The Why and the How of Deep Metric Learning.

Diving deep into metric based deep learning.

Aakash Agrawal · Follow
Published in Towards Data Science
17 min read · Jan 11, 2021

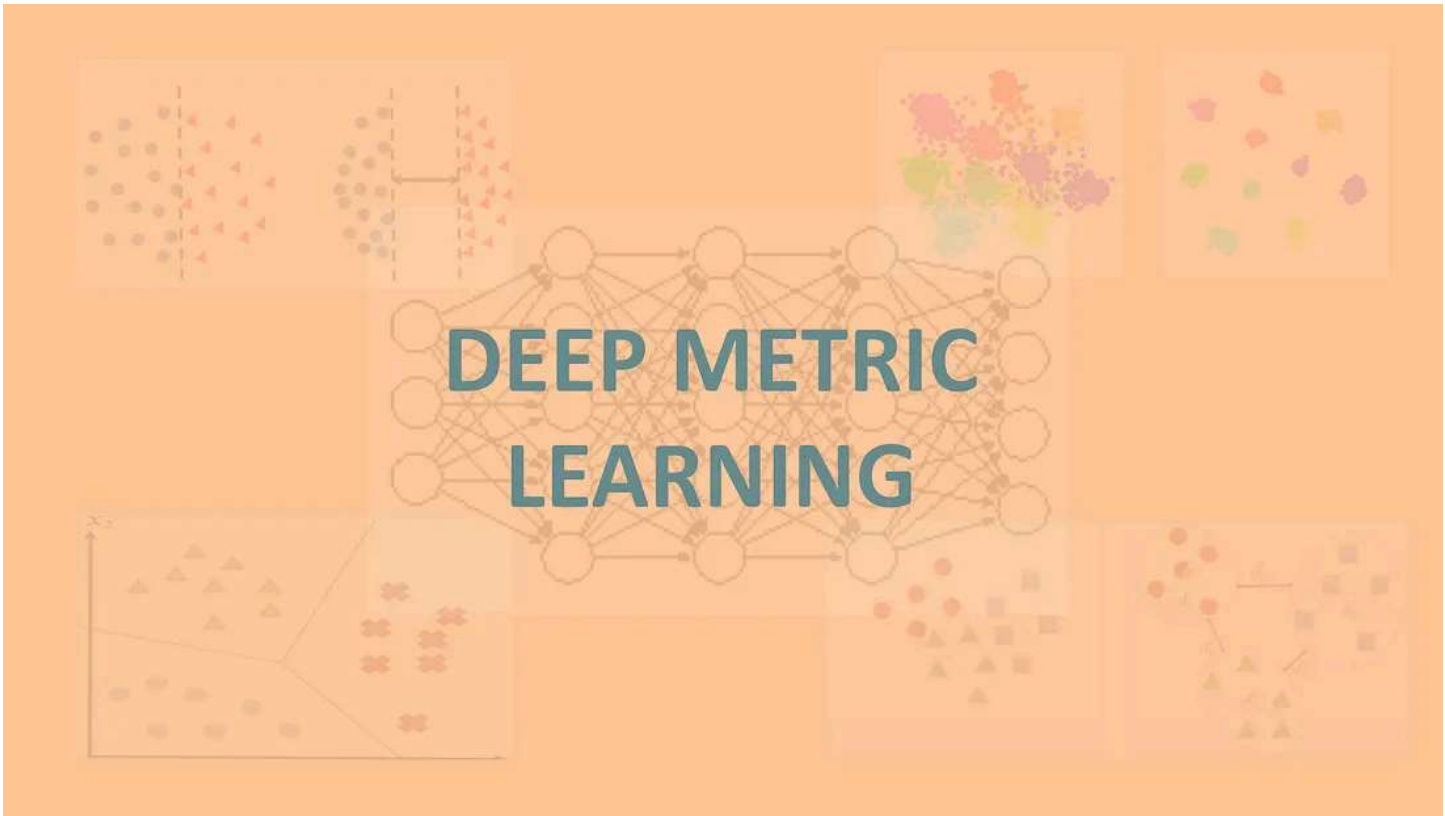


Image by Author

This article provides insights into the very basics of Deep Metric learning methods and how it helps in achieving state of the art results for tasks like face verification and face recognition. During my past internship, I have been working on these tasks, and seeing the power of deep metric learning methods for such tasks has influenced me in writing this article. Besides Face Recognition, Verification there are a number of other applications

where Deep Metric Learning has proven to be quite effective; Anomaly Detection, three-dimensional(3D) modelling, are a few of them.

Table of Contents

Prerequisites

The reader requires a good knowledge of linear algebra and familiarity with the basic concepts in machine learning to understand this article. I hope you will enjoy learning deep metric learning.

Keywords: Metric learning, Triplet loss, softmax loss, Face recognition, Face verification, DCNN, Arcface, Sphereface, Cosface.

A Face Verification System. source by Xenia on Dribble.com.

A tour of the basics

Let us first understand a few basic terminologies and establish a solid ground to enhance our understanding of deep metric learning.

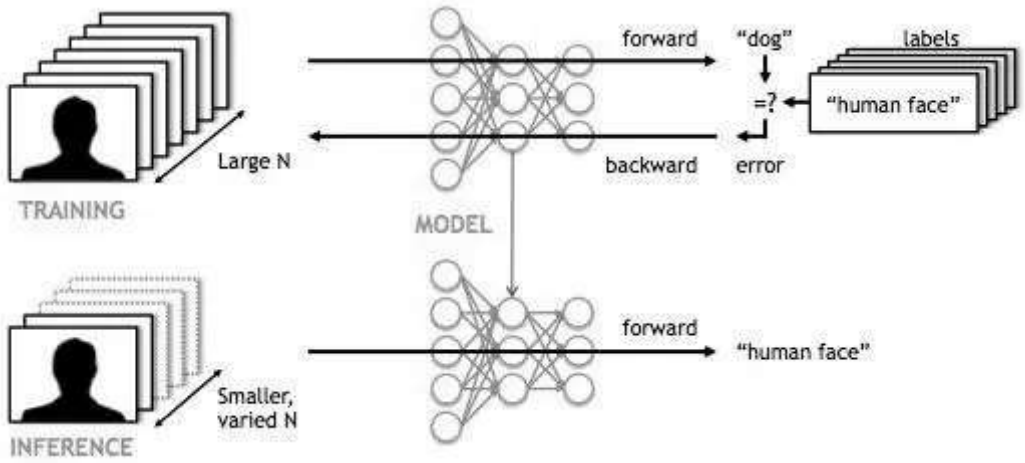
Face verification is the task of determining whether a given pair of images belong to the same person or not. In simple words, it is a task where given an image, we try to answer the question. **Is that you?.** A **1:1** authentication problem.

Face Recognition, on the other hand, is a combination of two tasks: **Face identification** and **Face verification.** Face identification is the task of recognizing a person from a database of images. A task where given an image database (let's say a gallery of k persons) we try to answer the question, **who are you?.** So, Face recognition is a **1:k** authentication problem.

It is quite helpful to know the basic training and inference process of a simple **Artificial neural network.** {please skip this part if you are familiar with the topic}.

TRAINING PHASE

1. We **randomly initialize** the weights and biases according to some probability distribution,
2. Feed the data set into the **untrained** neural network architecture.
3. **Forward propagation:** the hidden layer accepts the input data, applies an **activation** function, and passes the activations to the next successive layer. We propagate the results until we get the predicted result.
4. After generating the predictions, we calculate the **loss** by comparing the **predicted** results to the ground truth labels.
5. **Backward propagation:** We compute the **gradients** of the loss wrt to weights and biases, and adjust them by subtracting a small quantity proportional to the gradient.
6. We repeat steps 1 to 5 for the entire training set — this is one epoch. Repeat for more epochs until eventually our error is minimized or our prediction score is maximized.



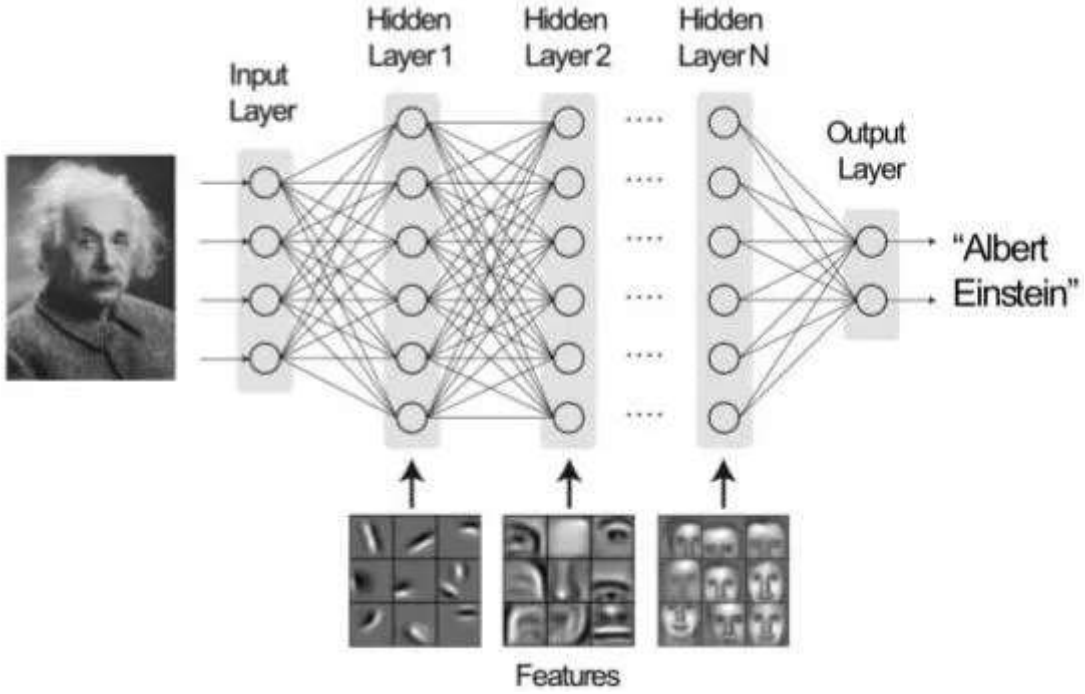
A general training and inference phase of a Deep Learning Model. Image referred from here.

INFERENCE PHASE

In this phase, no calculation of gradients or adjustment of parameters takes place. It uses the same sets of weights and biases for the evaluation on an unknown test dataset which the network hasn't seen before.

Deep neural networks are the go-to models for achieving the state of art performance when it comes to computer vision tasks. Learning **invariant** and **discriminative features** from data is the very fundamental goal for achieving good results for any computer vision task. Deep learning methods have been proven to be

quite effective for **feature learning,** the very reason being the ability of deep learning methods to learn **hierarchical** feature representations by building high-level features from low-level ones.



Deep Neural Networks learn Hierarchical Feature Representations. Source: Deep Learning, Journal of NeuroScience.

IMAGE CLASSIFICATION:

Can we use Image Classification to solve the task of Image Verification? We might be able to learn a quite robust Deep Convolutional Neural network that performs excellently on classifying all the employee images in an organization and also takes into account factors like poses, expression, and illumination.

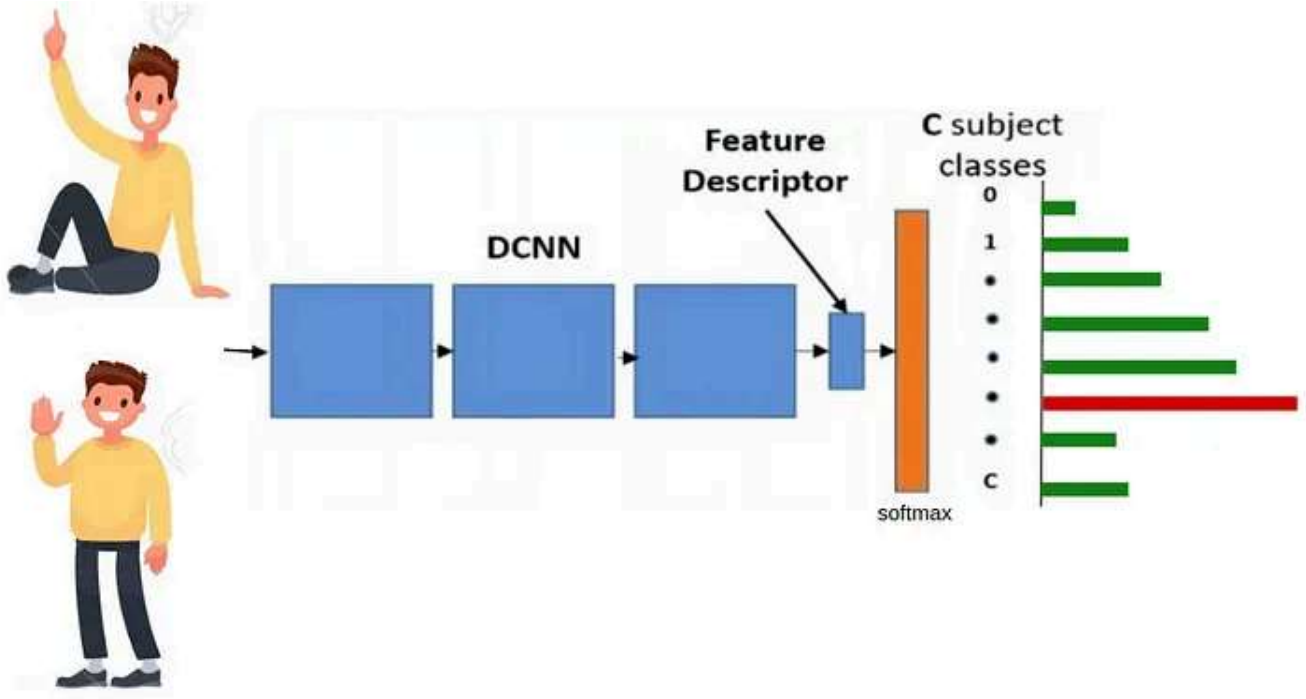


Image by Author: DCNNs for Image Classification.

But, this is usually achieved when we have a good amount of data. By good amount I mean 1000's of examples for each class/employee. In Image Classification, if the number of **data points per class** is small, it might lead to **overfitting** and yield very poor results. Also, Image Classification generally works well when the number of classes is small.

However, It is generally not the case with Person/Image verification tasks. In fact, it is quite opposite. Here, we usually have a very large number of classes and the number of examples per class is quite small. And this is where one-shot learning comes into the picture.

One-shot Learning: A classification problem that aims to learn about object categories from one/few training examples/images. [Wikipedia]. In simple words, given just one example/image of a person, you need to recognize him/her. To build a face recognition system, we need to solve this one-shot learning problem.

But deep neural nets usually require vast amounts of data to train on to excel at a particular task, which is not always available. Deep learning models won't work well with just one training example, {one-shot learning problem}. How to address this issue? We learn a **similarity function,** which helps us to solve the one-shot learning problem.

Let us start to understand deep metric learning by understanding the basics.

METRIC:

A Metric is a non-negative function between two points x and y{say **g(x,y)**} that describes the so-called notion of '**distance**' between these two points. There are several properties that a metric must satisfy:

1. **Non-negativity** => d(x,y) ≥ 0 and d(x,y) = 0, **iff x = y.**
2. **Triangular inequality** => d(x,y) ≤ d(x,z) + d(z,y).
3. **Symmetry** => g(x,y) = g(y,x).

EXAMPLES:

A. The Euclidean Metric: In a '**d**' dimensional vector space, the metric is:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}.$$

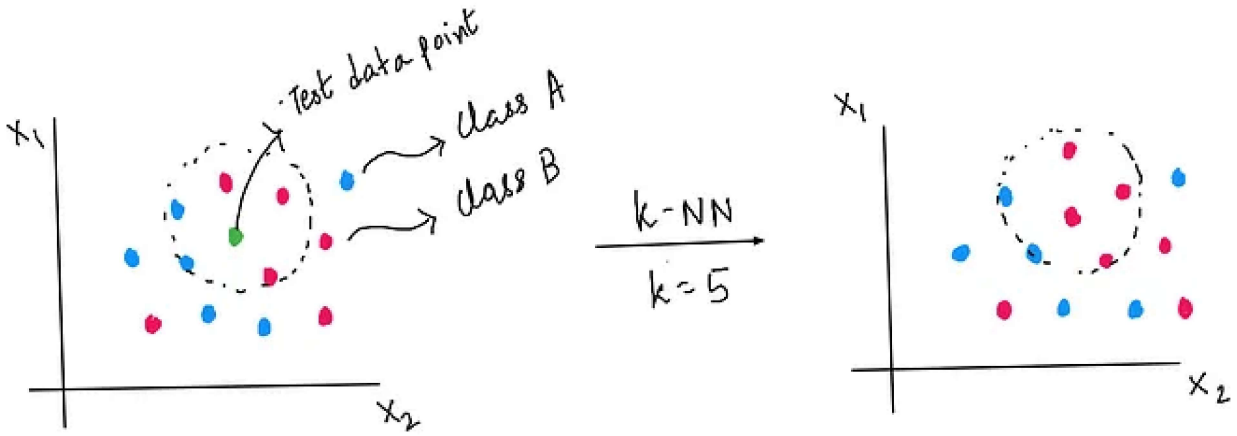
B. The Discrete Metric: The metric is given by:

$$d(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } x \neq y \\ 0 & \text{if } x = y. \end{cases}$$

METRIC LEARNING:

What does a basic machine learning algorithm do? — Given the data and the corresponding output labels, the goal is to come up with a set of **rules** or some **complex function** that maps those inputs to the corresponding output labels.

One of the simplest machine learning algorithms where **distance** information is captured is the **KNN** (k- Nearest Neighbours) algorithm, where the idea is to find a neighborhood of **k** nearest data points for a new data point and assign this data point to the class to which the majority of the **k** data points belong.



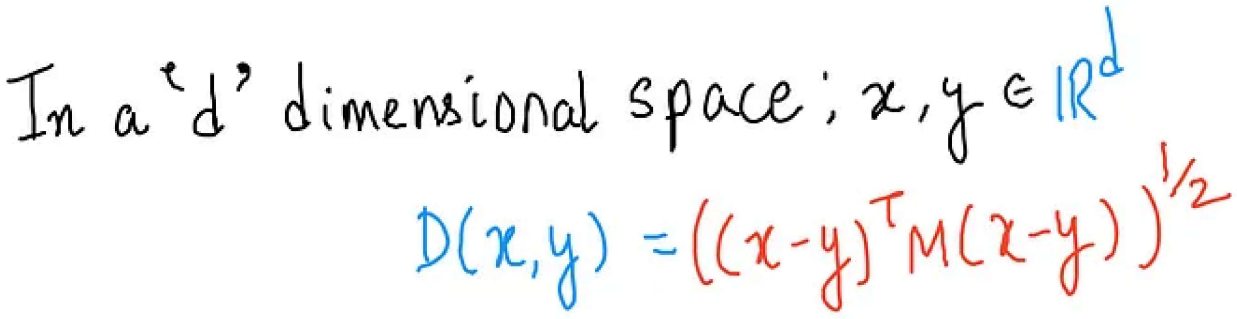
A simple explanation of k-NN

Similarly, the goal of metric learning is to learn a **similarity function** from data. Metric Learning aims to learn data embeddings/feature vectors in a way that reduces the distance between feature vectors corresponding to faces belonging to the same person and increases the distance between the feature vectors corresponding to different faces.

Euclidean distances are less meaningful in high dimensions. They fail to capture nonlinearity in data because of a number of reasons:

1. They represent an **isotropic** (same in every direction) distance metric.
2. These distances don't capture the class structure.

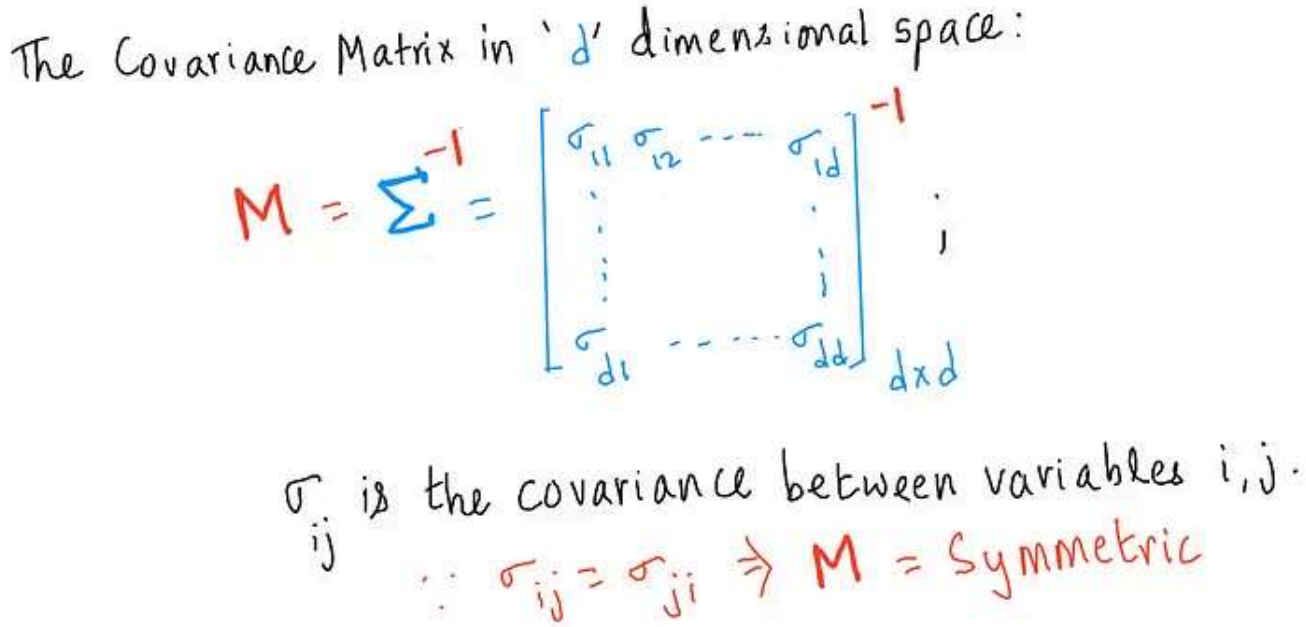
One can use non-isotropic distances that use properties of the data to capture the structure of class relationships. **Mahalanobis Distance Metric** is one such metric. The distance between two samples on the metric space is given by:



Mahalanobis Distance Metric

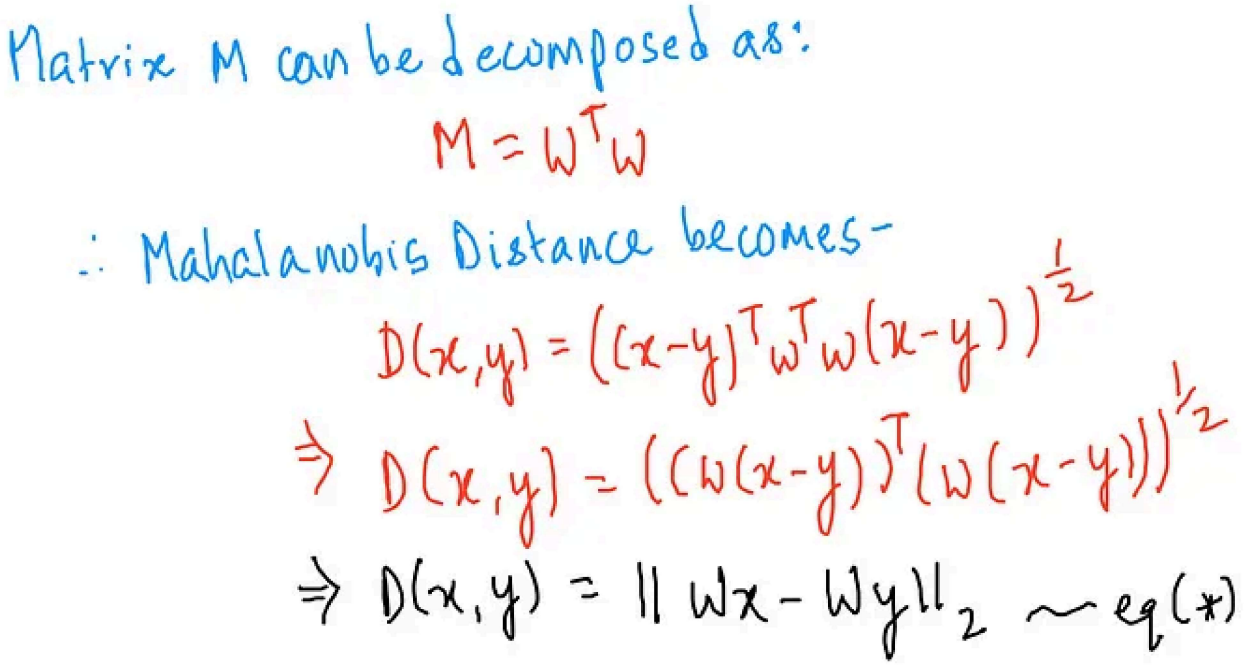
Here, M is the inverse of the covariance matrix and acts as a weight term to the square Euclidean distance. The Mahalanobis distance equation represents a

dissimilarity measure between two random vectors **x** and **y**, following the same distribution with Covariance Matrix **Σ.**



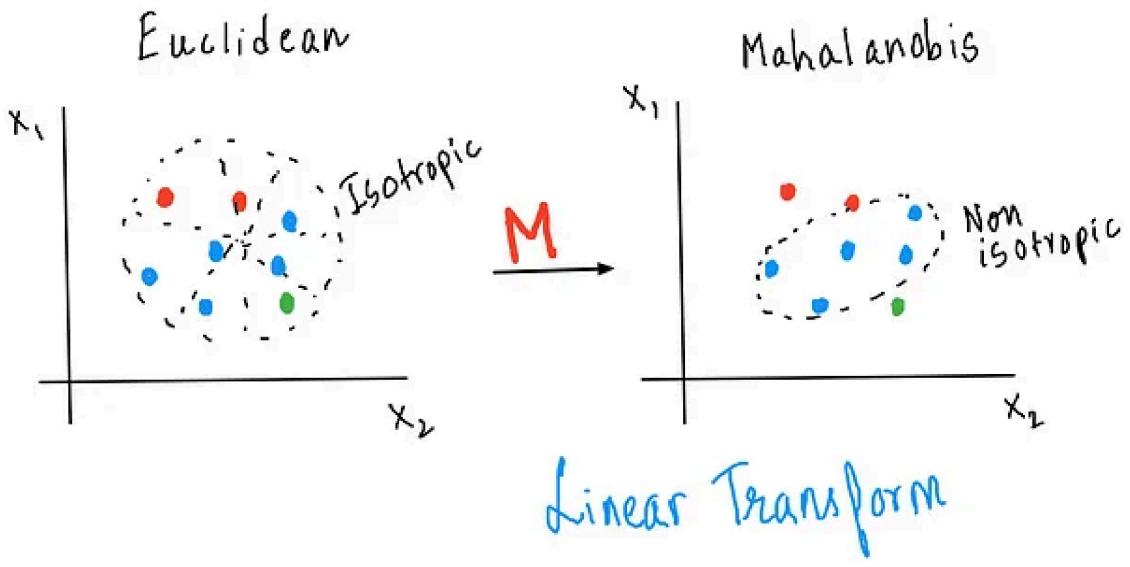
M is an inverse of the Covariance Matrix.

If M = **I**(Identity Matrix) we have the special case of Mahalanobis distance — Euclidean Distance, which proves an underlying assumption of the Euclidean Distance that the features are independent of each other.



Linear Transformation property of the Mahalanobis distance metric

The equation(*) can be interpreted as a linear projection in euclidean space. So, W has a linear transformation property because of which Euclidean distance in transformed space is equal to the Mahalanobis distance in the original space.



Isotropic Euclidean distance V/S Non-isotropic Mahalanobis distance metric

This distance metric provides a new data representation in the transformed space which is easily able to distinguish the items of different classes. This Linear transformation shows the inter-class separability power this approach posses. So, Metric Learning is basically a goal to learn this transformation matrix **W**.

Metric learning is an approach based directly on a distance metric that aims to establish similarity or dissimilarity between images. **Deep Metric Learning** on the other hand uses Neural Networks to automatically learn discriminative features from the images and then compute the metric.

Why there is a need for deep metric learning?

1. Faces of the same person when presented in different poses, expressions, illuminations, etc. might be able to fool a face verification/face recognition system very efficiently.

2. The task of building an efficient, large scale Face Verification system is radically a task of designing appropriate loss functions that best discriminate the classes under study. In recent years, Metric/Distance learning using Deep learning has been shown to output highly satisfying results for many computer vision tasks

such as face recognition, face verification, image classification, Anomaly detection, etc.

3. Metric Learning only has a limited capability to capture non-linearity in the data.

Deep Metric Learning helps capture Non-Linear feature structure by learning a non-linear transformation of the feature space.

DEEP METRIC LEARNING

There are two ways in which we can leverage deep metric learning for the task of face verification and recognition:

1. Designing appropriate loss functions for the problem.

Most widely used loss functions for deep metric learning are the contrastive loss and the triplet loss.

** Contrastive Loss — Siamese Networks:

One of the very fundamental ideas where **explicit metric learning** is performed is the siamese network model. Siamese network is a Symmetric neural network architecture consisting of two identical subnetworks that share the same sets of parameters (hence computing the same function) and learns by calculating a metric between highest level feature encodings of each subnetwork each with a distinct input.

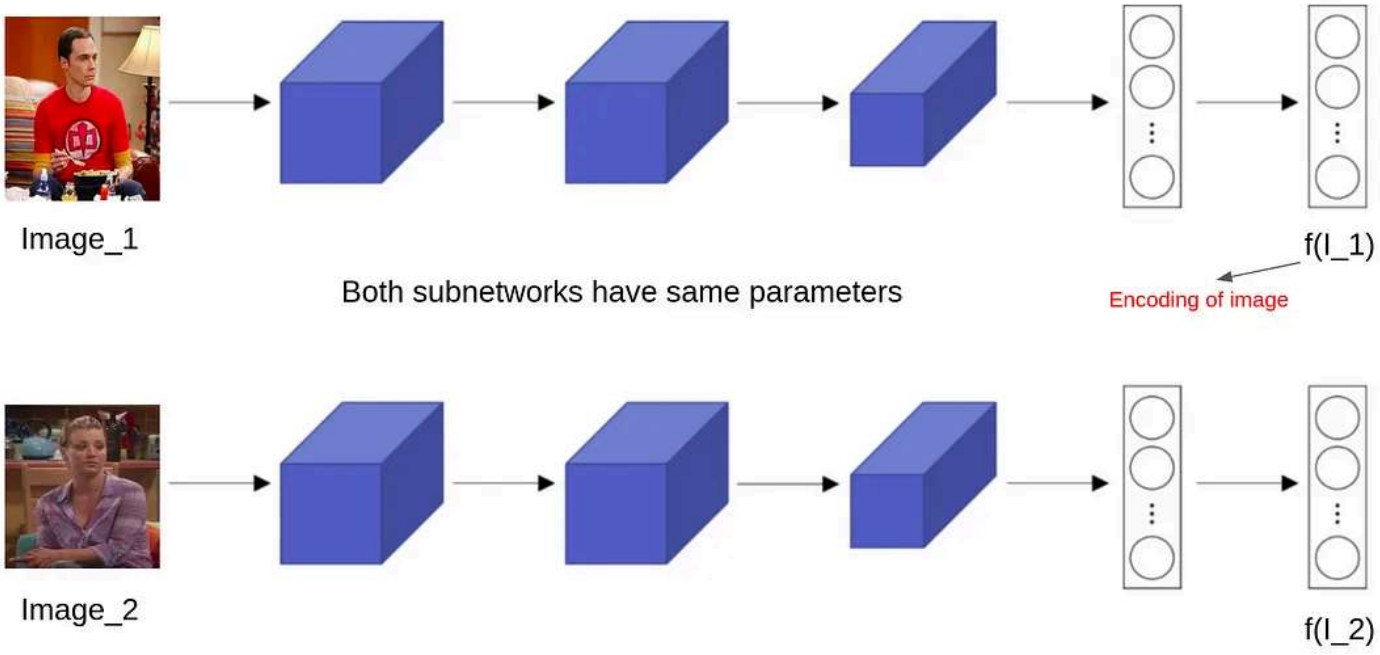
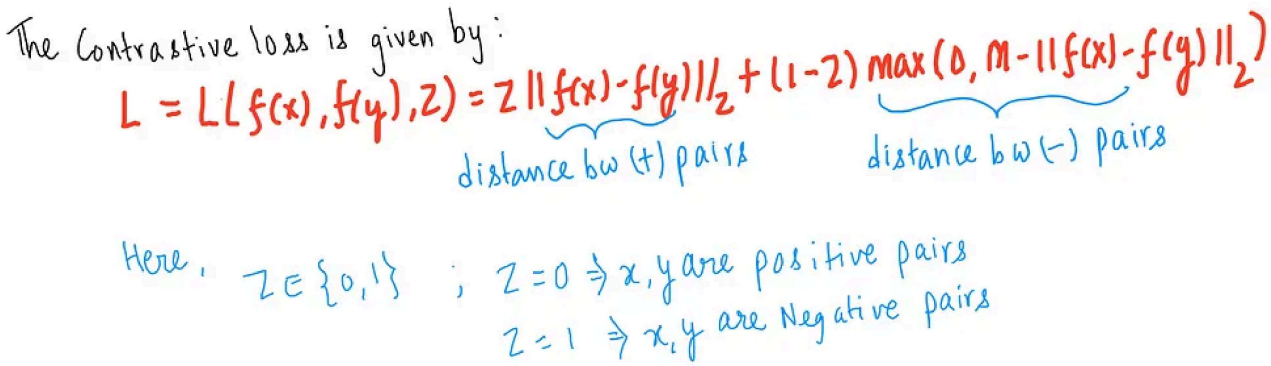


Image by Author: Subnetworks in a siamese net share the same sets of parameters

Mathematically, $\mathbf{d(x, y) = ||f(x)-f(y)||^2}$, Here, "**x**" and "**y**" represent the training examples/images. "**d(.)**" represents the metric distance between two images and "**f**" represents the encoding of the input images. The goal is to learn parameters so that:

- If **x** and **y** are the same person — d(x, y) is **small**,
- If **x** and **y** are a different person — d(x, y) is **large**.

Siamese Networks are mainly used in combination with a **Contrastive Loss function aka Pairwise ranking Loss.** The loss function is mainly used to learn embeddings (feature vectors) in a way that the metric distance between two examples from the same class is small and that between different classes is large in a metric space. Now, let's understand the maths.



Contrastive Loss

Here, m>0 is the margin, z is a boolean which is 1 if x, y are positive pairs and 0 otherwise, ||.|| is the L2 norm, other notations are the same as above.

Let's understand this equation. If x and y are actually positive pairs, **z is 1,** So, the second term vanishes (**margin m** has no effect). If the network computes a large distance between x and y, the loss will be high as the first term has a significant contribution to the loss.

Now, if x and y are actually negative pairs, **z is 0.** So, the first term vanishes. If the network computes a small distance (~0) between x and y, we will have, **L=m,** again a significant contribution to the loss.

Role of margin m: It is to be noted that the representations of negative pairs will only contribute to the loss if the estimated distance **||f(x)-f(y)|| < m.** Meaning that it

will no longer care how far the negative pairs x and y are once this limit reaches. So, it can focus more on the difficult to embed points.

Note, Siamese Network is mainly used when the number of data points per class is very small — **One-Shot Learning.**

****Triplet Loss — Triplet Networks:**

Triplet network is also a Symmetric neural network architecture but consists of three identical subnetworks that share the same sets of parameters. The learning is performed as a set of three images **1)** The **Anchor** image (The baseline image), **2)** The **positive** image (truth images of a specific person belonging to the same class as an anchor), **3)** The **negative** image (images not belonging to the anchor class).

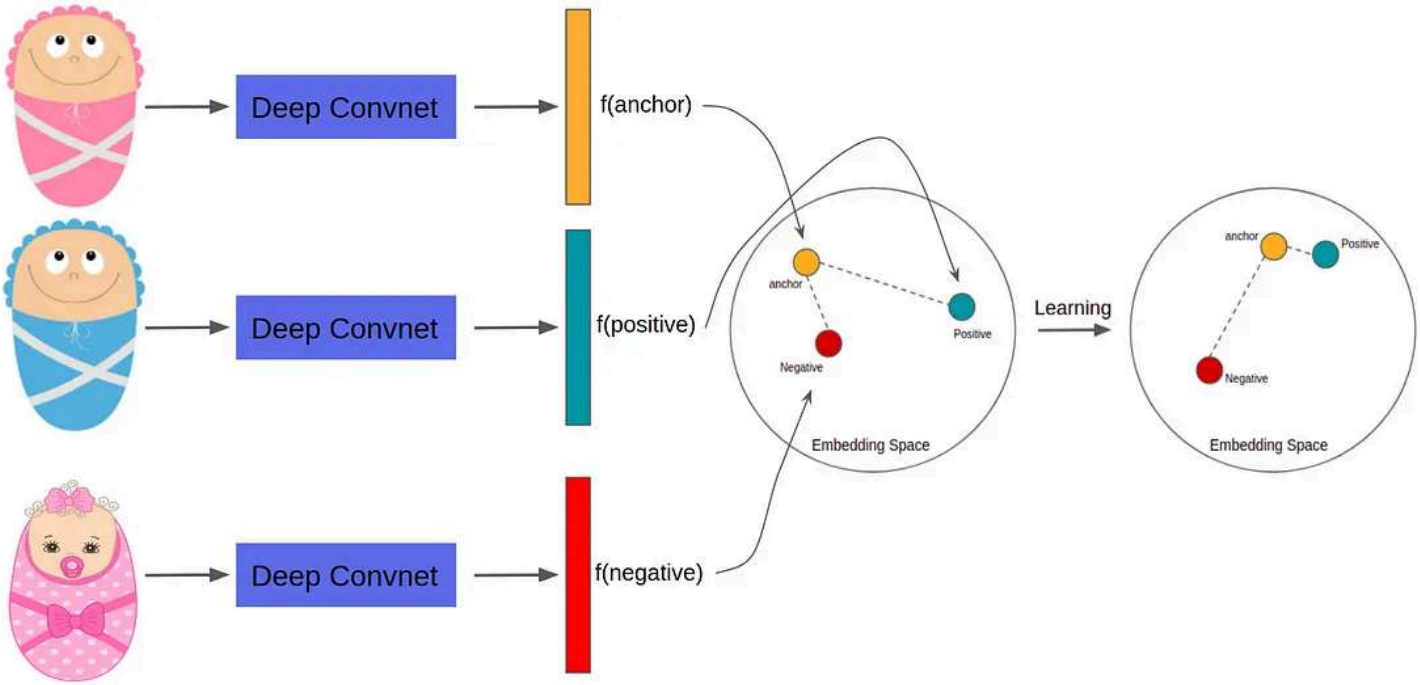


Image by Author: An intuition behind triplet loss

The goal of triplet loss is to ensure that the anchor/baseline image of a specific person is **closer** to all the positive sets of images (Mathematically, meaning that the euclidean distance between an anchor and a positive image is small) and far away from the negative sets of examples.

The Triplet Loss is given by:

$$L = L(a, p, n) = \max(0, m + \|f(a) - f(p)\| - \|f(a) - f(n)\|)$$

Triplet Loss

Here, a = anchor image, p = positive image, n = negative image, m= margin. The embedding is represented by a '**d**' dimensional vector **f(.)**{the encoding}. Let, **d(a, p)=||f(a)-f(p)||**.

Let's understand this equation by asking when the loss will be 0. The second term inside max(.) should be <0, i.e. d(a, n) > d(a, p) + m. If this happens, the triplet network will be quite lucky as there will be no learning. These triplets are known as Easy triplets.

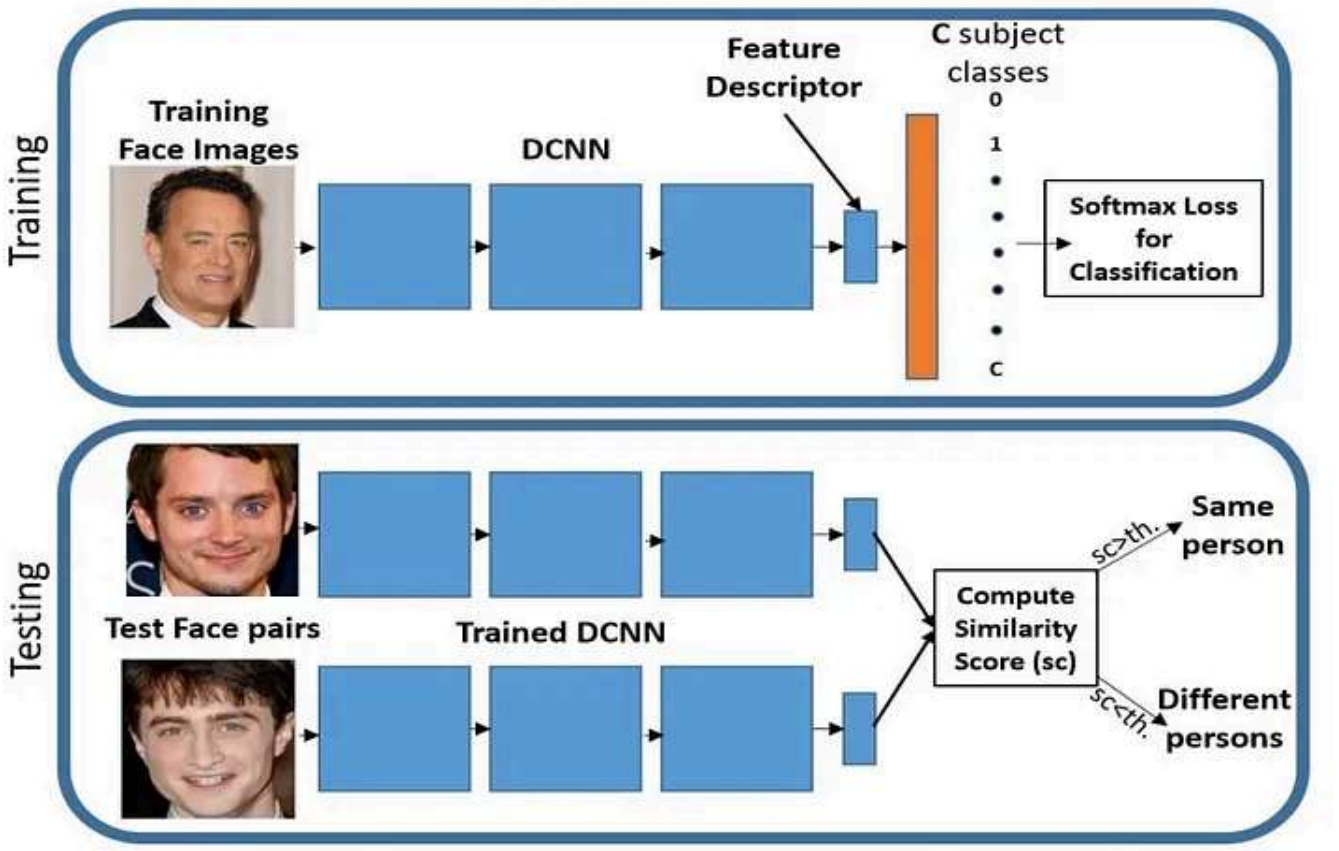
When the loss will be high? When, d(a, n) << d(a, p), which has a significant contribution to the loss. This will be a hard to train on triplet. Such triplets are known as Hard triplets. Another Case: d(a, p) < d(a, n) < d(a, p) + m, Again, the loss will be positive.

Issues with triplet loss & contrastive loss — These require a time consuming and performance-sensitive pair/triplet mining procedure to train the model.

2. Adding a custom layer to an easy to learn classification model:

One can achieve distance learning simply by learning the usual image classification problem. What we need to do is just add an additional layer on top of an easy to learn classification model.

General pipeline for training and testing a Face verification system using DCNN.



[source]: arXiv:1703.09507v3 [cs.CV] 7 Jun 2017

1. A DCNN is trained as an image classification task using the ground truth training dataset face images and the corresponding labels.
2. A Softmax loss function is used to optimize the parameters of the DCNN model.
3. At inference time, feature embeddings are extracted for the test face pairs using the trained DCNN.
4. We then compute the cosine similarity between these feature embeddings. If the score is greater than a set threshold, we say that the test pairs are the person and otherwise different.

This is a very simple face verification pipeline without the need for any metric learning.

Softmax Loss or Cross-Entropy loss:

Let's unravel some mathematics of Softmax loss so that it will be easy for us to understand the later sections. Let's be clear on notations:
C is the total number of classes,
$\mathbf{a_j}$ denotes the output of the $\mathbf{j}^{th}$ neuron of the last fully connected layer before softmax activation,

$\mathbf{p_j}$ denotes the softmax activation of the $\mathbf{j^{th}}$ neuron,
$\mathbf{x_i}$ denotes the $\mathbf{i^{th}}$ image in a training batch of size $\mathbf{M}$,
$\mathbf{t_i}$ denotes the target one-hot encoded vector of image $\mathbf{i}$,
$\mathbf{f_i}$ be an encoding of $\mathbf{x_i}$ and it is also the input of the last fully connected layer,
$\mathbf{W}$ is the weight of the last fully connected layer, (bias = 0 (let)),
$\mathbf{L_i}$ is the cross-entropy loss for image $\mathbf{i}$, we have:

DERIVATION SOFTMAX LOSS -

The output of softmax activation is given by:

$$p_j = \frac{e^{a_j}}{\sum_{k=1}^{C} e^{a_k}} \quad \{\text{softmax activation}\}.$$

The Categorical Cross Entropy loss for data point i is :-

$$L_i = -\sum_{j=1}^{C} t_i \log(p_j) \quad \{p_j \text{ contains } x_i\}.$$

$\because t_i$ is a one hot encoding vector, let the non-zero element be at position y_i. $y_i \in [1, C]$. (label).

$\therefore$ Rest all the elements are 0 in t_i, we have,

$$L_i = -\log\left(\frac{e^{a_{y_i}}}{\sum_{k=1}^{C} e^{a_k}}\right).$$

Now, $a_j = W_j^T f_i \quad \{f_i = f(x_i)\}$

overall Loss $\mathcal{L} = \frac{1}{M} \sum_i^M L_i$

$$\mathbf{L} = -\frac{1}{M} \sum_{i=1}^{M} log(\frac{e^{W_{y_i}^T f_i}}{\sum_{k=1}^{C} e^{W_k^T f_k}})$$

Derivation of softmax loss

Note, $\mathbf{f_i}$ is the learned feature vector which is also an input to the last fully connected layer, $\mathbf{a_j}$ is the output of the last fully connected layer. Since $\mathbf{W}$ is the weight of the

last fully connected layer, W_j can be interpreted as a linear classifier of class **j**.

Issues with the Softmax loss:

Though Softmax loss might help in easy convergence to the optima, unlike Contrastive loss and triplet loss, softmax loss doesn't attempt to keep the positive pairs closer and negative pairs farther. Softmax loss only learns separable features that are not discriminative enough. Due to this reason, many methods have been proposed that apply metric learning on top of the softmax feature.

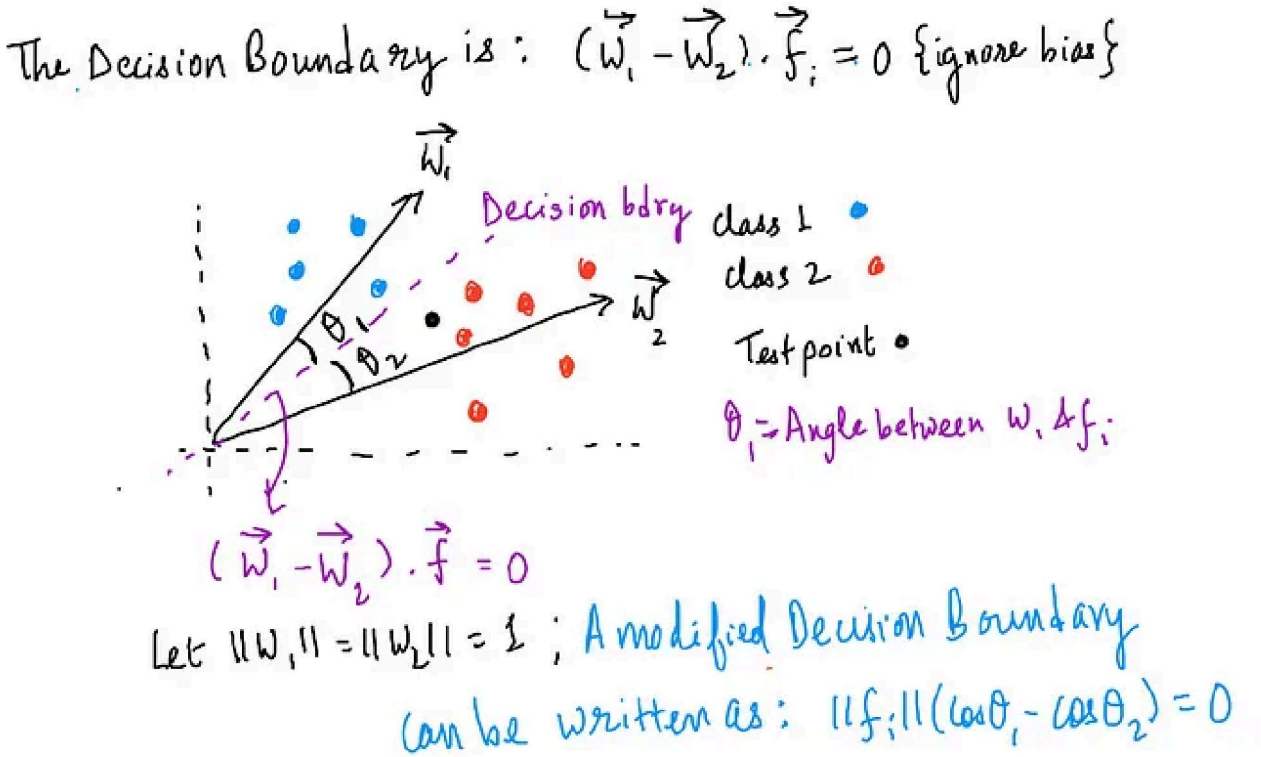
Now, the goal is to modify the existing softmax loss function in a way that we can achieve better discriminative power in the feature space.

A-Softmax loss — SPHEREFACE loss

Intuition: Consider a binary classification problem trained using softmax loss function. Let W_1 and W_2 be the linear classifiers of class **1** and **2** respectively. A data point x_i is predicted as class 1 if the feature vector f_i follows:

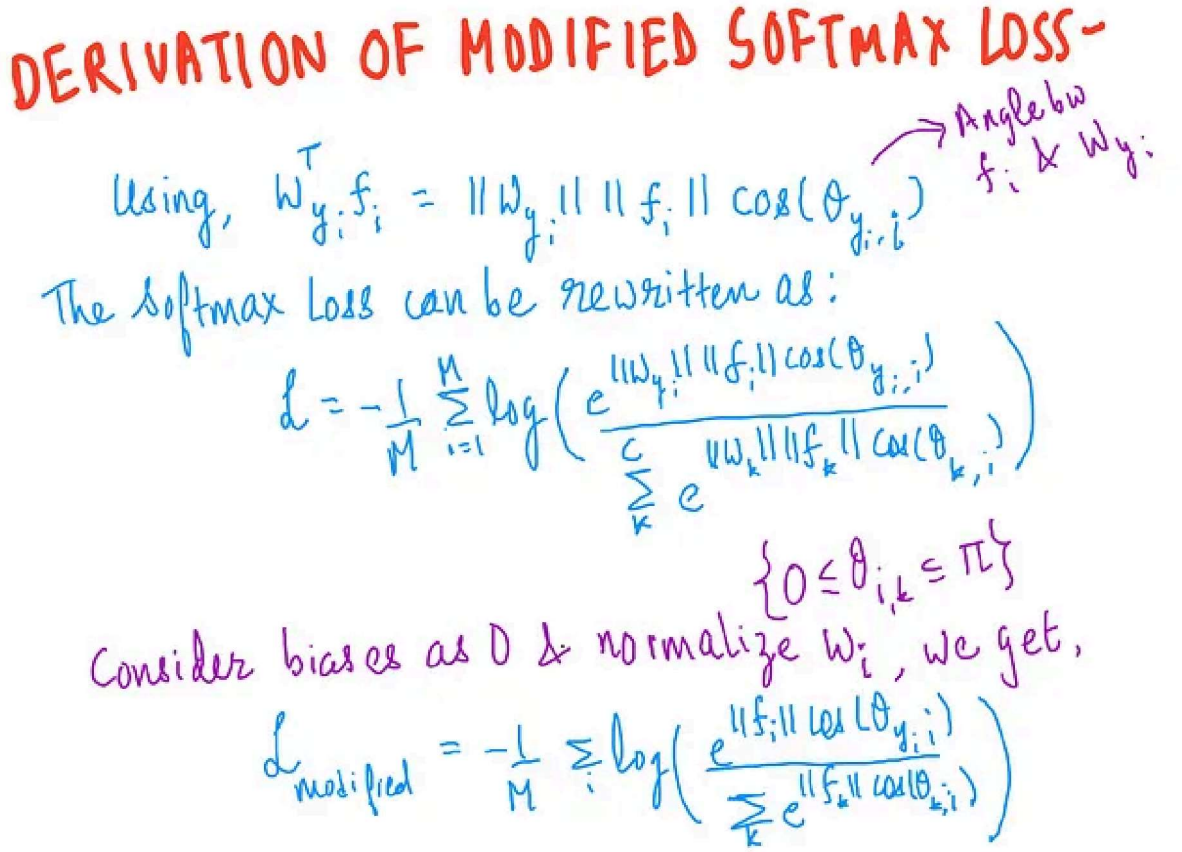
$$||W_1||||f_i||cos(\theta_1) > ||W_2||||f_i||cos(\theta_2)$$

Condition for x_i to belong to class 1



Decision boundary for modified softmax loss

So, a learned feature **f** from class 1 is correctly classified if $\cos\theta_1 > \cos\theta_2$. This is where an Angle based distance metric comes into the picture.



Derivation of Modified Softmax loss

To develop effective feature learning, the norm of W should be necessarily invariable and hence $\|W_i\|=1$. Compared to original softmax loss, the features learned by Modified softmax loss are angularly distributed, but not necessarily more discriminative.

The authors of the paper *SphereFace: Deep Hypersphere Embedding for Face Recognition* introduce a way to make the decision boundary more stringent and discriminative with the help of **Angular Margin**.

Consider the same binary classification scenario. We saw a learned feature **f** from class **1** is correctly classified if $\cos\theta_1 > \cos\theta_2$. Now, what if we instead require $\cos(m\theta_1) > \cos(\theta_2)$; where $m \geq 1$ is an integer. Clearly, this will make the decision boundary more discriminative, as the value of cosine decreases with the increasing angle and there is a lower bound for $\cos\theta_1$ to be larger than $\cos\theta_2$. Now, from an angular perspective correctly classifying f {class 1}, requires $m\theta_1 < \theta_2$ OR $\theta_1 < \theta_2/m$;

which is more difficult to learn than original $\theta_1 < \theta_2$, but, this will ensure more discriminative features. Here is a comparison between different decision boundaries.

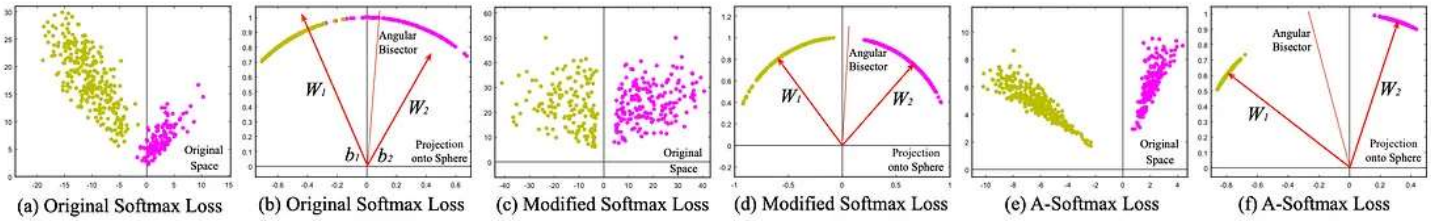
Loss Function	Decision Boundary
Softmax Loss	$(\boldsymbol{W}_1-\boldsymbol{W}_2)\,\boldsymbol{f}+b_1-b_2=0$
Modified Softmax Loss	$\|\boldsymbol{f}\|(\cos\theta_1-\cos\theta_2)=0$
A-Softmax Loss	$\|\boldsymbol{f}\|(\cos m\theta_1-\cos\theta_2)=0$ for class 1 $\|\boldsymbol{f}\|(\cos\theta_1-\cos m\theta_2)=0$ for class 2

Comparison of decision boundary in the binary case. [source].

Incorporating this idea into the modified softmax loss we have:$\{\theta_1 \in [0,\pi/m]\}$,

$$\mathbf{L_{ang}} = -\frac{1}{M}\sum_{i=1}^{M} log\left(\frac{e^{||f_i||cos(m\theta_{y_i,i})}}{e^{||f_i||cos(m\theta_{y_i,i})} + \sum_{k\neq y_i}^{C} e^{||f_k||cos(\theta_{k,i})}}\right)$$

A-Softmax loss function



Comparison between softmax loss, modified softmax loss, and A-Softmax loss. We can see that features learned by original softmax loss cannot be classified simply via angles, while modified softmax and A-softmax loss can. We can see the higher discriminative power in the case of A-Softmax loss [source]

Note, A-Softmax loss requires $||W_i||=1$, $b_i = 0$; Since, in the testing stage, the face recognition score of a testing face pair is usually calculated according to cosine similarity between the two feature vectors. This suggests that the norm of feature vector f ($||f||$)is not contributing to the scoring function and hence the prediction depends solely on the angle between W and f_i.

For any given problem, we can control the hyper-parameter m to enhance the discrimination between the learned parameters, since m controls the angular decision boundary.

Issues with A-Softmax loss:

The margin of A-Softmax loss is not consistent for all values of θ(the angle between weight vectors of 2 classes), the margin is different for different classes and so, some inter-class features have a larger margin whereas some smaller, this reduces the discriminative power of the loss function.

Large Margin Cosine Loss — COSFACE Loss

LMCL proposed in paper — CosFace: Large Margin Cosine Loss for Deep Face Recognition defines a decisive margin in cosine space rather than the angle space. Since cosine similarity is one of the most prominent similarity measure in face recognition, it is more reasonable to introduce cosine margin between different classes to improve cosine related discriminative information.

Consider the same binary classification problem. We saw a learned feature **f** from class **1** is correctly classified if $\mathbf{cos\theta_1 > cos\theta_2}$ for modified softmax loss. To develop a large margin classifier, LMCL further requires:

$$cos(\theta_1) - m \geq cos(\theta_2)$$

Since $\cos(\theta_i) - m$ is lower than $\cos(\theta_i)$, the constraint is more stringent for classification. Hence, for a large margin classification, we maximize $\mathbf{cos(\theta_1)}$ and minimize $\mathbf{cos(\theta_1)}$. Note, LMCL guides DCNN to learn features with a large cosine margin rather than a large angular margin. Unlike A-Softmax, LMCL has a fixed magnitude of the cosine margin **m**.

LMCL also requires $\mathbf{||W_i||=1}$, $\mathbf{b_i = 0}$, and since, in the testing stage, norm of the feature vector **f** is not contributing to the scoring function, the prediction depends solely on the angle between **W** and $\mathbf{f_i}$. Thus, in the training stage we fix $\mathbf{||f||= s}$ (known as the scaling parameter), which removes variations in radial directions and the resulting model learns features that are separable in the angular space. The learned embedding features are thus distributed on a hypersphere with a radius of s. The modified softmax loss becomes:

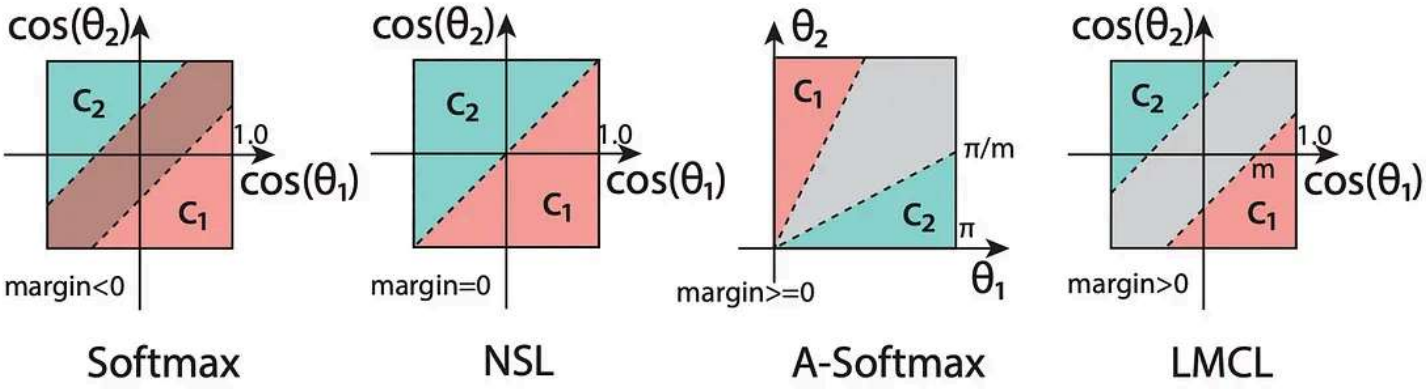
$$\mathbf{L_{modified}} = -\frac{1}{M}\sum_{i=1}^{M} log(\frac{e^{s(cos\theta_{y_i,i})}}{\sum_k e^{s.cos(\theta_{k,i})}})$$

The modified softmax loss

Finally, the Large Margin Cosine Loss (LMCL) is defined using the intuition discussed above as{notations are same as discussed above}:

$$\mathbf{L_{lmc}} = -\frac{1}{M}\sum_{i=1}^{M} log(\frac{e^{s(cos\theta_{y_i,i}-m)}}{e^{s(cos\theta_{y_i,i}-m)} + \sum_{k\neq y_i}^{C} e^{s.cos(\theta_{k,i})}})$$

The Large Margin Cosine Loss



The comparison of decision margins in cosine space for different loss functions in the binary-class scenario. NSL(Normalised Softmax loss) is the modified softmax loss. The dashed line represents the decision boundary, and the grey areas are the decision margins [source]

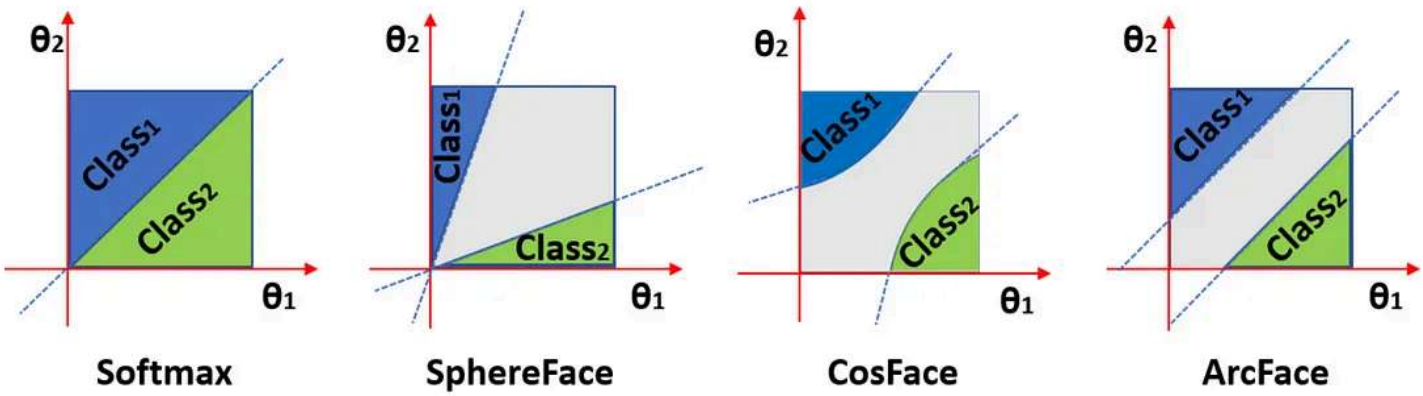
ARCFACE Loss

Unlike Cosface, where we add an **additive cosine margin** penalty, the paper ArcFace: Additive Angular Margin Loss for Deep Face Recognition proposes an **additive angular margin** penalty (Sphereface uses **multiplicative angular margin** penalty) to further improve the discriminative power of the model.

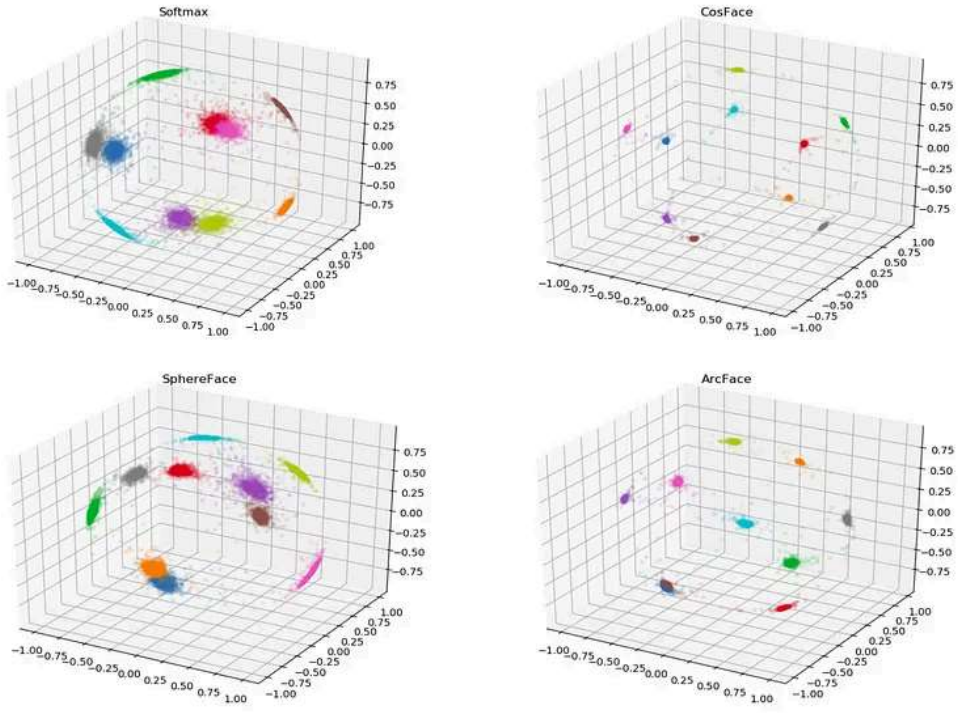
Similar to LMCL and A-Softmax, Arcface loss also requires weights to be l2-normalized and zero bias so that $||\mathbf{W_i}||=1$, $\mathbf{b_i} = 0$. We also l2-normalize the embedding feature $||\mathbf{f_i}||$ and re-scale it to **s**. The Arcface loss is given as:{notations are same as discussed above}:

$$\mathbf{L_{arcface}} = -\frac{1}{M}\sum_{i=1}^{M} log(\frac{e^{s(cos(\theta_{y_i,i}+m))}}{e^{s(cos(\theta_{y_i,i}+m))} + \sum_{k\neq y_i}^{C} e^{s.cos(\theta_{k,i})}})$$

Arcface loss function



The comparison of decision margins in angular space for different loss functions in the binary-class scenario. The dashed line represents the decision boundary, and the grey areas are the decision margins [source]



An experiment of different loss functions on the MNIST dataset [source]

All these methods Sphereface, Cosface, Arcface achieve distance learning simply by learning the usual image classification problem. Although they differ in terms of margin penalties, no matter add on the angle or cosine space, all enforce the intra-class compactness and inter-class diversity.

References

[1].https://www.researchgate.net/publication/335314481_Deep_Metric_Learning_A_Survey.

[2]. W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In CVPR, 2017.

[3]. W. Liu, Y. Wen, Z. Yu, and M. Yang. Large-margin softmax loss for convolutional neural networks. In ICML, 2016.

[4]. H. Wang, Y. Wang, Z. Zhou, X. Ji, Z. Li, D. Gong, J. Zhou, and W. Liu. Cosface: Large margin cosine loss for deep face recognition. In CVPR, 2018.

[5]. Jiankang Deng, Jia Guo, Niannan Xue, Stefanos Zafeiriou. ArcFace: Additive Angular Margin Loss for Deep Face Recognition.

[6]. Rajeev Ranjan, Carlos D. Castillo, Rama Chellappa. L2-constrained Softmax Loss for Discriminative Face Verification.

I hope you enjoyed the ride through Deep Metric Learning along with me. I would love to know the feedback of anyone reading this article. I would be happy to answer doubts/questions on any of the concepts mentioned above. Feedbacks are greatly welcomed (A clap 👏 will also be good feedback 😇). You can reach me via Linkedin.

ThankYou!

Data Science | Machine Learning | Deep Learning | Artificial Intelligence | Mathematics

Follow

Written by Aakash Agrawal

191 Followers · Writer for Towards Data Science

Data Scientist @CRED | IIT Guwahati 2020

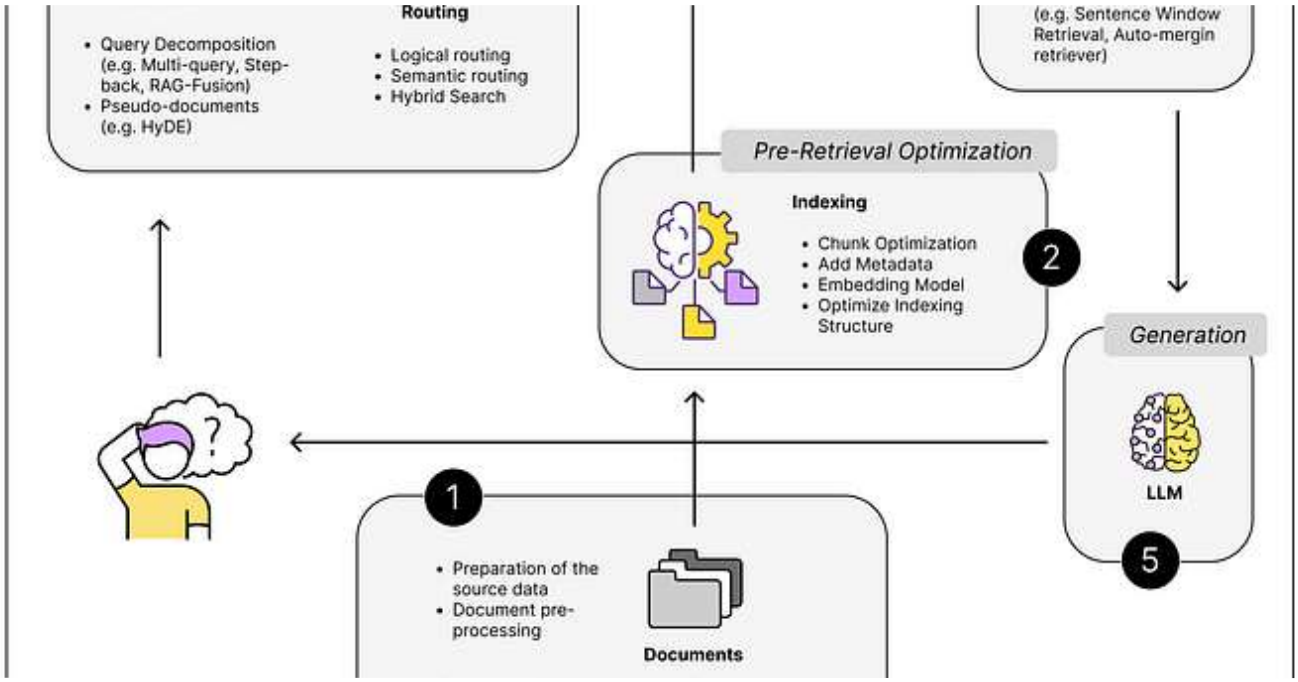
More from Aakash Agrawal and Towards Data Science

Aakash Agrawal in Towards Data Science

14 Probability problems for acing Data Science interviews

Decrypting probability questions in data science interviews with style

Jun 1, 2022 435 2

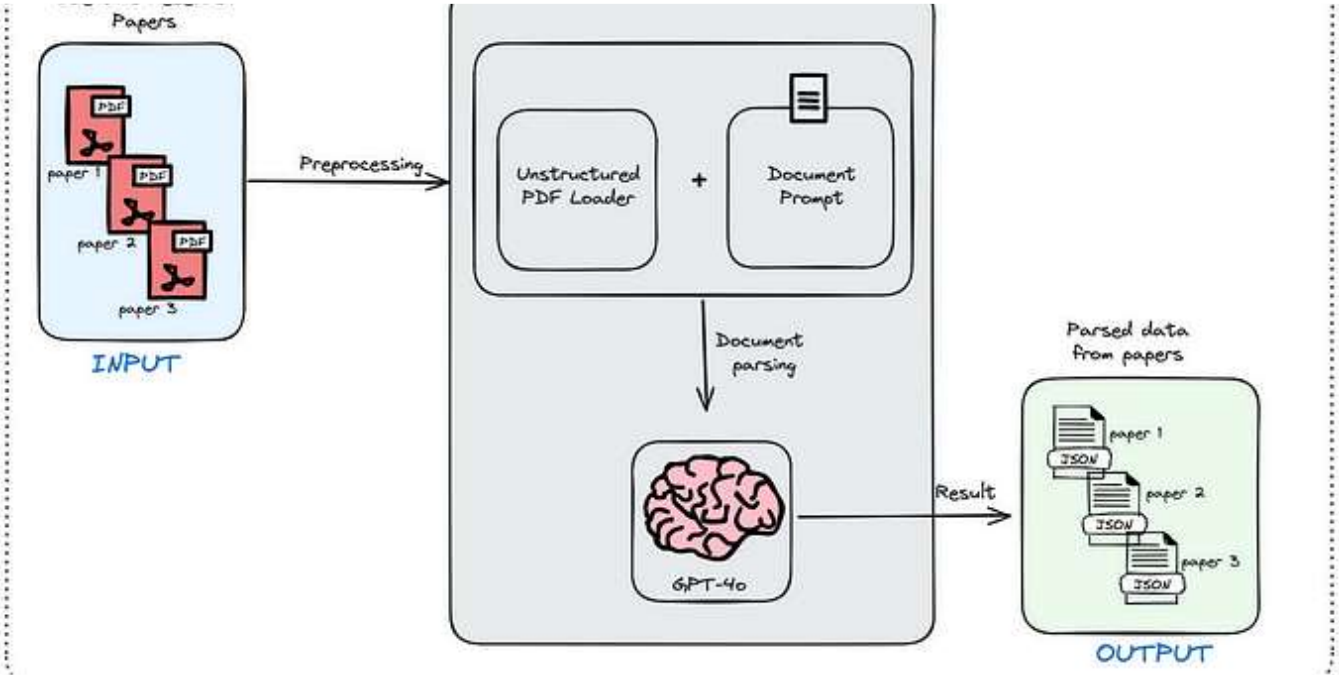


Dominik Polzer in Towards Data Science

17 (Advanced) RAG Techniques to Turn Your LLM App Prototype into a Production-Ready Solution

A collection of RAG techniques to help you develop your RAG app into something robust that will last

Jun 26 2.3K 22

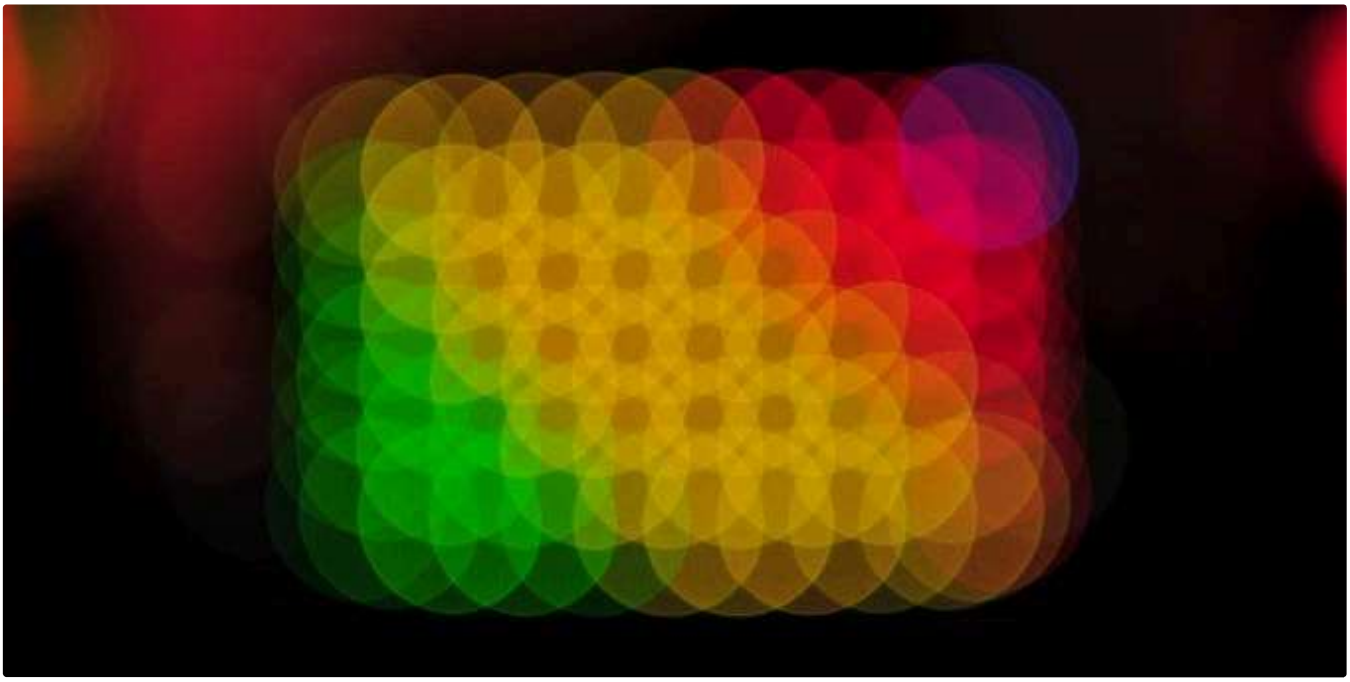


Zoumana Keita in Towards Data Science

Document Parsing Using Large Language Models—With Code

You will not think about using Regular Expressions anymore.

Jul 25 727 5



Aakash Agrawal in Towards Data Science

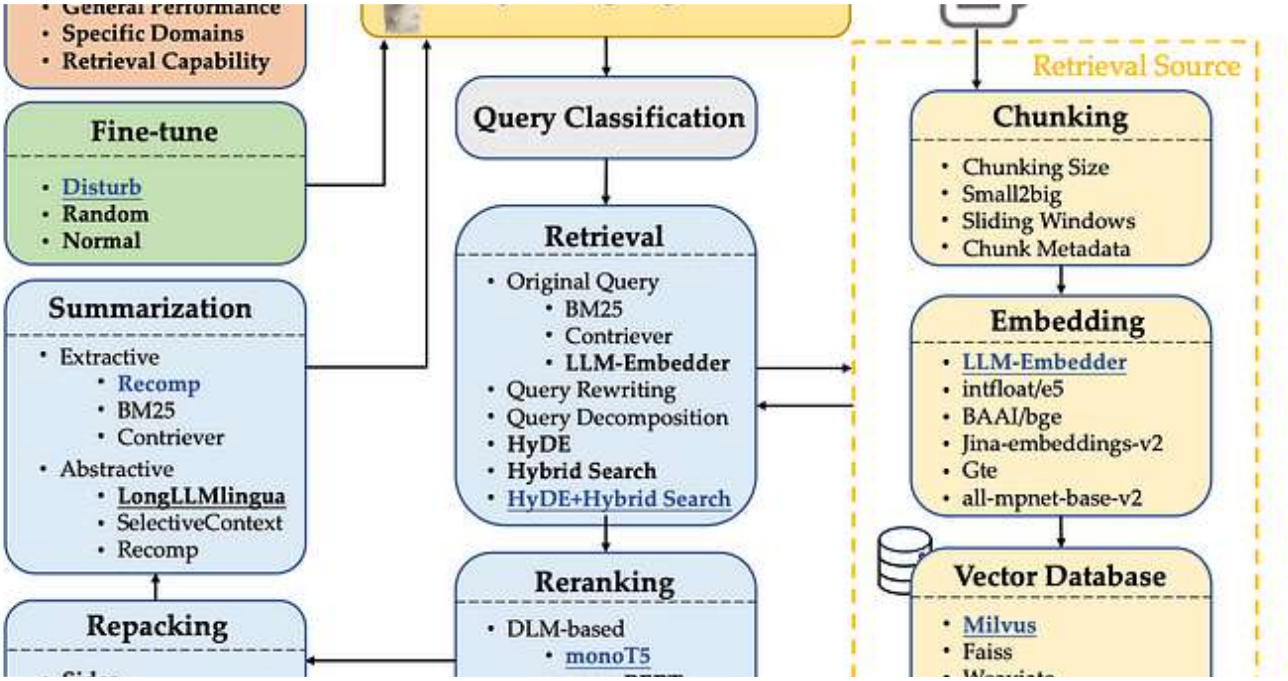
Color Swapping techniques in Image Processing

May 25, 2021 91

See all from Aakash Agrawal

See all from Towards Data Science

Recommended from Medium

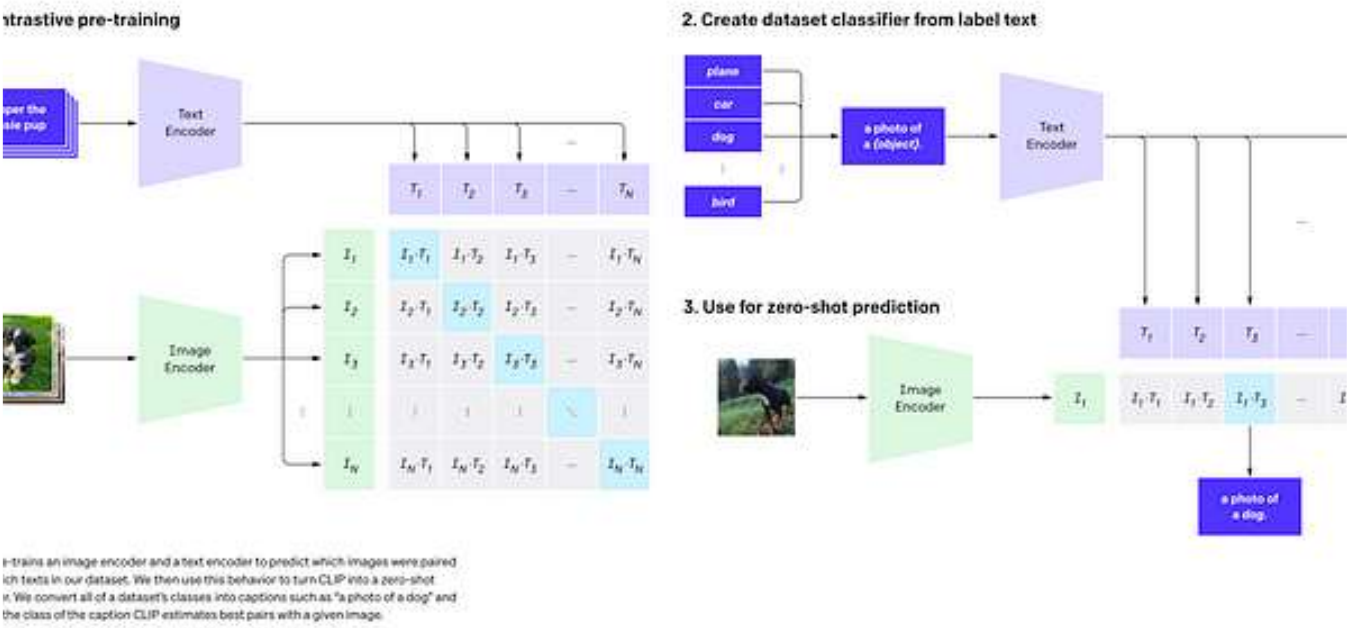


Florian June in Towards AI

The Best Practices of RAG

Typical RAG Process, Best Practices for Each Module, and Comprehensive Evaluation

Aug 8 498 2



Szymon Palucha

Understanding OpenAI's CLIP model

CLIP was released by OpenAI in 2021 and has become one of the building blocks in many multimodal AI systems that have been developed since...

Feb 24 139 2

Lists

Predictive Modeling w/ Python

20 stories · 1443 saves

Natural Language Processing

1645 stories · 1209 saves

Practical Guides to Machine Learning

10 stories · 1756 saves

data science and AI

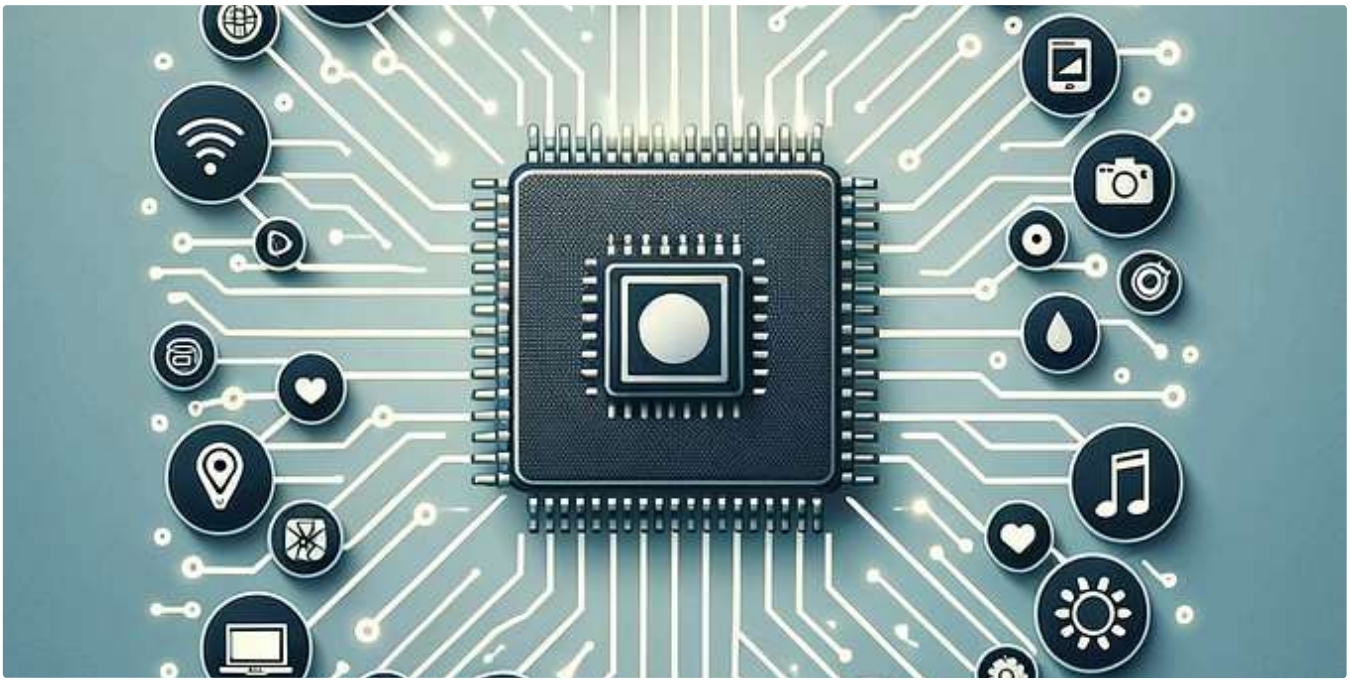
40 stories · 213 saves

Alexander Dickbauer

How to reduce the training time of an AI model by up to 2153%

Most people reading this will be AI enthusiasts without access to high-end AI GPUs like a H100. Most of you will know the struggle with...

Jul 1 89

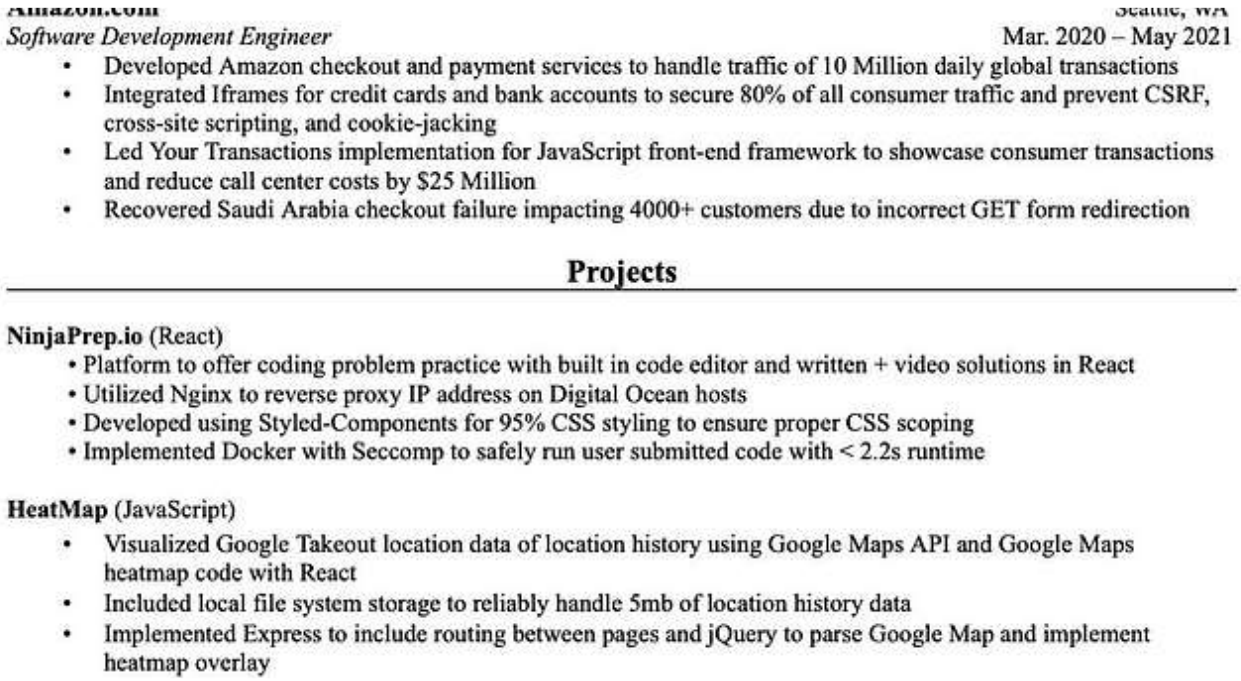


Raj Pulapakura

Multimodal Models and Fusion - A Complete Guide

A detailed guide to multimodal models and strategies to implement them

Feb 20 73 1



Software Development Engineer Mar. 2020 – May 2021

- Developed Amazon checkout and payment services to handle traffic of 10 Million daily global transactions
- Integrated Iframes for credit cards and bank accounts to secure 80% of all consumer traffic and prevent CSRF, cross-site scripting, and cookie-jacking
- Led Your Transactions implementation for JavaScript front-end framework to showcase consumer transactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 17.8K 285

```
*__, a, b, *_ = [1, 2, 3, 4, 5, 6]
print(__, _)
```

What does this print?

- A) Syntax error
- B) `[1] [4, 5, 6]`
- C) `[1, 2] [5, 6]`
- D) `[1, 2, 3] [6]`
- E) `<generator object <genexpr> at 0x1003847c0>`

Liu Zuo Lin

You’re Decent At Python If You Can Answer These 7 Questions Correctly

No cheating pls!!

Mar 6 6.9K 31