110-2-one-shot-facial-2020-7-27.pptx

CTI One Corporation

Date: July 27, 2020
Project Lead: Harry Li, Ph.D.

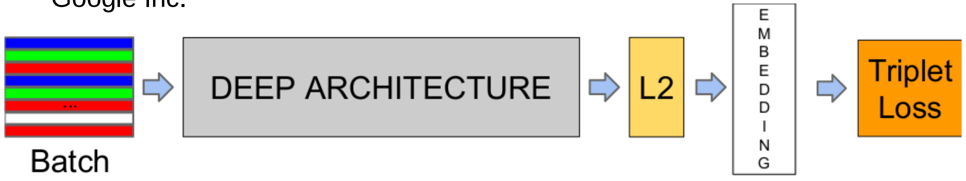
Team Member:

FaceNet from Google

A Unified Embedding for Face Recognition and Clustering

Florian Schroff Dmitry Kalenichenko James Philbin
fschroff@google.com dkalenichenko@google.com
jphilbin@google.com
Google Inc.

FaceNet learns the mapping from the images and creates embeddings rather than using any output layer for recognition



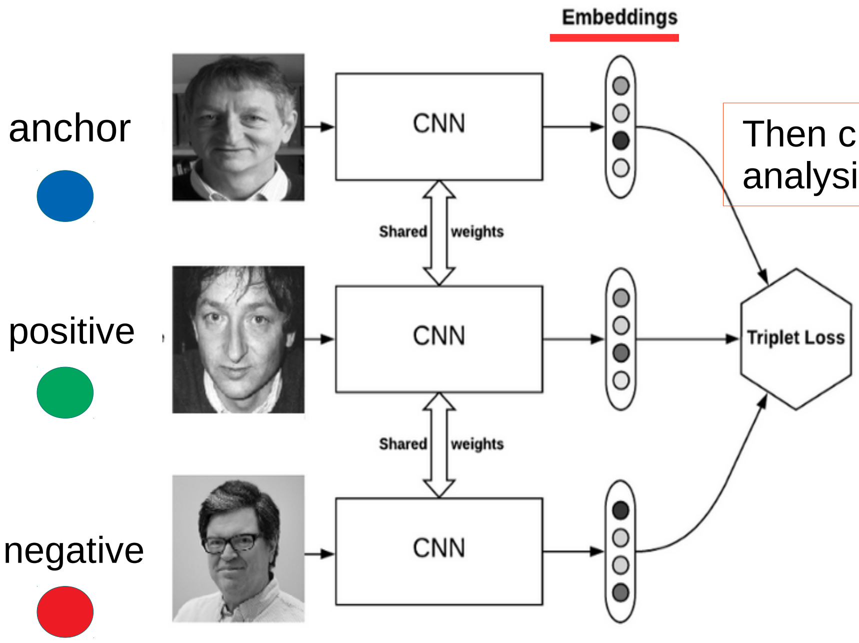
Block 1: Batch input layer

Block 2: deep CNN

Block 3: L2 distance e.g., normalization

L2, e.g, the Euclidean norm, a positive distance, calculated as the square root of the sum of the squared vector values.

Namely, we strive for an embedding $f(x)$, from an image x into a feature space $\mathbb{R}^d$, such that the squared distance between *all* faces, independent of imaging conditions, of the same identity is small, whereas the squared distance between a pair of face images from different identities is large.

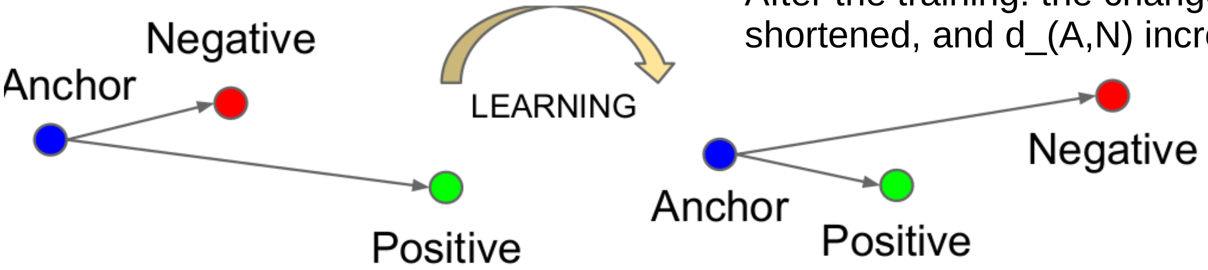


Then cluster analysis

Harry Li, Ph.D.

Triplet Loss Function

After the training: the change of the distance d_(A,P) shortened, and d_(A,N) increased



The objective:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \quad \text{… (1)}$$

$$\forall \left(f(x_i^a), f(x_i^p), f(x_i^n)\right) \in \mathcal{T} \quad \text{… (2)}$$

The loss function to be minimized:

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

… (3)

CNN1 Architecture

Table 1. NN1.
This table show the structure of our
Zeiler&Fergus [22] based model with 1×1 convolutions in-
spired by [9].
The input and output sizes are described
in rows × cols × #f ilters.
The kernel is specified as
rows × cols, stride and the maxout [6] pooling size as p = 2.

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

One way to reduce the training time is to normalize the activities of the neurons.

https://arxiv.org/abs/1607.06450

CNN2 Architecture

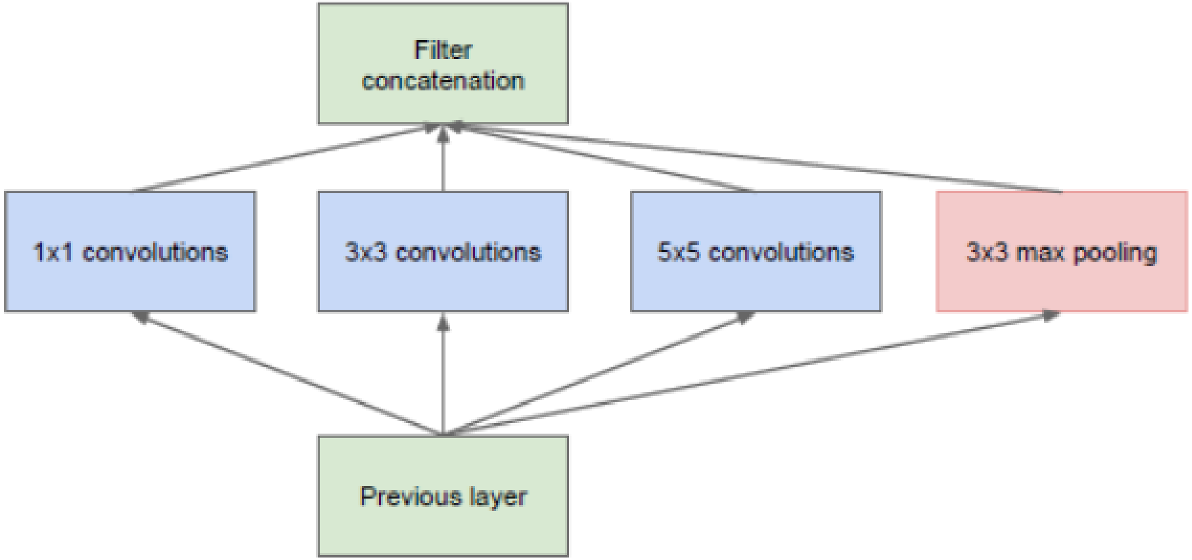
type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

CNN2 Architecture

Details of the NN2 Inception architecture. This model uses L 2 pooling instead of max pooling (m), where specified. The pooling is always 3×3 (aside from the final average pooling) and in parallel to the convolutional modules inside each Inception module. If there is a dimensionality reduction after the pooling it is denoted with p. 1×1, 3×3, and 5×5 pooling are then concatenated to get the final output.

"(Inception Layer) is a combination of all those layers (namely, 1×1 Convolutional layer, 3×3 Convolutional layer, 5×5 Convolutional layer) with their output filter banks concatenated into a single output vector forming the input of the next stage."

https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/



Triple Lose Function for Learning/Training

a

anchor

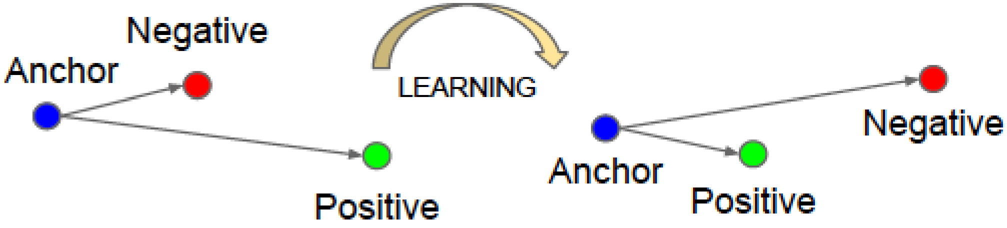
p

positive

n

negative

FaceNet: A Unified Embedding for Face Recognition and Clustering



$$\boxed{\sum_{i}^{N}\left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+}$$

X_j^a image j from anchor group a

$f(X_j^a)$ embedding, e.g., feature vectors

Choose Extreme Triple Lose Function for Training

$$\text{Argmax} \ || \ f(x_i^a) - f(x_i^p) \ ||_2^2 \quad \dots (1)$$

$$\text{Argmin} \ || \ f(x_i^a) - f(x_i^n) \ ||_2^2 \quad \dots (2)$$

Use equations (1) and (2) to train CNN to make sure the worst positive matching is not so “far away”, and the worst negative matching is not so “close together”.

Choose around 1000–2000 samples (In most experiments the batch size was around 1800).

Use adaptive learning rate

$$W_t = W'_{t-1} - \eta * g_t \quad \dots (3)$$

Eqn (3) is the regular Stochastic Greatest Descent method, eta is a constant

$$W_t = W'_{t-1} - \eta'_t * g_t \quad \dots (4)$$

Changing eta prime leads to adaptive learning rate, where

$$\eta'_t = \eta / \text{sqrt}(a_{t-1} + \epsilon) \quad \dots (5)$$

$$a_{t-1} = \sum^{t-1} g_i^2 \quad \text{and} \quad g_i = (\partial L / \partial W)$$

Choose Extreme Triple Lose Function for Training

$$\text{Argmax } \| f(x_i^a) - f(x_i^p) \|_2^2 \quad \dots (1)$$

$$\text{Argmin } \| f(x_i^a) - f(x_i^n) \|_2^2 \quad \dots (2)$$

Use equations (1) and (2) to train CNN to make sure the worst positive matching is not so “far away”, and the worst negative matching is not so “close together”.

Choose around 1000–2000 samples (In most experiments the batch size was around 1800).

Use adaptive learning rate

$$W_t = W'_{t-1} - \eta * g_t \quad \dots (3)$$

Eqn (3) is the regular Stochastic Greatest Descent method, eta is a constant

$$W_t = W'_{t-1} - \eta'_t * g_t \quad \dots (4)$$

Changing eta prime leads to adaptive learning rate, where

$$\eta'_t = \eta / \text{sqrt}(a_{t-1} + \epsilon) \quad \dots (5)$$

a_{t-1}: $\sum^{t-1} g_i^2$ and $g_i = (\partial L / \partial W)$

Harry Li, Ph.D.

TA (True Acceptance) and FA

Pair

$$\mathrm{TA}(d) = \{(i,j) \in \mathcal{P}_{\mathrm{same}}, \text{with } D(x_i, x_j) \leq d\}$$

True accepts are the face pairs that were correctly classified as same at threshold ‘d’.

$$\mathrm{FA}(d) = \{(i,j) \in \mathcal{P}_{\mathrm{diff}}, \text{with } D(x_i, x_j) \leq d\}$$

False accepts are the face pairs that were incorrectly classified as same

```
P same - It represents the pair of same identities
P diff - It represents the pair of different identities
D(xi,xj) -  It is the square L2 distance between the pair of images
d - It is the distance threshold
```

$$\mathrm{VAL}(d) = \frac{|\mathrm{TA}(d)|}{|\mathcal{P}_{\mathrm{same}}|}, \quad \mathrm{FAR}(d) = \frac{|\mathrm{FA}(d)|}{|\mathcal{P}_{\mathrm{diff}}|}$$

The validation rate (VAL) and false accept rate (FAR) for a given face distance ‘d’ is defined as

References:

1. FaceNet: A Unified Embedding for Face Recognition and Clustering

https://arxiv.org/abs/1503.03832 Cornell University

Zeiler & Fergus Architecture in the FaceNet Research Paper

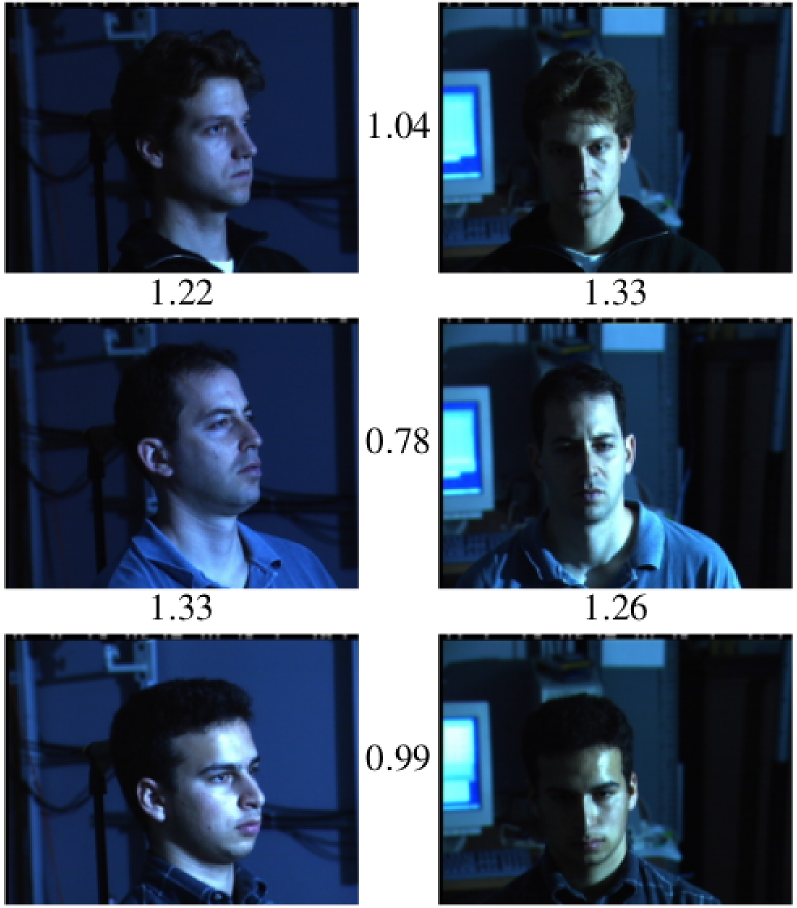
layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

https://medium.com/analytics-vidhya/introduction-to-facenet-a-unified-embedding-for-face-recognition-and-clustering-dbdac8e6f02

140 million parameters and 1.6 billion FLOPS (Floating point operations per second) per image.

Academic Datasets: Labeled Faces in the Wild (LFW) is the de-facto academic test set for face verification [7]

Pose and Illumination Invariant



Illumination and Pose invariance. Pose and illumination have been a long standing problem in face recognition. This figure shows the output distances of FaceNet between pairs of faces of the same and a different person in different pose and illumination combinations. A distance of 0.0 means the faces are identical, 4.0 corresponds to the opposite spectrum, two different identities. You can see that a threshold of 1.1 would classify every pair correctly

One Shot Facial Recognition

https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face-recognition/

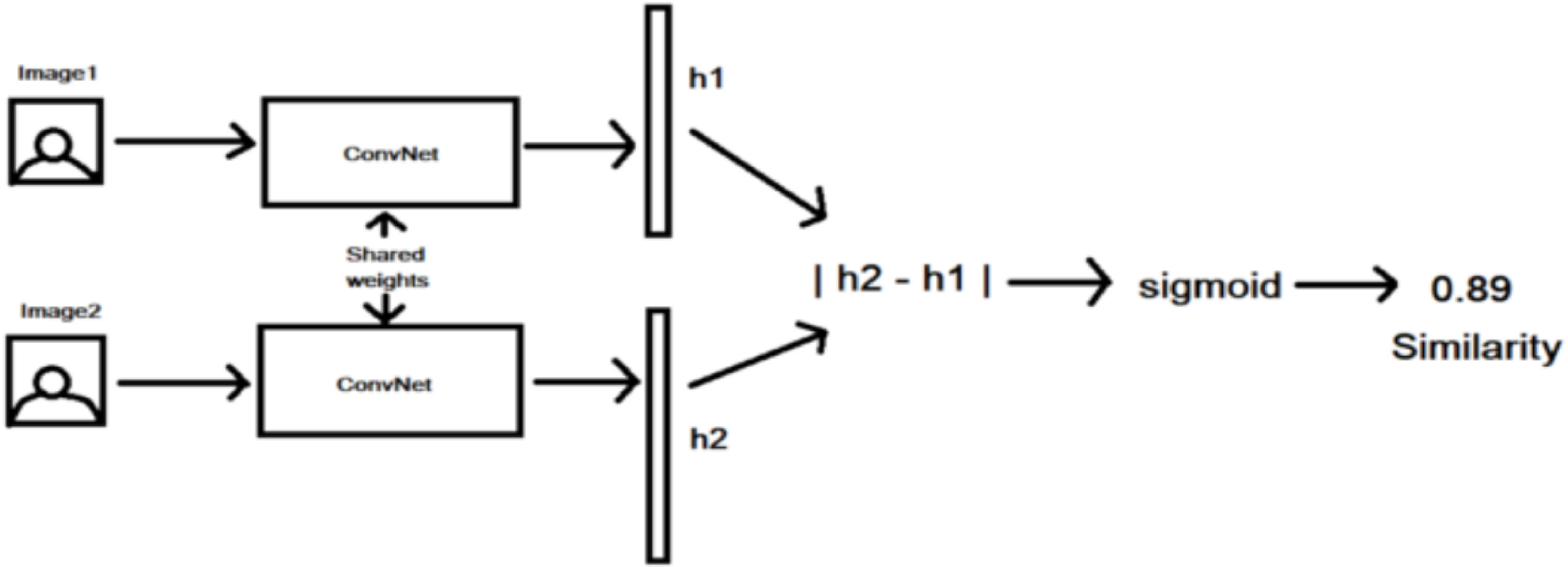
One, or a few, examples are used

To classify many new examples in the future.

Low-dimensional feature representation, called a face embedding

Embeddings were learned using a Siamese network.

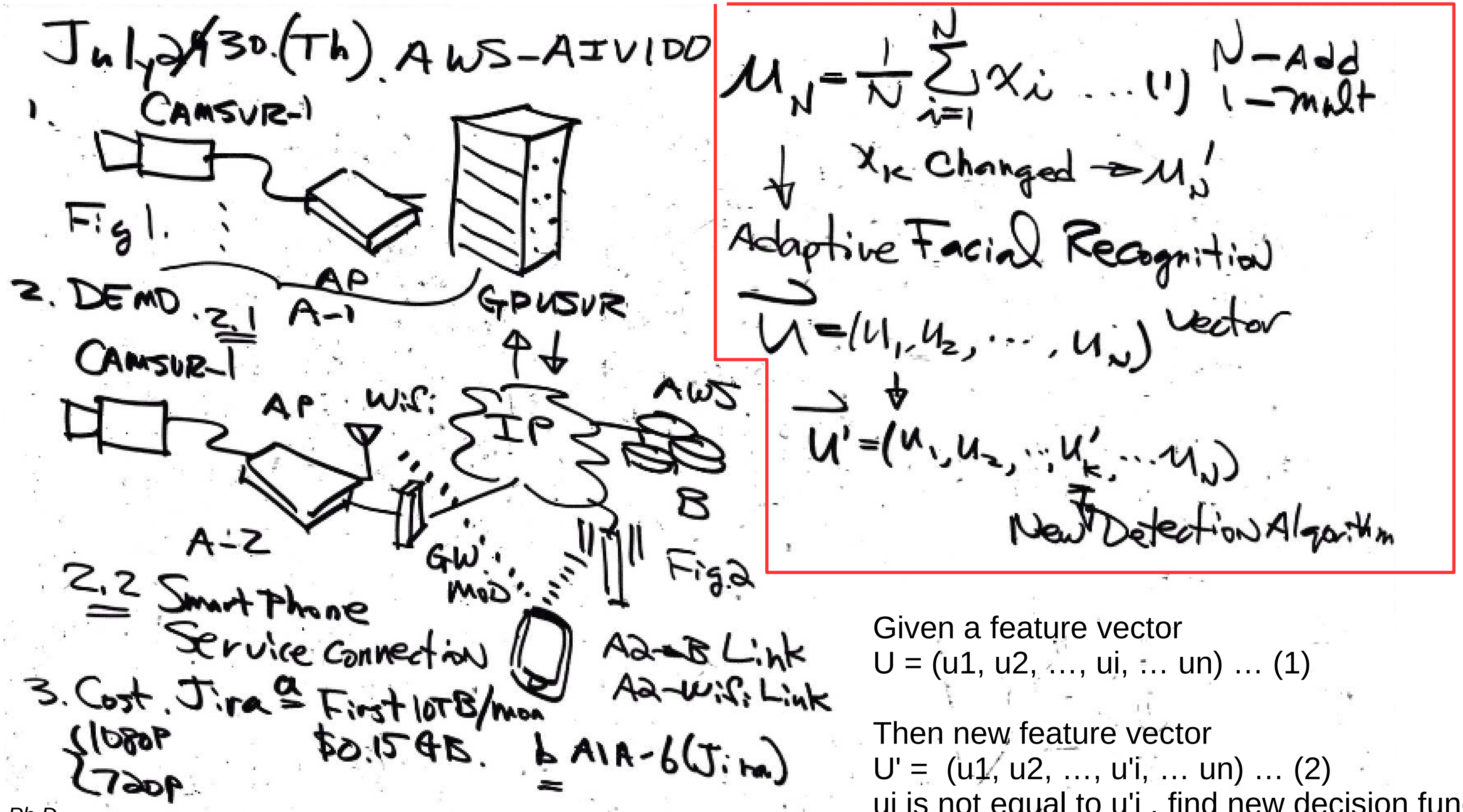
Siamese For One Shot Facial Recognition



1. Siamese networks consist of two symmetrical neural networks sharing the same weights and architectures, then joined together at the end using an energy function, E which acts as a distance function.

2. E function's objective is to learn whether two input images are similar or different.

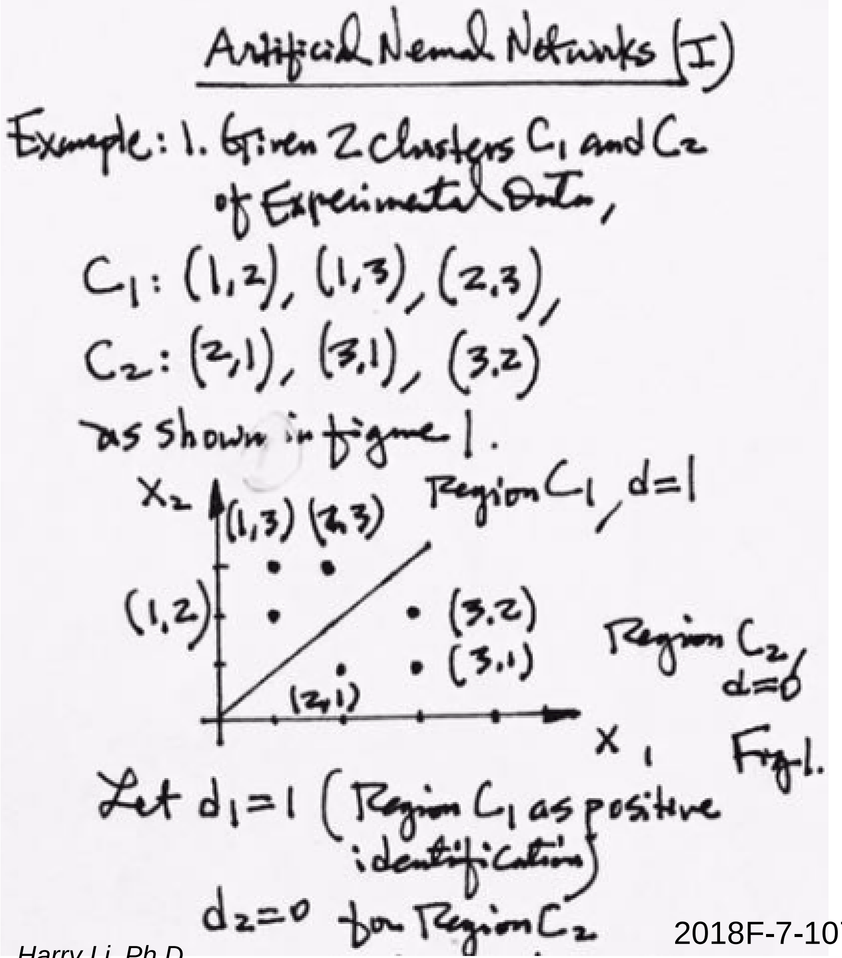
July 31 Adaptive Facial Recognition



Given a feature vector
U = (u1, u2, …, ui, … un) … (1)

Then new feature vector
U' = (u1, u2, …, u'i, … un) … (2)
ui is not equal to u'i , find new decision function

July 31 Adaptive Facial Recognition Example


Artificial Neural Networks (I)

Example: 1. Given 2 clusters C_1 and C_2 of Experimental Data,

C_1: (1,2), (1,3), (2,3),

C_2: (2,1), (3,1), (3,2)

as shown in figure 1.

Let $d_1=1$ (Region C_1 as positive identification)

$d_2=0$ for Region C_2

2018F-7-107-Feedford1.jpg

Step 1. train this neural networks with the given data, record the result {w}_orig and number of computation N.

Step 2. change one of the data set from any one of the classes, say, C1, make (1,2) to (1,4); Retrain the network, and record the result {w}_prim, and record the number of computations N_prim;

Step 3. now use new adaptive training approach by injecting {w}_orig to train the network to reach {w}_prim, record the number of computations N_adpt.

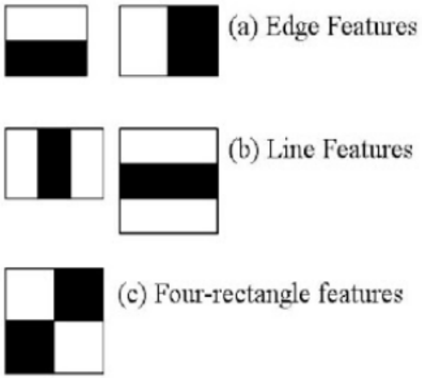
N and P_prim should be about the same, while N_adpt should be ideally much smaller.

Adaptive Training Example

https://towardsdatascience.com/inroduction-to-neural-networks-in-python-7e0b422e6c24

Haar Cascade for Facial Detection

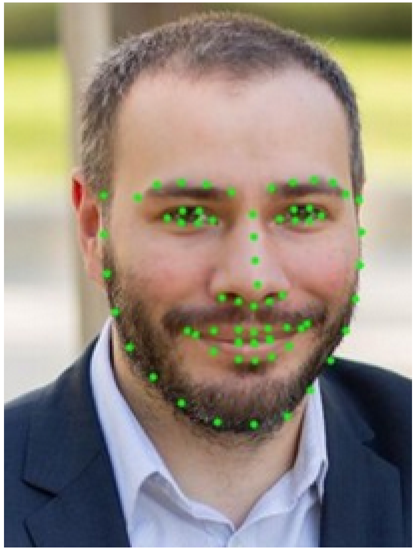
https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html

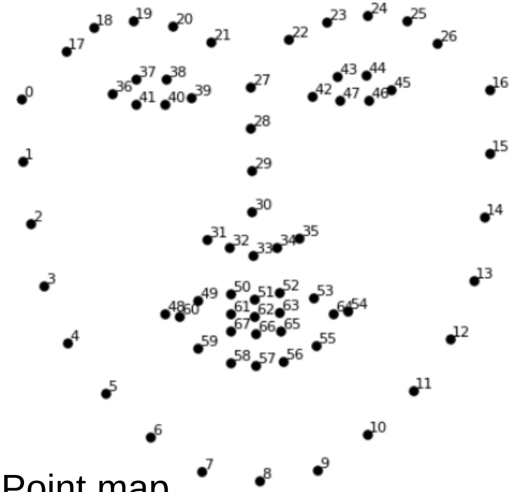


Sample Detecting Facial Features

https://towardsdatascience.com/detecting-face-features-with-python-30385aee4a8e

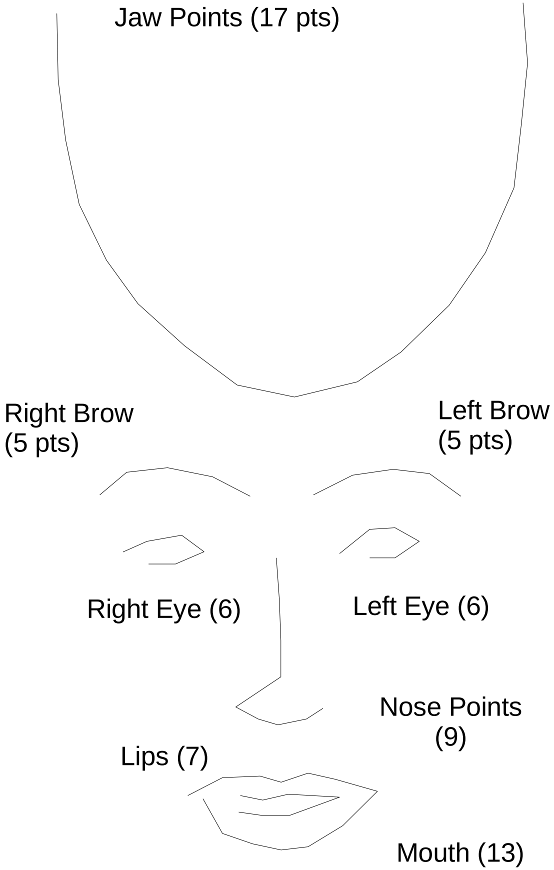
Feature Vectors





Point map
(67 points)

- Jaw Points = 0–16
- Right Brow Points = 17–21
- Left Brow Points = 22–26
- Nose Points = 27–35
- Right Eye Points = 36–41
- Left Eye Points = 42–47
- Mouth Points = 48–60
- Lips Points = 61–67



$pip -V (to check your pip version)

Note this model is simplified version with no upper face

Harry Li, Ph.D.

Using http://dlib.net/ For Feature Extraction

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments. Dlib's open source licensing allows you to use it in any application, free of charge.

Sample code:

```
import cv2
import numpy as np
import dlib   #for facial feature extraction
```

1. Machine learning
2. Numerical algorithms
3. Graphical model inference algorithms
4. Image processing
5. Threading
6. Networking
7. GUI
8. Testing (Unit testing framework)
9. XML Parser etc.

```
# Load the predictor
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

From dlib, additional features are possible

Use Python3.7

$python3.7 --version (to check your python 3.7 installation, if you have not installed python3.7, then install 3.7. To do so you will need to
$sudo apt-get update
$sudo apt install software-properties-common
$sudo add-apt-repository ppa:deadsnakes/ppa
$sudo apt-get update (I did this again, otherwise error message, python3.7 can not be found)
$sudo apt install python3.7

```
inal-py$ python3.7 --version
Python 3.7.0
```

Make python3.7 your default
Locate .bashrc file (in your home directory)_ and edit
$vi ~/.bashrc (to add the following line)
alias python3=python3.7

Make modifed .bashrc effective without login out and back again
$source ~/.bashrc

Then check:
$python3

```
/0-video-image/final-py$ python3 --version
Python 3.7.0
```

OpenCV Python3.7

Make python3.7 your default, as described in the previous slide.

Jupyter Notebook for iPython

https://askubuntu.com/questions/847263/install-jupyter-notebook-for-python-2-7

To install jupyter notebook

```
sudo apt-get update
sudo apt-get -y install python3-pip python3-dev
sudo -H pip3 install --upgrade pip
sudo apt-get -y install ipython3 ipython3-notebook
pip3 install --user jupyter
```

Teaching Tool to tell a story

$puypter --version

$puypter notebook

It is possible to use Python 3 in Jupyter Notebook for Python 2 by adding the kernel for Python 2. If you're running Jupyter on Python 3, you can set up a Python 2 kernel like this:

```
python2 -m pip install ipykernel
python2 -m ipykernel install --user
```

Tutorial

https://www.youtube.com/watch?v=jZ952vChhuI

Pycharm IDE for Python

PyCharm is probably the only Python dedicated IDE that supports the vast expanse of features Python has. Sublime, Atom and Sypder do exist, but they only exist! The integrated development environment experience that PyCharm provides is way better than the others.

To install pycharm:

https://medium.com/@singh.shreya8/how-to-install-pycharm-in-ubuntu-16-04-ubuntu-14-04-ubuntu-18-04-linux-easiest-way-5ae19d052693

To start pycharm:
$sh pycharm.sh

https://www.youtube.com/watch?v=BPC-bGdBSM8&list=PLQ176FUIyIUZ1mwB-uImQE-gmkwzjNLjP

Pycharm hello the world

https://www.jetbrains.com/help/pycharm/creating-and-running-your-first-python-project.html#summary

use Cmake to build and install OpenCV and Extra Modules from source and configure your Pycharm IDE

https://towardsdatascience.com/how-to-install-opencv-and-extra-modules-from-source-using-cmake-and-then-set-it-up-in-your-pycharm-7e6ae25dbac5

OpenCV Build

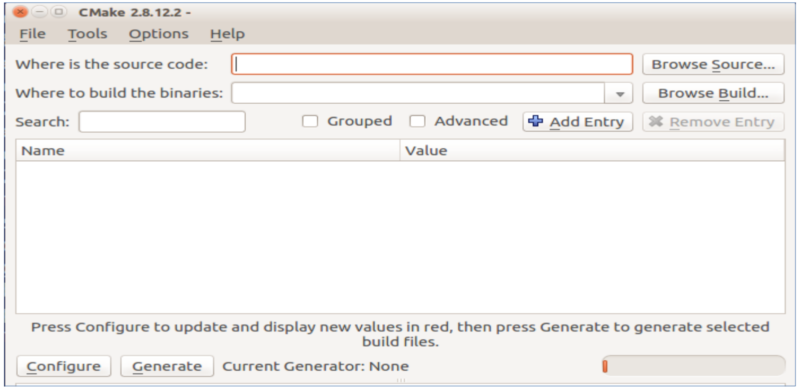
https://github.com/opencv/opencv_contrib

```
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_C_EXAMPLES=ON \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D WITH_TBB=ON \
-D WITH_V4L=ON \
-D WITH_QT=ON \
-D WITH_OPENGL=ON \
-D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
-D BUILD_EXAMPLES=ON ..
```

Use cmake-gui to Build OpenCV Extra Modules

$sudo apt-get Install cmake-qt-gui

$cmake-gui (to start cmake gui)



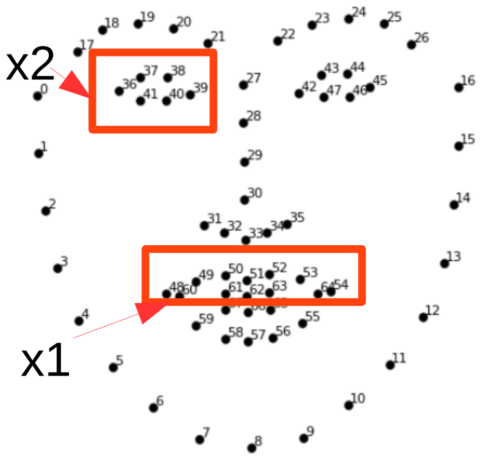
https://www.pyimagesearch.com/2015/07/27/installing-opencv-3-0-for-both-python-2-7-and-python-3-on-your-raspberry-pi-2/

Installing OpenCV 3.0 for both Python 2.7 and Python 3+ on your Raspberry Pi 2

```
$ cd ~/.virtualenvs/cv3/lib/python2.7/site-packages/
$ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so
```

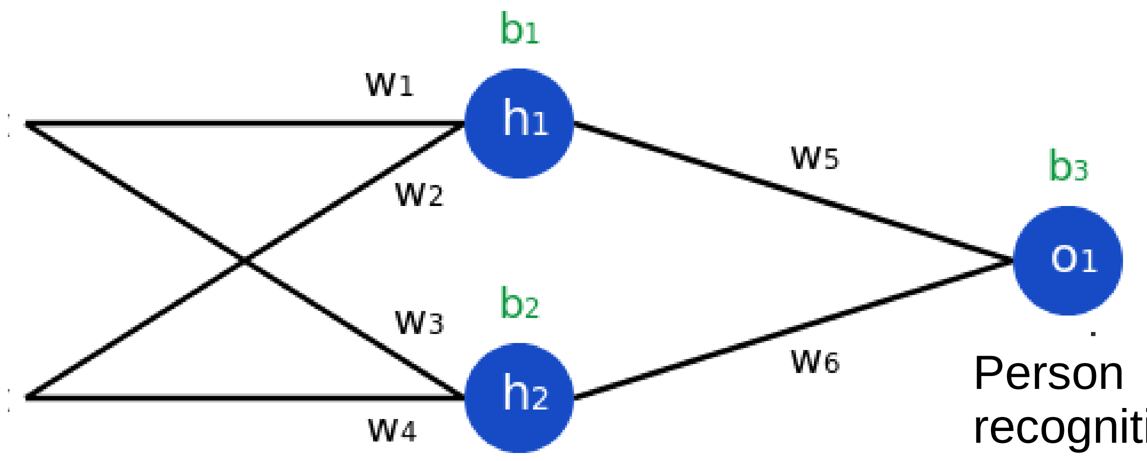
Simple Feed Forward Neural Networks

Input Layer Hidden Layer Output Layer



x1 (lip)

x2 (right eye brow)



Person recognition

```
def feedforward(self, x):
  # x is a numpy array with 2 elements.
  h1 = sigmoid(self.w1 * x[0] + self.w2 * x[1] + self.b1)
  h2 = sigmoid(self.w3 * x[0] + self.w4 * x[1] + self.b2)
  o1 = sigmoid(self.w5 * h1 + self.w6 * h2 + self.b3)
  return o1
```

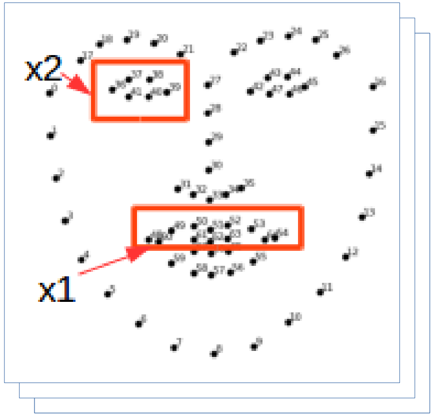
To do: (1) add h3 hidden layer, (2) add o2 at output layer, modify the code

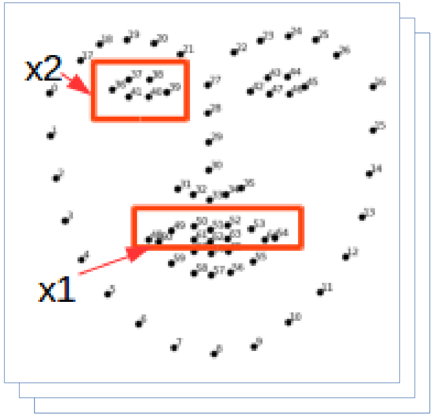
Prepare the Data Set

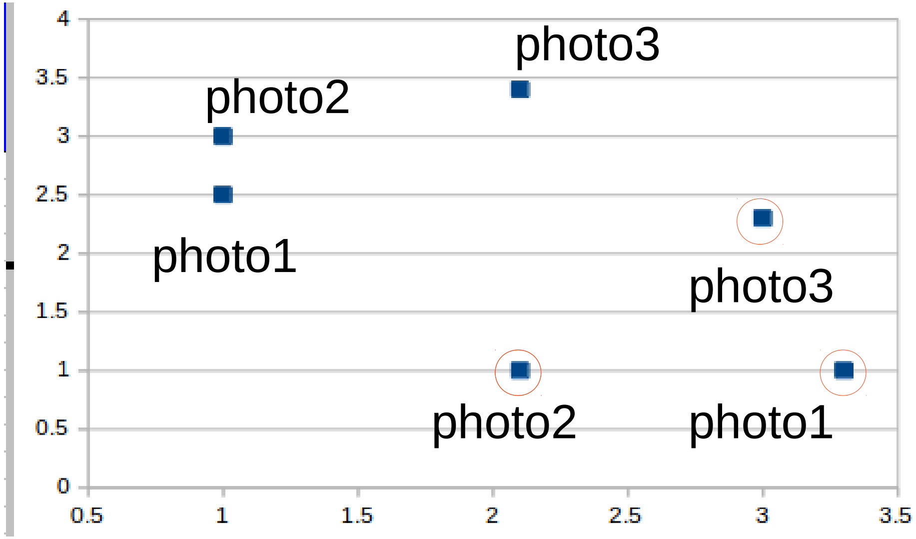
For 2 persons

```
#-----------------------------------
# Define dataset and all_y_trues
#-----------------------------------
data = np.array([
  [1, 2.5],     # person A
  [1, 3],       # person A
  [2.1, 3.4],   # Person A
  [2.1, 1], # person B
  [3.3, 1], # person B
  [3, 2.3], # person B
])
all_y_trues = np.array([
  1, # person A
  1, # person A
  1, # person A
  0, # person B
  0, # person B
  0, # person B
])
```

For 2 persons







Prepare the Output Layer

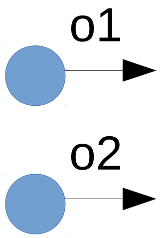
For 2 persons

```
#----------------------------------
# Define dataset and all_y_trues
#----------------------------------
data = np.array([
  [1, 2.5],     # person A
  [1, 3],       # person A
  [2.1, 3.4],   # Person A
  [2.1, 1], # person B
  [3.3, 1], # person B
  [3, 2.3], # person B
])
all_y_trues = np.array([
  1, # person A
  1, # person A
  1, # person A
  0, # person B
  0, # person B
  0, # person B
])
```

For 4 persons

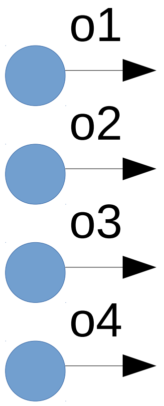
(1) Output layer with 2 neurons,

o1	o2
0	0
0	1
1	0
1	1



(2) output layer with 4 neurons

o1	o2	o3	o4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1 t



Output Layer from ImageNet

https://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-18.html

ImageNet Training in Minutes

Yang You, Zhao Zhang, Cho-Jui Hsieh, James Demmel and Kurt Keutzer

EECS Department

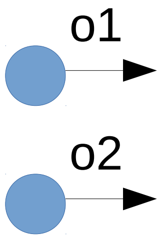
University of California, Berkeley

Technical Report No. UCB/EECS-2020-18

January 25, 2020

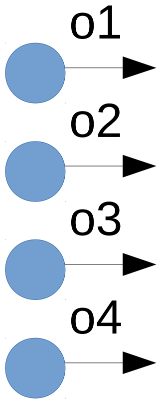
(1) Output layer with 2 neurons,

o1	o2
0	0
0	1
1	0
1	1



(2) output layer with 4 neurons

o1	o2	o3	o4
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1 t



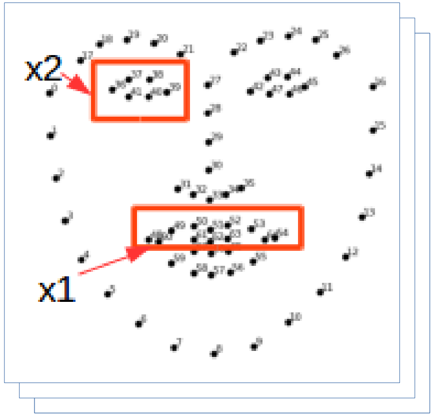
For 4 persons

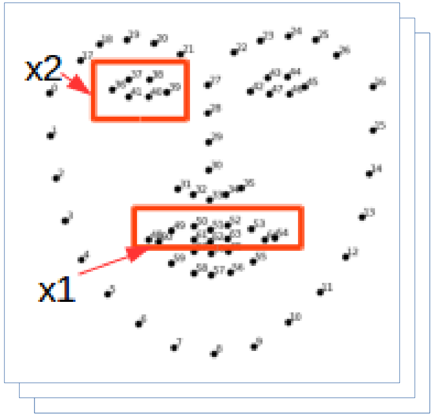
Prepare the Data Set

For 2 persons

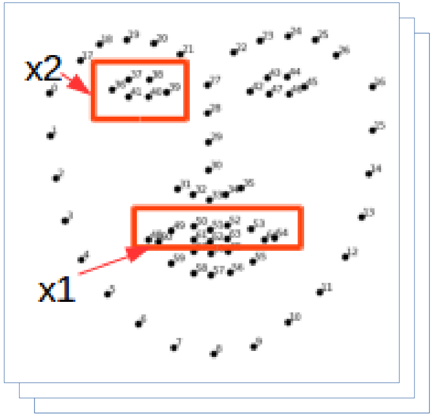
```
#-----------------------------------
# Define dataset and all_y_trues
#-----------------------------------
data = np.array([
  [1, 2.5],     # person A
  [1, 3],        # person A
  [2.1, 3.4],   # Person A
  [2.1, 1], # person B
  [3.3, 1], # person B
  [3, 2.3], # person B
])
all_y_trues = np.array([
  1, # person A
  1, # person A
  1, # person A
  0, # person B
  0, # person B
  0, # person B
])
```

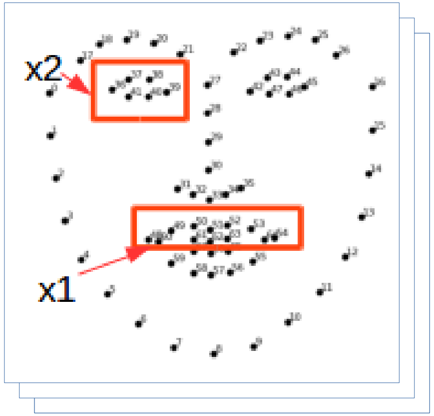
For 2 persons

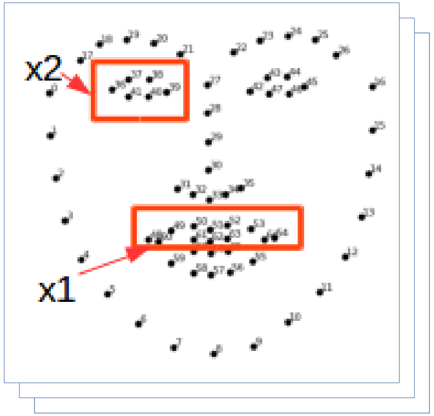


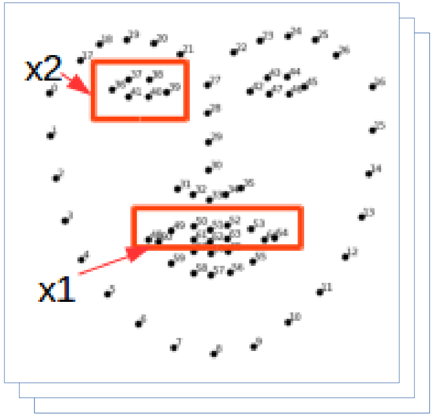


For 4 persons









4 Person Data Set

For 4 persons

```
#-----------------------------------
# Define dataset and all_y_trues
#-----------------------------------
data = np.array([
  [1, 2.5],     # person A
  [1, 3],       # person A
  [2.1, 3.4],   # Person A
  [2.1, 1], # person B
  [3.3, 1], # person B
  [3, 2.3], # person B
  [3, 2.5],       # person C
  [3, 3.6],       # person C
  [4.1, 5.4],    # Person C
  [4.5, 3.1], # person D
  [3.3, 1.9], # person D
  [5.9, 2.3], # person D
])
```

```
all_y_trues = np.array([
  01, # person A
  01, # person A
  01, # person A
  00, # person B
  00, # person B
  00, # person B
  11, # person C
  11, # person C
  11, # person C
  10, # person D
  10, # person D
  10, # person D
])
```

For 4 persons