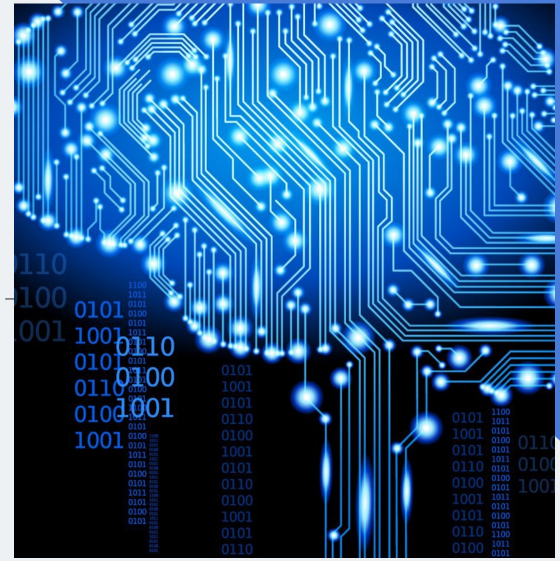
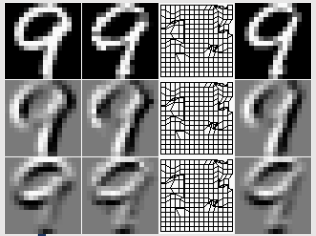
Northeastern SMILE Lab - Recognizing Faces in the Wild

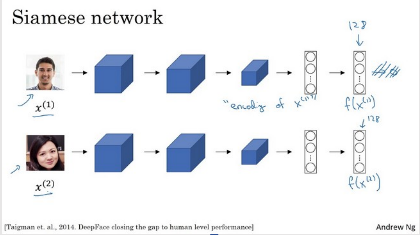


CONTENT

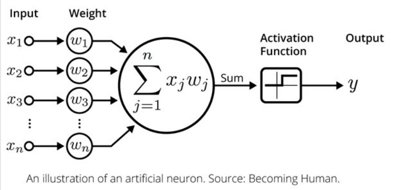
Data



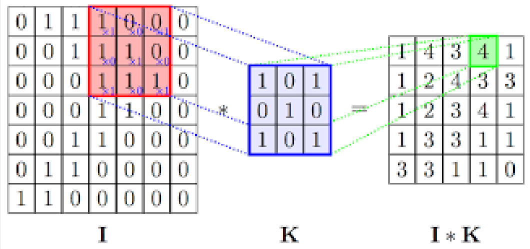
Siamese

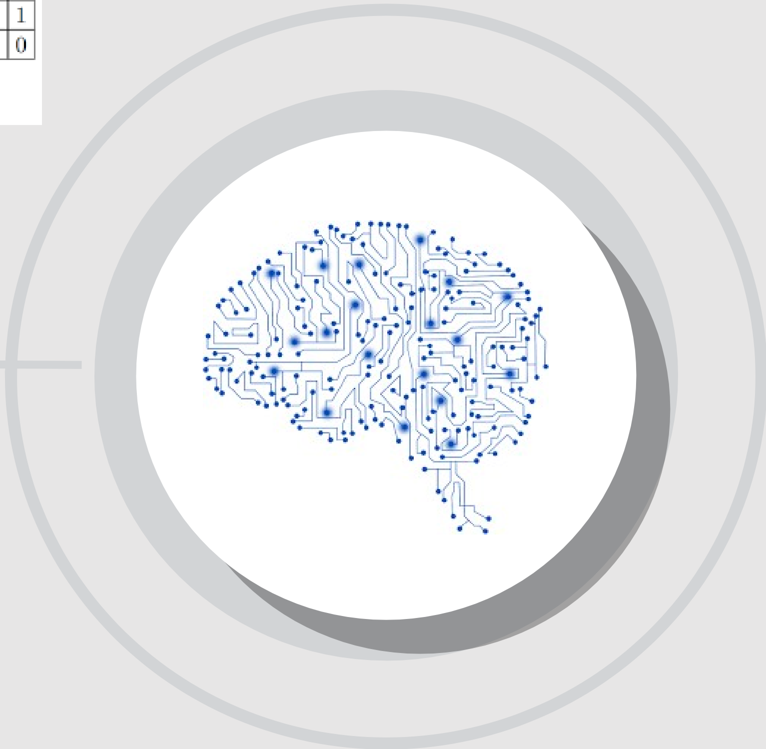


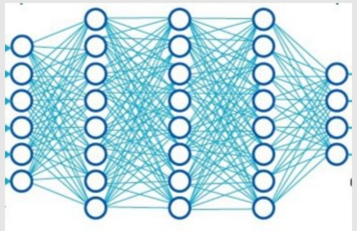
Activation function



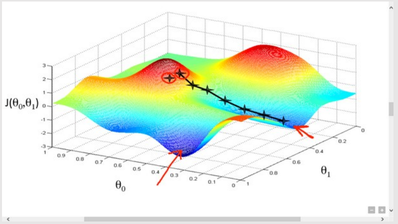
Siamese-





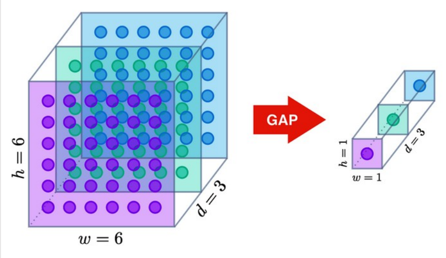


Vggface-Facenet



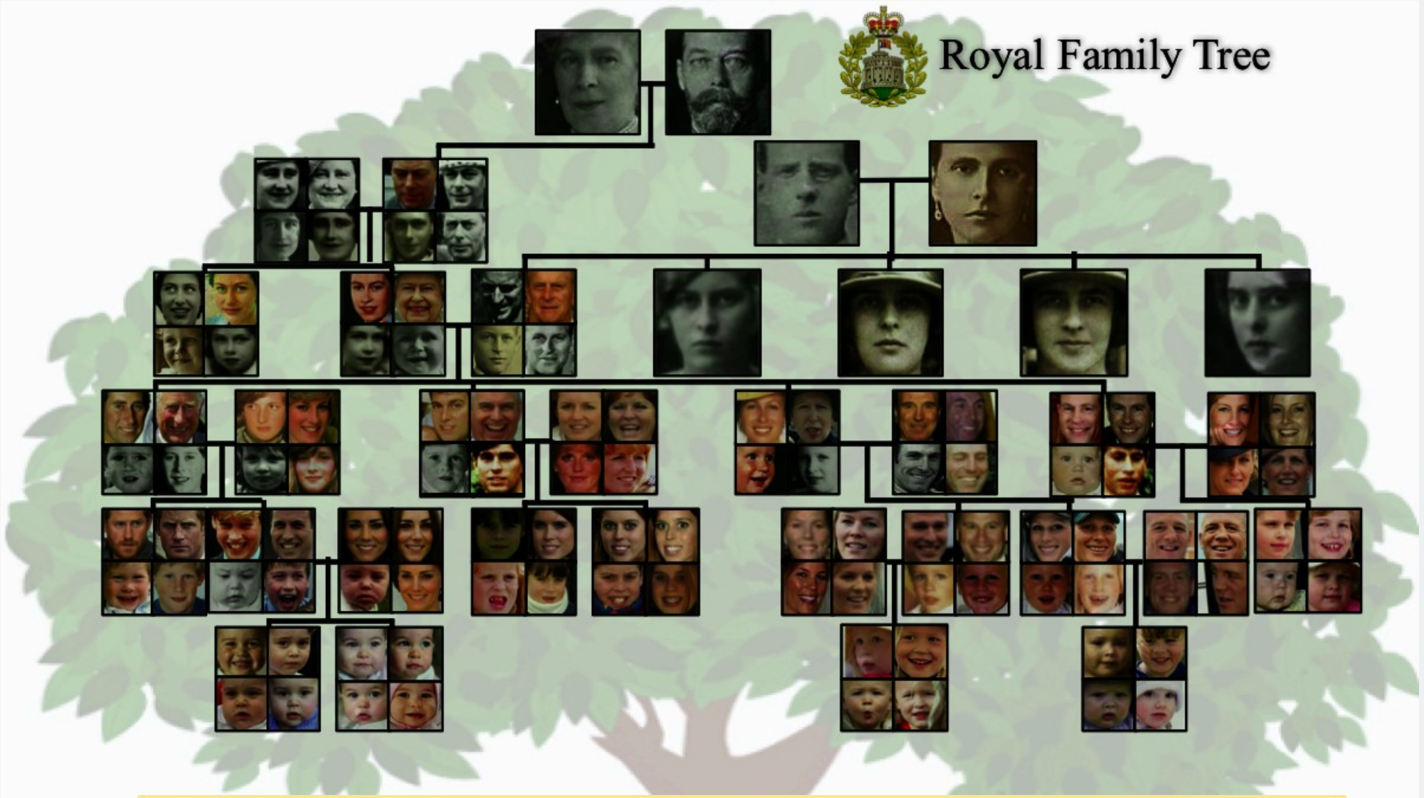
Optimiser

Global Average Pooling



INTRODUCTION

Northeastern SMILE Lab - Recognizing Faces in the Wild



VISUAL kinship recognition has an abundance of practical uses, such as issues of human trafficking and in missing children, problems from today's refugee crises, and

Dataset

Families In the Wild (FIW) is the largest and most comprehensive image database for automatic kinship recognition

Two tasks:

- Kinship recognition, that is predicting what kind of relation do two individuals share. Very difficult, we did not focus on it;
- Kinship verification, that is determine whether a pair of facial images of different subjects are kin of a specific type. The focus of the challenge.

DATA

over 12379 photos

photos

members

4-to-38 members

470 families

Families

person

2316 person in total.

Characters are public figures

Targeted to ensure diversity in terms of ethnicity, country, occupation

In the Wild refers to variation in pose, illumination, expression, even age for member people

The original dataset included 11 kinship types; unfortunately, they have not been made available for the challenge

Different types of kin share different familial features, so pair types and modeled and evaluated independent of all others*

* J. P. Robinson, *Recognizing Families in the Wild*, Data Challenge Workshop in conjunction with ACM MM, 2017

Data description

Train-faces - the training set is divided in Families

then individuals

Images in the same individul folder belong to the same person

Images in the same family folder belong to the same family

Test-faces   - the test set contains face images of unknown individuals

Kinship Verification

The goal of kinship verification is to determine whether a pair offacial images of different subjects are kin of a specific type

#of kineship in our data set =3598

Image pre-processing

Pre-processing has been tailored to the feature extractor used (more on this later)

Face detection and cropping has already been performed by the dataset curators

Deep Learning models have achieved superior performance on this task, compared to other approaches

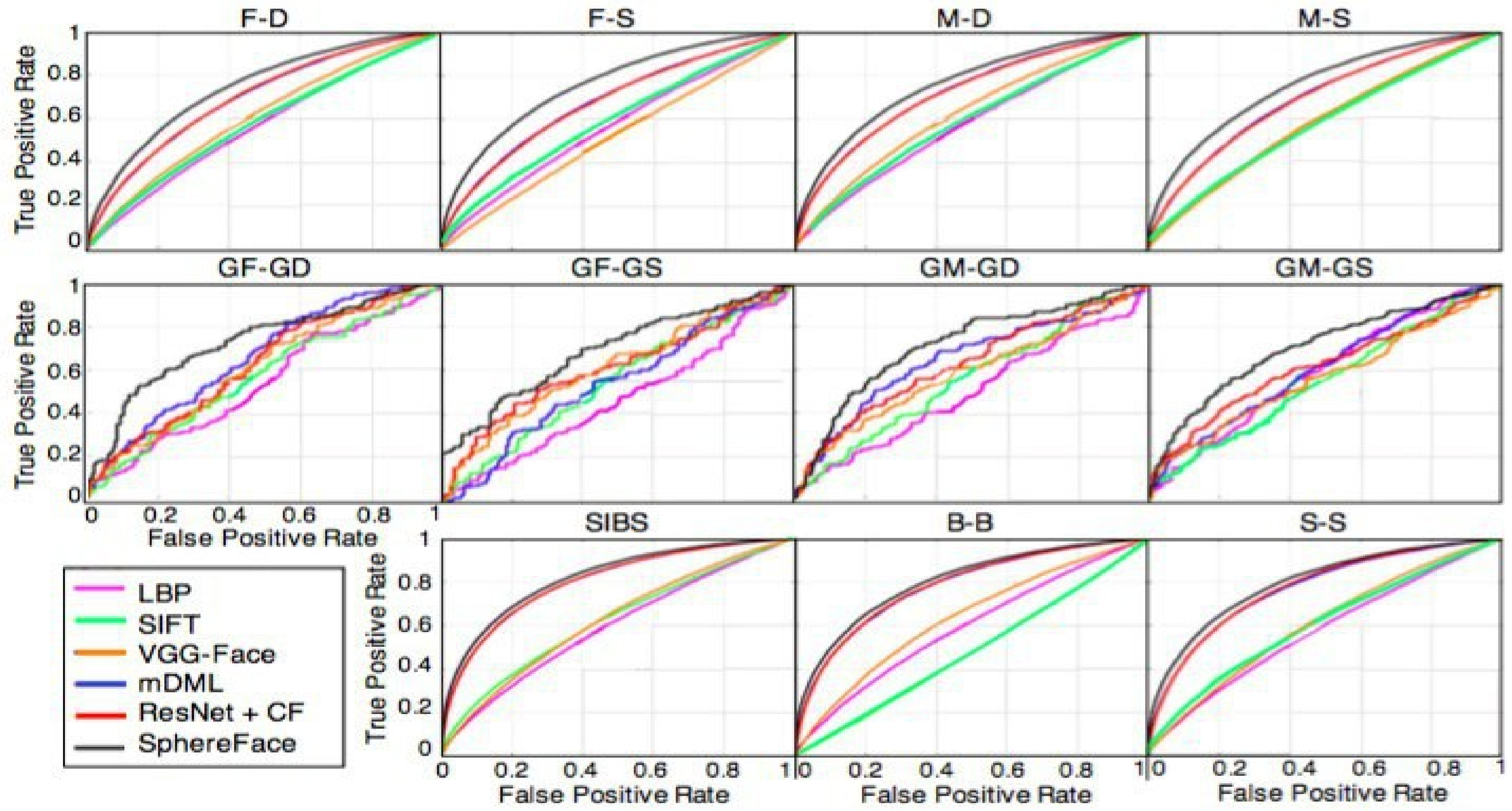


Fig. 6. Relationship type specific ROC curves.

Figure taken from J. P. Robinson et al., *Visual Kinship Recognition of Families in the Wild*, 2017

Existing Baseline Methods

Centerface (CF)-22-layer residual convolutional neural network (CNN)

Center-loss

Face pairs were scored using cosine similarity

MODEL 1

VGG-Face

An embedding size of 512, instead of the last fully-connected

Triplet-loss function

MODEL 2

Exsiting Baseline Methods

The authors proposed KinNet,

A softmax loss was added after thef c−layer to decide between the 41,856 subjects. While fine-tuning, thef c−layer was replaced with a new f c−layerof size1024D, which was followed by an L2-norm layer to normalize features to unit length

a soft triplet-loss

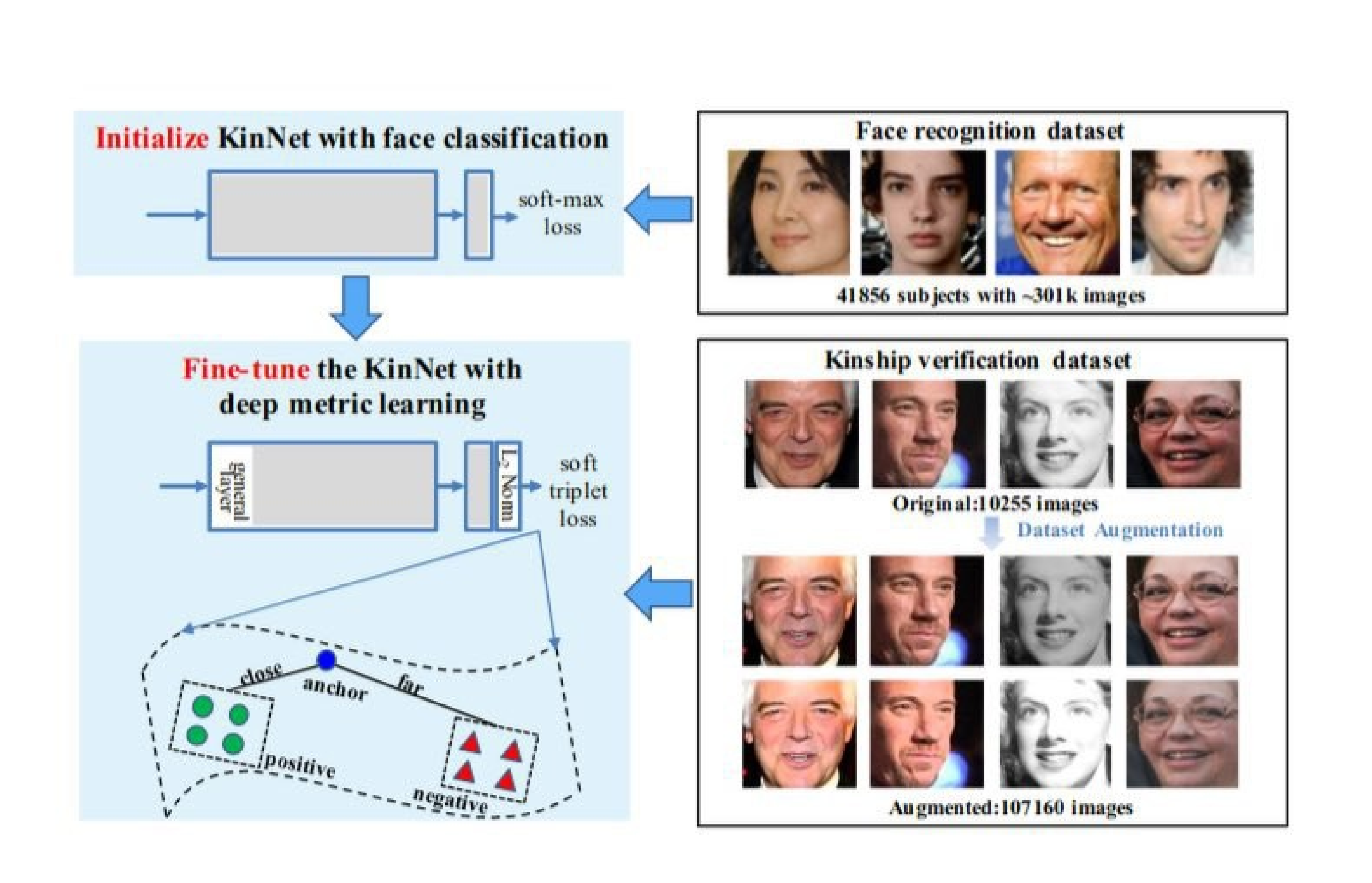
MODEL 3

Since thef irst layer learns the most general representations (lower-layerstend to learn more basic filters) the authors froze the bottommost7×7convolution layer and updated other layers to adapt the model for kinship verification.

To increase the number of training images : augmentation strategy

Exsiting Baseline Methods

MODEL 3



Exsiting Baseline Methods

MODEL 4

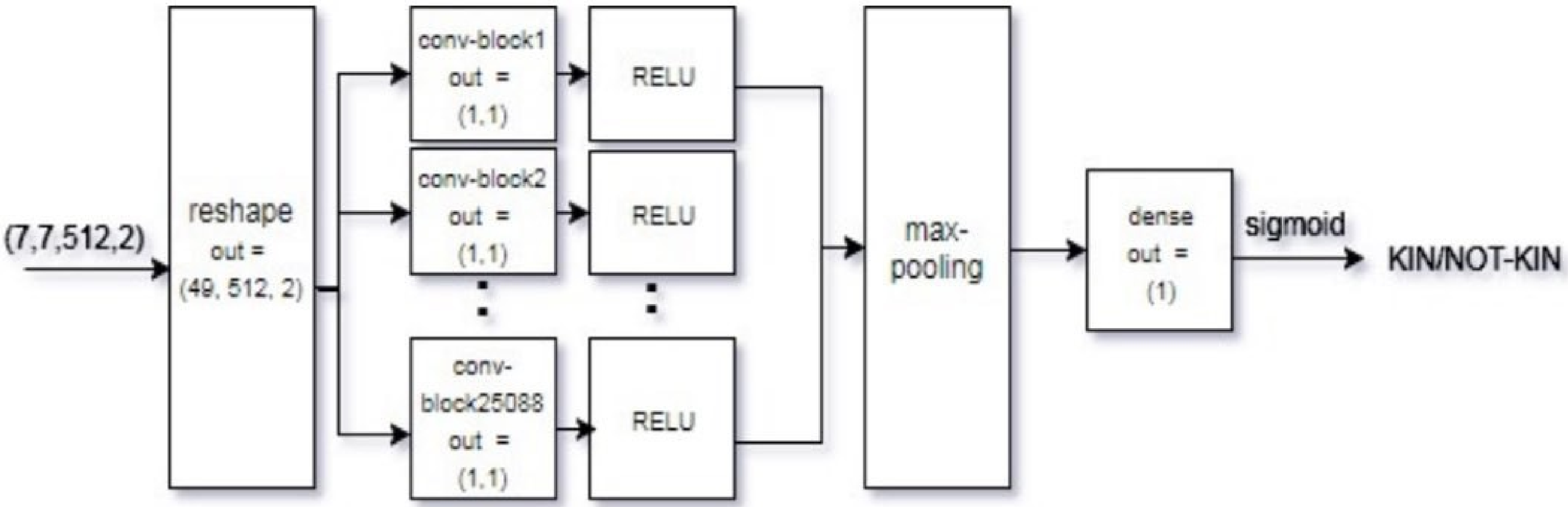


Figure 7: Modified convolution layer proposed by Team from Bar-Ilan University (Ref 1.F).

Human performance

According to experiments conducted by the curators of the dataset*, human performance on this task is 57.5% (accuracy)

Points to the difficulty of solving such a task

* J. P. Robinson et al., *Visual Kinship Recognition of Families in the Wild*, 2017

Vgg-Face and Facenet

VGGFACE

The VGG-Face architecture visualizing low to high level features captured as a facial expression is propagated through-out the network, stating the dimensions produced by each layer.

It has 22 layers.

VGG-Face

The VGG-Face architecture has been developed by the Oxford Visual Geometry Group as an extension to existing pre-trained models

It uses one of the mainstream pre-trained architectures, usually VGG-16 or Resnet-50, trained on massive amounts of face images

The *Labeled Faces in the Wild* (13,233 images of 5,749 people) and the *YouTube Faces* (3,425 videos of 1,595 people) had been used for training*

* http://www.robots.ox.ac.uk/~vgg/software/vgg_face/

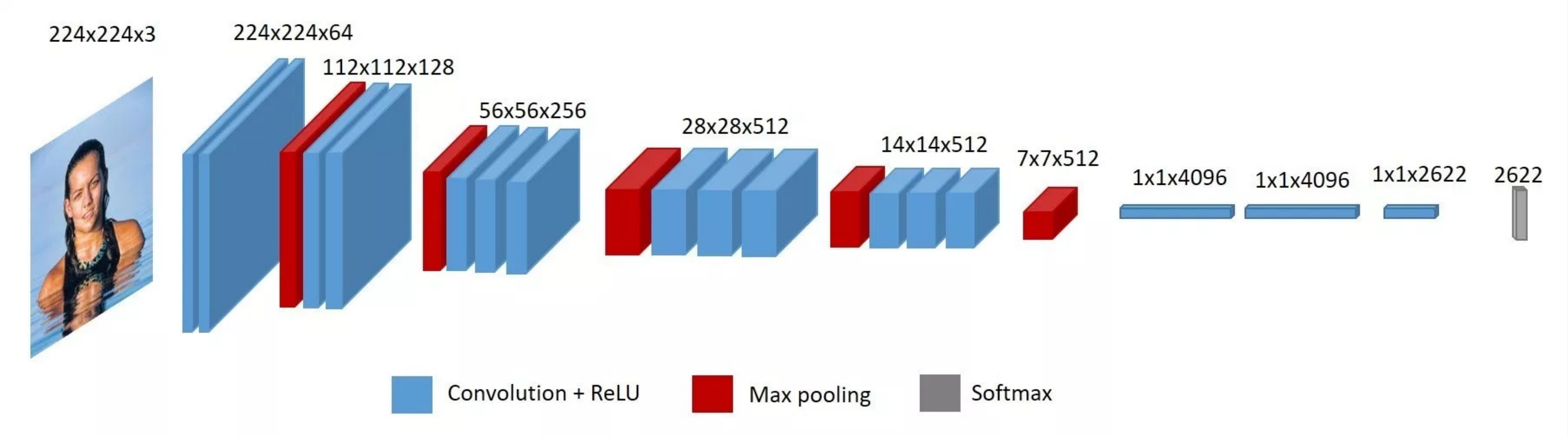
VGG-Face

Keras comes with a *keras_vggface* module to translate the original Caffe models into a Tensorflow backend

It contains everything we need, including a *preprocess_vggface* function

We used the Resnet-50 architecture

VGG-Face - full view at image level



VGGFACE -CODE

Let's construct the VGG Face model in Keras

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(224,224, 3)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation='relu'))
```

Face-Net*

FaceNet learns a mapping from face images to a compact Euclidean Space where distances directly correspond to a measure of face similarity

Once the FaceNet model having been trained with triplet loss for different classes of faces to capture the similarities and differences between them, the 128 dimensional embedding returned by the FaceNet model can be used to clusters faces effectively

Facenet has been introduced in 2015 by Google researchers and is now the backbone of many open-source face recognition systems like OpenFace, etc.

* F. Schroff, D. Kalenichenko, J. Philbin, *FaceNet: A Unified Embedding for Face Recognition and Clustering*, 2015

Face-Net

Input layer and a deep CNN followed by L2 normalization, which results in the face embedding. This is followed by the triplet loss during training

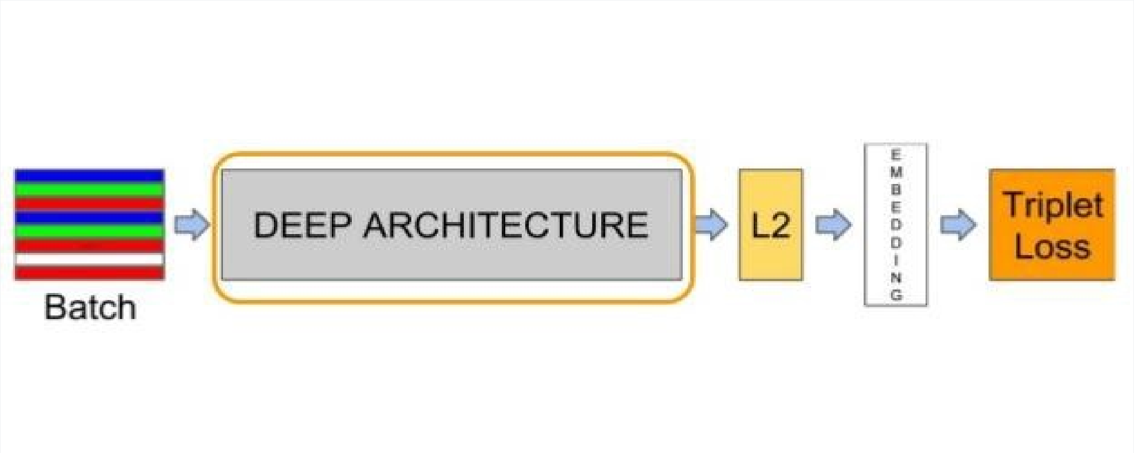


Image taken from the paper mentioned at previous slide

Face-Net

In order to compare two images, create the embedding for both images by feeding through the model separately. Then we can use above formula to find the distance which will be lower value for similar faces and higher value for different face.

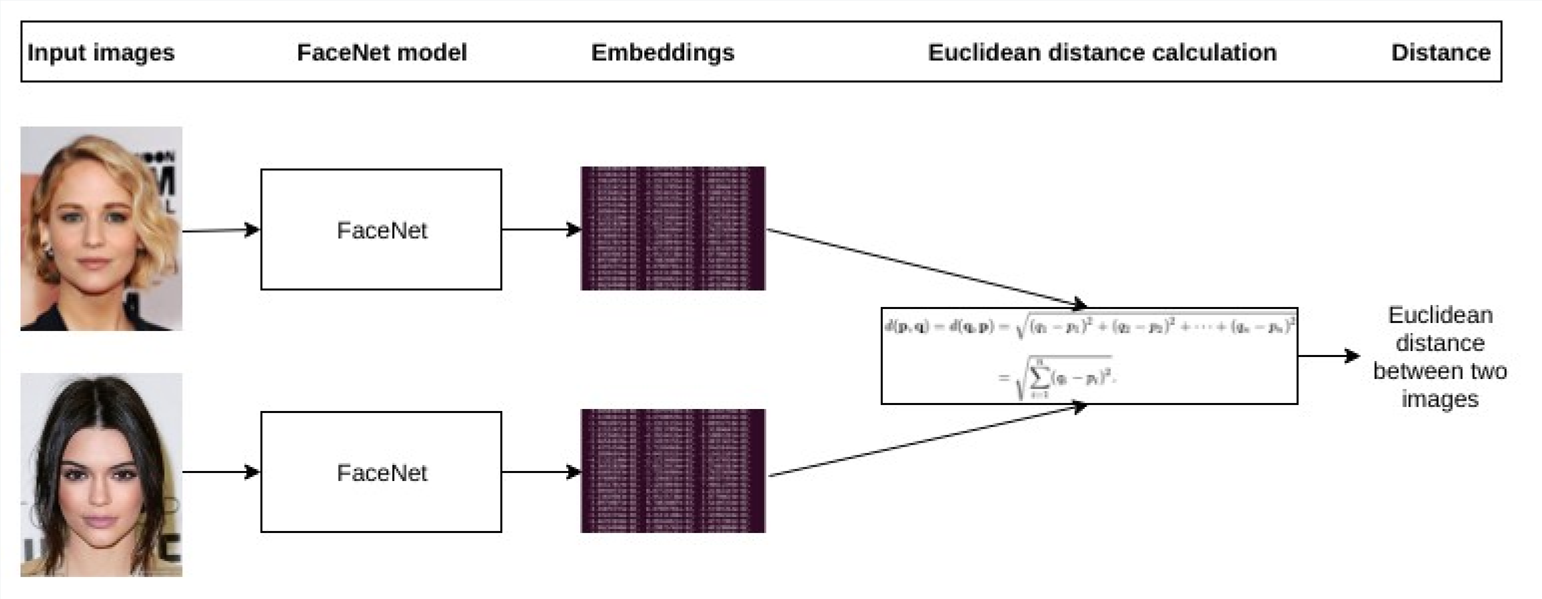


Image taken from the paper mentioned at previous slide

Face-Net

We used a pre-trained Facenet model hosted at: https://github.com/nyoki-mtl/keras-facenet

Training had been performed on *MS-Celeb-1M* dataset, a massive collection of 6,464,018 pictures taken from 64,682 celebrities (mined from online material)

Pre-processing: we need to resize the image to (160, 160, 3) and standardize the pixels to take values in [0,1]

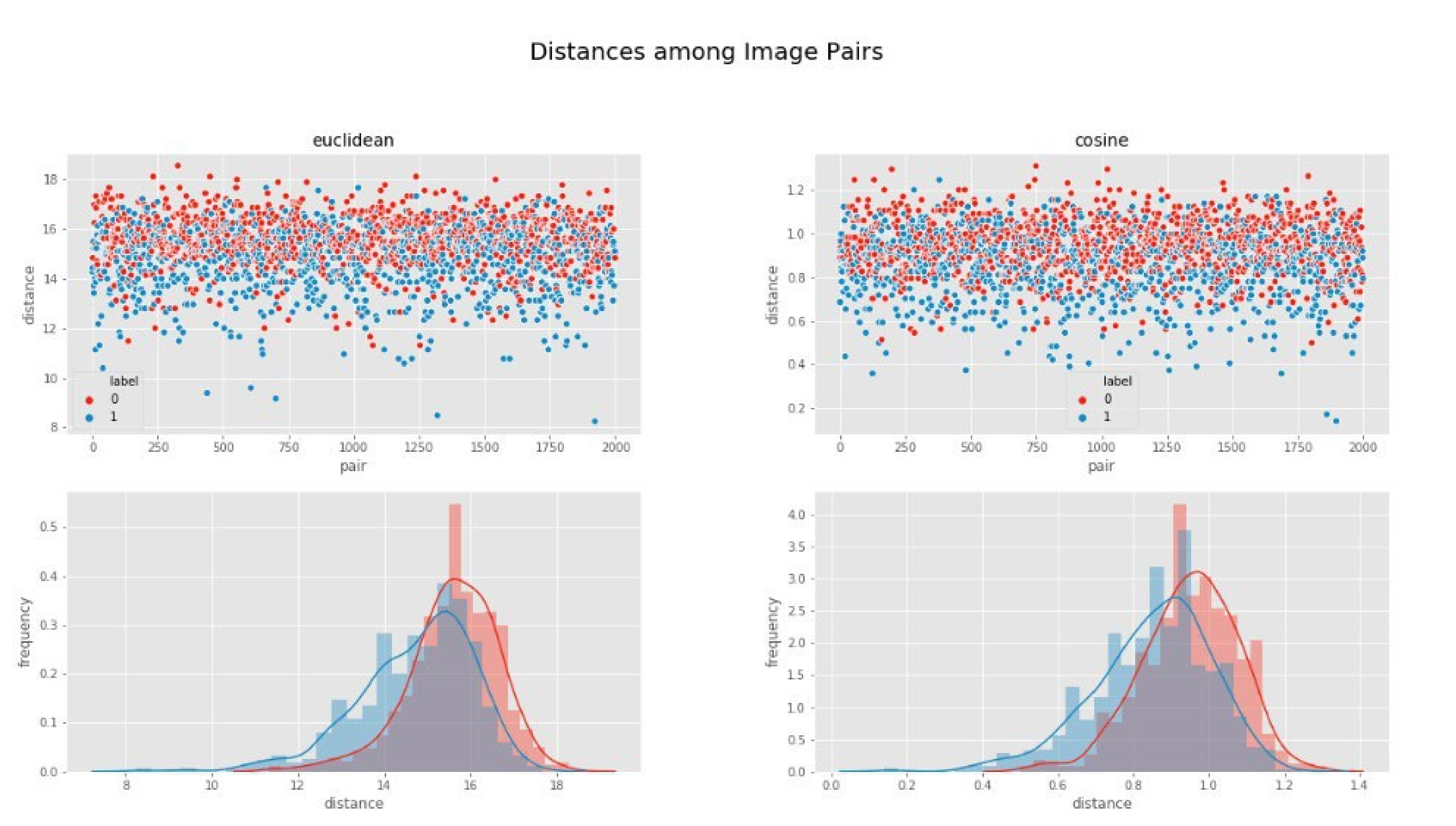
It employs an Inception-ResNet-v1 architecture, and includes layers such as BatchNormalization, together with an intricated topology made up of branches and blocks

Experiment*: sample k(=2000) pairs, produce embeddings using VGG-Face, plot different distance metrics between the encoded vectors. Pairs are colored according to kin/no-kin.

Boundaries are fuzzy, pointing to the difficulty of the task. Histogram frequencies are basically overlapping

* We are reproducing the plots of Q. Duan, *Adversarial Contrastive Residual Net for 1 Million Kinship Recognition*, 2017 (figure 5 of page 7)

Experiment: same as before, but using Facenet as feature extractor. Results are similar

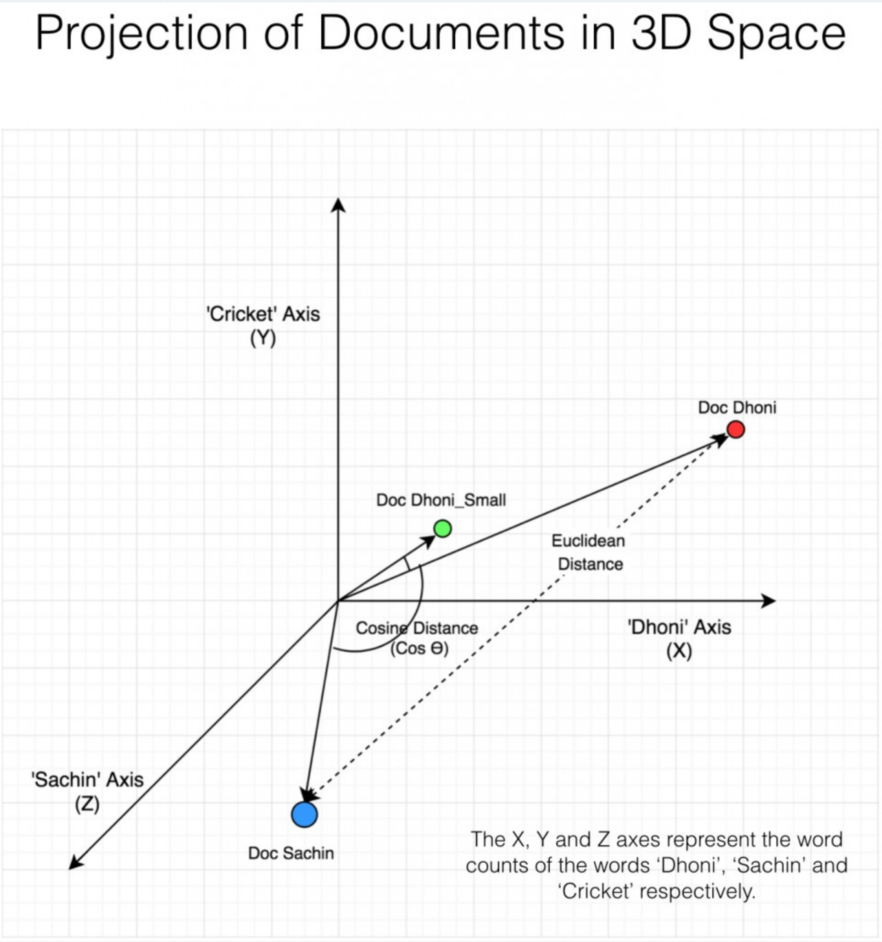


cosine similarity

cosine similarity function can be defined as follows:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

Cosine similarity



Siamese neural network

Siamese neural network

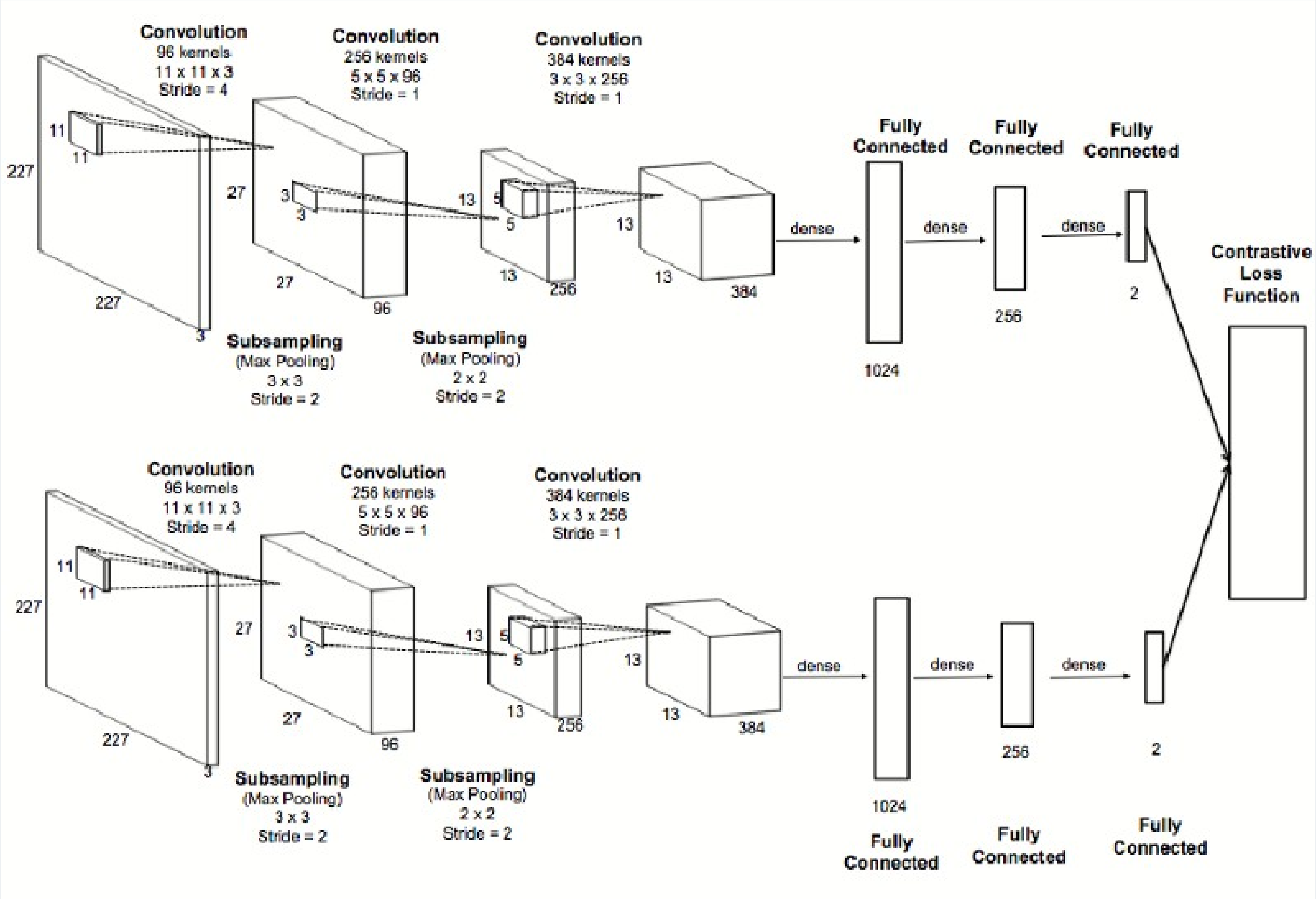
Siamese neural network are neural networks containing two or more identical sub network components.

The main idea behind siamese networks is that they can learn useful data descriptors that can be further used to compare between the inputs of the respective sub networks

Inputs can be anything from :

numerical data - image data - sequential data

Siamese neural network



Siamese neural network

Usually, siamese networks perform binary classification at the output.

loss functions :

cross-entropy loss: This loss can be calculated as

$$L = -y \log p + (1 - y) \log (1 - p)$$

where L is the loss function

y the class label (0 or 1) and p is the prediction. In order to train the network to distinguish between similar and dissimilar objects, we may feed it one positive and one negative example at a time and add up the losses:

L=+L +-L

siamese networks-Triplet loss

Triplet loss :

d is a distance function

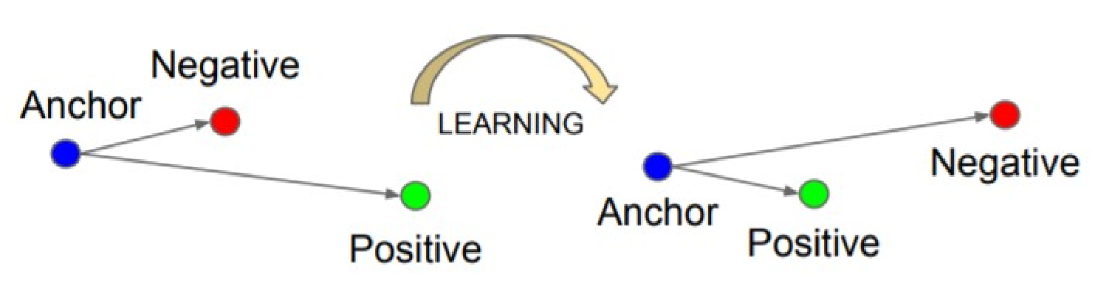
a is a sample of the data set

p is a random positive sample

n is a negative sample

m is an arbitrary margin

$$L = \max(d(a,p) - d(a,n) + m, 0)$$



separation between the positive and negative scores.

Weight Initialization

To break symmetries in the weigth space, we start from an asymmetric point

Weights are sampled from a N(0, 0.01), biases come from a N(0.5, 0.01)

Variability is crucial; we followed the specification for siamese networks found in

GAP

Global Average Pooling*

GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor.

A tensor with dimensions h×w×d is reduced in size to have dimensions 1×1×d. GAP layers reduce each h×w feature map to a single number by simply taking the average of all h*w values.

Sums out spatial information, so more robust to spatial translation (not really important here)

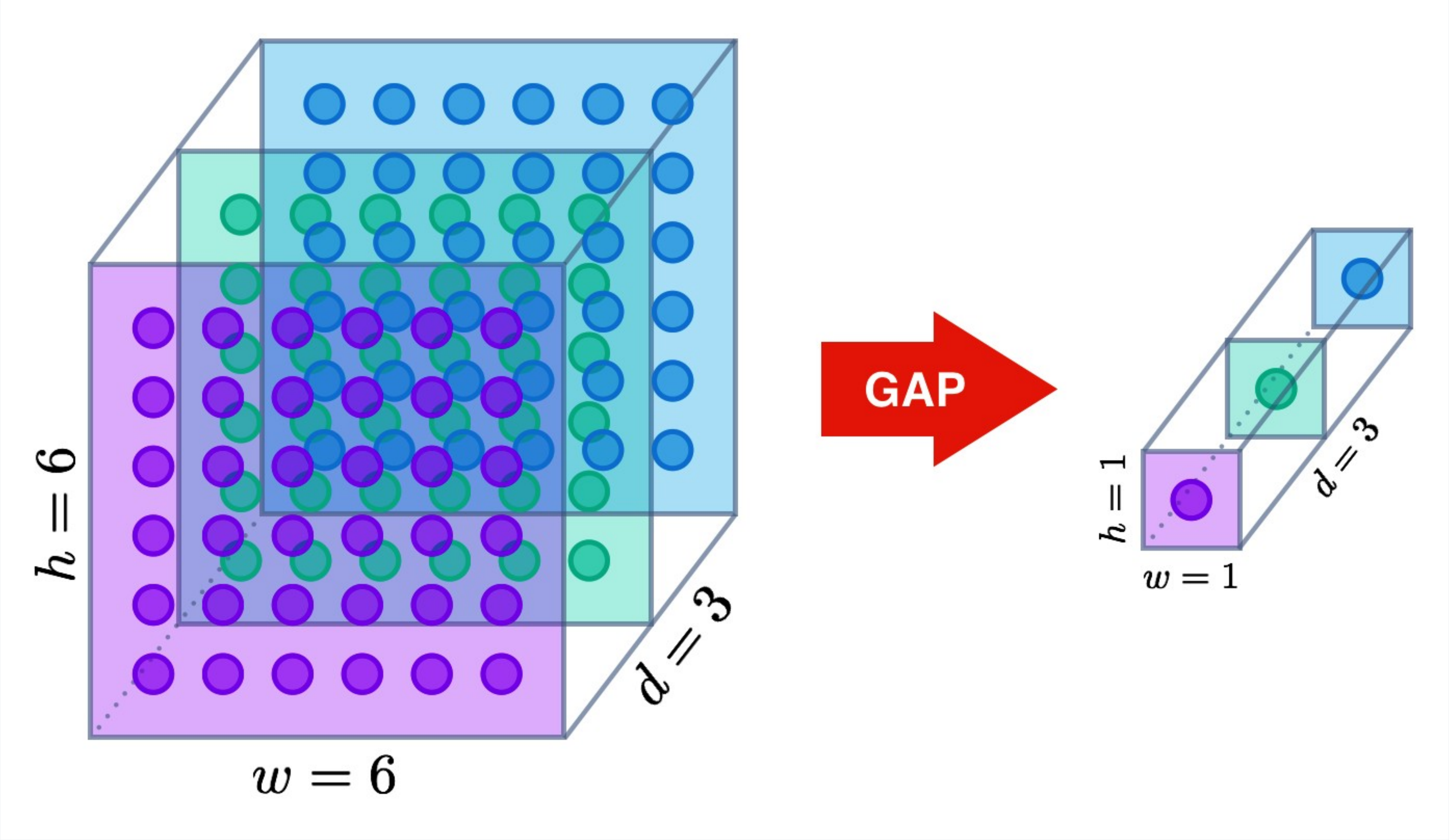
No parameter to optimize, so no risk of overfitting

No *pool_size* to tune (as in plain pooling)

No need to call *Flatten()*

In object detection, can be used to enforce correspondences between feature maps and categories

* M. Lin at al., *Network in Network*, 2013 (see section 3.2 especially)



$h = 6$
$w = 6$
$d = 3$
GAP
$h = 1$
$w = 1$
$d = 3$

Baseline

Validation Philosophy

We used a 10% hold-out validation strategy

A subset of the families, representing approx. 10% of all images was selected and set aside for validation. We made sure there was no family overlapping between training and validation sets

We used generators to feed images to the network, making use of the *fit_generator* functionality of Keras

For each batch, we randomly sample k = batch_size // 2 positive pairs, and the remainings will be negative. As a result, the network sees an equal amount of positive and negative examples (shuffled to make learning smoother)

We keep track of each sampled pair, so that the network is fed with all the training pairs only once. Same for validation

Metrics

Accuracy and AUC score were monitored during training and in validation

AUC was easily appended to the list of metrics by wrapping the *roc_auc_score* function of Sklearn into a Tensorflow tensor

Since the challenge required to use the AUC, we mostly focused our attentions on it

[explain AUC]

Feature Extractor

Both Facenet and VGG-Face have been tested as feature extractors

Since Facenet reported (slightly) superior performance in every instance, we will consider only it from now on (VGG-Face stays a valid feature extractor as well)

The reasons for this is rather intuitive: Facenet has been trained on an incredibly bigger number of images; moreover, Facenet employs an architecture that has proved superior on the ImageNet task

Fine-tuned the whole network. Freezing layers did not really help

Early Stopping

As a preliminary regularization technique, we applied early stopping on the model

Validation AUC score was monitored, with a patience of 5 epochs

EarlyStopping callback of Keras turned out useful

Loss and Optimizer

Binary cross-entropy was chosen as loss function

Adam optimizer was used

Initial learning rate was set to 0.00001

Moreover:

- Batch size was set to 32. 64+ did not fit in memory and 16 showed a too erratic behavior
- Number of epochs was set to 20 (without considering early stopping)

Learning Strategy

We first try to boost model capacity and achieve excellent training performance, then we will regularize to look after validation performance (as usually suggested)*

*See for example I. Courville, Y. Bengio, I. Goodfellow, *Deep Learning*, 2015

```
input_left (InputLayer)         (None, 224, 224, 3)  0
________________________________________________________________________________________________
input_right (InputLayer)        (None, 224, 224, 3)  0
________________________________________________________________________________________________
vggface_resnet50 (Model)        multiple             23561152    input_left[0][0]
                                                                 input_right[0][0]
________________________________________________________________________________________________
global_average_pooling2d_1 (Glo (None, 2048)         0           vggface_resnet50[1][0]
________________________________________________________________________________________________
global_average_pooling2d_2 (Glo (None, 2048)         0           vggface_resnet50[2][0]
________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 2048)         0           global_average_pooling2d_1[0][0]
                                                                 global_average_pooling2d_2[0][0]
________________________________________________________________________________________________
lambda_2 (Lambda)               (None, 2048)         0           lambda_1[0][0]
________________________________________________________________________________________________
multiply_1 (Multiply)           (None, 2048)         0           global_average_pooling2d_1[0][0]
                                                                 global_average_pooling2d_2[0][0]
________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 4096)         0           lambda_2[0][0]
                                                                 multiply_1[0][0]
________________________________________________________________________________________________
dense_together1 (Dense)         (None, 100)          409700      concatenate_1[0][0]
________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 100)          0           dense_together1[0][0]
________________________________________________________________________________________________
output (Dense)                  (None, 1)            101         dropout_1[0][0]
================================================================================================
Total params: 23,970,953
Trainable params: 23,917,833
Non-trainable params: 53,120
```

Baseline

We use siamese network as a baseline of our algorithm.

Batch size, 224,224,3 | VGGFACE OR FACE NET | Global average pooling | Size of tensor (2048)

+

Batch size, 224,224,3 | VGGFACE OR FACE NET | Global average pooling | Size of tensor (2048)

Baseline

Lambda1 : we took the absolute value of first image and then we substruct the second tensor

Lambda2: we square the lambda 1 to increase the difference

Multiply : the output of two tensors to get 1 output .

Concatenate : we put together multiply tensor with lambda 2

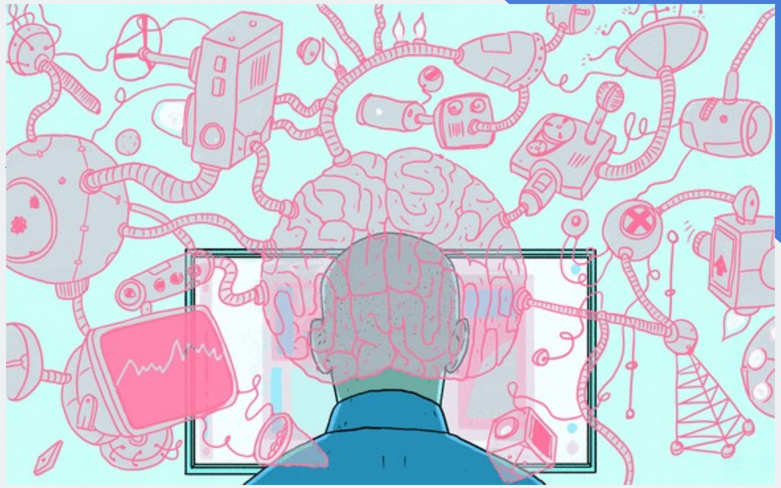
Add a **dense layer**

Add **dropout layer**

output: there is just one neuron with sigmoid activation function that tells the probablity (0,1) of being kin.

TOTAL PARAMETERS

# OF TOTAL PARAMETER (WEIGHT AND BIAS)	23 970 953
FACENET	2[illegible] 561 152



THANK YOU