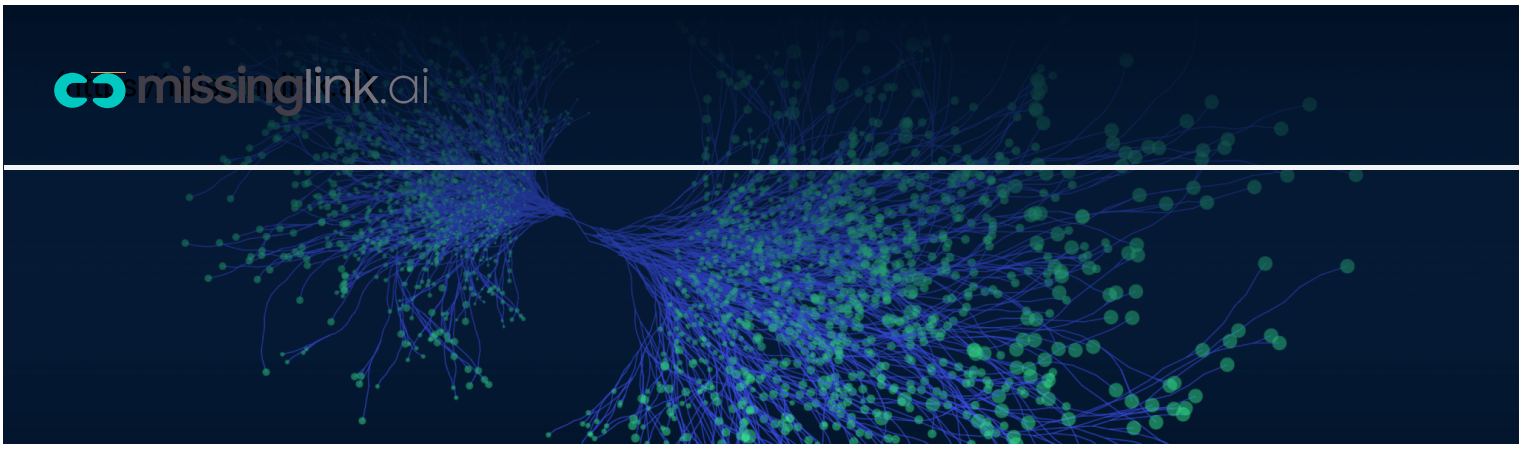


Deep Learning Frameworks

TensorFlow Face Recognition: Three Quick Tutorials

The popularity of face recognition is skyrocketing. Apple recently introduced its new iPhone X which incorporates Face ID to validate user authenticity; Baidu has done away with ID cards and is using face recognition to grant their employees entry to their offices.

In this article, we'll show you how to develop a deep learning network for facial recognition network using Tensorflow, via three community tutorials, all of which use the Google FaceNet face recognition framework. To speed up the process, you can use MissingLink's deep learning platform (https://missinglink.ai/features/) to run models on multiple machines or GPUs.

In this article you will learn

What is Facial Recognition?

Facial recognition maps the facial features of an individual and retains the data as a faceprint. The software uses deep learning algorithms to contrast an archived digital image of a person, or live capture of a person's face, to the faceprint to authenticate the identity of an individual.

How face recognition software works:

- **Initial scan**—scans the digital image or live capture until it identifies a face. Before matching a face with a name, the system checks the image is a face or may perform segmentation to identify which part of the image contains a face (see our guide on neural networks for image segmentation).

- **Face analysis**—locks on a face, analyses the features, and looks for distinguishing facial landmarks.

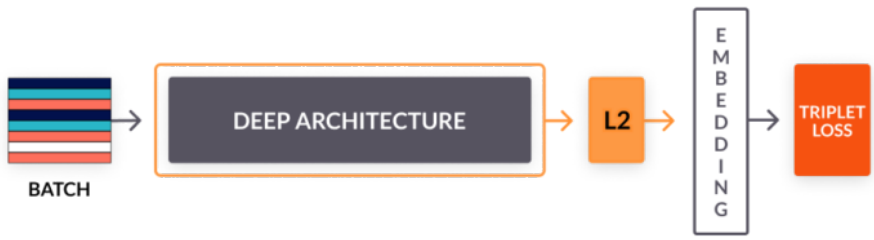
- **Data matching**—converts the data into feature vectors. The deep learning model interprets the data and finds a match, provided the face exists in the database.

Facial Recognition Applications

Industries	Application
Retail security	Retail stores can integrate facial recognition with surveillance cameras to prevent shoplifting.
Smartphone security	Companies like Apple use facial recognition as a key mechanism that allows access to the phone.
Company security	Tech-savvy companies use facial recognition systems to admit people into facilities.
Airport security	Facial recognition systems can help monitor people entering and exiting airports.
Social media	Facebook uses a face recognition algorithm to match faces in photos uploaded to the platform.

The Google FaceNet Algorithm

In this article, we present tutorials that use the Google FaceNet algorithm, a common framework for implementing modern face recognition applications. FaceNet uses a technique called “one shot learning”. Its network consists of a batch input layer and a deep Convolutional Neural Network (CNN) followed by L2 normalization (learn more about normalization in our guide to neural network hyperparameters).



The FaceNet convolutional neural network relies on image pixels as the features, rather than extracting them manually. The main idea behind the algorithm is representing a face as a 128-dimensional embedding, mapping input features to vectors.

Since these vector embeddings are represented in shared vector space, vector distance can be used to calculate the similarity between two vectors. This is a technique for calculating how similar two faces are.

The last stage in the FaceNet architecture is “triplet loss”, which minimizes the distance between an anchor and a known positive (similarity between the two faces), while maximizing the distance between the anchor and a known negative (dissimilarity).

Scaling Up Face Recognition on TensorFlow with MissingLink

If you’re working on image recognition, you probably have a large dataset of face images and need to run experiments on multiple machines and GPUs. Setting up these machines, copying data and managing experiments on an ongoing basis will become a burden.

MissingLink is a deep learning platform that lets you **effortlessly scale TensorFlow face recognition models** across hundreds of machines, whether on-premises or on AWS and Azure. It also helps manage and update your training datasets without having to manually copy files, view hyperparameters and metrics across your entire team, manage large data sets, and manage large scale experiments easily.

Learn more **to see how easy it is.**

Quick Tutorial #1: Face Recognition on Static Image Using FaceNet via Tensorflow, Dlib, and Docker

> This tutorial shows how to create a face recognition network using TensorFlow, Dlib, and Docker. The network uses FaceNet to map facial features as a vector (this is called embedding). Once facial features are encoded numerically, the network calculates the distance between two vectors and decides if two images belong to the same individual.

(https://missinglink.ai). The following steps are summarized, see the full tutorial by Cole Murray (https://hackernoon.com/building-a-facial-recognition-pipeline-with-deep-learning-in-tensorflow-66e7645015b8).

1. Environment setup – preprocessing data using Dlib and Docker

Image pre-processing addresses lighting differences, alignment, occlusion, segmentation and more. The pre-processing scripts in the Docker image address segmentation and alignment:

Segmentation isolates the largest face in an image, to eliminate the need to crop the training data.

Alignment standardizes the input and centers all images on the face.

Here is the Dockerfile provided as part of this tutorial, which you can use to run a Docker container to process the images for analysis.

```
# Project Structure
├── Dockerfile
├── etc
│   ├── 20170511-185253
│   │   ├── 20170511-185253.pb
├── data
├── medium_facenet_tutorial
│   ├── align_dlib.py
│   ├── download_and_extract_model.py
│   ├── __init__.py
│   ├── lfw_input.py
│   ├── preprocess.py
│   ├── shape_predictor_68_face_landmarks.dat
│   └── train_classifier.py
├── requirements.txt
```

Download the LFW (Labeled Faces in the Wild) dataset using this command:

```
$ curl -O http://vis-www.cs.umass.edu/lfw/lfw.tgz$
tar -xzvf lfw.tgz
```

You can use any face dataset as training data. Structure it like the LFW dataset:

```
├── Tyra_Banks
│   ├── Tyra_Banks_0001.jpg
│   └── Tyra_Banks_0002.jpg
├── Tyron_Garner
│   ├── Tyron_Garner_0001.jpg
│   └── Tyron_Garner_0002.jpg
```

Use Docker to install TensorFlow, Dlib, and OpenCV. Dlib has a library which enables alignment and facial detection.

To install Docker, run:

```
curl https://get.docker.com | sh
```

Create the image by running the following:

```
$ docker build -t colemurray/medium-facenet-tutorial -f Docke
rfile
```

2. Detect, crop and align with Dlib

Begin preprocessing by downloading Dlib's face landmark predictor.

```
$ curl -O http://dlib.net/files/shape_predictor_68_face_landm
arks.dat.bz2
$ bzip2 -d shape_predictor_68_face_landmarks.dat.bz2
```

Use this predictor to locate the inner eyes and bottom lips of a face. These coordinates will be used to center the image.

```
align_dlib.py
```

This file enables face detection, finding facial landmarks and alignment. It reads each image into memory, center aligns, tries to find the largest face, and writes the file to output. If a face cannot be identified, the console will log an error with the filename.

Run the preprocessor in the environment to use the installed libraries. Set the project directory as a volume inside the docker container, and run the preprocessing script on your input data.

```
$ docker run -v $PWD:/medium-facenet-tutorial \
-e PYTHONPATH=$PYTHONPATH:/medium-facenet-tutorial \
-it colemurray/medium-facenet-tutorial python3 /medium-facene
t-tutorial/medium_facenet_tutorial/preprocess.py \
--input-dir /medium-facenet-tutorial/data \
--output-dir /medium-facenet-tutorial/output/intermediate \
--crop-dim 180
```

The results will be written to the directory you specify in the command line arguments.

3. Create embeddings in Tensorflow

Generate vector embeddings of each identity, used as input to a classification, clustering, or regression task.

a. Download weights

Use Inception ResNet V1 and create a file to download the weights to the model.

Pre-trained weights let you apply transfer learning to a dataset (here the LFW dataset):$

```
$ docker run -v $PWD:/medium-facenet-tutorial \
-e PYTHONPATH=$PYTHONPATH:/medium-facenet-tutorial \
-it colemurray/medium-facenet-tutorial python3 /medium-facene
t-tutorial/medium_facenet_tutorial/download_and_extract_mode
l.py \
--model-dir /medium-facenet-tutorial/etc
```

b. Load embeddings

Use TensorFlow's Queue API (https://www.tensorflow.org/api_guides/python/threading_and_queues) to load the preprocessed images in parallel using multi-threading. When you use a GPU, image preprocessing can be conducted on CPU, while matrix multiplication is conducted on GPU.

4. Evaluate the results

Feed in the images the classifier has not trained on. Remove the `is_train` flag from the previous command and check your results. You will see the results for each image on the console.

Quick Tutorial #2: Face Recognition via the Facenet Network and a Webcam, with Implementation Using Keras and Tensorflow

This tutorial uses Keras with a Tensorflow backend to implement a FaceNet model that can process a live feed from a webcam.

It uses the following utility files created by deeplearning.ai (the files can be found here (https://github.com/JudasDie/deeplearning.ai/tree/master/Convolutional%20Neural%20Netw

py with functions to feed images to the network and get image encoding

py with functions to prepare and compile the FaceNet network

The following steps are summarized, for full instructions and code see Sigurður Skúli (https://medium.freecodecamp.org/making-your-own-face-recognition-system-29a8e728107c).

1. Compiling the FaceNet network

Compile FaceNet as follows. Start with these imports:

```
import os
import glob
import numpy as np
import cv2
import tensorflow as tf
from fr_utils import *
from inception_blocks_v2 import *
from keras import backend as K
```

We initialize the network with an input shape of (3, 96, 96). The Red-Green-Blue (RGB) channels are the initial components of the image volume fed to the network. All these images should be 96×96 pixels.

```
K.set_image_data_format('channels_first')
FRmodel = faceRecoModel(input_shape=(3, 96, 96))
```

Now define the Triplet Loss function:

```
def triplet_loss(y_true, y_pred, alpha = 0.3):
    anchor, positive, negative = y_pred[0], y_pred[1], y_pred
[2]

    pos_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,
               positive)), axis=-1)
    neg_dist = tf.reduce_sum(tf.square(tf.subtract(anchor,
               negative)), axis=-1)
    basic_loss = tf.add(tf.subtract(pos_dist, neg_dist), alph
a)
    loss = tf.reduce_sum(tf.maximum(basic_loss, 0.0))

    return loss
```

Use the Keras Adam optimizer (https://keras.io/optimizers/#adam) to minimize the loss calculated by the Triplet Loss function:

```
FRmodel.compile(optimizer = 'adam', loss = triplet_loss, metr
ics = ['accuracy'])
load_weights_from_FaceNet(FRmodel)
```

2. Creating a face image database

Prepare a database of face images, as follows:

```
def prepare_database():
    database = {}
```

Convert the image data, for each image, to an encoding of 128 float numbers. We do this by calling the function img_path_to_encoding.

```
for file in glob.glob("images/*"):
        identity = os.path.splitext(os.path.basename(file))
[0]
        database[identity] = img_path_to_encoding(file, FRmod
el)

    return database
```

The function takes in a path to an image and inputs the image to the network. It provides the output from the network, which is the encoding of the image.

3. Recognizing a face

FaceNet is trained to minimize the distance between the images of the same person and to maximize the distances between images of different people. The implementation applies this information and checks which person the new face probably belongs to.

```
def who_is_it(image, database, model):
    encoding = img_to_encoding(image, model)

    min_dist = 100
    identity = None

    # Loop over the database dictionary's names and encoding
s.
    for (name, db_enc) in database.items():
        dist = np.linalg.norm(db_enc - encoding)
        print('distance for %s is %s' %(name, dist))
        if dist < min_dist:
            min_dist = dist
            identity = name

    if min_dist > 0.52:
        return None
    else:
        return identity
```

The function above feeds the new image into a utility function called img_to_encoding. The function uses FaceNet to process the image and provides the encoding of the image. To identify the person, we calculate the distance between our recent image and the people in the database. The individual with the lowest distance to the new image is selected as the most probable match.

We must then see if the probable match is an actual match., i.e. if the new image includes the same as the candidate image, as follows:

```
if min_dist > 0.52:
    return None
else:
    return identity
```

If the distance is more than 0.52, we conclude that the individual in the new image does not exist in our database. However, if the distance equals or is less than 0.52, then we conclude that they are the same person, and there is a match!

The best value (here 0.52) may vary. It may be much lower or slightly higher depending on your implementation and data. We suggest trying several values and seeing what value best fits your system.

Quick Tutorial #3: Face Recognition Tensorflow Tutorial with Less Than 10 Lines of Code

Let's implement Facenet with Tensorflow, via the framework created by Tirmidzi Aflahi (https://thedatamage.com/face-recognition-tensorflow-tutorial/).

1. Install the Easy Facenet Interface

Install the library created by Tirmidzi:

```
pip install easyfacenet
```

2. Import files

Import the FaceNet file:

```
from easyfacenet.simple import facenet
```

Define images:

```
images = ['images/image1.jpg', 'images/image2.jpg', 'images/i
mage3.jpg']
```

3. Face recognition steps

a. Alignment

Use:

```
aligned = facenet.align_face(images)
```

The library will attempt to isolate the face in the image, crop, and pre-whiten the face. Pre-whitening will make it easier to train the system. At inference time, you will also need to pre-whiten the image.

b. Embeddings

(https://missinglink.ai)

The compare methods have embedding inside, however, if you wish to use embedding separately use:

```
embeddings = facenet.embedding(aligned)
```

c. Comparison

```
comparisons = facenet.compare(aligned)
```

If you are comparing 3 images, the comparison variable will have 3×3 values, which are three variation of each images compared to each other.

For example, to compare image 1 to image 2 use:

```
print("Is image 1 and 2 similar? ", bool(comparisons[0][1]))
```

To compare image 1 to image 3 use:

```
print("Is image 1 and 3 similar? ", bool(comparisons[0][2]))
```

The comparison technique used here is cosine similarity. You can use any similarity method, including clustering or classification.

TensorFlow Face Recognition in the Real World

In this article, we provided three tutorials that illustrate how to perform face recognition with Google FaceNet in TensorFlow. When you start working on real-life face recognition projects, you'll run into some practical challenges:

Tracking experiments

Each experiment you run may have its own source code, hyperparameters and configuration. To achieve good results, you'll need to run hundreds or thousands of experiments, and tracking parameters and configuration for each experiment is challenging. Not to mention that you won't have visibility across your entire team.

Scaling up experiments

Google FaceNet and other face recognition models require fast GPUs to run. To run production scale models you'll need to distribute experiments across multiple GPUs and machines, either on-premises or in the cloud. Provisioning these machines, setting them up and running experiments on each one can be very time consuming.

Manage training data

Face recognition typically involves large datasets. Managing large quantities of images, copying them to each training machine, then re-copying them when you modify your dataset or incorporate new training images, wastes precious time that could be spent building your face recognition model.

MissingLink is a deep learning platform that does all of this for you and lets you concentrate on building the most accurate model. Learn more to see how easy it is.

Learn More About Deep Learning Frameworks

TensorFlow Image Recognition with Object Detection API: Tutorials (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-image-recognition-object-detection-api-two-quick-tutorials/)

TensorFlow Image Classification: Three Quick Tutorials (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-image-classification/)

TensorFlow Distributed Training: Introduction and Tutorials (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-distributed-training-introduction-tutorials/)

TensorFlow Image Segmentation: Two Quick Tutorials (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-image-segmentation-two-quick-tutorials/)

Building Convolutional Neural Networks on TensorFlow: Three Examples (https://missinglink.ai/guides/deep-learning-frameworks/building-convolutional-neural-networks-tensorflow-three-examples/)

TensorFlow Conv2D Layers: A Practical Guide (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-conv2d-layers-practical-guide/)

TensorFlow ResNet: Building, Training and Scaling Residual Networks on TensorFlow (https://missinglink.ai/guides/deep-learning-frameworks/tensorflow-resnet-building-training-scaling-residual-networks-tensorflow/)

Building Faster R-CNN on TensorFlow: Introduction and Examples (https://missinglink.ai/guides/deep-learning-frameworks/building-faster-r-cnn-on-tensorflow-introduction-and-examples/)

(https://missinglink.ai)

(https://www.facebook.com/MissingLink-243295859743111/) (https://twitter.com/missinglinkai) (https://www.linkedin.com/company/missinglinkai/)

Product

Blog (/blog/)

Hyperparameters (/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/)

Neural Network Bias (/guides/neural-network-concepts/neural-network-bias-bias-neuron-overfitting-underfitting/)

Company

About us (/about/)

Join the team (/jobs/)

Contact us (/contact-us/)

Legal

Privacy policy (/privacy/)

Terms of service (/terms/)

Terms of Service (/terms/)

(https://missinglink.ai)