OpenFace: A general-purpose face recognition library with mobile applications

Brandon Amos, Bartosz Ludwiczuk,[†] Mahadev Satyanarayanan

June 2016
CMU-CS-16-118

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]Poznan University of Technology

Abstract

Cameras are becoming ubiquitous in the Internet of Things (IoT) and can use face recognition technology to improve context. There is a large accuracy gap between today's publicly available face recognition systems and the state-of-the-art private face recognition systems. This paper presents our OpenFace face recognition library that bridges this accuracy gap. We show that OpenFace provides near-human accuracy on the LFW benchmark and present a new classification benchmark for mobile scenarios. This paper is intended for non-experts interested in using OpenFace and provides a light introduction to the deep neural network techniques we use.

We released OpenFace in October 2015 as an open source library under the Apache 2.0 license. It is available at: `http://cmusatyalab.github.io/openface/`

This research was supported by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Crown Castle, the Conklin Kistler family fund, Google, the Intel Corporation, and Vodafone. NVIDIA's academic hardware grant provided the Tesla K40 GPU used in all of our experiments. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to their employers or funding sources.

Keywords: face recognition, deep learning, machine learning, computer vision, neural networks, mobile computing

1 Introduction

Video cameras are extremely cheap and easily integrated into today's mobile and static devices such as surveillance cameras, auto dashcams, police body cameras, laptops, smartphones, GoPro, and Google Glass. Video cameras can be used to improve context in mobile scenarios. The identity of a person is a large part of context in humans and modulates what people say and how they act. Likewise, recognizing people is a primitive operation in mobile computing that adds context to applications such as cognitive assistance, social events, speaker annotation in meetings, and person of interest identification from wearable devices.

State-of-the-art face recognition is dominated by industry- and government-scale datasets. Example applications in this space include person of interest identification from mounted cameras and tagging a user's friends in pictures. Training is often an offline, batch operation and produces a model that can predict in hundreds of milliseconds. The time to train new classification models in these scenarios isn't a major focus because the set of people to classify doesn't change often.

Mobile scenarios span a different problem space where a mobile user may have a device performing real-time face recognition. The context of the mobile user and people around them provide information about who they are likely to see. If the user attends a meetup, the system should quickly learn to recognize the other attendees. Many people in the system are transient and the user only needs to recognize them for a short period of time. The time to train new classification models now becomes important as the user's context changes and people are added and removed from the system.

Towards exploring transient and mobile face recognition, we have created OpenFace as a general-purpose library for face recognition. Our experiments show that OpenFace offers higher accuracy than prior open source projects and is well-suited for mobile scenarios. This paper discusses OpenFace's design, implementation, and evaluation and presents empirical results relevant to transient mobile applications.

2 Background and Related Work

2.1 Face Recognition

Face recognition has been an active research topic since the 1970's [Kan73]. Given an input image with multiple faces, face recognition systems typically first run **face detection** to isolate the faces. Each face is **preprocessed** and then a low-dimensional **representation** (or **embedding**) is obtained. A low-dimensional representation is important for efficient classification. Challenges in face recognition arise because the face is not a rigid object and images can be taken from many different viewpoints of the face. Face representations need to be resilient to **intrapersonal** image variations such as age, expressions, and styling while distinguishing between **interpersonal** image variations between different people [Jeb95].

Jafri and Arabnia [JA09] provide a comprehensive survey of face recognition techniques up to 2009. To summarize the regimes, face recognition research can be characterized into feature-based and holistic approaches. The earliest work in face recognition was feature-based and sought

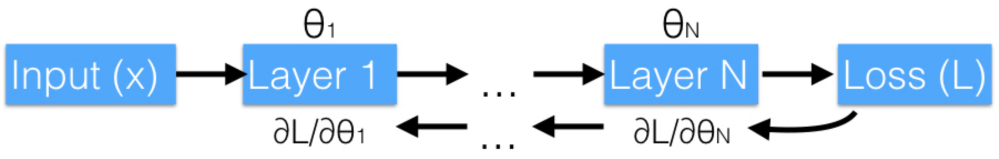


Figure 1: Training flow for a feed-forward neural network.

to explicitly define a low-dimensional face representation based on ratios of distances, areas, and angles [Kan73]. An explicitly defined face representation is desirable for an intuitive feature space and technique. However, in practice, explicitly defined representations are not accurate. Later work sought to use holistic approaches stemming from statistics and Artificial Intelligence (AI) that learn from and perform well on a dataset of face images. Statistical techniques such as Principal Component Analysis (PCA) [Hot33] represent faces as a combination of eigenvectors [SK87]. Eigenfaces [TP91] and fisherfaces [BHK97] are landmark techniques in PCA-based face recognition. Lawrence et al. [LGTB97] present an AI technique that uses convolutional neural networks to classify an image of a face.

Today's top-performing face recognition techniques are based on convolutional neural networks. Facebook's DeepFace [TYRW14] and Google's FaceNet [SKP15] systems yield the highest accuracy. However, these deep neural network-based techniques are trained with private datasets containing millions of social media images that are orders of magnitude larger than available datasets for research.

2.2 Face Recognition with Neural Networks

This section provides a deeper introduction to face recognition with neural networks from the techniques used in Facebook's DeepFace [TYRW14] and Google's FaceNet [SKP15] systems that are used within OpenFace. This section is intended to give an overview of the two most impactful works in this space and is not meant to be a comprehensive overview of the thriving field of neural network-based face recognition. Other notable efforts in face recognition with deep neural networks include the Visual Geometry Group (VGG) Face Descriptor [PVZ15] and Lightened Convolutional Neural Networks (CNNs) [WHS15], which have also released code.

A feed-forward neural network consists of many function compositions, or layers, followed by a loss function $\mathcal{L}$ as shown in Figure 1. The loss function measures how well the neural network models the data, for example how accurately the neural network classifies an image. Each layer i is parameterized by θ_i, which can be a vector or matrix. Common layer operations are:

- **Spatial convolutions** that slide a kernel over the input feature maps,
- **Linear** or fully connected layers that take a weighted sum of all the input units, and
- **Pooling** that take the max, average, or Euclidean norm over spatial regions.

These operations are often followed by a nonlinear activation function, such as Rectified Linear Units (ReLUs), which are defined by $f(x) = \max\{0, x\}$. Neural network training is a (nonconvex) optimization problem that finds a θ that minimizes (or maximizes) $\mathcal{L}$. With differentiable layers, $\partial L/\partial\theta_i$ can be computed with backpropagation. The optimization problem is then solved with a first-order method, which iteratively progress towards the optimal value based on $\partial L/\partial\theta_i$. See [BGC15] for a more thorough introduction to modern deep neural networks.

Figure 2 shows the logic flow for face recognition with neural networks. There are many face detection methods to choose from, as it is another active research topic in computer vision. Once a face is detected, the systems preprocess each face in the image to create a normalized and fixed-size input to the neural network. The preprocessed images are too high-dimensional for a classifier to take directly on input. The neural network is used as a feature extractor to produce a low-dimensional representation that characterizes a person's face. A low-dimensional representation is key so it can be efficiently used in classifiers or clustering techniques.

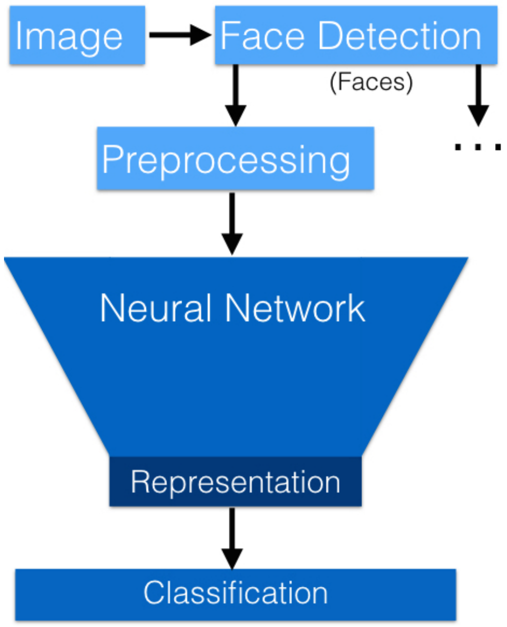


Figure 2: Logic flow for face recognition with a neural network.

DeepFace first preprocesses a face by using 3D face modeling to normalize the input image so that it appears as a frontal face even if the image was taken from a different angle. DeepFace then defines classification as a fully-connected neural network layer with a softmax function, which makes the network's output a normalized probability distribution over identities. The neural network predicts some probability distribution $\hat{p}$ and the loss function $\mathcal{L}$ measures how well $\hat{p}$ predicts the person's actual identity i.[1] DeepFace's innovation comes from three distinct factors: (a) the 3D alignment, (b) a neural network structure with 120 million parameters, and (c) training with 4.4 million labeled faces. Once the neural network is trained on this large set of faces, the final classification layer is removed and the output of the preceding fully connected layer is used as a low-dimensional face representation.

Often, face recognition applications seek a desirable low-dimensional representation that generalizes well to new faces that the neural network wasn't trained on. DeepFace's approach to this works, but the representation is a consequence of training a network for high-accuracy classification on their training data. The drawback of this approach is that the representation is difficult to use because faces of the same person aren't necessarily clustered, which classification algorithms can take advantage of. FaceNet's triplet loss function is defined directly on the representation. Figure 3 illustrates how FaceNet's training procedure learns to cluster face representations of the same person. The unit hypersphere is a high-dimensional sphere such that every point has distance

[1] Formally, this is done with the cross-entropy loss $\mathcal{L}(\hat{p}, i) = -\log \hat{p}_i$, where $\hat{p}_i$ is the ith element of $\hat{p}$.

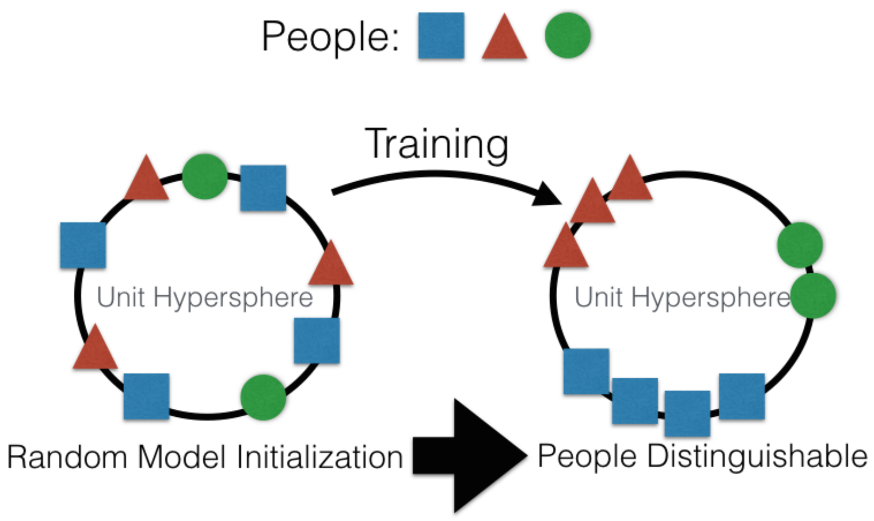


Figure 3: Illustration of FaceNet's triplet-loss training procedure.

1 from the origin. Constraining the embedding to the unit hypersphere provides a structure to a space that is otherwise unbounded. FaceNet's innovation comes from four distinct factors: (a) the triplet loss, (b) their triplet selection procedure, (c) training with 100 million to 200 million labeled images, and (d) (not discussed here) large-scale experimentation to find an network architecture. For reference, we formally define FaceNet's triplet loss in Appendix A.

2.3 Face Recognition in Mobile Computing

The mobile computing community studies and improves off-the-shelf face recognition techniques in mobile scenarios. Due to lack of availability, these studies often use techniques with an order of magnitude less accuracy than the state-of-the-art. Soyata et al. [SMF$^+$12] study how to partition an Eigenfaces-based face recognition system between the mobile device, cloudlet, and cloud. Hsu et al. [HC15] study the accuracies of cloud-based face recognition services as a drone's distance and angle to a person is varied. There has also been a rise of efficient GPU architectures for mobile devices, such as NVIDIA's Jetson TK1.

These studies and research directions are complementary to our studies in this paper. We do not study the impacts of executing techniques on different architectures, such as on an embedded GPU or offloaded to a surrogate. We instead present performance experiments showing that OpenFace's execution time is well-suited for mobile scenarios compared to other techniques.

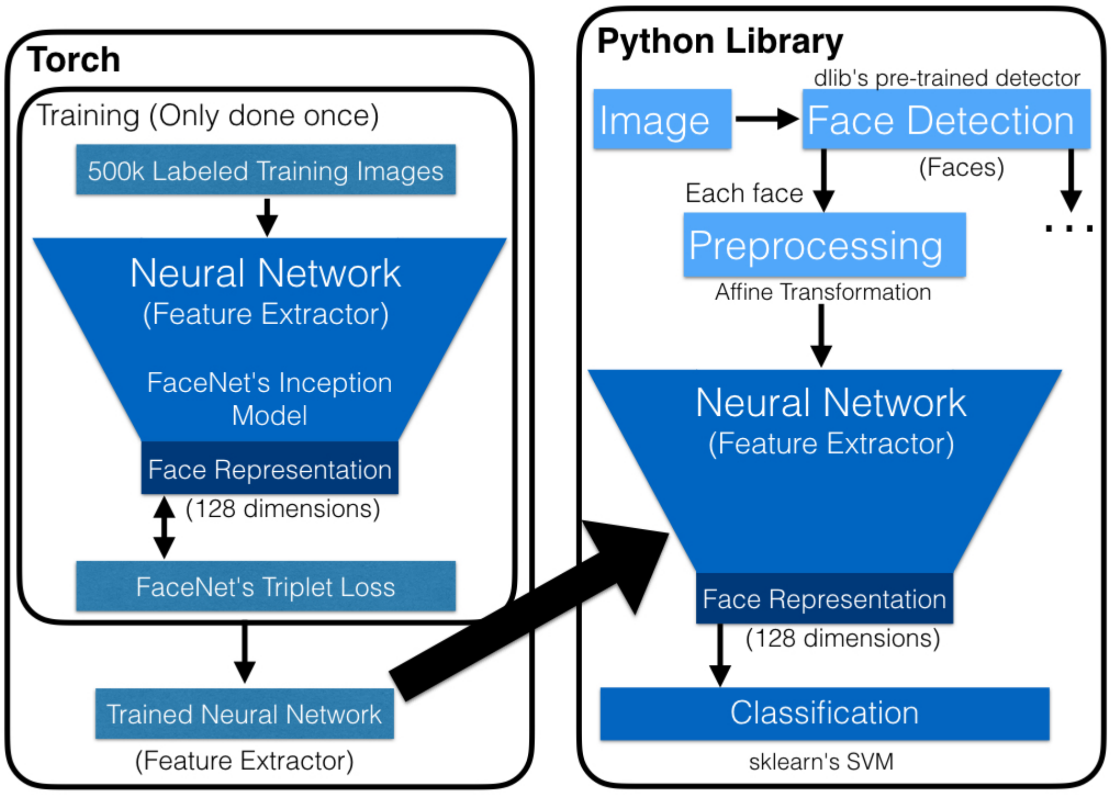


Figure 4: OpenFace's project structure.

3 Design and Implementation

Our interest in building OpenFace is in mobile scenarios where a user's real-time face recognition system adapts depending on context. Our key design consideration is a system that gives high accuracy with low training and prediction times.

OpenFace provides the logic flow presented in Figure 2 to obtain low-dimensional face representations for the faces in an image. Figure 4 highlights OpenFace's implementation. The neural network training and inference portions use Torch [CKF11], Lua [IDFCF96] and luajit [Pal08]. Our Python [VRDJ95] library uses numpy [Oli06] for arrays and linear algebra operations, OpenCV [B$^+$00] for computer vision primitives, and scikit-learn [PVG$^+$11] for classification. We also provide plotting scripts that use matplotlib [H$^+$07]. The project structure is agnostic to the neural network architecture and we currently use FaceNet's architecture [SKP15]. We use dlib's [Kin09] pre-trained face detector for higher accuracy than OpenCV's detector. We compile native C code with gcc [Sta89] and CUDA code with NVIDIA's LLVM-based [LA04] nvcc.

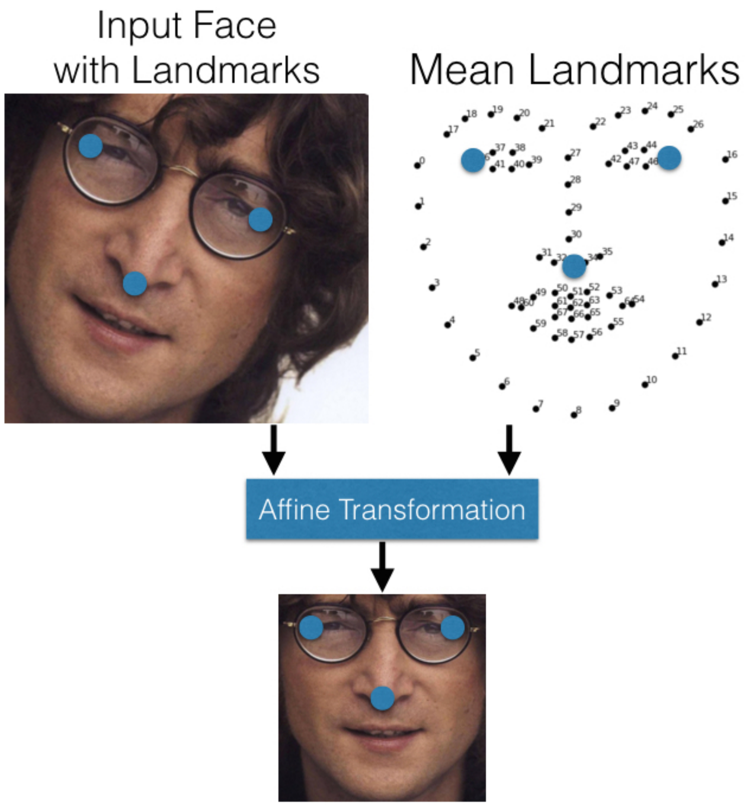


Figure 5: OpenFace's affine transformation. The transformation is based on the large blue landmarks and the final image is cropped to the boundaries and resized to 96 $\times$ 96 pixels.

3.1 Preprocessing: Alignment with an Affine Transformation

The face detection portion returns a list of bounding boxes around the faces in an image that can be under different pose and illumination conditions. A potential issue with using the bounding boxes directly as an input into the neural network is that faces could be looking in different directions or under different illumination conditions. FaceNet is able to handle this with a large training dataset, but a heuristic for our smaller dataset is to reduce the size of the input space by normalizing the faces so that they eyes, nose, and mouth appear at similar locations in each image. Normalization of faces is an active research topic in computer vision. Many modern techniques such as DeepFace [TYRW14] frontalize the face to a 3D model so the image appears as if the face is looking directly towards the camera. OpenFace uses a simple 2D affine transformation to make the eyes and nose appear in similar locations for the neural network input.

Figure 5 illustrates how the affine transformation normalizes faces. The 68 landmarks are detected with dlib's face landmark detector [Kin09], which is an implementation of Kazemi et al. [KS14]. Given an input face, our affine transformation makes the eye corners and nose close to the mean locations. The affine transformation also resizes and crops the image to the edges of the landmarks so the input image to the neural network is 96 $\times$ 96 pixels.

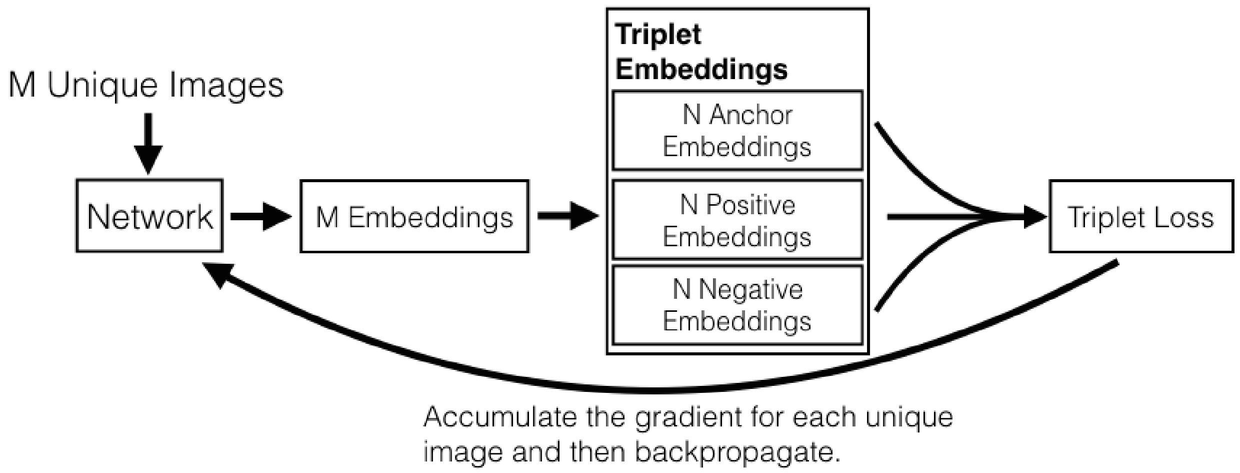


Figure 6: OpenFace's end-to-end network training flow.

3.2 Training the Face Representation Neural Network

Training the neural network requires a lot of data. FaceNet [SKP15] uses a private dataset with 100M-200M images and DeepFace [TYRW14] uses a private dataset with 4.4M images. OpenFace is trained with 500k images from combining the two largest labeled face recognition datasets for research, CASIA-WebFace [YLLL14] and FaceScrub [NW14].

The neural network component from Figure 2 maps a preprocessed (aligned) image to a low-dimensional representation. The best neural network structure for a task is an unsolved and thriving research topic in computer vision. OpenFace uses a modified version of FaceNet's nn4 network presented in Appendix C. nn4 is based on the GoogLeNet [SLJ+15] architecture and our modified nn4.small2 variant reduces the number of parameters for our smaller dataset. OpenFace uses FaceNet's triplet loss as defined in Section 2.2 so the network provides an embedding on the unit hypersphere and Euclidean distance represents similarity.

Figure 6 shows how we train networks. We map unique images from a single network into triplets. The gradient of the triplet loss is backpropagated back through the mapping to the unique images. In each mini-batch, we sample at most P images per person from Q people in the dataset and send all $M \approx PQ$ images through the network in a single forward pass on the GPU to get M embeddings. We currently use $P = 20$ and $Q = 15$. We take all anchor-positive pairs to obtain $N = Q\binom{P}{2}$ triplets. We compute the triplet loss and map the derivative back through to the original image in a backwards network pass. If a negative image is not found within the α margin[2] for a given anchor-positive pair, we do not use it.

Appendix B provides a description of our original network training technique that takes an order of magnitude longer than the one presented here.

[2]The α margin is defined in Appendix A.

Technique	Accuracy
Human-level (cropped) [KBBN09]	0.9753
Eigenfaces (no outside data) [TP91][3]	0.6002 ± 0.0079
FaceNet [SKP15]	0.9964 ± 0.009
DeepFace-ensemble [TYRW14]	0.9735 ± 0.0025
OpenFace (ours)	0.9292 ± 0.0134

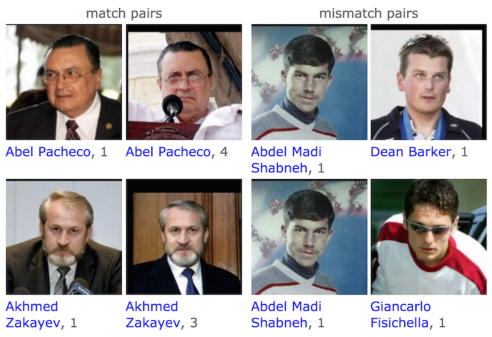


Figure 7: LFW accuracies and example image pairs.

4 Evaluation

Our evaluation studies OpenFace's accuracy and performance in comparison to other face recognition techniques. The LFW dataset [HRBLM07] is a standard benchmark in face recognition research and Section 4.1 presents OpenFace's accuracy on the LFW verification experiment. Section 4.2 presents a new classification benchmark using the LFW dataset for transient mobile scenarios.

All experiments in this section use the nn4.small2.v1 OpenFace model described in Appendix C. The identities in our neural network training data does not overlap with the LFW identities.

4.1 LFW Verification

The LFW verification experiment [HRBLM07] predicts whether pairs of images are of the same person. Figure 7 shows example pairs and accuracies. The LFW has 13,233 images from 5,750 people and this experiment provides 6,000 pairs broken into ten folds.

The accuracy in the restricted protocol is obtained by averaging the accuracy of ten experiments. The data is separated into ten equally-sized folds and each experiment trains on nine folds and computes the accuracy on remaining testing fold. The OpenFace results are obtained by computing the squared Euclidean distance on the pairs and labeling pairs under a threshold as being the same person and above the threshold as different people. The best threshold on the training folds is used as the threshold on the remaining fold. In nine out of ten experiments, the best threshold is 0.99. Figure 7 compares OpenFace's accuracy with other techniques. Unlike the modern deep neural network-based techniques, the Eigenfaces result uses no outside data.

The verification threshold can be varied and plotted as an (receiver operating characteristic) ROC curve as shown in Figure 8. The ROC curve shows the tradeoffs between the TPR and FPR. The perfect ROC curve would have a TPR of 1 everywhere, which is where todays state-of-the-art industry techniques are nearly at. The area under the curve (AUC) is the probability the classifier will rank randomly chosen faces of the same person higher than randomly chosen faces of different people. Every curve is an average of the curves obtained from thresholding each fold of data. The OpenFace folds are included to illustrate the variability of these experiments. The OpenBR curve

[3]Result is from `http://vis-www.cs.umass.edu/lfw/results.html#eigenfaces`

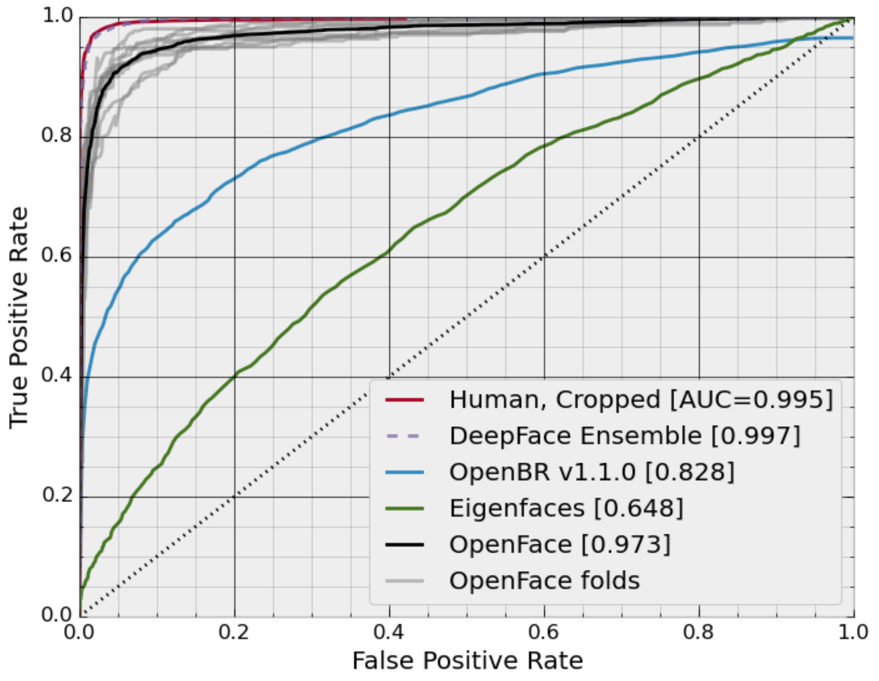


Figure 8: ROC curve on the LFW benchmark with area under the curve (AUC) values.

is from their LFW script[4] and the others are from the LFW results page. Kumar et al. [KBBN09] provides human-level performance results. The cropped version crops LFW images around the faces in the images to reduce contextual information. The FaceNet curve has not been released.

These results show that OpenFace's accuracy is close to to the accuracy of state-of-the-art deep learning techniques.

4.2 LFW Classification

Many scenarios using face recognition involve classifying who a person is, not just if two faces are the same. This section presents an experiment that measures the classification accuracies on a subset of LFW images. We present results comparing OpenCV's [B$^+$00] face recognition techniques (eigenfaces [TP91], fisherfaces [BHK97], and Local Binary Pattern Histograms (LBPH) [AHP04]) to OpenFace's.

Figure 9 overviews the experiment setup. Person i corresponds to the person in the LFW with the ith most images. Then, 20 images are sampled from the first N people. If 20 images aren't provided of a person, all of their images are used. Next, the sampled data is split randomly ten times by putting 90% of the images in the training set and 10% of the images in the testing set. The random number seed should be initialized to the same value for sampling different experiments. A

[4]`https://github.com/biometrics/openbr/blob/v1.1.0/scripts/evalFaceRecognition-LFW.sh`

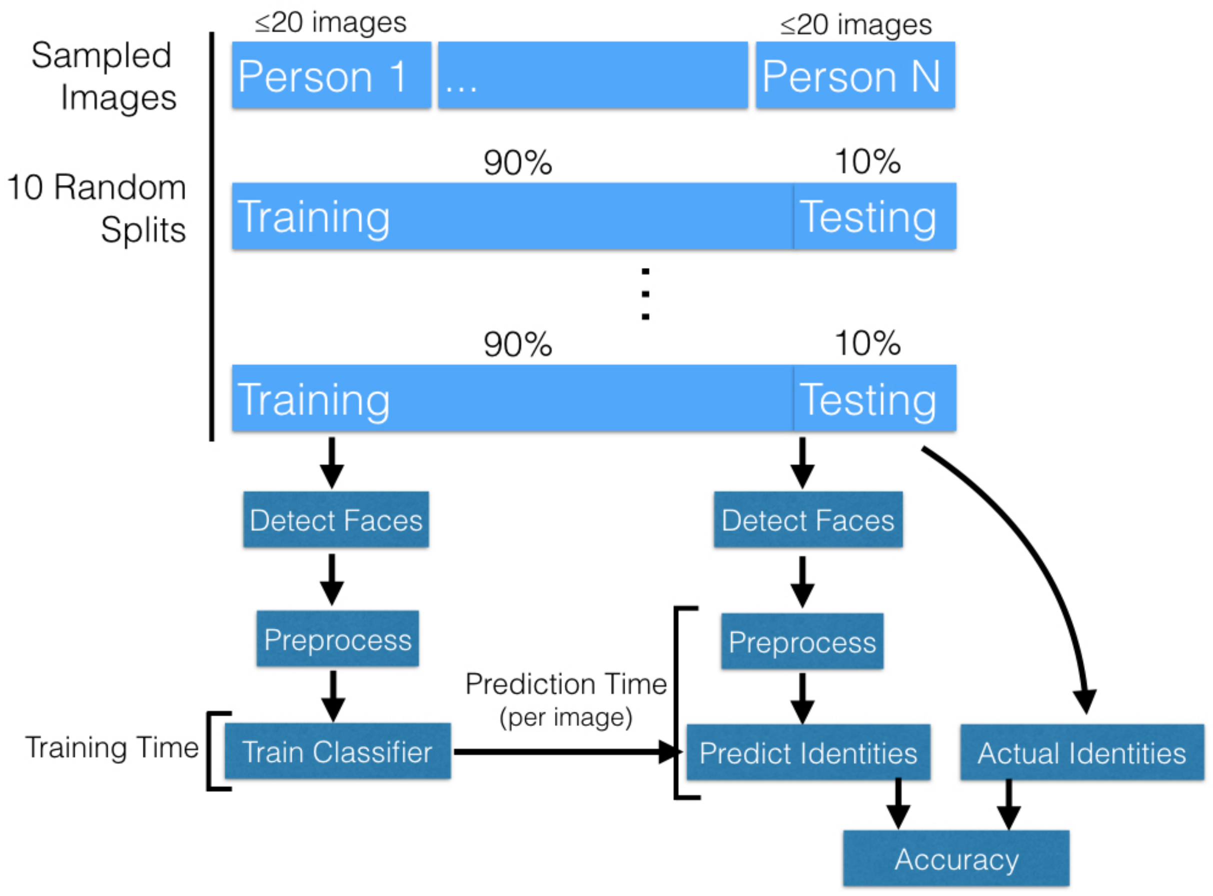


Figure 9: Overview of the LFW classification accuracy and performance benchmark.

classifier is trained on the training set and the accuracy is obtained from predicting the identities in the testing set.

Measuring the runtime performance of training classifiers and predicting who new images belong to is an important consideration for mobility. On wearable devices, the performance of predicting who a face belongs to is important so there isn't a noticeable lag when a user looks at another person. Face detection isn't included as part of the prediction time in our analysis because it is the same between all techniques. The prediction time includes the preprocessing and prediction times.

Users may also want to add or remove identities from their recognition system for transient scenarios. This benchmark studies how long it takes to re-train a classifier. We assume the face and identity data is data that has already been collected and pre-processed and that the user has the ability to choose a subset of identities to classify. The training time only reflects the time to train the classifier.

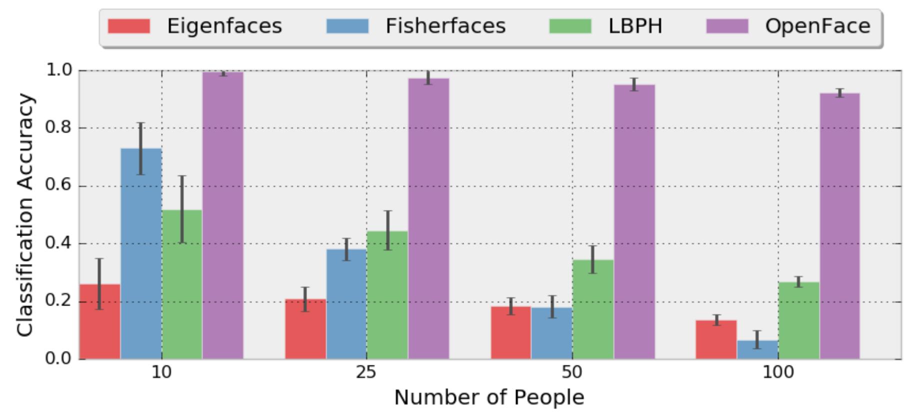
All of OpenCV's techniques have the same interface. The preprocessing converts the image a grayscale. OpenFace's preprocessing in this context means the affine transformation for alignment followed by the neural network forward pass for the 128-dimensional representation. OpenFace classification uses a linear SVM with a regularization weight of 1, which consistently performs the same or better than other regularization weights and RBF kernels.

Figure 10 presents our experimental results from classifying between 10 and 100 people on an 8-core Intel Xeon E5-1630 v3 @ 3.70GHz CPU with OpenBLAS and a NVIDIA Tesla K40 GPU. Figure 10a shows that adding more people decreases the accuracy and that OpenFace always has the highest accuracy by a large margin. Figure 10b shows that adding more people increases the training time. OpenFace's SVM consistently has the fastest training time. Only one result for OpenFace is shown here because the numbers do not include the neural network representation preprocessing time and the SVM library only uses the CPU. Figure 10c shows the per-image prediction times. The execution time of eigenfaces, fisherfaces, and LBPH slightly increase as more faces are added while OpenFace's prediction time remains constant. OpenFace's prediction involves a neural network forward pass that takes substantially more time than the PCA and histogram-based techniques. Executing on a GPU instead of a CPU offers slight performance improvements.

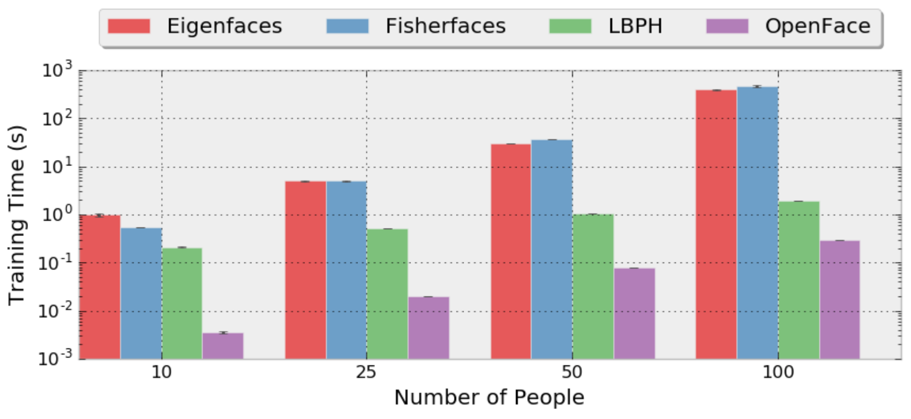
5 Conclusion

This paper presents OpenFace, a face recognition library. OpenFace is open sourced under the Apache 2.0 license and can be obtained from `http://cmusatyalab.github.io/openface/`. We trained a network on the largest datasets available for research, which is one order of magnitude smaller than DeepFace [TYRW14] and two orders of magnitude smaller than FaceNet [SKP15], the state-of-the-art private datasets that have been published. We show competitive accuracy and performance results on the LFW verification benchmark despite our smaller training dataset. We introduce a LFW classification benchmark and show competitive performance results on it.

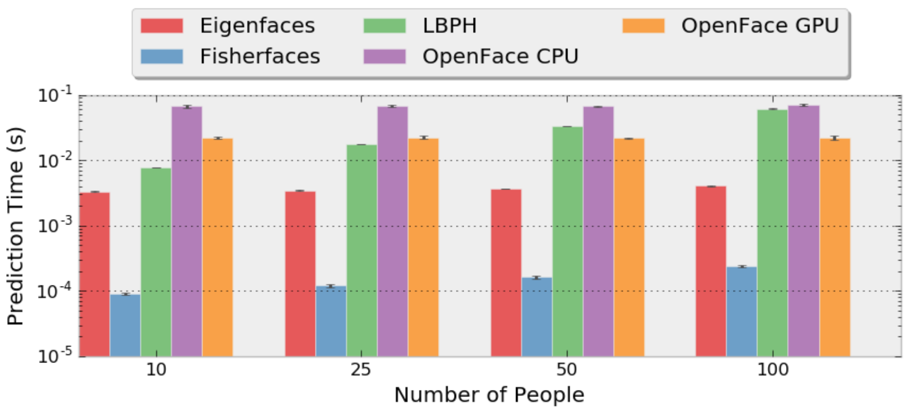
We intend to maintain OpenFace as a library that stays updated with the latest deep neural network architectures and technologies for face recognition.



(a) Classification accuracy.



(b) Training times.



(c) Per-image prediction times.

Figure 10: Accuracy and performance comparisons between OpenFace and prior non-proprietary face recognition implementations (from OpenCV).

Acknowledgments

We are grateful for the insightful discussions and strong contributions that have made OpenFace possible. Hervé Bredin helped us remove a redundant face detection after the affine transformation for alignment. The Torch ecosystem and community have provided many of the neural network components, including Alfredo Canziani's implementation[5] of FaceNet's loss function. Nicholas Lèonard quickly merged our pull requests to dpnn[6] that modified the inception layer for FaceNet's structure. Francisco Massa and Andrej Karpathy quickly released Torch's `nn.Normalize` layer after we expressed interest in using it. Early in the project Soumith Chintala provided helpful Torch advice and Davis King helped with dlib usage. We also thank Zhuo Chen, Kiryong Ha, Khalid Elgazzar, Jan Harkes, Wenlu Hu, J. Zico Kolter, Padmanabhan Pillai, Wolfgang Richter, Daniel Siewiorek, Rahul Sukthankar, and Junjue Wang for insightful discussions and feedback.

This research was supported by the National Science Foundation (NSF) under grant number CNS-1518865. Additional support was provided by Crown Castle, the Conklin Kistler family fund, Google, the Intel Corporation, and Vodafone. NVIDIA's academic hardware grant provided the Tesla K40 GPU used in all of our experiments. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and should not be attributed to their employers or funding sources.

[5] `https://github.com/Atcold/torch-TripletEmbedding`
[6] `https://github.com/Element-Research/dpnn`

References

[AHP04] Timo Ahonen, Abdenour Hadid, and Matti Pietikäinen. Face recognition with local binary patterns. In *Computer vision-eccv 2004*, pages 469–481. Springer, 2004.

[B$^+$00] Gary Bradski et al. The opencv library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

[BBB$^+$93] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[BGC15] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[BHK97] Peter N Belhumeur, João P Hespanha, and David J Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):711–720, 1997.

[CKF11] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[H$^+$07] John D Hunter et al. Matplotlib: A 2d graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[HC15] Hwai-Jung Hsu and Kuan-Ta Chen. Face recognition on drones: Issues and limitations. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, DroNet '15, pages 39–44, New York, NY, USA, 2015. ACM.

[Hot33] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

[HRBLM07] Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[IDFCF96] Roberto Ierusalimschy, Luiz Henrique De Figueiredo, and Waldemar Celes Filho. Lua-an extensible extension language. *Softw., Pract. Exper.*, 26(6):635–652, 1996.

[JA09] Rabia Jafri and Hamid R Arabnia. A survey of face recognition techniques. *JIPS*, 5(2):41–68, 2009.

[Jeb95] Tony S Jebara. *3D pose estimation and normalization for face recognition*. PhD thesis, McGill University, 1995.

[Kan73] Takeo Kanade. Picture processing system by computer complex and recognition of human faces. *Doctoral dissertation, Kyoto University*, 3952:83–97, 1973.

[KBBN09] Neeraj Kumar, Alexander C Berg, Peter N Belhumeur, and Shree K Nayar. Attribute and simile classifiers for face verification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 365–372. IEEE, 2009.

[Kin09] Davis E King. Dlib-ml: A machine learning toolkit. *The Journal of Machine Learning Research*, 10:1755–1758, 2009.

[KS14] Vahid Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[LA04] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Code Generation and Optimization, 2004. CGO 2004. International Symposium on*, pages 75–86. IEEE, 2004.

[LGTB97] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[NW14] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. *IEEE International Conference on Image Processing (ICIP)*, 265(265):530, 2014.

[Oli06] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[Pal08] Mike Pall. The luajit project. *Web site: http://luajit. org*, 2008.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PVZ15] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. *Proceedings of the British Machine Vision*, 1(3):6, 2015.

[SK87] Lawrence Sirovich and Michael Kirby. Low-dimensional procedure for the characterization of human faces. *JOSA A*, 4(3):519–524, 1987.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[SLJ$^+$15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[SMF$^+$12] Tolga Soyata, Rajani Muraleedharan, Colin Funai, Minseok Kwon, and Wendi Heinzelman. Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000059–000066. IEEE, 2012.

[Sta89] Richard M Stallman. Using and porting the gnu compiler collection. *Free Software Foundation*, 51:02110–1301, 1989.

[TP91] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1701–1708. IEEE, 2014.

[VRDJ95] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[WHS15] Xiang Wu, Ran He, and Zhenan Sun. A lightened cnn for deep face representation. *arXiv preprint arXiv:1511.02683*, 2015.

[YLLL14] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014.

A FaceNet's Triplet Loss Formulation

This section presents the FaceNet [SKP15] triplet loss formulation. Let $f_\Theta(I)$ be a neural network parameterized by Θ that maps an image I onto a unit hypersphere of dimension m. A triplet consists of an anchor image a, a positive image of the same person p, and a negative image of a different person n. To achieve the clustering illustrated in Figure 3, the distance between the anchor and positive should be less than the distance between the anchor and negative. Adding a threshold α, all triplets should satisfy

$$||f_\Theta(a) - f_\Theta(p)||_2^2 + \alpha < ||f_\Theta(a) - f_\Theta(n)||_2^2,$$

where $||z||_2^2 = \sum_i z_i^2$ is the squared Euclidean norm of z. The loss function to make triplets meet this condition is

$$\mathcal{L}(a,p,n) = \left[||f_\Theta(a) - f_\Theta(p)||_2^2 + \alpha - ||f_\Theta(a) - f_\Theta(n)||_2^2\right]_+,$$

where $[z]_+ = \max\{0, z\}$. In this paper we use $\alpha = 0.2$ and $m = 128$ as suggested in the FaceNet paper.

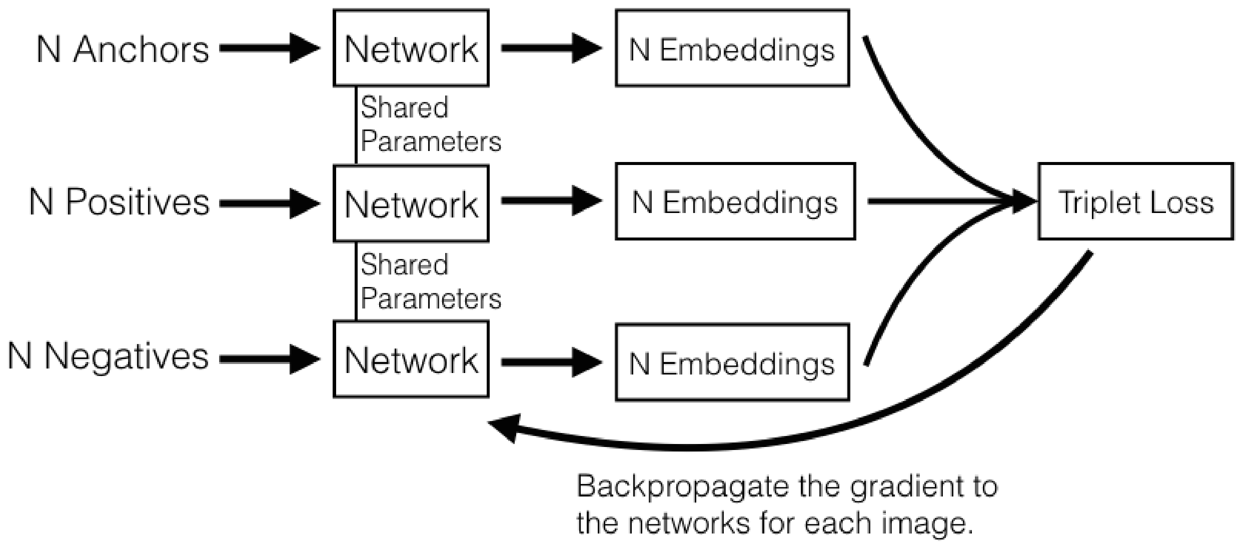


Figure 11: Original (suboptimal) OpenFace triplet loss training technique.

B Original (Sub-Optimal) Network Training Technique

Our original attempt at training a network used an architecture similar to training Siamese networks [BBB+93]. Figure 11 illustrates how we used three separate neural networks with shared weights that process triplets. The gradient of the loss function can be taken with respect to the anchors, positives, and negatives and back-propagated separately through each network. After training is done, a single network can be taken and used for prediction because the weights are shared.

While using three separate networks is straightforward, duplicate images are inefficiently processed multiple times. We can only send 100 triplets through three networks at a time on our Tesla K40 GPU with 12GB of memory. In each mini-batch, suppose we sample 20 images per person from 15 people in the dataset. Selecting every combination of 2 images from each person for the anchor and positive images and then selecting a negative from the remaining images gives $15\binom{20}{2} = 2850$ triplets. This requires 29 forward and backward passes to process 100 triplets at a time, even though there are only 300 unique images. In attempt to remove redundant images, our original technique randomly selects two images from each person for the anchor and positive.

Section 3.2 presents our current training technique that improves the time to train networks by an order of magnitude.

Table 1: The OpenFace nn4.small2 network definition.

type	output size	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
conv1 $(7 \times 7 \times 3, 2)$	$48 \times 48 \times 64$						
max pool + norm	$24 \times 24 \times 64$						m 3×3, 2
inception (2)	$24 \times 24 \times 192$		64	192			
norm + max pool	$12 \times 12 \times 192$						m 3×3, 2
inception (3a)	$12 \times 12 \times 256$	64	96	128	16	32	m, 32p
inception (3b)	$12 \times 12 \times 320$	64	96	128	32	64	ℓ_2, 64p
inception (3c)	$6 \times 6 \times 640$		128	256,2	32	64,2	m 3×3, 2
inception (4a)	$6 \times 6 \times 640$	256	96	192	32	64	ℓ_2, 128p
inception (4e)	$3 \times 3 \times 1024$		160	256,2	64	128,2	m 3×3, 2
inception (5a)	$3 \times 3 \times 736$	256	96	384			ℓ_2, 96p
inception (5b)	$3 \times 3 \times 736$	256	96	384			m, 96p
avg pool	736						
linear	128						
ℓ_2 normalization	128						

C OpenFace's Neural Network Definition

Table 1 shows our nn4.small2 neural network model, which is a version of the nn4 model from FaceNet [SKP15] hand-tuned to have less parameters. We include it here for completeness and reproducibility. Each row is a layer in the neural network and the last six columns indicate the parameters of pooling or the inception layers from [SLJ+15]. Dimensionality reductions to N dimensions after pooling is denoted with "Np". The normalization is local response normalization. OpenFace's full definition of the nn4.small2 model, in addition to FaceNet's nn2 and nn4 models, is publicly available at:
`https://github.com/cmusatyalab/openface/tree/master/models/openface`