INFO 7375 - Neural Networks & AI

Home Work to Chapter - 18

Submitted By:

Abdul Haseeb Khan
NUID: 002844724
khan.abdulh@northeastern.edu

What is face verification and how does it work?

Face verification is a computer vision task that determines whether two facial images belong to the same person. Unlike face identification, which attempts to recognize a person's identity from a database, face verification focuses only on confirming if two given faces match. It is commonly used in security systems, smartphone authentication, and online identity verification.

The process begins with face detection, where the system locates and extracts the face region from an image. Techniques like MTCNN or RetinaFace are often used to identify the face while ignoring the background. Once detected, the face undergoes alignment, ensuring that features such as the eyes, nose, and mouth are consistently positioned. This step helps reduce errors caused by variations in head angle, lighting, or scale.

After alignment, the face image is passed through a deep neural network that generates a compact numerical representation known as a face embedding. This embedding captures unique facial features in a high-dimensional vector space. Popular models for this step include FaceNet, ArcFace, and VGGFace, each trained on large datasets of human faces.

Next, the system compares the embeddings of the two input faces by measuring their similarity or distance. Metrics such as cosine similarity or Euclidean distance are used to quantify how close the two vectors are. A smaller distance indicates that the faces are likely from the same person, while a larger distance suggests they are different. A decision threshold is then applied, if the distance is below this threshold, the system verifies the faces as matching.

Underlying this process is a type of deep learning architecture called a Siamese network or a triplet network, which learns to minimize the distance between embeddings of the same person and maximize it for different individuals. This training approach ensures that the model becomes highly sensitive to subtle facial differences while remaining robust to variations in expression or lighting.

In summary, face verification involves five main stages: detection, alignment, embedding generation, similarity computation, and final decision. Together, these steps enable systems to verify identity accurately and efficiently, forming the backbone of many modern facial authentication technologies.

Describe the difference between face verification and face recognition?

Face verification and face recognition are closely related but serve different purposes within facial analysis systems. The key difference lies in what each system aims to determine and how many identities it compares against.
Face verification is a one-to-one comparison task. Its goal is to determine whether two facial images belong to the same person. For example, when unlocking a smartphone

using Face ID, the system compares the live image of the user's face with the stored face template. It answers a simple binary question: *"Is this the same person?"* The process involves generating feature embeddings for both faces and computing their similarity. If the similarity exceeds a predefined threshold, the system verifies the identity as a match.
In contrast, face recognition is a one-to-many identification task. It aims to determine *who* the person in an image is by comparing the detected face against a large database of known faces. For instance, in a security or surveillance system, a captured face might be compared against thousands of stored identities to find a match. The system not only needs to extract embeddings but also classify them to the correct identity within the database. This task involves a more complex decision process and often requires larger datasets and higher computational resources.
In essence, face verification asks **"**Are these two faces the same person?", while face recognition asks "Whose face is this?". Verification is typically used for authentication and access control, whereas recognition is used for identification in applications like law enforcement, social media tagging, and public surveillance.

How do you measure similarity of images?

Measuring the similarity of images is a fundamental task in computer vision, and the method used depends on the type of images and the level of detail required. Broadly, image similarity can be measured either through pixel-level comparison or through feature-based comparison using learned representations from deep neural networks.
At the simplest level, pixel-based methods directly compare the intensity values of corresponding pixels between two images. Techniques such as Mean Squared Error (MSE) and Peak Signal-to-Noise Ratio (PSNR) fall into this category. MSE calculates the average squared difference between pixel values of the two images; smaller values indicate higher similarity. PSNR, on the other hand, expresses this difference in decibels, where a higher PSNR value means the images are more alike. These methods work well when the images are perfectly aligned and have the same size and illumination but fail when there are transformations such as rotation, translation, or scaling.
A more perceptually aligned metric is the Structural Similarity Index (SSIM), which evaluates image similarity based on three aspects of visual perception: luminance, contrast, and structural information. SSIM models the way humans perceive image quality and is therefore more robust than MSE or PSNR when dealing with small distortions or compression artifacts.
In modern computer vision applications, however, feature-based or embedding-based similarity is more common. Instead of comparing raw pixels, images are passed through a deep neural network (such as a convolutional neural network) that extracts high-level features edges, textures, shapes, and semantic information. Each image is represented as a vector (embedding) in a high-dimensional feature space. The similarity between two images is then computed using distance metrics such as cosine similarity or Euclidean distance between their embeddings. Smaller distances (or higher cosine similarity) indicate greater visual similarity.
For example, in face verification systems, the embeddings of two face images are compared using cosine similarity to determine if they belong to the same person.

Similarly, in image retrieval or visual search systems, feature-based similarity allows matching images even under variations in lighting, orientation, or background.
In summary, image similarity can be measured using:

- Pixel-based metrics (MSE, PSNR) — direct and simple but sensitive to changes.
- Perceptual metrics (SSIM) — closer to human perception.
- Feature-based metrics (cosine similarity, Euclidean distance on embeddings) — robust and widely used in deep learning-based systems.

Feature-based methods are now the preferred choice for most real-world applications, as they capture both visual content and semantic meaning beyond raw pixel comparisons.

Describe Siamese networks.

A Siamese network is a type of neural network architecture designed to learn similarity or relationship between two inputs rather than performing direct classification. It consists of two (or sometimes more) identical subnetworks that share the same architecture and weights. These subnetworks process two different inputs in parallel, and their outputs are then compared using a distance or similarity function to determine how alike the inputs are.

In this setup, each subnetwork extracts a feature representation (embedding) from its input for example, from two images, audio signals, or text samples. Because the subnetworks share weights, they learn to map inputs into the same feature space consistently. After feature extraction, the network computes the distance between the two embeddings using a metric such as Euclidean distance or cosine similarity. The goal during training is to make the distance small for similar pairs and large for dissimilar pairs.

Siamese networks are often trained using a contrastive loss function, which encourages the network to minimize the distance between embeddings of similar inputs and maximize it for dissimilar ones. In some variations, such as in face verification models, triplet loss is used instead. Triplet loss compares three inputs, an anchor, a positive example (same class), and a negative example (different class) and forces the embedding of the anchor to be closer to the positive than to the negative by a defined margin.

A key advantage of Siamese networks is that they can generalize well even with limited labeled data because they learn a distance metric rather than specific class boundaries. This makes them especially useful in tasks like face verification, signature or fingerprint matching, document comparison, and one-shot learning, where examples per class are few.

What is triplet loss and why is it needed?

Triplet loss is a loss function designed to learn embeddings where similar items are close together and dissimilar items are far apart in the embedding space. It works by training on triplets of examples: an anchor sample, a positive sample (similar to the anchor), and a negative sample (dissimilar to the anchor). The loss function encourages the network to learn representations where the distance from the anchor to the positive is smaller than the distance from the anchor to the negative by at least a specified margin.

The fundamental reason triplet loss is needed stems from the limitations of traditional classification approaches for tasks like face verification. If you trained a face verification system as a standard classification problem with softmax cross-entropy loss, you'd need one output neuron per person in your training set. This becomes impractical when dealing with millions of identities, and more critically, the model cannot generalize to new identities it hasn't seen during training.

What is neural style transfer (NST) and how does it work?

Neural style transfer is a computer vision technique that combines the content of one image with the artistic style of another image to create a new, synthesized image. The goal is to produce an output that preserves the recognizable objects and structure from the content image while adopting the textures, colors, and artistic patterns from the style image. For example, you could take a photograph of a building and render it in the style of Van Gogh's "Starry Night," maintaining the building's structure but with swirling brushstrokes and vibrant colors characteristic of Van Gogh's painting.

Describe style cost function.

Neural style transfer relies on a carefully designed cost function that balances preserving the content of one image while adopting the style of another. This cost function consists of three main components: content loss, style loss, and sometimes a total variation loss for smoothness. The optimization process minimizes this combined cost by iteratively updating the generated image's pixel values, guided by gradients computed through a pretrained convolutional neural network.

The content loss ensures that the generated image preserves the semantic content and structure of the original content image. It's computed by comparing the feature representations of the generated image and content image at specific layers of the CNN. Mathematically, for a chosen layer l with feature maps of dimension (number of filters × height × width), the content loss is defined as $L_{content} = 1/(2 * N_l * M_l) * \Sigma_{ij} (F_{ij}^l - P_{ij}^l)^2$, where F_ij^l represents the activation of the ith filter at position j in layer l for the generated image, P_ij^l is the corresponding activation for the content image, and N_l and M_l are the number of filters and the spatial size of the feature maps respectively.

The style loss is more complex and captures the artistic style independent of spatial arrangement. The key innovation is using the Gram matrix, which encodes the correlations between different filter responses within a layer. For a layer l with N_l

filters and M_l spatial positions (height × width), we first reshape the feature maps into a matrix F_l of shape (N_l × M_l). The Gram matrix is then computed as `G_l = F_l * F_l^T`, resulting in an N_l × N_l matrix where each element G_ij represents the correlation between filter i and filter j across all spatial locations.