CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Representation Learning (Siamese and Autoencoder)

Meenu Menon(2019201026) Nisarg Mankodi(2019201065) Srinivasa Rao Pegadaraju(2019900066)

1 Transfer Learning

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting (i.e., distribution P1) is exploited to improve generalization in another setting (say distribution P2).

In transfer learning, the learner must perform two or more different tasks, but we assume that many of the factors that explain the variations in P1 are relevant to the variations that need to be captured for learning P2. This is typically understood in a supervised learning context, where the input is the same but the target may be of a different nature. For example, we may learn about one set of visual categories, such as cats and dogs, in the first setting, then learn about a different set of visual categories, such as ants and wasps, in the second setting.If there is significantly more data in the first setting (sampled from P1), then that may help to learn representations that are useful to quickly generalize from only very few examples drawn from P2. Many visual categories share low-level notions of edges and visual shapes, the effects of geometric changes, changes in lighting, etc. In general, transfer learning, multi-task learning, and domain adaptation can be achieved via representation learning when there exist features that are useful for the different settings or tasks, corresponding to underlying factors that appear in more than one setting.

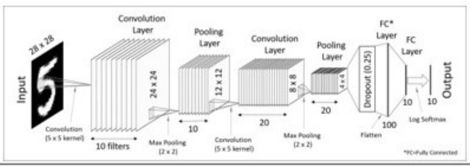
In the related case of domain adaptation, the task (and the optimal input-to-output mapping) remains the same between each setting, but the input distribution is slightly different. For example, consider the task of sentiment analysis, which consists of determining whether a comment expresses positive or negative sentiment. Comments posted on the web come from many categories. A domain adaptation scenario can arise when a sentiment predictor trained on customer reviews of media content such as books, videos and music is later used to analyze comments about consumer electronics such as televisions or smartphones. One can imagine that there is an underlying function that tells whether any statement is positive, neutral or negative, but of course the vocabulary and style may vary from one domain to another, making it more difficult to generalize across domains.

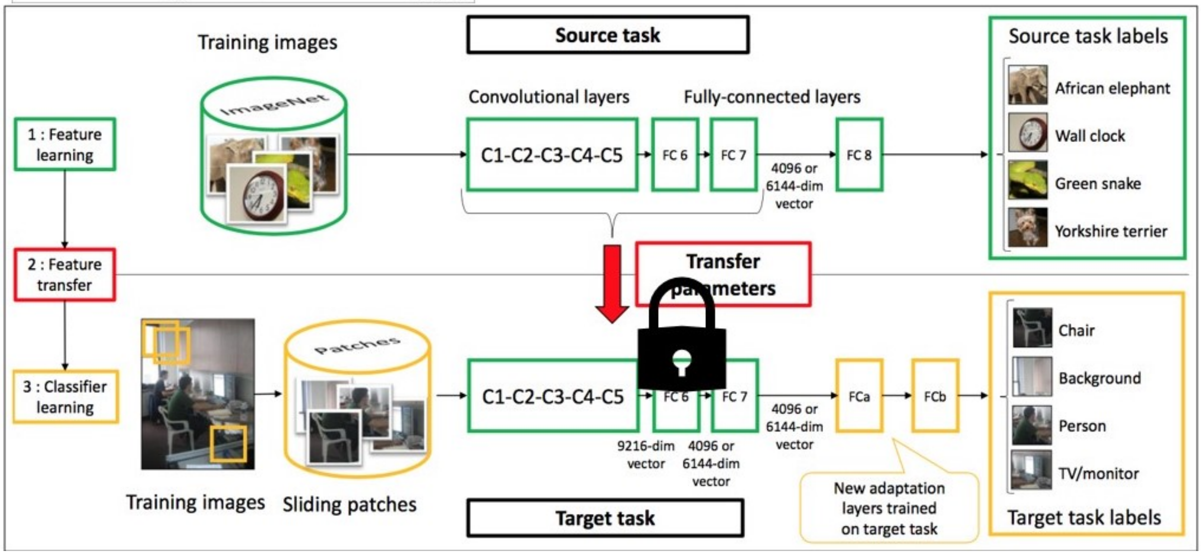
A related problem is that of concept drift, which we can view as a form of transfer learning due to gradual changes in the data distribution over time. Both concept drift and transfer learning can be viewed as particular forms of multi-task learning. While the phrase "multi-task learning" typically refers to supervised learning tasks, the more general notion of transfer learning is applicable to unsupervised learning and reinforcement learning as well.

In all of these cases, the objective is to take advantage of data from the first setting to extract information that may be useful when learning or even when directly making predictions in the second setting. The core idea of representation learning is that the same representation may be useful in both settings. Using the same representation in both settings allows the representation to benefit from the training data that is available for both tasks.

1.1 Transfer Learning: Approach 1:

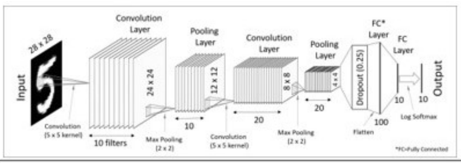
In the first approach new domain data is small in quantity. In this approach we take the source network, copy the weights and add adaptation layers and focus on learning the weights for the adaptation layers.As seen in the picture below, the weights carried from the previously trained network are locked from updating.

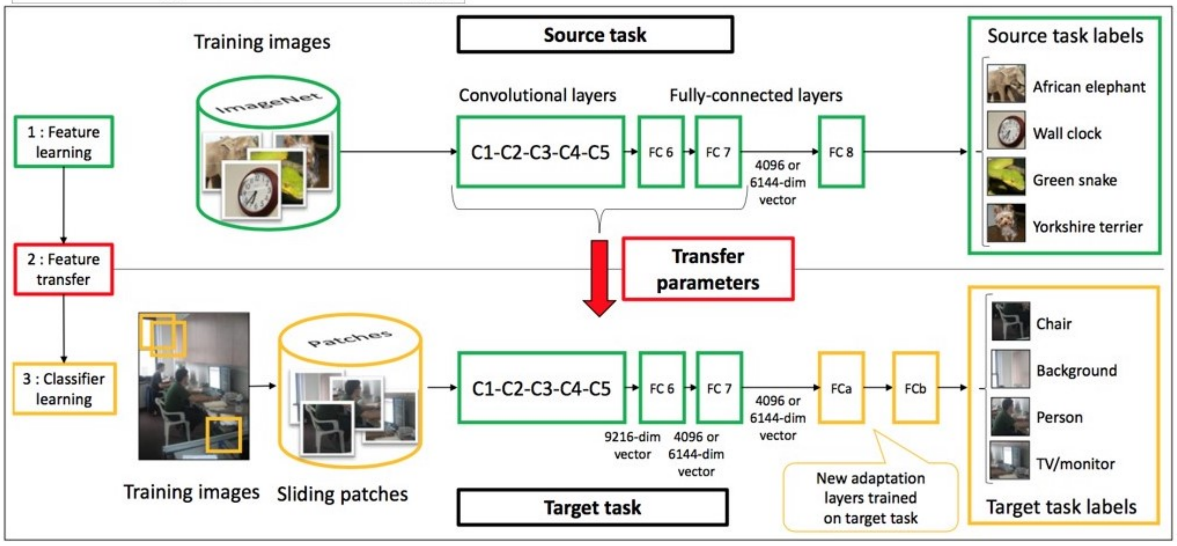




1.2 Transfer Learning: Approach 2:

In the second approach, where the new domain data is significantly different, where the lower level features are usable, we will have to fine tune all the weights.

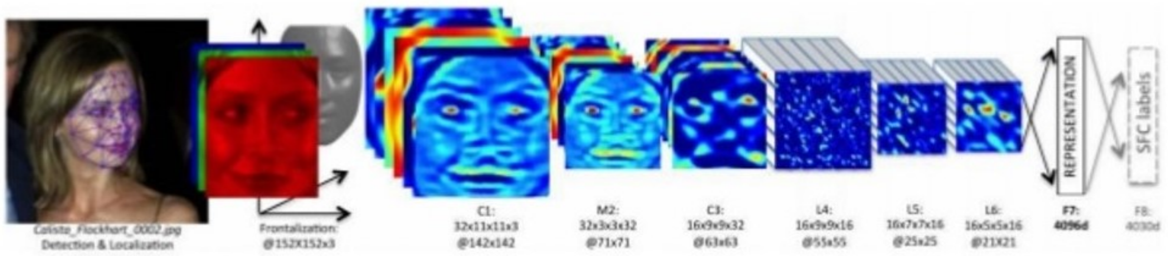




1.3 Classification: Face recognition using fine tuning of weights

Given a dataset of faces, we want to recognize a new face. What is the ID for a given face? This is something similar to idetifying a person in a database like Adhaar.

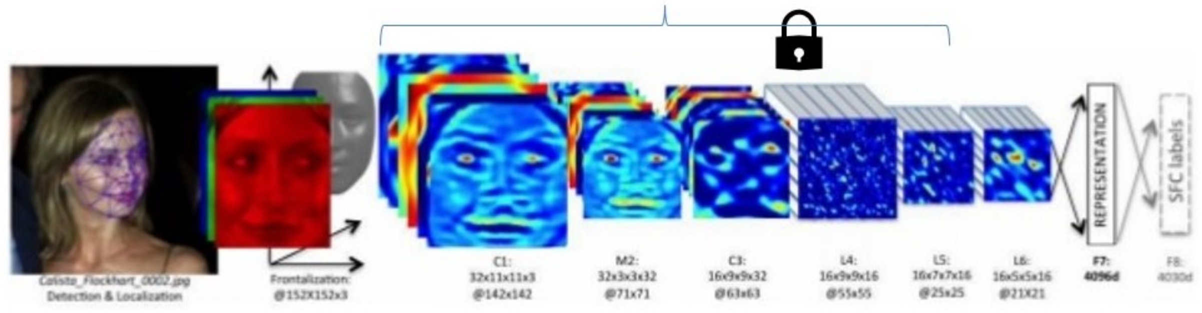
Deep face is a network trained on celebrities. If we want to use this network to identify a person of say our group, lab team or the class, we will employ approach 1 of transfer learning discussed above. In this case we freeze most of the weights and learn only the last couple of layers. The number of ids that we want to recognize may be small. We need to have few samples for each person and pass them through the network.



1.4 Classification: Face recognition using no fine tuning of weights

In extreme cases, where we don't have enough samples of each person we want to identify,we can still do classification, without performing fine tuning of weights. In the process of training celebrities, the network has learnt some representation of faces. In this case, we have to pass the image of each person to be classified and store the representation. When a new face comes in, it is treated as a query sample and use KNN classification. In this manner, we can use pre-trained network for feature extraction. In many cases the representation learned in pre-trained networks are good enough. Just as KNN is a good baseline before trying out anything costlier, we can use this method for base lining and use more complex and advanced techniques when resources are available.

Face Identification/Recognition (1:N matching)


Calisto_Flockhart_0002.jpg
Detection & Localization
Frontalization:
@152X152x3
C1:
32x11x11x3
@142x142
M2:
32x3x3x32
@71x71
C3:
16x9x9x32
@63x63
L4:
16x9x9x16
@55x55
L5:
16x7x7x16
@25x25
L6:
16x5x5x16
@21X21
F7:
4096d
F8:
4030d
REPRESENTATION
SFC labels

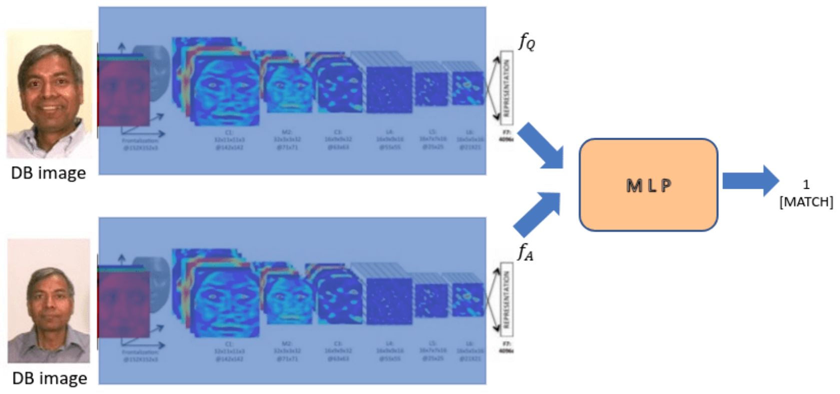
2 Verification

Verification aims to check if two given images match or do not match.For verification/ image matching, we compare the representations generated by the neural network.We assume that there is a good representation of the images, using a proper deep learning model as feature extractor.

1. **Approach 1a :**Here, we have a database of multiple image representations.

   A given query image is checked against all the images in the database to check if it matches with any one or not. This has a network has a Multi Layer Perceptron that does the mapping of the two feature vectors of the images to a two class classification problem.It generates output as 1 if there is a match and 0 if the query image doesn't match any of the given db images. The out put of this network is:

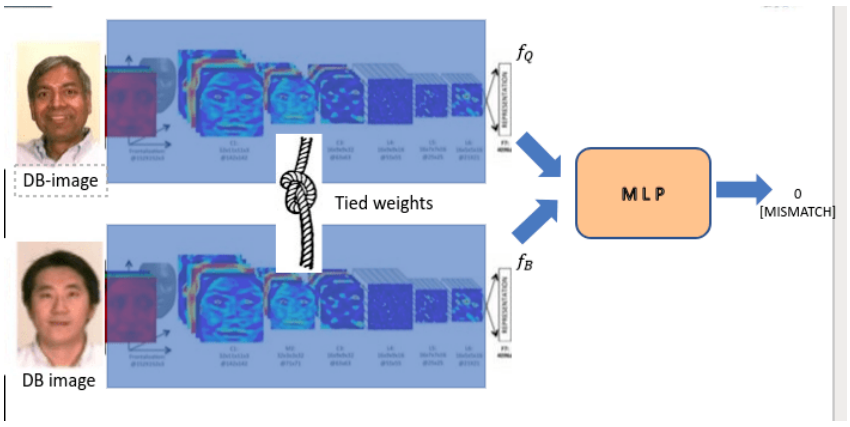
$$Output = \begin{cases} 1 & \text{if Query image = DB image} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$



Approach 1a

   The MLP used here should be designed so as to generate the binary classification.A sigmoid function could be used as the last layer,that gives the degree of match or mismatch. The threshold, above which it gives a match can be set accordingly by the designer of the MLP.

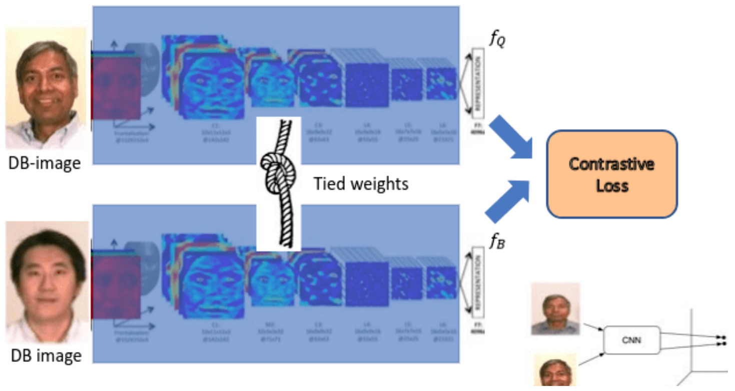
2. **Approach 1b :**This involves fine tuning the deep learning network model to get a better representation of the feature vectors. Suppose there are two copies of the same network present.Pass one image to top network one to bottom and obtain obtain their feature vector.Now, calculate binary cross entropy loss based on the ground truth.Say it is L1, once it is propagated back up to the MLP's first layer. This l1 is passed through the deep learning networks and their respective gradients are calculated as $\frac{\partial L1}{\partial W1}$ and $\frac{\partial L1}{\partial W2}$. Then average the two gradients obtained to update the weights in the deep learning network. This is the notion of **tied weights**. In reality,there is only a single set of weights.So we pass first image and after one forward pass and back propagation we get a gradient as x and for the next image we get say y as the gradient.Then, $(x + y)/2$ is used to update the weights of the network.



Approach 1b

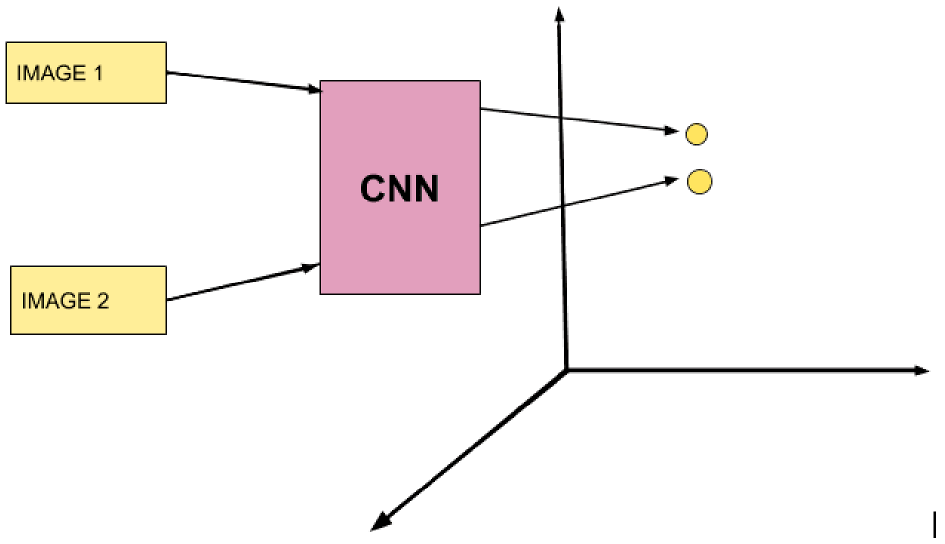
How different is it from mini batch?
Well, in mini batch two samples are treated as two independent entities.When we pass two independent images, we update the weights also independently . But in tied weight, we are doing that only once.

3. **Approach 1c :** In the previous two approaches, MLP was making a decision about two feature representation and classify them as match or mismatch. It was more specific to the database of images that we had.This approach mentions about building a network for matching that is general purpose.i.e. a reusable matching network by exploiting some property of the verification network. This is done using the concept of **contrastive loss**.

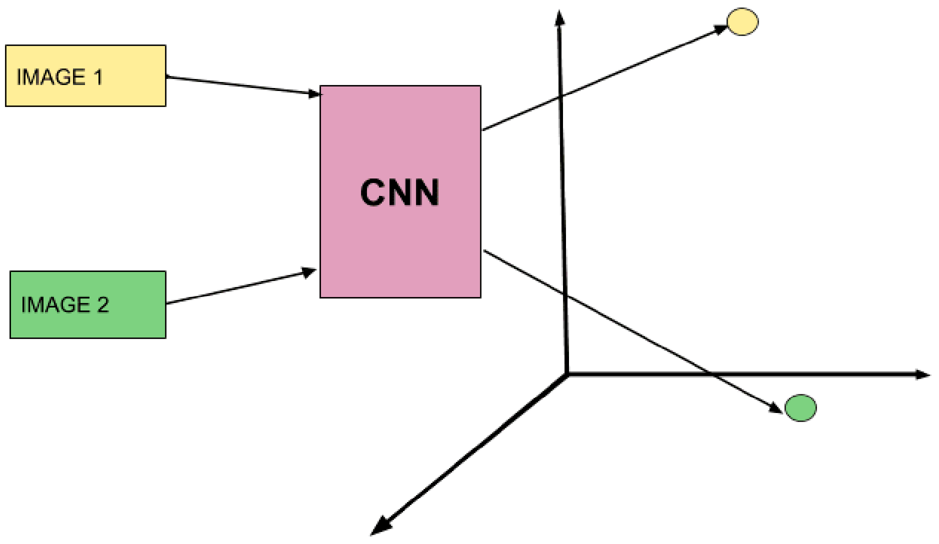


Approach 1c

We need to modify the representations of images in such a format, that for representation with same ID(i.e. different representations of the same person), the coordinates of the representation lie closer to each other and for representations with different ID, the coordinates lie far from each other.



Coordinates of two images of same person lie close



Coordinates of two images of different persons lie farther

The contrastive loss function used here is :
LOSS=yd^2+(1-y)max(margin-d,0)

$$y = \begin{cases} 1 & \text{if Query image} = \text{DB image} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The feature vectors are learnt so that:

d=dist(fq,fb) is large when the ids mismatch (y=0)
d=dist(fq,fb) is small when the ids match (y=1)

where fq=feature vector of query image and
fb=feature vector of db image

The aim is to minimise the loss.

(a) **y=1 :** Contrastive loss=(1)d^2=d^2
This is the case where the two representations match.Now contrastive loss needs to be minimised. Thus, d^2 needs to minimised i.e. the representations must be brought closer together (since d=dist(fq,fb)).

(b) **y=0 :** Contrastive loss=(1-y)max(margin-d,0)=max(margin-d,0)
This is when we have a mismatch.So, the coordinates of the representations should lie as far as possible.The contrastive loss$\geq$ 0.

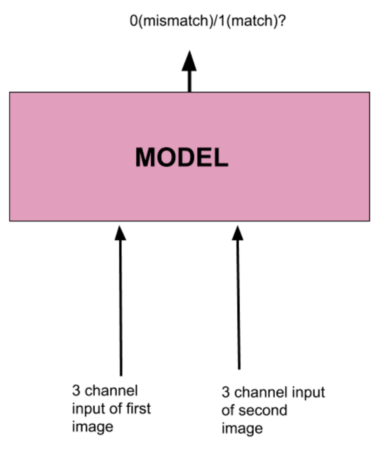
- If the images lie closer than they should be(i.e.d< margin), then the penalty is (margin-d).This happens when $d < margin$.
- If the images already lie far apart (i.e.$d \geq$ the margin) then no efforts are put in to make them any farther.This happens when $d \geq margin$.

Margin is a hyper parameter here.

- Too small margin :It doesn't do a good job on pushing away representations of different images
- Too large margin :It makes training difficult which is usually a clue to reduce the margin.

So, we train the network to obtain a good distance function. The network now works like a similarity matcher.Once we obtain the final representations, we could use the Euclidean distance on the modified representations.
We are learning a similarity metric.So, how different is it from clustering ?
In clustering we had a given set of feature representation which are just clustered together based on some parameter. But here, we have the representation itself being modified or fine tuned that behaves in a sensible manner when there is a labeled data set.

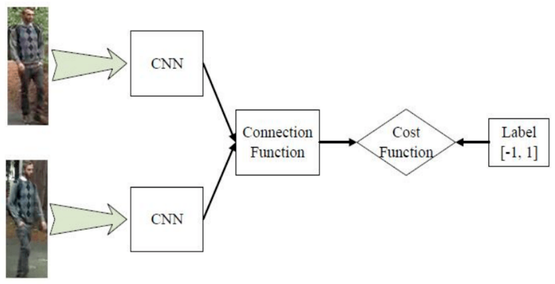
4. **Approach 2 :** Here two images with three channels each are fed to the model together which gives the output as match /mismatch.
Siamese network
This follows Approach 2.One of the feature vectors is pre-computed, thus forming a baseline against which the feature vector is compared. This network is not learning to classify an image directly to any of the output classes. Rather, it is learning a similarity function, which takes two images as input and expresses how similar they are.[4]


0(mismatch)/1(match)?
MODEL
3 channel input of first image
3 channel input of second image

Approach 2

Applications

- *Signature Verification :* Used to identify forgery in bank cheques. Digital pen's inputs are fed to the network and cosine distance is used for the calculation similarity score.It gives out +1(match) or -1(mismatch).
- Person re-identification : Person re-identification is the task of associating images of the same person taken from different cameras or from the same camera in different occasions.Feeding CCTV images to identify a person is done here.Here, two images are fed to the Siamese network .The network has a symmetry structure with two sub-networks which are connected by Cosine function. [2]



The Siamese convolution neural network

- Image Retrieval:This is used to rank images according to their similarity to a given image.Ranking loss is used, where we train models to rank items in an specific order.
- Street view to overhead view matching:Train the network on street view and overhead view of a place and given an image of say street view, we can retrieve its overhead view.

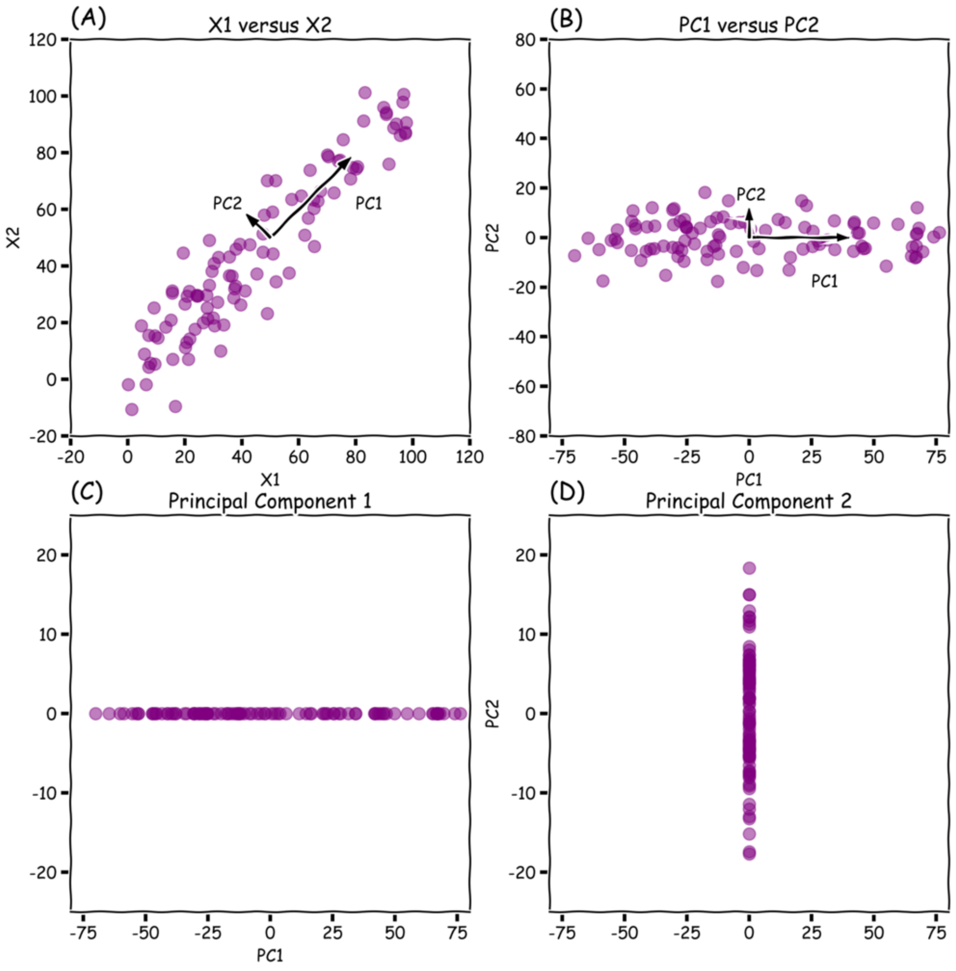
Different types of losses:

- **Ranking loss:** This name comes from the information retrieval field, where we want to train models to rank items in an specific order.
- **Margin Loss:** This name comes from the fact that these losses use a margin to compare samples representations distances.
- **Contrastive Loss:** Contrastive refers to the fact that these losses are computed contrasting two or more data points representations. It is a distance-based loss function.
- **Triplet Loss:** Often used as loss name when triplet training pairs are employed.We can train the network by taking an anchor image and comparing it with both a positive sample and a negative sample. The dissimilarity between the anchor image and positive image must be low and the dissimilarity between the anchor image and the negative image must be high.[3]

3 Auto encoders

Unsupervised learning is a type of machine learning algorithm used to draw inferences from data sets consisting of input data without labeled responses. tasks like clustering, density estimation, visualization etc. can be solved using unsupervised learning algorithms.

Data Visualization and Dimensionality reduction : The main goal of these types of algorithms is to study the intrinsic and hidden structure of the data in order to get meaningful insights, segment the datasets in similar groups or to simplify them. techniques like PCA (principle component analysis), SVD (Singular value decomposition), are used to compress a dataset onto a lower-dimensional feature subspace with the goal of maintaining most of the relevant information. Underlying assumption these techniques make, is that the data in the higher dimension lies on a linear manifold. So by doing orthogonal transformation and projection on these new orthogonal basis. we can retain most of the information.

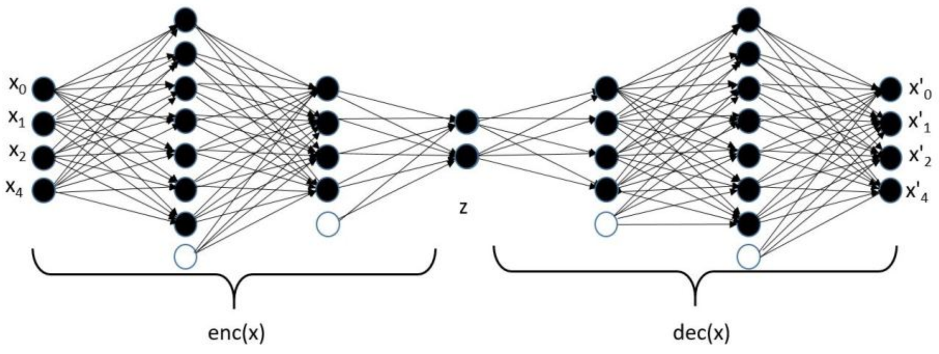


PCA dimensionality reduction

But in most of the cases Data in higher dimensions doesn't lie on a linear surface. PCA and SVD can't preserve the topology of the data in these cases. methods like LLE (local linear embedding) and Isomaps can do better a job of mapping those points lying on non-linear surface to a lower dimensions. However, several problems in the LLE algorithm still remain open, such as its sensitivity to noise, inevitable ill-conditioned eigen-problems, and lack of knowledge on how to deal with novel problems

By exploiting neural network architecture it is possible to learn the non linear transformation for any lower dimensional manifold residing in the data.

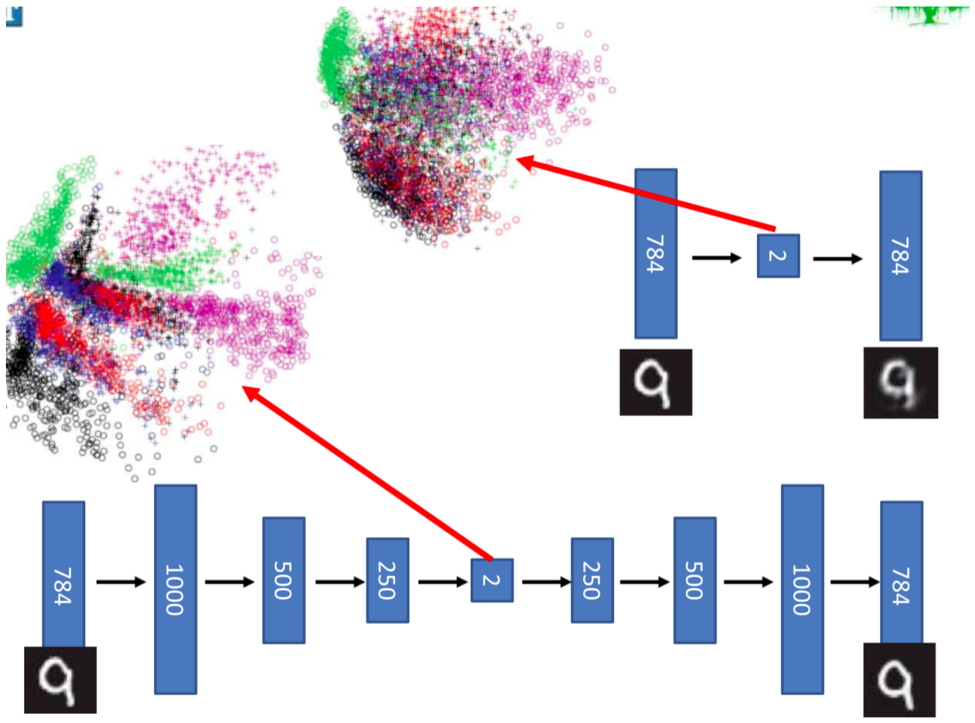
Deep autoencoders : At its most superficial level autoencoder is a feed-forward neural network with a middle layer of much smaller dimension that the input and an output the same dimension as the input as shown in Figure below



auto encoder: multi-layer perception

Let the dimension of the input vector be N and the dimension of the middle layer, which is referred to as z, to be k. Represent the first half of the network by the function $z = enc(x)$ and the second half the network be $x' = dec(z)$. An autoencoder network is trained to be the identity function. output of the hidden unit in between encoder and decoder modules is our lower dimensional embedding.

In other words, suppose we have m points xi, $i = 0, m-1$, in R^N, then we train the network so that xi' $= dec(enc(xi))$ approximates xi for each i. The function $enc(x)$ is our encoder function that maps R^N to R^k , and the function $dec(z)$ is the decoder that maps from R_k to R^N. Of course, $dec()$ and $enc()$ are not unique and the depend on the way we train the network. Figure bellow shows such an example. In one architecture neural net with many hidden layers is used and it performs better than the one with only one hidden layer.



auto encoder: data visualization

SVD with autoencoders The SVD decomposition of A is given by $A = UVW$ where U is m by m, V is a diagonal matrix of size m by N and W is a N by N matrix. U and V have the property that $U^TU = I$, where U^T is the transpose of U and $WW^T = I$. The values on the diagonal of V are the singular values of A in decreasing size. Multiplying $A = UVW$ on the right by W^TW yields the equation $AW^TW = UVWW^TW = UVW = A$. Another way of saying this is if x is a row of A, then $x = (xW^T)W$. Pick the k largest singular values and define $A' = U_kV_kW_k$ where U_k is the first k columns of U and W_k are the first k rows of W and V_k is our k by k matrix of top eigenvalues. In these terms, $z_i = x_iW_k^T$ is the encoding of x_i into R_K and $x'_i = z_iW_k$ corresponds to the decoder function. In fact, if the neural net for the autoencoder is linear and only one layer these two concepts are identical. In other words, the truncated SVD is a degenerate form of a neural network autoencoder.

Convolutional Autoencoders : Convolutional AutoEncoders (CAEs) approach the filter definition task from a different perspective: instead of manually engineer convolutional filters we let the model learn the optimal filters that minimize the reconstruction error. later part of the network is called deconvolution network, which uses unpooling layers. To perform unpooling, we need to remember the position of each maximum activation value whilst we did max pooling in convolution layers. Then, the remembered position is used for unpooling as shown above.

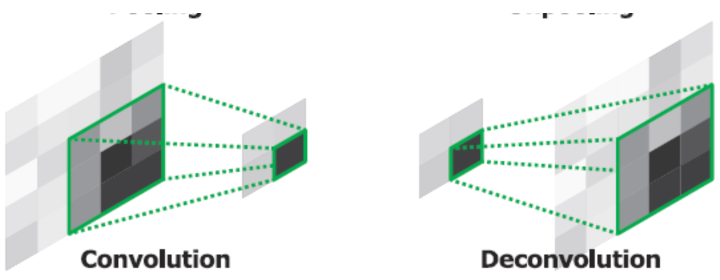


Figure 1: Unpooling

CAEs are the state-of-art tools for unsupervised learning of convolutional filters. Once these filters have been learned, they can be applied to any input in order to extract features. These features, then, can be used to do any task that requires a compact representation of the input, like classification.

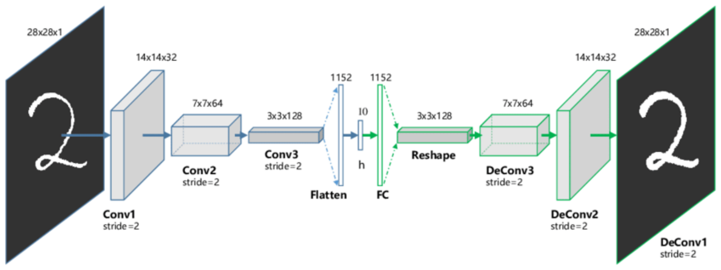


Figure 2: Convolutional auto encoder

Similar Image Retrieval using Autoencoders It turns out that encoded representations (embeddings) given by the encoder are magnificent objects for similarity retrieval. Most raw and highly unstructured data, for example an image of your face, is typically embedded in a non-human interpretable vector space representation. So instead of operating painstakingly in the space of RGB pixels, we can use a trained encoder to convert the image of your face to lower-dimensional embedding sitting in a hopefully more meaningful dimensions such as "image brightness", "head shape", "location of eyes", "color of hair", etc. With such a condensed encoding representation, simple vector similarity measures between embeddings (such as cosine similarity) will create much more human-interpretable similarities between images. Had we done PCA on the data we'd have got eigenfaces instead of these embeddings. and the data would have been represented by the linear combination of those eigenfaces.

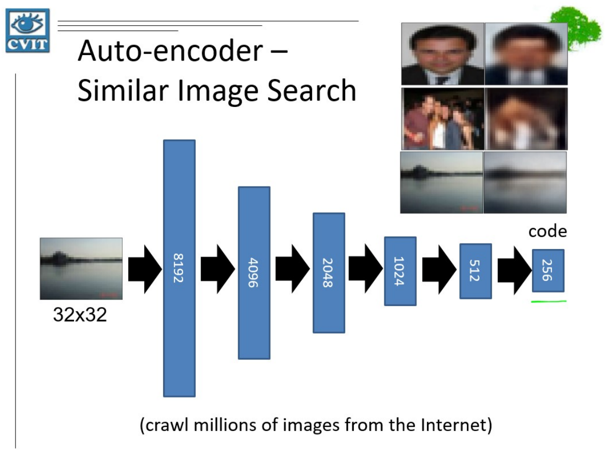


Figure 3: Autoencoder used for special embedding

Variational Autoencoders : A variational autoencoder (VAE) is one that is based on creating an encoding that has a probabilistic distribution that generates a good model of the input data from some system. In a VAE, one views the encoding vector z as a latent variable with a probability density function $P(z)$ such that if we sample z from $P(z)$ we have a high probability that the decoded vector $d(z)$ is a good example from or very near the manifold for our data X. Given this density function we can model the system that generated X in two way.

1. we can pick values from the distribution that generate new potential samples of our system.
2. or we can take a candidate x as a possible member of the system, we can then use the computed $P(enc(x))$ as probabilistic evidence that the candidate is a likely member.

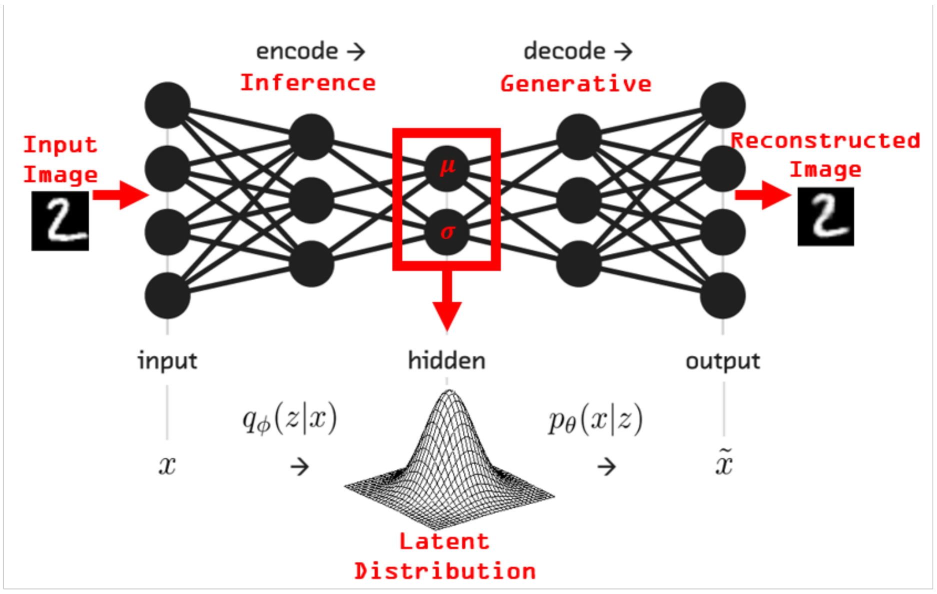


Figure 4: auto encoder: Autoencoder used on MNIST dataset

The architecture of the VAE is similar to the standard with one major exception. Rather than generating an explicit vector in the embedded space we generate a Gaussian density function defined by a mean and a standard deviation (actually the log of the squared standard deviation which is assumed to be diagonal) for each of our sample points. We then sample from this distribution and generate a value for the output of the encoder which can be input to the decoder. This is illustrated in the diagram above.

References

[1] Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville

[2] http://www.fubin.org/research/Person R eID/P erson R eID.html

[3] https://gombru.github.io/2019/04/03/ranking$_loss/$

[4] http://www.fubin.org/research/Person R eID/P erson R eID.html