Determining familial resemblances from face images

Kaushik Sah (ksah@stanford.edu)

(computer vision)

Problem statement and motivation

Looking for similarity in faces of people is a difficult and broad task. The subjectivity might stem from varying degrees of similarity between facial features. Identifying the same individual person in different circumstances can be viewed as a stricter similarity problem, whereas determing if a child resembles either or both parents is a wider similarity problem. Assessing whether two people share the same ancestry is an even wider similarity problem. In all cases the solution can be challenged by obscuration or evolving facial features, e.g. with age. Engineered features or metrics have been used in the past [4]. In recent works [3] [2] deep neural networks have been used. In this project, the objective is to explore and improve upon current methods in deep learning on facial images to asses similarity between family members. The scope will be limited to the classification problem spanning one generation: *whether a child resembles his/her biological parents*? A further extension of the solution will be to assess similarities between persons in same ancestry. Development in this area can be helpful in biometric or forensic verifications, anthropological studies, social media tagging, etc. A more ambitious application would be to understand how subtle genetic or environmental factors manifest as variations in facial features.

Keywords: familial resemblance, kinship verification, genetic similarity, computer vision

Labeled datasets

A. UB- Kinface Dataset v2 http://www1.ece.neu.edu/~yunfu/research/Kinface/Kinface.htm

B. Family101 http://chenlab.ece.cornell.edu/projects/KinshipClassification/index.html

C. RFIW2018 https://web.northeastern.edu/smilelab/RFIW2018/ – the most comprehensive dataset till date.

Approach (methods and algorithms)

I. Introduction: Past and current approaches to solve the problem

After studying current state of the art methods [6] [3] [4], around kinship verification, it is evident that as quality dataset size representing familial relationships increases, deep-learning based methods can outperform conventional metric-based approaches. Carefully labeled dataset, Families in the Wild (FIW), [C] is the first large scale dataset released for kinship recognition and verification tasks. This dataset has cropped and labelled face photos of members of 1000 famous families (celebrities) capturing 11 relationships across 3 generations, for example, parent child pairs such as father-son (F-S), father-daughter (F-D), mother-son (M-S) and mother-daughter (M-D), sibling pairs and grandparent-grandchild. The motivation for separating relationship pairs come from research in both psychology and computer vision revealing that different familial relations render different familial features which motivated researchers to model different relationship types independently [3]. The current database contains 656,954 face pairs across these relationships.

II. Choice of model architecture and loss function:

The most promising approach so far, [3] [4] is to use transfer learning on a pre-trained face classification network like VGG. For this project, we propose a Siamese network with triplet loss as a suitable approach to find positive similarities and negative dissimilarities between three set of input faces, namely, anchor, positive, negative. The goal of the learning problem is to minimize a similarity distance between anchor and positive examples while maximizing the distance between anchor and negative example. For training the Siamese network, we use transfer learning with a pretrained network (in this case vgg-16) to encode each (224x224x3) face image into a 4096-dimensional feature vector also called as encoding vector in this paper. Next, few custom fully connected layers are added. The intent is to train the custom part with mini-batches of triplet examples. Since vgg has ~134 million parameters (until the 4096-encoding layer), it may not be trivial to train and fine-tune the entire network. For the custom part we use fully connected three-layer architecture. The output is a 128-dimensional feature vector called here as 'descriptor'. If the model can learn well, each descriptor will encode the right features need for familial recognition task. It is to be noted that we tried to formulate the problem as binary classification (family vs non-family) on the triplet training examples and used standard binary-cross entropy loss but found that the model was not able to learn effectively, hence formulated the problem with triplet loss function.

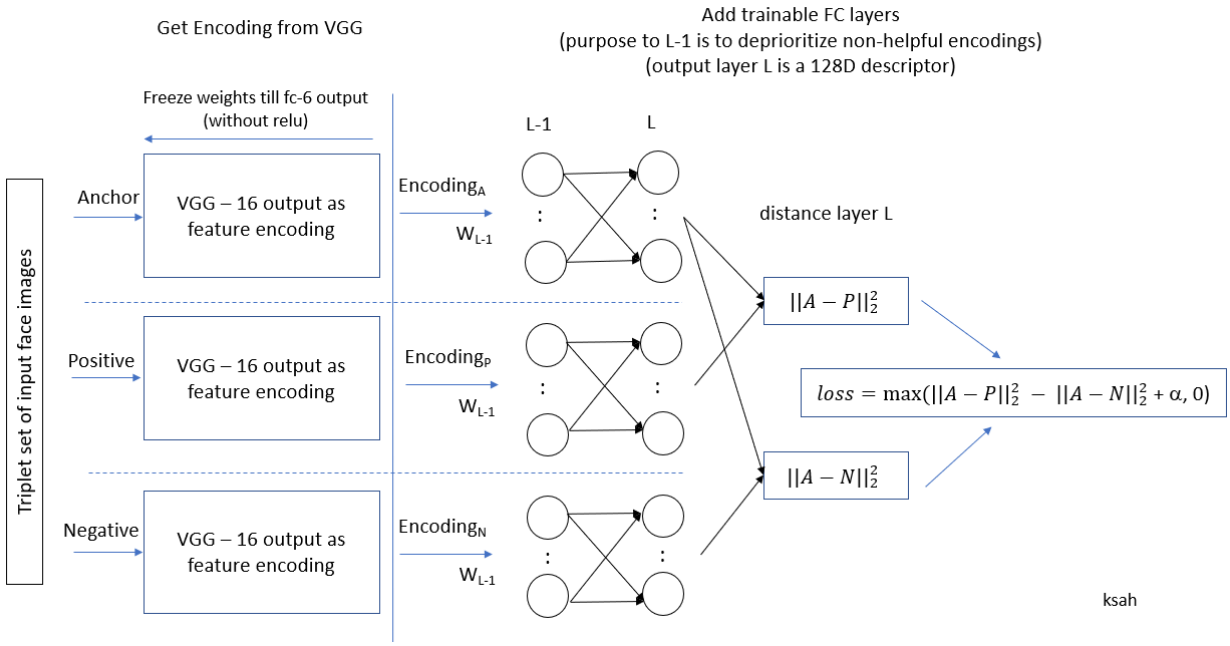

Figure 1: Siamese Model Architecture based on VGG-16

III. Dataset preparation:

'Family In the Wild' kinship dataset [C] was downloaded and parsed. Brief overview of dataset structure is shown in figure 2. We are only concerned with relationships spanning only one generation for the time being, hence the data hierarchy of original set was restructured into parent-child positive pairs, with each sub-folder containing exactly one-pair of images. This structure also makes it feasible to generate triplet image set dynamically in batches using a python generator. During the data prep process, we also augmented the dataset to have multiple positive combinations of each pair by age, pose, etc.

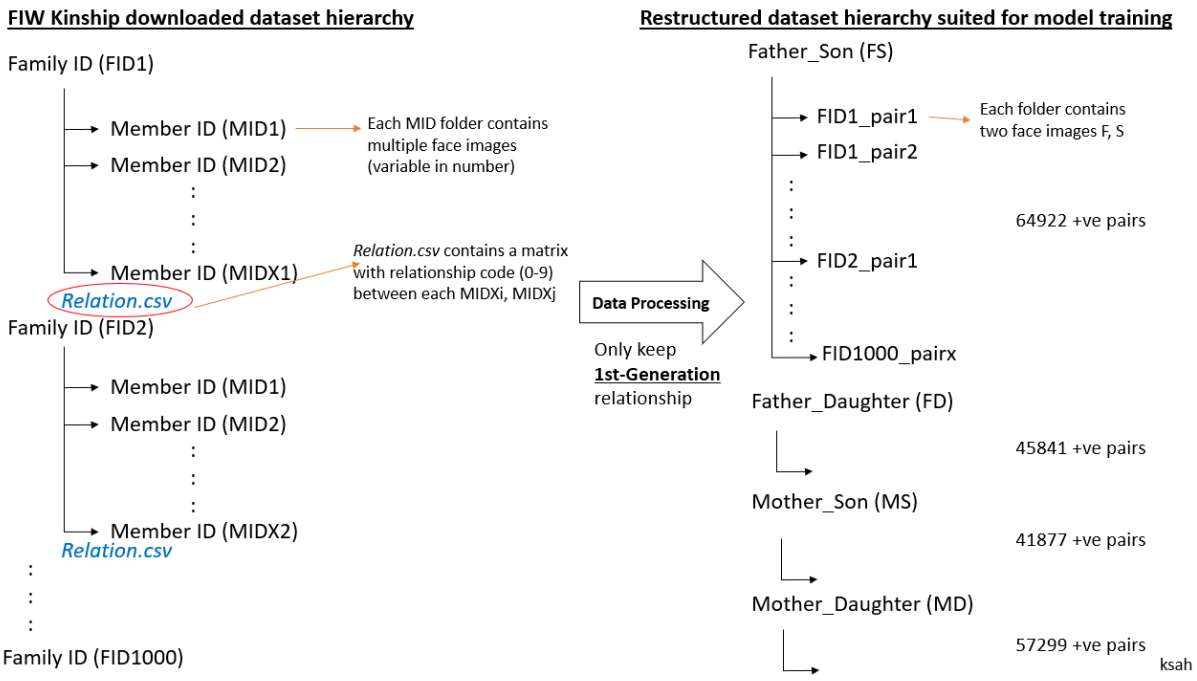

Figure 2: Originally downloaded and post processed dataset structure

IV. Model Training:

Model implementation was done in Keras with TensorFlow as backend. VGG-16 pre-trained network on ImageNet is used as base model. We remove end layers which are used for classification purposes and retain until layer 'fc6' which provide a 4096D encoding for an image input of (226, 226, 3). It is important that we remove the 'relu' activation layer associated with 'fc6' and retain only the logits. Then, we freeze the trained weights of this base network. There are ~134 M parameters in this model which are not intended to be fine tuned at this moment. Next, we add a dense (1024) hidden layer to fc6 output encoding, followed by another dense (512) hidden layer. This adds ~4.7 M trainable parameters. The intent is to have the network learn weights such that important features are given higher importance for this image matching task. At the output of this layer we have (512,1) activation vector. We have used 'relu' non-linearity and 'batch-normalization' for the two layers. Finally, we add a dense 128 output layer with no activation and L2 normalization. We can think of this output layer as a "descriptor layer" outputting (128,1) features for each input image. Since we are passing three inputs (A, P, N) as a triplet, we need three descriptors to compute the loss function. We use a Siamese network approach wherein only one model is used to get multiple outputs, one each for each of the input images. The difference between A, P and A, N is used to calculate triplet loss as per the loss formula shown in figure 1. The parameter alpha was set to 0.5 in our experiment. For generating the triplets, we used python generator with a given batch size allowed by memory of the hardware system. Since the data folders were pre-processed to each contain exactly one pair of related images per family, A and P are chosen together when a family folder is selected. For the negative example we chose N, again randomly, but such that it does not come from same family. We did not implement hard triplet generation at this point by considering dynamically trained model while generating the batches. We relied on the fact that if the model is trained long enough, there will be instances when hard triplets are seen by the model. Similar approach is taken for cross validation and entire validation set is used at the end of each epoch. An important point to

note is that (A, P, N) sequence from one training/validation instance to next will not be same. This can lead to small oscillations in the training and validation loss.

Specifically, for this training instance, we trained the model on mother-daughter (M-D) relation dataset. Same model can be trained for other relationships using respective datasets. For M-D case, we used 33,502 positive pairs in the training set and 8,375 positive pairs in the validation set. One of the major challenges faced during training is the large oscillation of training loss. The loss decreases steadily but after certain number of epochs, it starts to oscillate with large amplitude. Typical

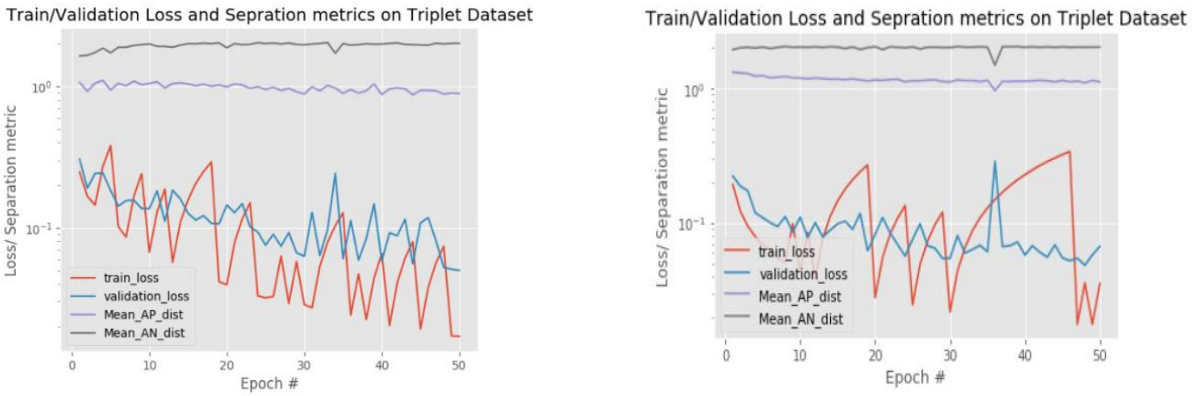

Figure 3a (left): Train and Validation loss with learning rate = 0.005, decay = 0.00025 and no gradient clipping;
Figure 3b (right): learning rate = 0.0005, decay 0.0025 and gradient clipping value = 1.0

example is shown in figure 3a (***note the log y scale***). We tried several learning rates with adjusting decay but did not result in significant success in reducing oscillations. We also tried gradient clipping suspecting exploding gradient problem which helped slightly for validation loss but made training loss oscillations wider. We finally chose the model hyperparameters corresponding to figure 3a (learning rate = 0.005, decay = 0.00025 and no gradient clipping in Keras). Training was done for 50 epochs. Both training and validation batch size was set to 64, which was maximum possible as per the hardware system. The model with lowest validation loss was saved as best model. System hardware included Intel i7 CPU, 32GB RAM, and 1x GeForce RTX 2080 8GB GPU. Average training and validation time for 50 epochs was 12.5 hours.

V. Results:

As seen from figure 3a, there is a quite good separation (on an average) between A-P distance and A-N distance. For concreteness, we tested the performance on each triplet in validation set (as there are 8375 positive examples, 8375 triplets can be constructed). Figure 4 shows the variation of A-P distance, A-N distance and triplet loss as a box plot. This shows us the degree of separation between positive and negative examples in detail. Even though the negative examples are constructed

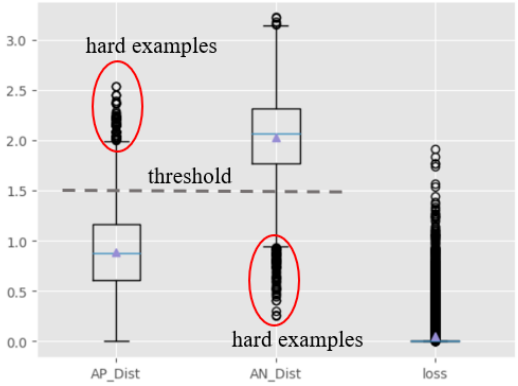

Figure 4: A-P, A-N and loss for each triplet example in validation set. Examples encircled in red are hard examples.

randomly, and A-N distance can vary run to run, A-P distances remain the same. From the figure a threshold of 1.5 is a good choice for verification (binary classification) tasks relating to mother-daughter relationship. The examples circled in red are hard examples that the model has not been able to separate and hence contribute to a positive loss value. Model performance can be summarized into a confusion matrix as shown in figure 5. We can now calculate the **accuracy**, **precision**, **recall** and **F1** values as **89.7%**, **88.4%**, **91.2%** and **89.8 %** respectively. **This is much higher than published accuracy value of 73.8% in [4]. The references [2] and [4] which present latest performance on the dataset must be studied in detail to ensure that out results and published results are obtained in similar manner, since neither paper use triplet loss function to train the model.**

Finally, we look at some of the examples from the validations set, including the hard examples that the model misclassified in figure 6. 6a shows two successful examples formulated using triplets (random selection). 6b shows positive examples that were

		Actual	
Predicted		M-D (Y)	M-D (N)
	M-D (Y)	7639	995
	M-D (N)	736	7380

N_Triplet = 8375

Figure 5: Confusion Matrix on validation set for mother-daughter relationship using a threshold of 1.5

misclassified to be negative (i.e. A_P > 1.5). Successful (A_P < = 1.5) versions are also shown from the validation set. (By data augmentation we have multiple positive combinations of each pair by age, pose, etc.)

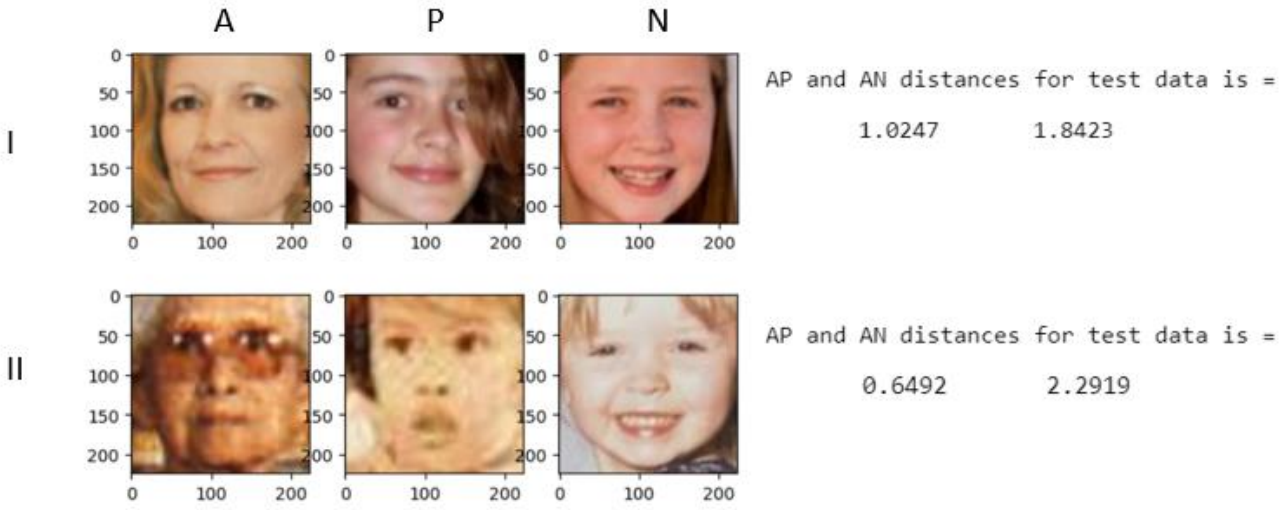

Figure 6a: Examples of successful verification

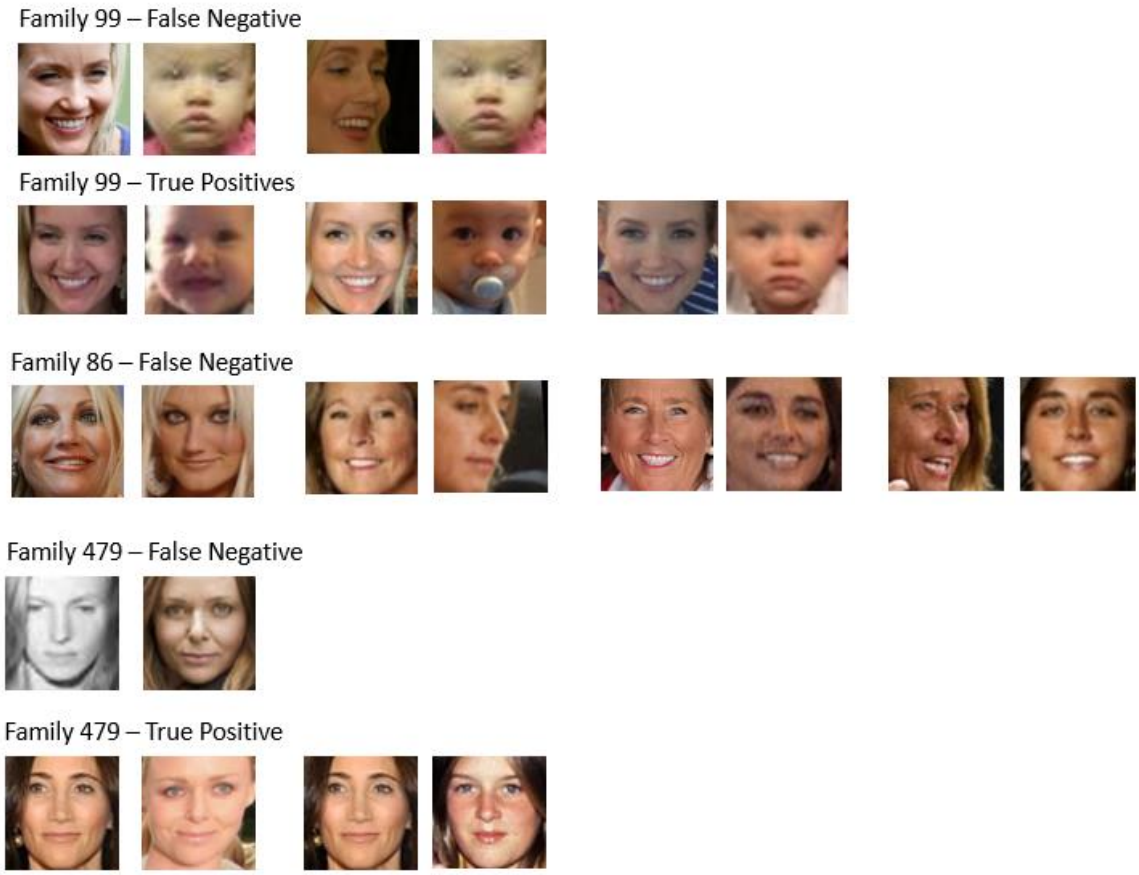

Figure 6b: Examples of successful and non-successful classifications from validation set

VI. Conclusions:

In this project, we described and trained a deep learning model for determining familial relationship using face images. We used a Siamese model with triplet loss function building on a pretrained VGG-16 model. The model was trained only for mother-daughter relationship and achieved good results on the validation set.

VII. Further work:

1. Decrease the oscillations in both training and validation loss (although some oscillations are expected due to changing negative examples in each epoch). One of the methods towards this will require checking the gradients for oscillations and take remedial action such as scheduled gradient clipping. In this project, we only implemented fixed clipping which did not yield any better result than no clipping.
2. Dynamically generate hard triplets using already trained model into account.
3. Train models for other relationships such as F – S, F – D, M – S.
4. Test model on different datasets (not used in training/ validation) to see robustness
5. Compare performance to published results in detail and attempt publication.

VIII. Acknowledgements:

I would like to thank Prof. Andrew Ng, the entire CS230 teaching staff and Stanford University for designing this fantastic course. As a non-practitioner in this field, this course has given me sound conceptual background, practical insights, hands on programming experience especially from the project (such as creating custom batches, Keras layers, training loops, metric constructions, etc.). I have got good confidence to be able to execute my own projects in the field of deep learning.

References

[1] Database: Families in the wild - https://web.northeastern.edu/smilelab/fiw/, https://github.com/visionjo/FIW_KRT

[2] J. Robinson, M. Shao, Y. Wu, H. Liu, Y. Fu, "Visual Kinship Recognition of Families in the Wild", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 40, No. 11, Nov 2018.

[3] J. Robinson, M. Shao, H. Zhao, Y. Wu, T. Gillis, Y. Fu. "Recognizing Families In the Wild (RFIW): Data Challenge Workshop in conjunction with ACM MM 2017," ACM Multimedia Conference: Workshop on RFIW (2017)

[4] E. Dahan, Y. Keller, "SelfKin: Self Adjusted deep Model for Kinship Verification", Computer Vision and Pattern Recognition, Sep 2018. doi: arXiv:1809.08493v1

[5] S. Wang, J. Robinson, and Y. Fu. "Kinship Verification on Families in the Wild with Marginalized Denoising Metric Learning," in IEEE Automatic Face and Gesture Recognition

[6] A. Puthenputhussery, Q. Liu, C. Liu, "SIFT FLOW BASED GENETIC FISHER VECTOR FEATURE FOR KINSHIP VERIFICATION", ICIP 2016.

[7] J. Robinson, M. Shao, Y. Wu, and Y. Fu. "Families In the Wild (FIW): large-scale kinship image database and benchmarks." in ACM on Multimedia Conference (2016)

[8] F. Schroff et al., FaceNet: A unified embedding for face recognition and clustering, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2015.

[9] O. Parkhi, A. Vedaldi, A. Zisserman, "Deep face Recognition", VGG, University of Oxford, 2015

[10] G. Koch, et. Al., Siamese Neural Networks for One-shot Image Recognition, Dept. of Computer Science, University of Toronto.

A note on code implementations

The entire programming code is written by me as my own work, with inspirations drawn from CS230 class, Stack Overflow, and following tutorials:
https://towardsdatascience.com/one-shot-learning-with-siamese-networks-using-keras-17f34e75bb3d
https://medium.com/@crimy/one-shot-learning-siamese-networks-and-triplet-loss-with-keras-2885ed022352

Appendix

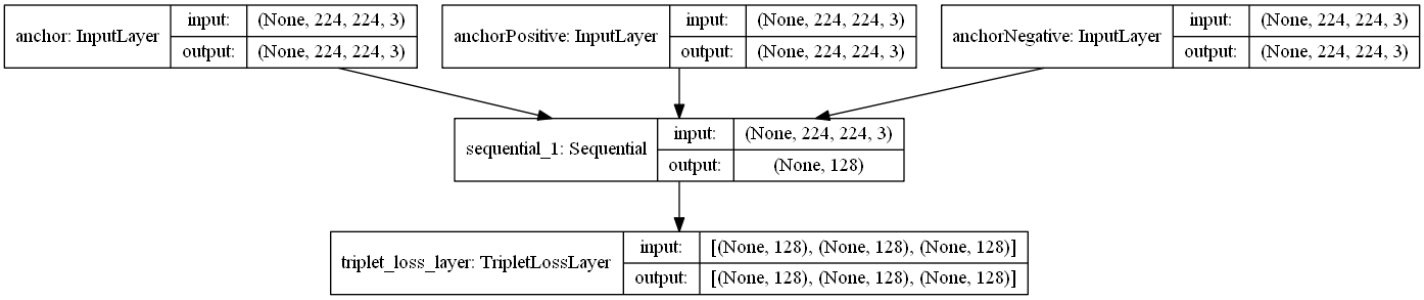

Figure A: Siamese Model with triplet loss leading to 128-dimensional output (descriptor) for input image of dimension (224,224,3)

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 224, 224, 3)	0
conv1_1 (Conv2D)	(None, 224, 224, 64)	1792
conv1_2 (Conv2D)	(None, 224, 224, 64)	36928
pool1 (MaxPooling2D)	(None, 112, 112, 64)	0
conv2_1 (Conv2D)	(None, 112, 112, 128)	73856
conv2_2 (Conv2D)	(None, 112, 112, 128)	147584
pool2 (MaxPooling2D)	(None, 56, 56, 128)	0
conv3_1 (Conv2D)	(None, 56, 56, 256)	295168
conv3_2 (Conv2D)	(None, 56, 56, 256)	590080
conv3_3 (Conv2D)	(None, 56, 56, 256)	590080
pool3 (MaxPooling2D)	(None, 28, 28, 256)	0
conv4_1 (Conv2D)	(None, 28, 28, 512)	1180160
conv4_2 (Conv2D)	(None, 28, 28, 512)	2359808
conv4_3 (Conv2D)	(None, 28, 28, 512)	2359808
pool4 (MaxPooling2D)	(None, 14, 14, 512)	0
conv5_1 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_2 (Conv2D)	(None, 14, 14, 512)	2359808
conv5_3 (Conv2D)	(None, 14, 14, 512)	2359808
pool5 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc6 (Dense)	(None, 4096)	102764544
fc6/relu (Activation)	(None, 4096)	0
fc7 (Dense)	(None, 4096)	16781312
fc7/relu (Activation)	(None, 4096)	0
fc8 (Dense)	(None, 2622)	10742334
fc8/softmax (Activation)	(None, 2622)	0

Total params: 145,002,878
Trainable params: 145,002,878
Non-trainable params: 0

Figure B: Pretrained VGG-16 network on ImageNet. We use truncated output at layer 'fc6 (Dense)' to feed our custom model