Special Applications

➢ Face Verification and Recognition
➢ Neural Style Transfer
➢ Generative Adversarial Networks
Face Verification and Recognition

• **Face Verification** - "is this the claimed person?".

For example, at some airports, you can pass through customs by letting a system scan your passport and then verifying that you (the person carrying the passport) are the correct person. A cell phone that unlocks using your face is also using face verification. This is a **1:1 matching problem**.

• **Face Recognition** - "who is this person?".

For example, the video lecture showed a [face recognition video](https://example.com) of Baidu employees entering the office without needing to otherwise identify themselves. This is a **1:K matching problem**.
Using a CNN to recognize (classify) persons is not a promising approach because:
- you don’t have a large database with lots of images for each person
- if a new person should be recognized, the CNN should be trained again

**One-shot learning: learning from one example to recognize the person again**

**Learning a “similarity” function**

\[ d(\text{img1}, \text{img2}) = \text{degree of difference between two images} \]

**Face verification**

\[
\begin{align*}
\text{If } & d(\text{img1}, \text{img2}) \leq \text{threshold}; & \text{same person} \\
\text{If } & d(\text{img1}, \text{img2}) > \text{threshold}; & \text{different person}
\end{align*}
\]

Siamese network

By using a 128-neuron fully connected layer as its last layer, the model ensures that the output is an **encoding vector** of size 128.

\[
d(x^{(1)}, x^{(2)}) = \| f(x^{(1)}) - f(x^{(2)}) \|^2_2
\]

Parameters of CNN define the encoding \( f(x^{(i)}) \)

**Learn parameters of CNN so that:**
\[
d(x^{(1)}, x^{(2)}) \quad \text{is small for the same person}
\]
\[
d(x^{(1)}, x^{(2)}) \quad \text{is large for different persons}
\]


So, an encoding is a good one if:

- The encodings of two images of the same person are quite similar to each other.
- The encodings of two images of different persons are very different.

The triplet loss function formalizes this and tries to "push" the encodings of two images of the same person (Anchor and Positive) closer together, while "pulling" the encodings of two images of different persons (Anchor, Negative) further apart.
**Training** will use triplets of images (A,P,N):

- A is an "Anchor" image - a picture of a person.
- P is a "Positive" image - a picture of the same person as the Anchor image.
- N is a "Negative" image - a picture of a different person than the Anchor image.

These triplets are picked from our training dataset. We will write \((A^{(i)}, P^{(i)}, N^{(i)})\) to denote the \(i\)-th training example.

You'd like to make sure that an image \(A^{(i)}\) of an individual is closer to the Positive \(P^{(i)}\) than to the Negative image \(N^{(i)}\) by at least a margin \(\alpha\):

\[
\| f(A^{(i)}) - f(P^{(i)}) \|_2^2 + \alpha < \| f(A^{(i)}) - f(N^{(i)}) \|_2^2
\]

You would thus like to minimize the following "triplet cost":

\[
J = \sum_{i=1}^{m} \left[ \| f(A^{(i)}) - f(P^{(i)}) \|_2^2 - \| f(A^{(i)}) - f(N^{(i)}) \|_2^2 + \alpha \right]_+
\]

Here, we are using the notation "\([z]_+\)" to denote \(\max(z, 0)\).


Today's face recognition systems especially the commercial ones are trained on very large datasets.

Datasets of a million images is not uncommon, some companies are using 10 million images and some companies 100 million images.

These are very large datasets even by modern standards and they are not easy to acquire.

Fortunately, some of these companies have trained these large networks and posted parameters online.

So, rather than trying to train one of these networks from scratch, this is one domain where because of the share data volume sizes, it might be useful for you to download someone else's pre-trained model, rather than do everything from scratch yourself.

Face verification as a binary classification problem

\[ \hat{y} = \begin{cases} 
1 \\
0
\end{cases} \]

Training dataset:

\[ (x^{(1)}, y) \]

\[ (x^{(2)}, y) \]


Neural style transfer

Neural style transfer is an optimization technique used to take two images:

✓ a content image

▪ a style reference image (such as an artwork by a famous painter)

and blend them together so the output image looks like the content image, but “painted” in the style of the style image.

<table>
<thead>
<tr>
<th>Content</th>
<th>Style</th>
<th>Generated image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="lion1.jpg" alt="Lion" /></td>
<td><img src="lion2.jpg" alt="Lion" /></td>
<td><img src="generated_lion.jpg" alt="Generated Image" /></td>
</tr>
<tr>
<td><img src="lion3.jpg" alt="Lion" /></td>
<td><img src="lion4.jpg" alt="Lion" /></td>
<td><img src="generated_lion2.jpg" alt="Generated Image" /></td>
</tr>
<tr>
<td><img src="lion5.jpg" alt="Lion" /></td>
<td><img src="lion6.jpg" alt="Lion" /></td>
<td><img src="generated_lion3.jpg" alt="Generated Image" /></td>
</tr>
<tr>
<td><img src="lion7.jpg" alt="Lion" /></td>
<td><img src="lion8.jpg" alt="Lion" /></td>
<td><img src="generated_lion4.jpg" alt="Generated Image" /></td>
</tr>
</tbody>
</table>
Most of the algorithms you've studied optimize a cost function to get a set of parameter values. In Neural Style Transfer, you'll optimize a cost function to get pixel values!

Neural Style Transfer (NST) is one of the most fun techniques in deep learning. It merges two images, namely: a "content" image (C) and a "style" image (S), to create a "generated" image (G). The generated image G combines the "content" of the image C with the "style" of image S.

Neural Style Transfer (NST) uses a previously trained convolutional network and builds on top of that. The idea of using a network trained on a different task and applying it to a new task is called transfer learning.

The model has already been trained on the very large database, and thus has learned to recognize a variety of low-level features (at the shallower layers) and high-level features (at the deeper layers).
High level architecture of a NST model

Neural Style Transfer with Eager Execution

https://colab.research.google.com/github/tensorflow/models/blob/master/research/nst_blogpost/4_Neural_Style_Transfer_with_Eager_Exe

TensorFlow's eager execution is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models, and it reduces boilerplate as well.

In Tensorflow 2.0, eager execution is enabled by default.
Generative Adversarial Networks (GANs)


Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models:

- **the generator model** that we train to generate new examples,
- **the discriminator model** that tries to classify examples as either real (from the domain) or fake (generated).

The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.
Introduction to Generative Adversarial Networks (GANs)

Aditya Sharma
JUNE 28, 2021

https://learnopencv.com/introduction-to-generative-adversarial-networks/?ck_subscriber_id=427927332
**The Generator Model**

The generator model takes a fixed-length random vector as input and generates a sample in the domain. The vector is drawn from a Gaussian distribution, and the vector is used to seed the generative process. After training, points in this multidimensional vector space will correspond to points in the problem domain, forming a compressed representation of the data distribution.

**The Discriminator Model**

The discriminator model takes an example from the domain as input (real or generated) and predicts a binary class label of real or fake (generated). The real example comes from the training dataset. The generated examples are output by the generator model. The discriminator is a normal (and well understood) classification model.

*After the training process, the discriminator model is discarded as we are interested in the generator.*
GANs as a Two Player Game

Generative modeling is an unsupervised learning problem, although a clever property of the GAN architecture is that the training of the generative model is framed as a supervised learning problem.

The two models, the generator and discriminator, are trained together.

The generator generates a batch of samples, and these, along with real examples from the domain, are provided to the discriminator and classified as real or fake.

The discriminator is then updated to get better at discriminating real and fake samples in the next round, and importantly, the generator is updated based on how well, or not, the generated samples fooled the discriminator.
Example of the GAN Model Architecture

In this case, zero-sum means:

✓ when the discriminator successfully identifies real and fake samples, it is rewarded or no change is needed to the model parameters, whereas the generator is penalized with large updates to model parameters.

✓ when the generator fools the discriminator, it is rewarded, or no change is needed to the model parameters, but the discriminator is penalized, and its model parameters are updated.
Conditional GANs

An important extension to the GAN is in their use for **conditionally generating an output**.

The generative model can be trained to generate new examples from the input domain, where the input is provided with (conditioned by) some additional input.

The additional input could be a class value, such as male or female in the generation of photographs of people, or a digit, in the case of generating images of handwritten digits.

![Example of a Conditional Generative Adversarial Network Model Architecture](image-url)
Perhaps the most compelling application of GANs is in conditional GANs for tasks that require the generation of new examples. Jan Goodfellow indicates three main examples:

- **Image Super-Resolution**: generates high-resolution versions of input images.
- **Creating Art**: creates new and artistic images, sketches, painting, and more.
- **Image-to-Image Translation**: translate photographs across domains, such as day to night, summer to winter, and more.

GANs have been able to generate photos so realistic that humans are unable to tell that they are of objects, scenes, and people that do not exist in real life.

Example of the Progression in the Capabilities of GANs From 2014 to 2017. Taken from *The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation*, 2018.
Why Generative Adversarial Networks?

One of the many major advancements in the use of deep learning methods in domains such as computer vision is a technique called data augmentation.

Data augmentation results in better performing models, both increasing model skill and providing a regularizing effect, reducing generalization error. It works by creating new, artificial but plausible examples from the input problem domain on which the model is trained.

The techniques are primitive in the case of image data, involving crops, flips, zooms, and other simple transforms of existing images in the training dataset.

Successful generative modeling provides an alternative and potentially more domain-specific approach for data augmentation. In fact, data augmentation is a simplified version of generative modeling, although it is rarely described this way.
GAN Lab. Play with Generative Adversarial Networks (GANs) in your browser!

https://poloclub.github.io/ganlab/

Deep Convolutional Generative Adversarial Network