

[Sign In](#)[Get started](#)

Published in Analytics Vidhya



Milind Deore

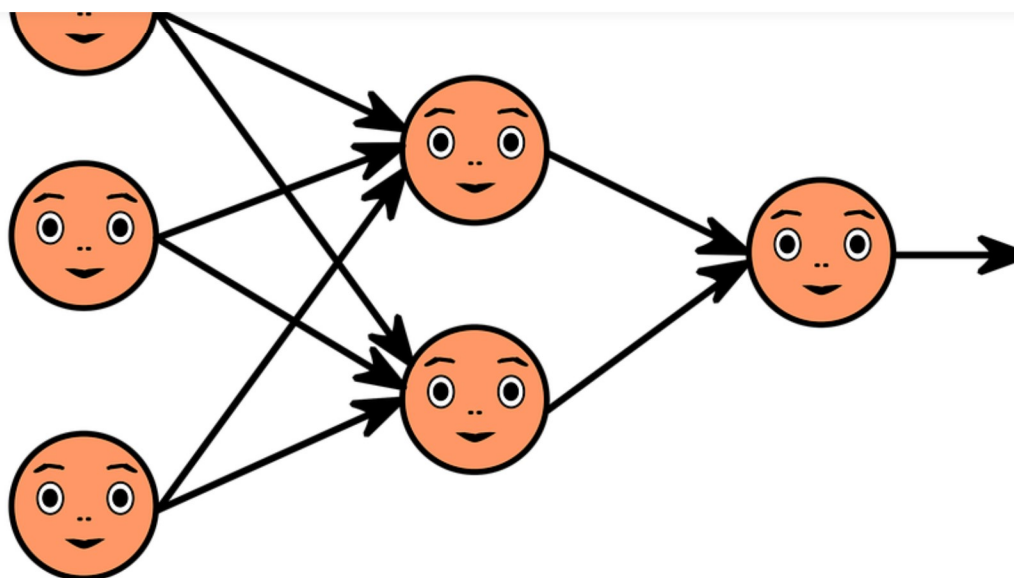
[Follow](#)Apr 4, 2019 · 7 min read · [Listen](#)

FaceNet Architecture

Part 1: Architecture and running a basic example on Google Colab

Help out the underdog! Beat the Cybercrime learn how.



[Sign In](#)[Get started](#)

The comprehension in this article comes from [FaceNet](#) and [GoogleNet](#) papers. This is a two part series, in the first part we will cover FaceNet architecture along with the example running on Google Colab and later part will cover mobile version.

FaceNet is a start-of-art face recognition, verification and clustering neural network. It is 22-layers deep neural network that directly trains its output to be a 128-dimensional embedding. The loss function used at the last layer is called triplet loss.



[Sign In](#)[Get started](#)

Fig 1: High Level Modal Structure (Source - FaceNet)

FaceNet is comprised of above building blocks and therefore we will go through each of them in sequence.

The deep net shown in Fig-1 is from GoogleNet architecture (*it has many revisions, but 'Inception-Resenet-v1' is the one that we will use in our coding example*). FaceNet paper doesn't deal much with the internal workings of GoogleNet architecture, it considers deep neural network as a black box, but we will touch base on the important concepts to know how its being used and for what purpose.

Deep Network — GoogleNet

GoogleNet is a winner of ImageNet 2014 challenge, this network has given some ground breaking results and improvements over the conventional Convolutional Neural Network (CNN). Few of these features are listed below:

- 22-layers deep network compared to 8-layers AlexNet.



[Sign In](#)[Get started](#)

- Significantly more accurate compared to AlexNet.
- Low memory usage and low power consumption.
- Network is bigger but number of parameter are smaller compared to AlexNet.
12 times less parameters compared to AlexNet.
- MUL-ADD Ops Budget was restricted to 1.5 billion (during inference), such that the architecture can be used for real world applications, specially for portable devices like: mobile phones.

In conventional CNN, convolution is done on an image with a given filter to construct a correlation statistics, layer-by-layer and then clustering these neurons that are highly correlated as an output. Important point to note is the correlation is local to the image patch and the highest correlation exist in the earlier layers of the network and hence large filter size and early pooling would reduce the important information hidden in the image patch.

This was the primary inspiration behind GoogleNet architecture and that got transformed into something called network-in-network, named as '**inception module**'.

The conventional CNN had few other challenges that GoogleNet solved quite elegantly and they are:



[Sign In](#)[Get started](#)

2. Deep network also suffer from vanishing gradient problem, because the gradient could not reach through the network till the initial layer during backpropagation cause weights unchanged and that is undesirable.
3. Linear increase in filters cause quadratic increase in operations, by which more computational power is required.
4. More number of parameter, would need more dataset and longer training time, even data-augmentation won't help much. Often, cosmetic data generating is not a suitable solution.
5. **Reduce representation bottleneck.** This can be understood as trade-off between *dimension reduction Vs information extraction*. In convolution, as we go deep in the network the dimension of the input reduces and information decay happens, therefore the information extraction should be effective with each passing layer, specially w.r.t. the local region is concern.

Solution:

1. GoogleNet use 1x1 filter for dimension reduction. The idea behind 1x1 convolution is to keep the input size (height and width) intact but shrink channels. Example: converting an 256x256x3 RGB image to 256x256x1 image.
2. Along with 1x1, other smaller but spatially spread-out filters are used like 3x3, 5x5 and 7x7. Since max-polling was successful to downsample the image, filters are applied in parallel and eventually all the intermediate



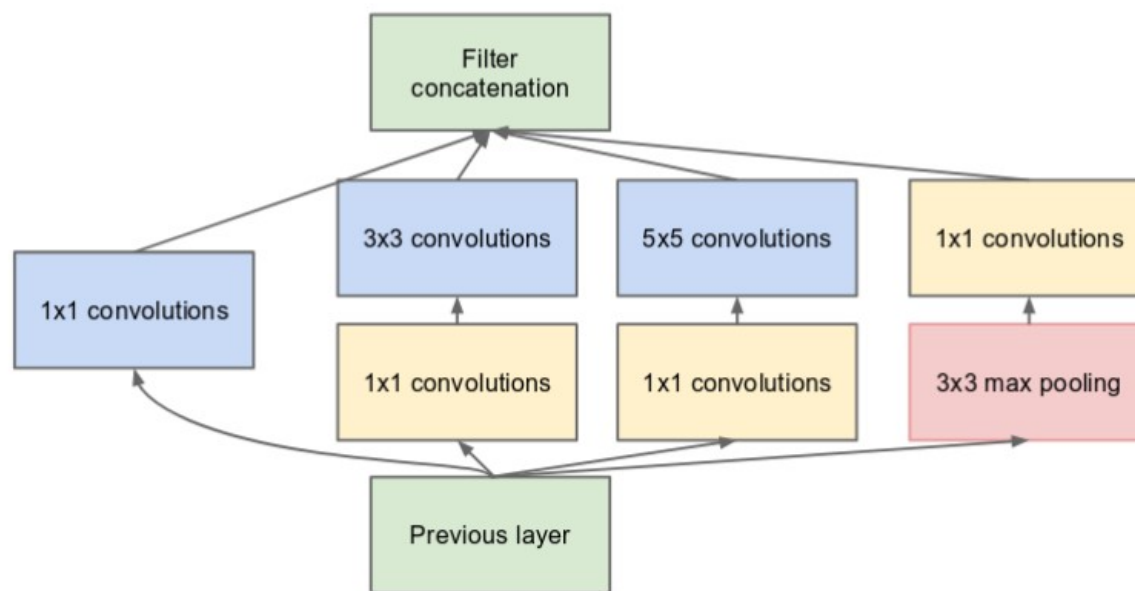
[Sign In](#)[Get started](#)

Fig 2: Inception module with dimension reductions (Source - GoogleNet)

3. Considering the depth of the network it was bound to vanishing gradient problem during back-propagation, hence two auxiliary outputs were tapped at middle layers and taken weighted average before adding it to total loss, that is:

$$\text{total_loss} = \text{final_loss} + (1/3 * \text{aux1_loss}) + (1/3 * \text{aux2_loss})$$

Since Inception v1 module has gone through various improvements, as



[Sign In](#)[Get started](#)

This version has **Factorization** which decreases the parameter and reduce the overfitting problem, **BatchNormalization** was introduced, **label smoothing** that prevent a particular logit from becoming too large compared to others hence regularizing is applied at the classifier layer.

Inception-v4 and Inception-ResNet-v1 ([paper](#))

This version simplified stem of the network (this is the preamble of the network that connects to the first inception module). The inception blocks are same as before just that they are named as A, B, C. For ResNet Version, Residual connection is introduces, replacing pooling from the inception module.

In [David Sandberg's FaceNet implementation](#), '[Inception-ResNet-v1](#)' version is being used.

During FaceNet training, deep network extracts and learns various facial features, these features are then converted directly to 128D embeddings, where same faces should have close to each other and different faces should be long apart in the embedding space (*embedding space* is nothing but *feature space*). This is just to give you intuition but implementation wise this is achieved using a loss function called Triplet Loss.



[Sign In](#)[Get started](#)

has two loss functions '**Triplet loss**' as well as '**Softmax activation with cross entropy loss**'. Triplet cost function looks as:

$$\text{Cost Function} = \sum_i^N \text{Triplet Loss Function} + \text{L2 Regularization}$$

Fig 4: Cost Function

Triplet Loss: Let us say, $f(x)$ creates embedding in d -dimensional space for an image x . Example images are:

- **Anchor** : Image of Elon Mask, that we want to compare with,
- **Positive** : Another image of Elon Mask, positive example,
- **Negative** : Image of John Travolta, negative example.





Sign In

Get started



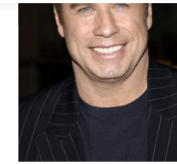
Anchor



Positive



Anchor



Negative

Fig 3: Three images, grouped.

Theoretically, Anchor image should be closer to positive image and away from negative one in the euclidean space this can be calculated as:

dist (A,P)**dist (A,N)**

$$||f(A) - f(P)||^2 + \alpha \leq ||f(A) - f(N)||^2$$

similarly,

$$||f(A) - f(P)||^2 + \alpha - ||f(A) - f(N)||^2 \leq 0 \quad \dots (1)$$

Here,

$||f(A) - f(P)||^2$ is distance between anchor and positive,

$||f(A) - f(N)||^2$ is distance between anchor and negative.





Sign In

Get started

The loss function (1) can be **zero** and in that case the equation would look like following (as we do not need value below zero):

$$L(A, P, N) = \max(|f(A) - f(P)|^2 + \alpha - |f(A) - f(N)|^2, 0) \quad \dots (2)$$

Triplet Selection: Obvious question comes to mind is to how would we choose the $f(A, P)$ and $f(A, N)$ pairs because if we select them randomly, the above equation (2) would quite easily be satisfied but our network won't learn much from it, moreover finding local minima would also be incorrect and gradient descent may converge to wrong weights.

Paper suggests, using **very hard** examples can cause convergence happening right in the beginning and may cause broken model. **Semi-hard** examples is preferred option. This can be done using reasonable mini-batch size, in the paper author used 40 face in a mini-batch.



Hence, it's good that we must pair the '**semi-hard**' examples and present it to the network. Such that:



[Sign In](#)[Get started](#)

FaceNet paper suggest two methods:

1. Offline on every n training steps: Where you compute the argmin and argmax on the latest checkpoint and apply it on the subset of the data.
2. Online: Where select a large mini-batch and computer argmin and argmax within the batch.

NOTE :- Training with triplet loss  272 |  4 come and hence David's FaceNet implementation suggest using 'Softmax with cross entropy loss', this theory comes from [paper](#).

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](#)

Your email

 [Get this newsletter](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

there.





Sign In

Get started

About Help Terms Privacy

