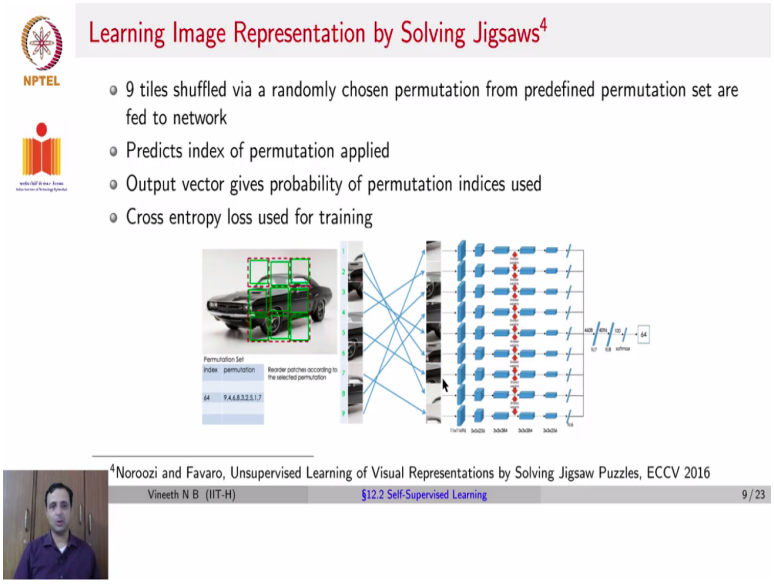
Using these two losses, this work showed fairly good results. You can see this example of an input where the central region is missing. This is an impression of a human artist on filling in the details. This is the same context encoder work with only the L_2 loss. You can see here that in the central patch, the L_2 loss has an averaging effect does not give sharp details at every pixel because L_2 loss minimizes error across the pixels and does not focus on each pixel individually. Adding an adversarial loss to the context auto encoder improves performance and makes the final In-painted image more realistic.

(Refer Slide Time: 07:22)



Another popular example of a Self supervision, Self supervised pretext task is solving Jigsaw puzzles. In this case, the objective is to teach a model that an image is made of multiple parts and to coax the model to learn certain mappings of parts to the objects as well as their spatial arrangement in an image. This is done by solving a 9-tiled jigsaw puzzle. But how do you teach a neural network to solve a jigsaw puzzle?

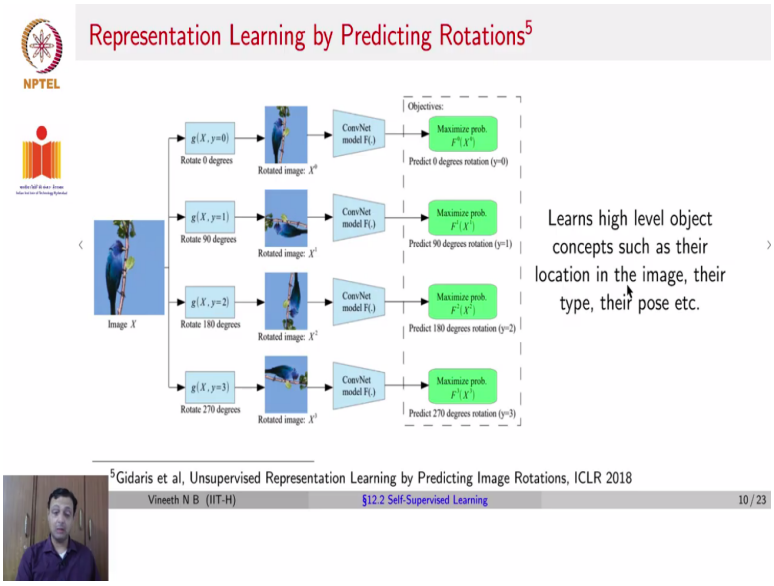
(Refer Slide Time: 08:08)



This is done using a Neural Network. 9 tiles are shuffled, taken from the image could be the entire image, could be part of an image are shuffled by a random permutation. And a neural

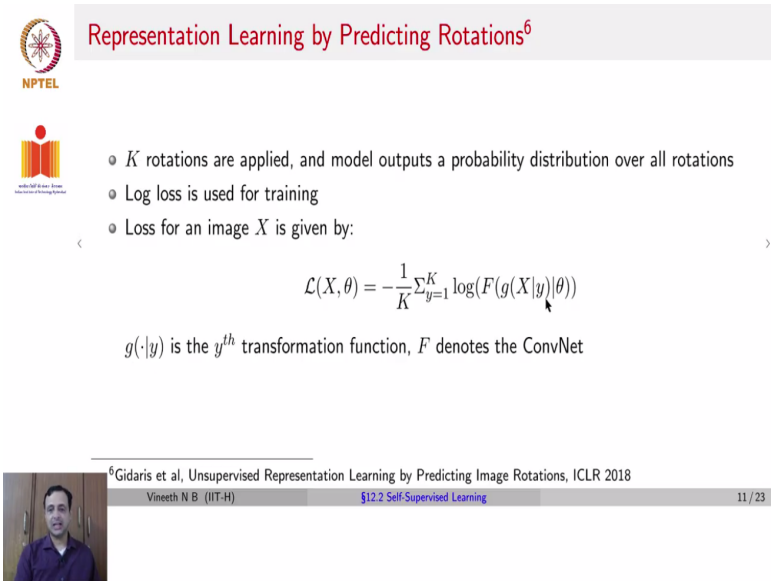
network is now asked to predict the right permutation. How is this implemented? Given 9 tiles, all possible permutations are given index values. And the job of the neural network is to predict the index associated with the right permutation for this jumbled up set of patches. What is the loss? A cross entropy loss on the index of permutation can be used to train the network.

(Refer Slide Time: 09:01)



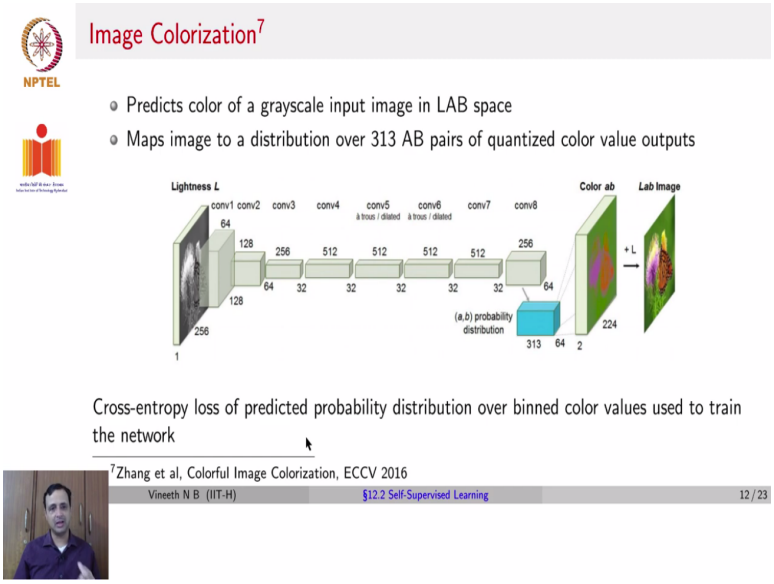
Another popular Self Supervised pretext task is predicting rotations. Given an image, multiple rotations of the image are performed. You can see here, this is the unrotated image. This is rotated by 90 degrees, rotated by 180 degrees, 270 degrees, so on and so forth. And the same CNN model in all of these cases, is used to predict the rotation angle. This can help the network learn images from the domain, as well as perhaps learn certain artifacts, such as an object's location in an image, its type, pose, etcetera.

(Refer Slide Time: 09:50)



So, K rotations are applied. And the model outputs a probability distribution over all rotations. Log loss is used for training your standard cross entropy loss, over the set of rotation categories that you have as outputs is the loss that is used for training. In this particular loss, g is the transformation function, which is our rotation, and F denotes the CNN that is given in this particular picture.

(Refer Slide Time: 10:26)

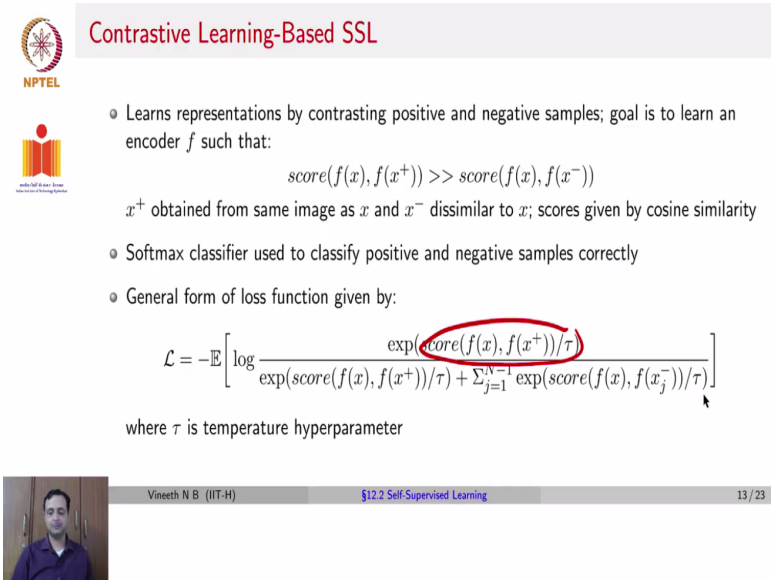


Another interesting Self Supervised task is Image Colorization. The goal here is to take a grayscale image, and then predict the colored version of the grayscale image. The assumption here is that you already have colored images in your dataset. So, you take grayscale versions of them and now ask your network to predict the color version of the grayscale image. Once you have trained such a network, remember, you could use this network as an initialization for say a semantic segmentation task or any other pixel wise classification task such as Surface normal prediction or depth estimation, so on and so forth, just as examples.

So, in this particular case, the image is mapped to a distribution over 313 AB pairs of quantized color value outputs. So, the model does not predict any real value in the AB output space. But about 313 bins are created and the model has to predict one of those 313 values, making this a classification problem. Why LAB space? Remember, LAB is one of the color spaces, where L is the grayscale intensity and AB bring the color into the final representation.

Why LAB space? Why not RGB space? The answer is simple. In LAB space, the L is given us input, you only have to predict AB and add L to get the image. Which means you need to predict only 2 channels instead of 3 channels, which makes it an easier task. Secondly, why not predict the color value as it is? Why should we quantize and then predict? The researchers in this paper found that quantizing it and then predicting give sharper colors, and predicting the actual color value necessitates an L_2 kind of loss, which can once again smoothen the colors over the image and not give us a sharp coloring effect.

(Refer Slide Time: 12:57)



More recently, over the last year, in particular, self supervised learning has been dominated by what are known as contrastive learning based methods where the overall goal is to learn representations by contrasting positive and negative samples, you can go back and revisit triplet loss, contrastive loss, margin loss, so on and so forth, ranking loss, so on and so forth. But the goal, but the approach here is slightly different because we do not have class labels.

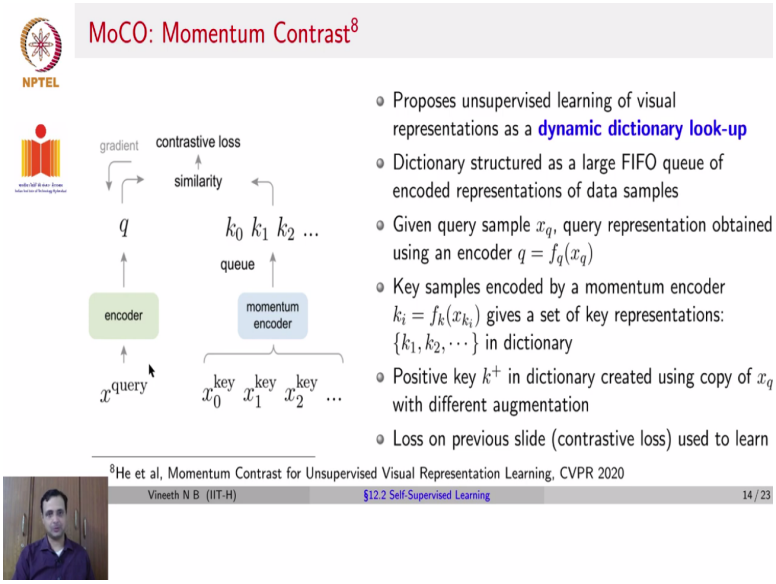
Let us see what the difference is. The goal here is to learn an encoder, which learns the representation of the an image such that some score, similarity score between an image and a similar sample is greater than a score between the same image and a negative sample. Now, if you had class labels, positive and negative, become easy to define, similar to what we saw with triplet loss. So, how do you go about it here is what we will talk about.

So, in this particular case, a softmax classifier is used at the end to classify the positive and negative samples, we will soon talk about how those positive and negative samples are obtained. And the general form of a loss function used for these set of tasks is given by a softmax loss and then having a log on top of that loss but the softmax is defined as where the numerator is between the current sample and a similar sample divided by the similarity of the score between the current sample and all samples including positive and negative.

And generally, which with such softmax operators, there is also a temperature hyper parameter, τ that you see here, which is also added to help learning. What is the role of a temperature parameter in softmax? It depend depending on what the value of τ is, it either helps smoothen the softmax distribution or sharpen the softmax distribution. If τ is greater than 1, it softens the softmax distribution, a distribution, which was perhaps concentrated in one label, now gets distributed over all possible labels.

On the other hand, if τ is less than 1, your value inside your exponent will increase. And any increase in the argument further increases the value of the exponential function and your softmax distribution gets sharper around certain nodes. τ greater than one softens the distribution or smoothens the distribution, tau less than one sharpens the modes of the distribution.

(Refer Slide Time: 16:09)

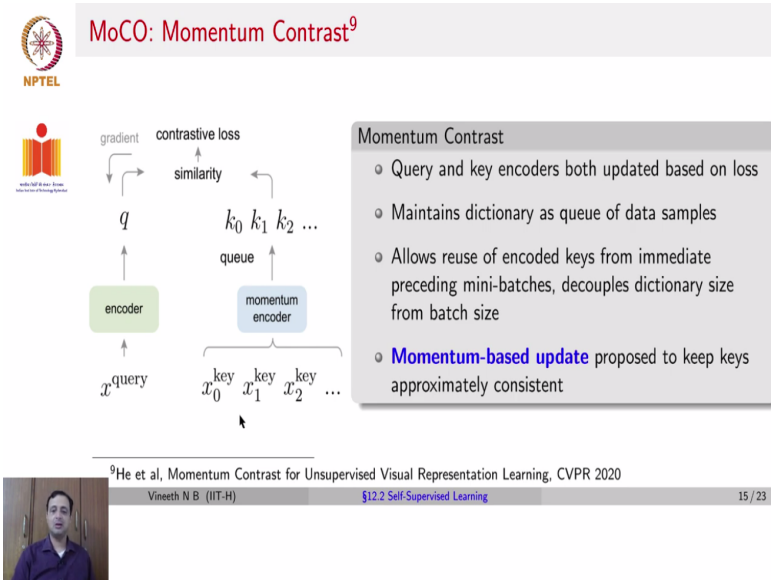


And one of the earliest methods in this space is known as MoCO, which was first made public about a year ago, but was published in CVPR of 2020. MoCO stands for Momentum Contrast and this method proposes an unsupervised learning of visual representations using the idea of a dynamic dictionary lookup. So, you can see here that the dictionary, which in this case is stored as x_0^{key}, x_1^{key}, x_2^{key} so on and so forth, a set of keys, is structured as a large first in first out queue of encoded representations of data samples.

So, given a query sample x_q, an encoder transforms it to a representation q. Similarly, all the key samples are transformed by another encounter encoder, which gives us k_0, k_1, k_2 so on and so forth. And then we measure the similarity between q k_0, q k_1, q k_2 so on and so forth pair wise and use the softmax log loss that we saw on the previous slide. One question is, how do we ensure that q is similar to one of these k's here?

Remember, when we use this for a Few-short learning, we would ensure that in a given meta learning episode, the query class was from one of the key classes. But now, how do you ensure that? That is ensured by making a one of the case as an augmented version of x_q. And as I just mentioned, it is trained using the loss that we saw on the previous slide, which is the log loss with a soft max with the temperature hyper parameter.

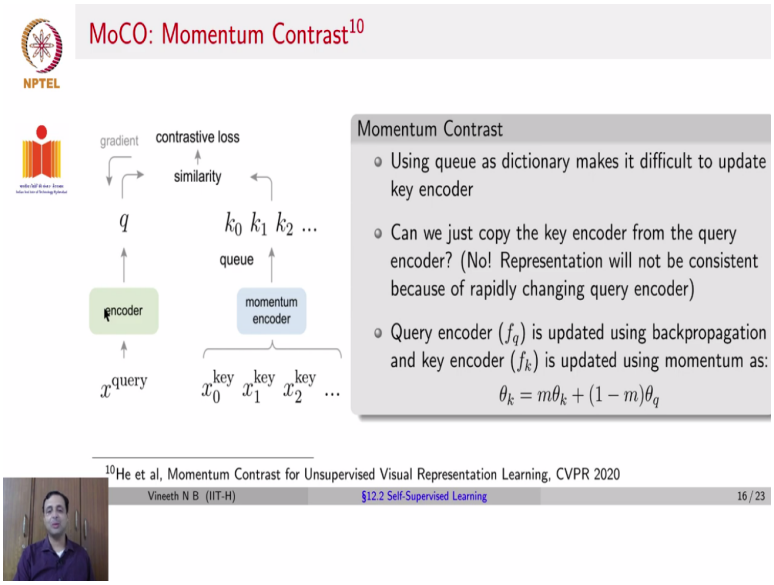
(Refer Slide Time: 18:21)



Why is this called Momentum Contrast in that particular case? The reason it is called Momentum Contrast is the full network is firstly trained end to end, which means both the query and the key encoders are updated based on the loss that we just talked about. And the dictionary is maintained as a queue of data samples. The interesting contribution of this work was to make this set or dictionary of encoded q's, as coming from the current mini-batch, as well as immediately preceding mini-batches.

So, the dictionary size would be slightly large than a mini batch size. So, all the data points that are not queued in a mini batch would become keys of that mini-batch. But you may also have a few images from the previous mini-batch, also as keys. And that is a queue, which where as newer keys come from a current mini-batch, the oldest keys from the oldest mini-batch that was visited, then is pushed out of the queue. And this is repeated over and over again. It is called Momentum, because this dictionary is based on the philosophy of momentum, where the idea is to use the previous iterations samples not the gradient this time in the current iteration.

(Refer Slide Time: 19:55)

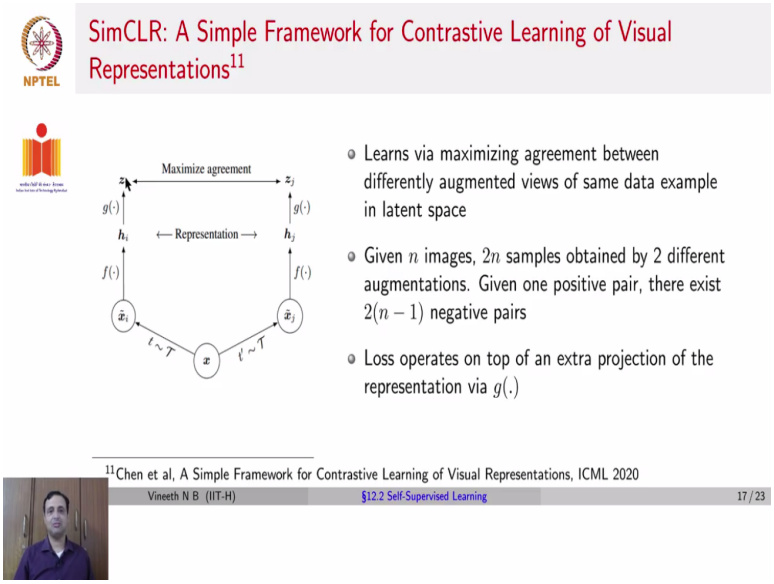


So one question here is, we say that we have an encoder, as well as a Momentum encoder. Why two encoders? Can't we use the same encoder in both places? It may be easy to update. Unfortunately, we cannot do that, because the representation may not be consistent for both the query and the keys. Because the momentum encoder is more stable than the query encoder in the

sense of maintaining a larger number of data points which stay constant for a couple of mini-batches at least whereas, the encoder can vary with each query image.

So, the task in both the encoders is slightly different. And that is why two different encoders to get the corresponding embeddings. While the query encoder is updated using normal backpropagation, the key or the momentum encoder is updated using a momentum concept where θ_k, the parameters at a particular iteration is $m\theta_k + (1 - m)\theta_q$, q comes as the parameters of the query encoder. So, that is the idea that is used here to update both the query encoder and the momentum encoder.

(Refer Slide Time: 21:27)



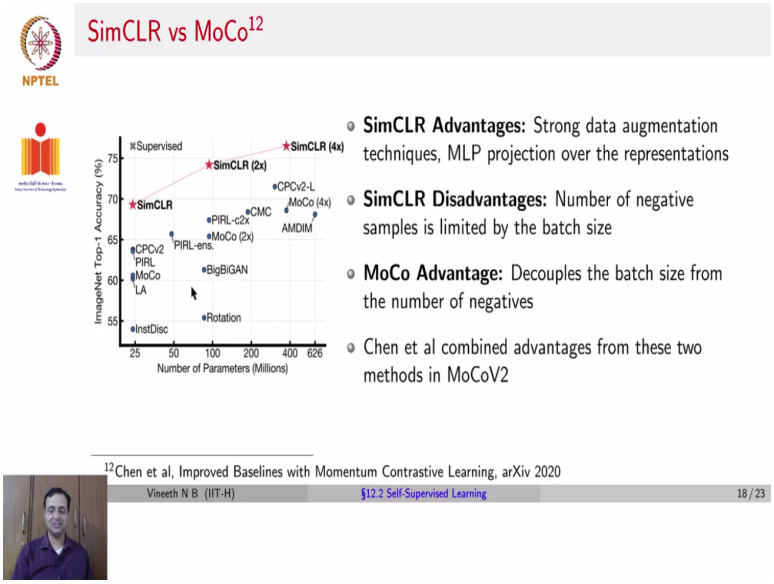
Another framework that came up around the same time was published in ICML of 2020. This year is known as SimCLR where the goal is more similar to standard contrastive learning, where the model learns via maximizing agreement between differently augmented views of the same data sample. How is this done? In a given mini batch, if you have n samples, you can obtain 2n samples up which can for which can be obtained using two different augmentations.

Now, given one positive pair, which is given an image, you give the image itself on one branch and give a rotated version of the image on the second branch, for instance. So, then for that one pair, now there exists $2(n - 1)$, where the other $(n - 1)$ images are the same mini batch, those

many negative pairs. So, we automatically have similar and dissimilar pairs in each mini batch without changing anything and this is useful to learn representations.

In SimCLR, you see that there are two networks an f network here, which performs a representation learning step and a g network here, which is then used to get a representation to maximize agreement. You could consider this similar to the relation network that we saw in few-shot learning, or zero-shot learning, where we first learned an embedding, and then learned a scoring mechanism to compare those embeddings. This is similar in that sense, but the goal here is not Few-shot posts, or Zero-short learning, but simply to learn the representations in an unsupervised setting, using the data in a mini batch itself.

(Refer Slide Time: 23:39)



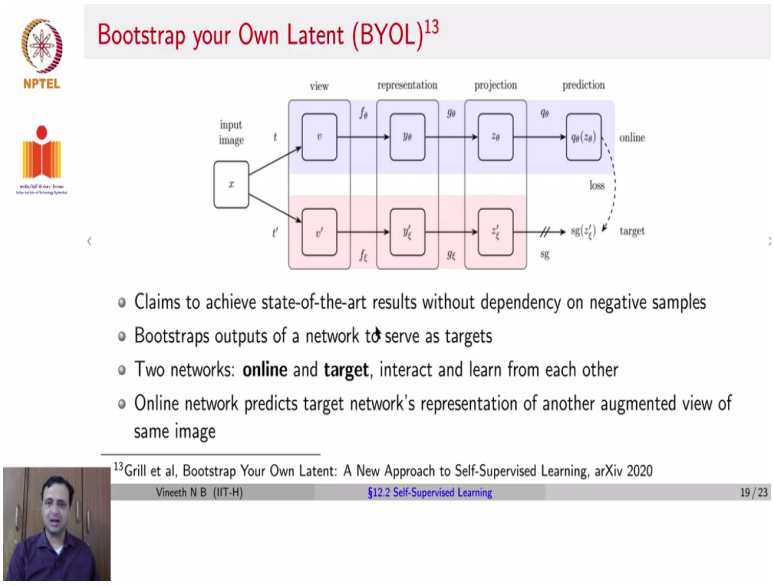
SimCLR and MoCO are direct competitors. So, one can compare the two. SimCLR has some advantages. It has strong data augmentation techniques, and uses an MLP projection over the representation layer. So, you have a q or a g on top of f, which is your embedding network. However, one of the disadvantages of SimCLR is that the number of negative samples you have is limited by the batch size. In MoCO that can be expanded to be a dictionary that is comprised of many mini-batches' images.

On the other hand, MoCO decouples the dictionary and the mini-batch, and does allows more negative samples to help better learning. So, you can see here that in terms of number of

parameters, when SimCLR has small number of parameters, it still gets a reasonable amount of accuracy when compared to a fully supervised setting, it gets close to 70 percent accuracy on ImageNet Top-1 whereas, supervised a fully supervised method is slightly over 75.

However, if the network of SimCLR has more parameters up to 4x, up to say 620 million, you can see the performance reaches close to supervised learning, without using any labels on ImageNet Top-1 accuracy. More recently, there has been a version known as MoCO V2, which combines the benefits of MoCO and SimCLR. We leave this for your reading after this lecture.

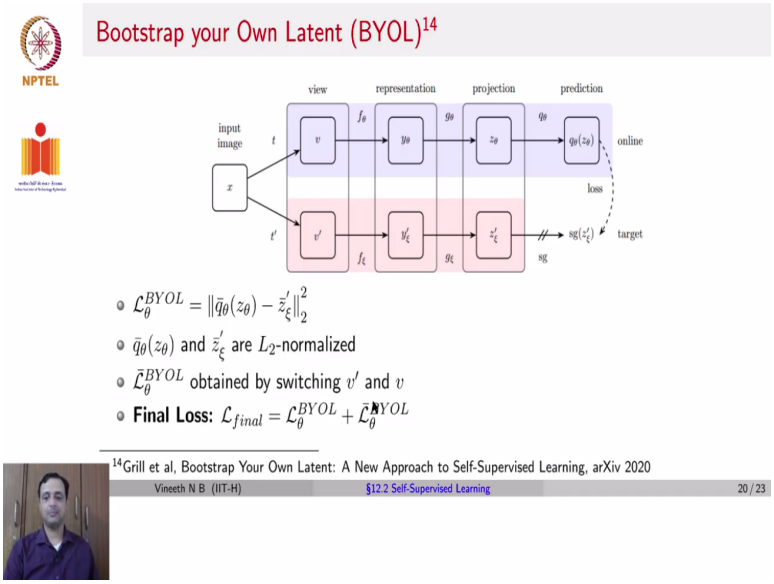
(Refer Slide Time: 25:38)



The last method that we will talk about in the context of self supervised learning is another recent method called Bootstrap your Own Latent or BYOL, where the method claims to achieve state of the art results without depending on any negative sample at all. How does it achieve this? It bootstraps the outputs of a network itself to serve as targets, remember, it is self supervised learning. So, every input is passed through two networks, an online network shown in blue, and a target network shown in red. So, what is the objective?

The objective is that the online network predicts the target network's representation of another augmentation of the same image. So, you give a certain image, let the network learn a series of representations and asked for it to provide a prediction and that prediction or output must match the output provided by the target network's prediction of an augmented version of the same input, t and t prime are two different transformations or augmentations of the same input. So, why is the bottom one used as the target?

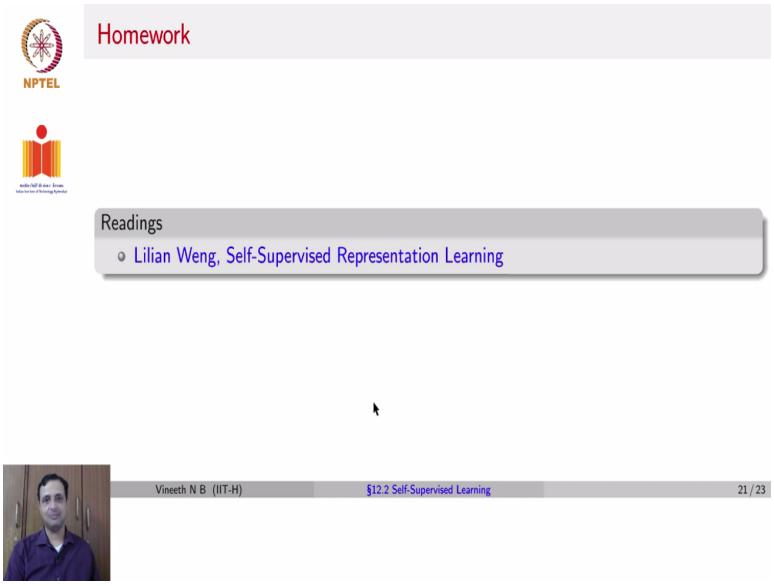
(Refer Slide Time: 27:15)



Not really, you can flip it as we will see soon. So, the loss for BYOL is an L_2 loss between the prediction of the online network $q_\theta$and the output of the target network, which in this case is given by z_ξ. Now, one can flip the network to also get the loss the other way. So, in both of these cases, q_θ and z are L_2 normalized. And when it is flipped, the lower network becomes the online network and the top network becomes the target network. So, one can switch the roles of v' and v.

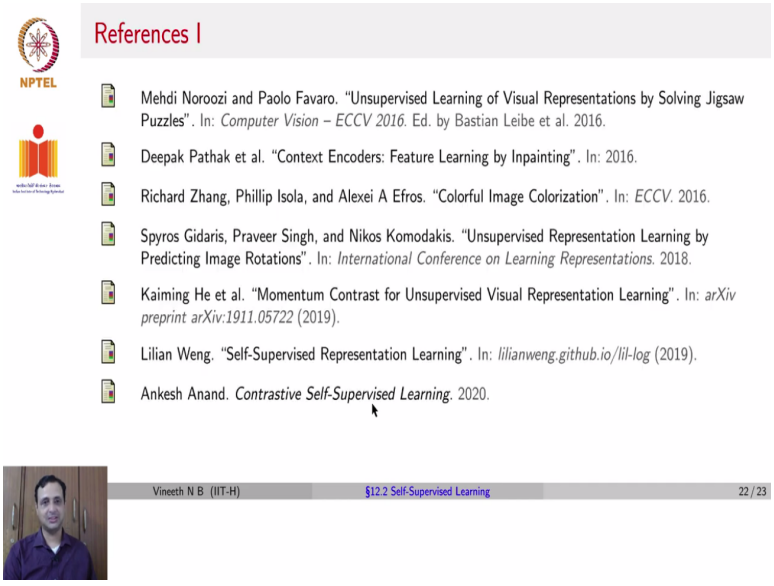
And the final learning is through a combination of the original loss and the loss obtained by flipping which is denoted as L, $\bar{L}$ where the bottom network is online, and the top network is target.

(Refer Slide Time: 28:19)



Your homework is to read this excellent article on Self Supervised Representation Learning by Lillian Weng. Also, try to go through how MoCOv2 combines MoCO and SimCLR.

(Refer Slide Time: 28:38)



Here are some references.