Face Verification with Depthwise Separable Convolutions and the Triplet Loss*

Eduardo S. C. de Souza

Universidade de São Paulo, São Carlos SP, Brazil

Abstract. This article explores the use of depthwise separable convolutions and the triplet loss for human face bounding box detection and face verification. The dataset used is the CelebA, consisting of photographic images of celebrities. Experiments executed show that depthwise separable convolutions work considerably well for bounding box detection (1.8%; 2.01 times faster), but combined with the triplet loss do not perform exceptionally (85% accuracy; 1.83 times faster).

Keywords: Depthwise Separable Convolutions · Triplet Loss · Face Verification

1 Introduction

In the field of computer vision, convolutional neural networks have become one of the most popular and effective methods, starting its popularity when AlexNet[4] won the ImageNet challenge[8]. It started a currently ongoing trend for such challenges where, in order to achieve better metrics, applicants increase the size, depth, and consequently the computational cost of their models. This results in highly accurate models, but with an unreasonable cost for many applications. As such, it has been a trend in research to take more into consideration the computational cost, and develop ways to optimize CNNs.

In this context, depthwise separable convolutions[10,2] have been widely used. This is due to their considerable reduction in both the number of parameters and computational costs of a CNN, without great loss in accuracy and other such metrics.

1.1 The Face Verification Problem

This article aims to apply depthwise separable convolutions to the face verification problem. This problem consists of being able to determine if two images have the face of the same person in them, or different people, without necessarily determining who those people are. It is a variation of the face classification problem, where given a single image, it must be determined to whom the face belongs to in a predetermined set of people.

* Supported by Universidade de São Paulo

To achieve this goal, the CNNs will be trained using the Triplet loss function. This is a loss function where, as it decreases, the distances between images belonging to the same class (in this scenario, the same person) decrease, and images belonging to the different classes (in this scenario, different people) increase. As such, the distance between two images is the variable that determines whether or not those two images are of the same person.

2 Depthwise Separable Convolutions

2.1 Number of Weights

Firstly, the idea of the depthwise separable convolution must be explained in detail. In a regular convolution, the input tensor has rank 3, with the axis being the height, the width, and the number of channels of an input image. Then, each convolutional filter also has rank 3, with the axis being the height and width of the filter and the number of channels of the input image. Each filter will then apply a convolution on the entire input image, each resulting in one output channel of a convolutional layer.

Let H_i, W_i, D_i, H_f, W_f, N_f be the height of the input image, the width of the input image, the depth (or number of channels) of the input image, the height of the filters, the width of the filters and the number of filters respectively. In a regular convolutional layer, the number of non-bias weights can be expressed by equation 1.

$$N_f * H_f * W_f * D_i \tag{1}$$

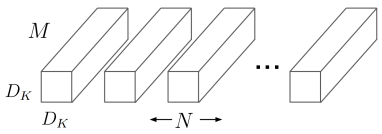

Fig. 1. Shape of a regular convolution's weights

For a depthwise separable convolution, the input tensor has the same shape. However, the convolution is applied in two steps. The first step consists of executing a convolution on each input channel separately, meaning that both the filters and the images they are applied to will have a depth of 1. Then, the resulting image of each filter will be used as an input channel for the next step. It consists of executing several regular convolutions of size 1x1, known as a pointwise convolution. The resulting image of each pointwise convolution will be one output channel of the depthwise separable convolutional layer.

Image 2, originally from [11], is a good visualization of the way that a depthwise separable convolution is applied.

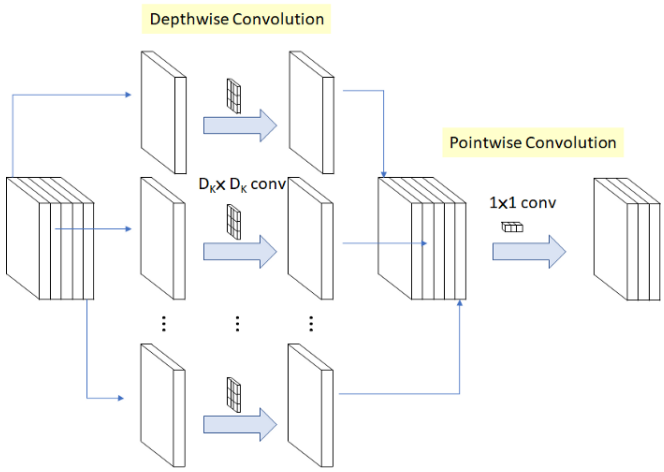

Fig. 2. Both steps of the depthwise separable convolution

The number of non-bias weights for a depthwise separable convolution can be expressed by equation 2. The first element of the sum refers to the weights on the first step of the process, and the second element to the second step.

$$D_i * H_f * W_f + D_i * N_f \tag{2}$$

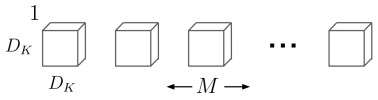

Fig. 3. Shape of a depthwise separable convolution's first step's weights

Therefore, the ratio between the number of weights can be expressed by equation 3, showing a considerable reduction in the number of weights.

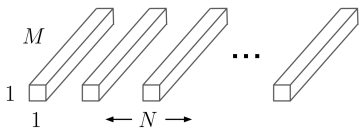

Fig. 4. Shape of a depthwise separable convolution's second step's weights

$$\frac{D_i * H_f * W_f + D_i * N_f}{N_f * H_f * W_f * D_i} = \frac{D_i * (H_f * W_f + N_f)}{N_f * H_f * W_f * D_i} = \\ \frac{H_f * W_f + N_f}{H_f * W_f * N_f} = \frac{1}{N_f} + \frac{1}{H_f * W_f} \tag{3}$$

Images 1, 3 and 4, originally from [2], are a good visualization of the weights in each type of convolution.

2.2 Number of Operations

Although the number of parameters is certainly important, in many scenarios the number of operations is considerably more so. Assuming a stride of 1, and sufficient padding so that the output has the same height and width as the input, the number of sums and multiplications executed by a regular convolution can be expressed by equation 4.

$$N_f * H_i * W_i * H_f * W_f * D_i \tag{4}$$

With the same assumptions, the number of operations executed by a depthwise separable convolution can be expressed by equation 5. The first element of the sum refers to the operations on the first step of the process, and the second element to the second step.

$$D_i * H_i * W_i * H_f * W_f + N_f * H_i * W_i * D_i \tag{5}$$

And the ratio between the number of operations can be expressed by equation 6, showing a considerable reduction in the computational cost.

$$\frac{D_i * H_i * W_i * H_f * W_f + N_f * H_i * W_i * D_i}{N_f * H_i * W_i * H_f * W_f * D_i} =$$

$$\frac{D_i * (H_i * W_i * H_f * W_f + N_f * H_i * W_i)}{N_f * H_i * W_i * H_f * W_f * D_i} =$$

$$\frac{H_i * W_i * H_f * W_f + N_f * H_i * W_i}{N_f * H_i * W_i * H_f * W_f} = \tag{6}$$

$$\frac{H_i * W_i * H_f * W_f}{N_f * H_i * W_i * H_f * W_f} + \frac{N_f * H_i * W_i}{N_f * H_i * W_i * H_f * W_f} =$$

$$\frac{1}{N_f} + \frac{1}{H_f * W_f}$$

3 Triplet Loss

The triplet loss[9] is a loss function that aims to move feature vectors belonging to the same class closer to each other. It does this by a fairly simple method, which is by explicitly minimizing the distance between samples of the same class, and maximizing the distance between samples of different classes. Its equation is the number 7.

$$l(A, P, N) = max(0, d(f(A), f(P)) - d(f(A), f(N)) + \alpha) \tag{7}$$

Where A is the anchor sample, P is the positive sample, which means it should be clustered with A (belongs to the same class), and N is the negative sample, which means it should be moved away from A (belongs to a different class). f is the function that generates a feature vector from an input, such as a CNN. d is a function that calculates the distance between 2 vectors, commonly being the euclidean distance or the euclidean distance squared. α is a constant, which serves as a margin, or a minimum difference, between the positive and negative examples. It exists so that in cases where the negative distance is just slightly greater than the positive the loss doesn't become zero. The maximum with 0 serves as a lower limit for the function.

The triplet loss decreases either when the distance between the anchor and the positive example decrease, or when the distance between the anchor and the negative example increase. As such, as its value reduces through the training process, the feature vectors start to become clustered based on their classes. The effects of triplet loss reduction is shown in image 5, originally from [9].

In the context of the face verification problem, the anchor would be an image of someone, the positive sample would be a different image of the same person,

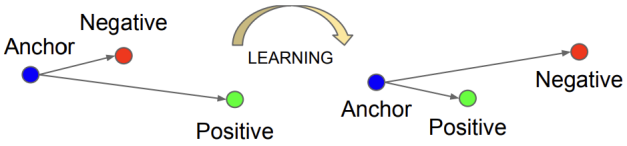

Fig. 5. Effects of training with the triplet loss

and the negative sample a tertiary image of a different person. It is expected then, that a feature extractor with a low triplet loss is capable of generating highly representative feature vectors of human faces.

3.1 Types of Triplets

Triplets can be separated into three groups: easy, hard, and semi-hard.

- The easy triplets occur when the distance from the anchor to the negative sample is already greater than the distance from the anchor to the positive sample plus the α margin.
- The hard triplets occur when the distance to the negative sample is smaller than the one to the positive sample.
- The semi-hard triplets occur when the distance to the negative sample is greater than the one to the positive sample, but not greater than the distance to the positive sample plus the α margin.

This distinction is easily visualized on image 6, originally from [6].

4 Experiments

The main goal of this article is to explore the use of both the previously mentioned techniques on the face verification problem. The problem is broken down into two steps.

1. **Bounding Box Detection and Segmentation:** The first step is to create a model capable of detecting the bounding box of someone's face on an image. It is limited to a single face per image.
2. **Verification Through Feature Vector Distance:** The next step is to create a model capable of generating feature vectors for the segmented faces. The distance between 2 extracted feature vectors is then used to differentiate between the same person and different people.

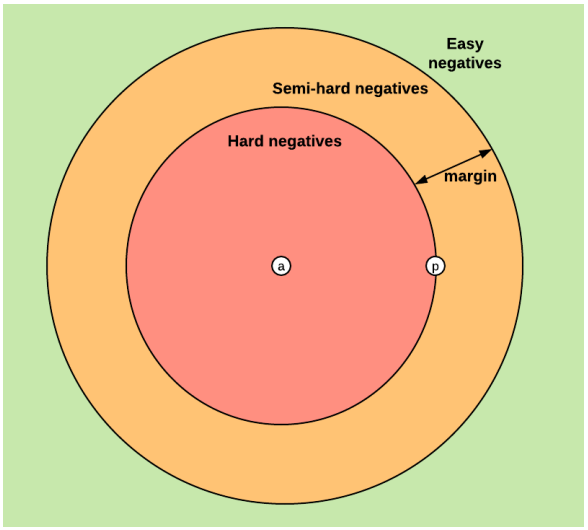

Fig. 6. Differences in types of triplets

4.1 Dataset and Tools Used

For both steps the CelebA[5] dataset is used, since it has all the annotation needed, and is extremely large and varied. The version of the dataset used is the in-the-wild one, since it represents a real-world scenario. For the verification step, the images are cropped using the ground-truth bounding boxes, so as not to propagate errors from the segmentation model. Also, only images of individuals with more than 5 images of them in the dataset were used. The implementation used the TensorFlow[1] platform.

4.2 Bounding Box Detection and Segmentation

This problem is modeled as a regression problem, where the input is the image, and the expected output is the location of each of the 4 edges of the bounding box. The positions of the edges are normalized by the image shape so that the regression is always in the $[0, 1]$ interval.

The models used for this step are all CNNs, separated into two groups; one that uses regular convolutions, and one that uses depthwise separable convolutions. Each group is a grid search of models architectures, varying the input image size, the depth, the number of convolutional filters, and the size of the dense layers. This resulted in 54 different models for each group.

The images were in the RGB color space, all input pixels where divided by 255 so they fall in the $[0, 1]$ interval, and the image resized to the CNNs input shape using the bi-linear interpolation. All data used 32-bit floating-point variables.

All models were trained using the Adamax optimizer, for a maximum of 1000 epochs, with the possibility of early stopping. Each epoch consisted of 200 batches of 32 samples, and after each epoch 50 batches of 32 samples were used to extract validation metrics. The mean squared error was used as the loss function. After training was over, the weights with the best metrics during training are restored and stored, and the model with those weights is evaluated on 2000 batches of 32 images.

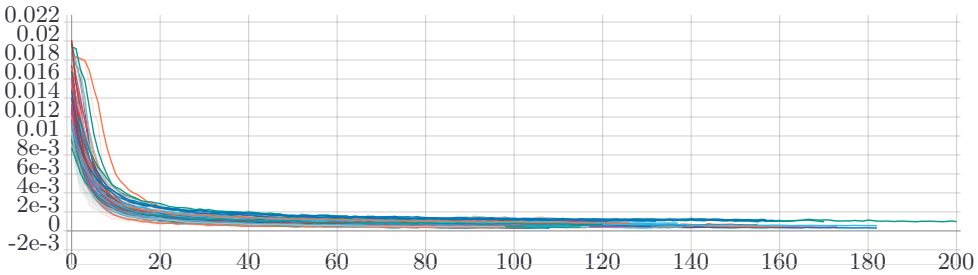

Fig. 7. Validation loss during training for all models using regular convolutions

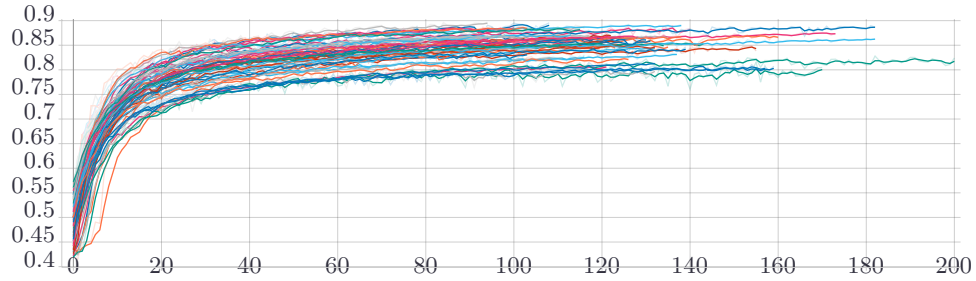

Fig. 8. Validation IoU during training for all models using regular convolutions

The models using depthwise separable convolutions generally took longer to train, as is more easily noticeable when comparing images 9 and 13. Also, they only had slightly worse metrics, shown in images 10 and 14.

One notable problem with depthwise separable convolutions is the possibility of the loss not decreasing due to a local minimum, as seen in image 11. During training, very similar models, or sometimes the same model with different initial weights, will have different behaviors regarding non-convergence. This points to the loss getting stuck on a local minimum being the cause.

This occurs in only one instance in all regular convolution models (kept out of the graph due to being in a very different range), but in many cases in the separable models, especially the deeper ones. The reason for this probably is

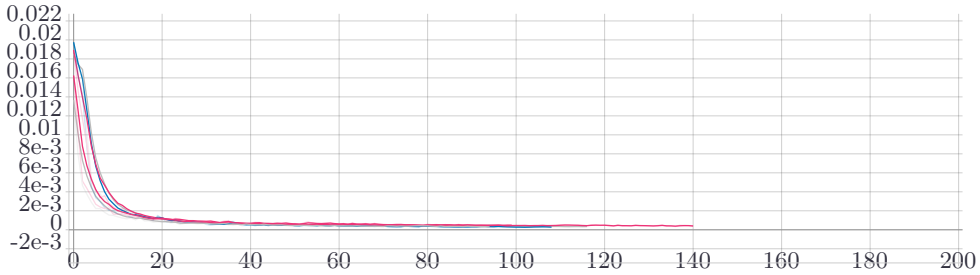

Fig. 9. Validation loss during training for the top 5 models using regular convolutions

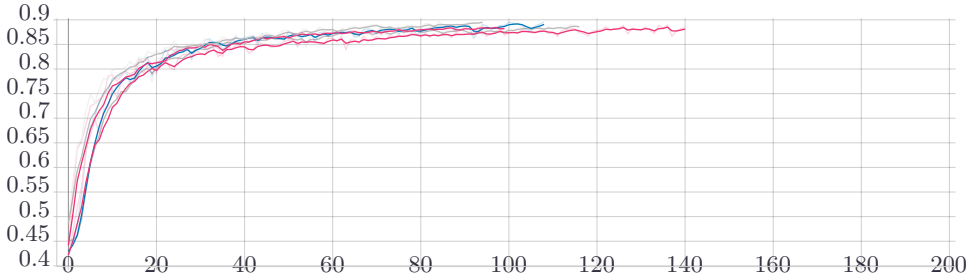

Fig. 10. Validation IoU during training for the top 5 models models using regular convolutions

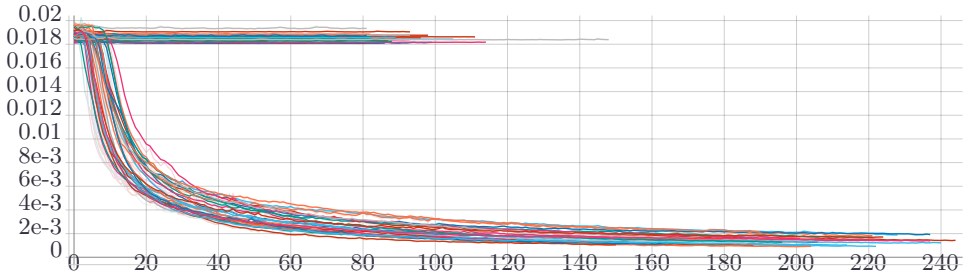

Fig. 11. Validation loss during training for all models using depthwise separable convolutions

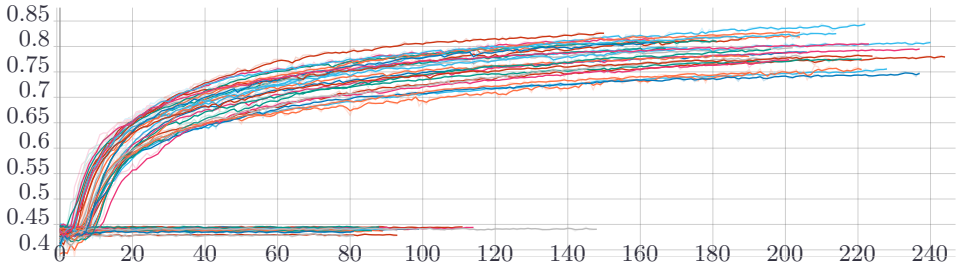

Fig. 12. Validation IoU during training for all models using depthwise separable convolutions

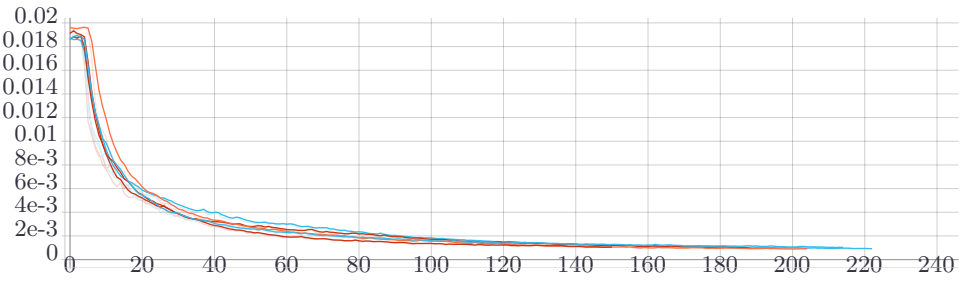

Fig. 13. Validation loss during training for the top 5 models models using depthwise separable convolutions

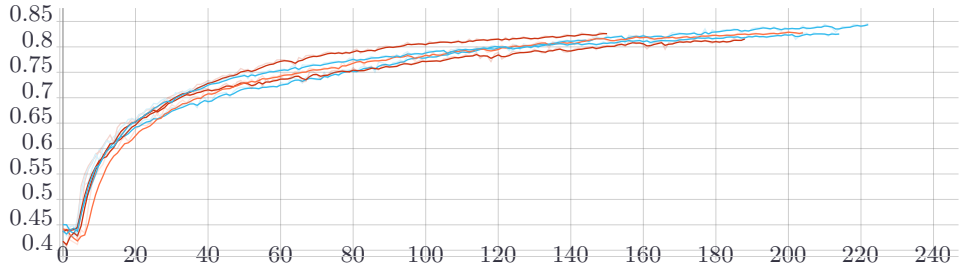

Fig. 14. Validation IoU during training for the top 5 models models using depthwise separable convolutions

that, due to the initial separable convolutions having a relatively very small amount of weights, all of them might get stuck on a local minimum. Models with more weights on the shallower convolutions have a decreased chance of this occurring. Techniques such as batch-normalization[3] and activation functions with non-zero gradients in their entire domain might improve this, which was attempted for the verification model.

Best Models Comparison

Two models were selected from the top 5 of each type. The ones with the most similar shapes were selected, in order to have a fair comparison.

For tables 1 and 2, all convolutional layers have a stride of 1x1, are padded with zeros, and use the ReLU activation function. All max-pooling layers have a stride of 2x2 and a size of 2x2. All dense layers use the ReLU activation function, except for the last one, which uses the logistic function. All dropout layers have a rate of 0.3. For table 3, the inference time was calculated on 2500 batches of 32 images, with processing limited to a single CPU core.

4.3 Verification Through Feature Vector Distance

This problem is modeled as a CNN which serves exclusively as a feature vector generator for faces. The input is an image, and the expected output is a feature

Table 1. Architecture of the bounding box detection model using regular convolutions

Layer Type	*Shape*	*Input Shape*	*Number of Weights*
2D Convolution	64x3x3	112x112x3	1792
2D Convolution	64x3x3	112x112x64	36928
2D Max Pooling	None	112x112x64	0
2D Convolution	128x3x3	56x56x64	73856
2D Convolution	128x3x3	56x56x128	147584
2D Max Pooling	None	56x56x128	0
2D Convolution	256x3x3	28x28x128	295168
2D Convolution	256x3x3	28x28x256	590080
2D Max Pooling	None	28x28x256	0
2D Convolution	512x3x3	14x14x256	1180160
2D Convolution	512x3x3	14x14x512	2359808
2D Max Pooling	None	14x14x512	0
Flatten	None	7x7x512	0
Dense	1024	25088	25691136
Dropout	None	1024	0
Dense	1024	1024	1049600
Dropout	None	1024	0
Dense	4	1024	4100
Total			31430212

Table 2. Architecture of the bounding box detection model using depthwise separable convolutions

Layer Type	*Shape*	*Input Shape*	*Number of Weights*
2D Separable Convolution	64x3x3	112x112x3	283
2D Separable Convolution	64x3x3	112x112x64	4736
2D Max Pooling	None	112x112x64	0
2D Separable Convolution	128x3x3	56x56x64	8896
2D Separable Convolution	128x3x3	56x56x128	17664
2D Max Pooling	None	56x56x128	0
2D Separable Convolution	256x3x3	28x28x128	34176
2D Separable Convolution	256x3x3	28x28x256	68096
2D Max Pooling	None	28x28x256	0
Flatten	None	14x14x256	0
Dense	256	50176	12845312
Dropout	None	256	0
Dense	256	256	65792
Dropout	None	256	0
Dense	4	256	1028
Total			13045983

Table 3. Metrics comparison between the best bounding box detection models

Convolution Type	*Mean Squared Error*	*Mean Absolute Error*	*Mean Bounding Box IoU*	*Inference Time (s)*
Regular	0.00031	0.01119	0.89682	123.6211
Separable	0.00081	0.01887	0.83563	61.3478

vector that represents someone's face, which ideally is representative enough to differentiate people's faces.

The CNNs used are all based on the VGG-Very-Deep-16 CNN architecture as described in [7]. There are 2 variations of the model, one using standard convolutions, and the other using depthwise separable convolutions. For each, both the euclidean distance squared and the cosine distance were attempted as the distance function. All models were trained from initially random weights, using the triplet loss function. As shown in [9], semi-hard triplets provide the best results, and therefore only those were used during training.

The images were in the RGB color space, all input pixels where divided by 255 so they fall in the $[0, 1]$ interval, and the image resized to the CNNs input shape using the bi-cubic interpolation. All data used 32-bit floating-point variables.

All models were trained using the Adamax optimizer, for a maximum of 1000 epochs, with the possibility of early stopping. Each epoch consisted of 200 batches of 16 images, and after each epoch 20 batches of 16 images were used to extract validation metrics. After training was over, the weights with the best metrics during training are restored and stored, and the model with those weights is evaluated on 2000 batches of 16 images.

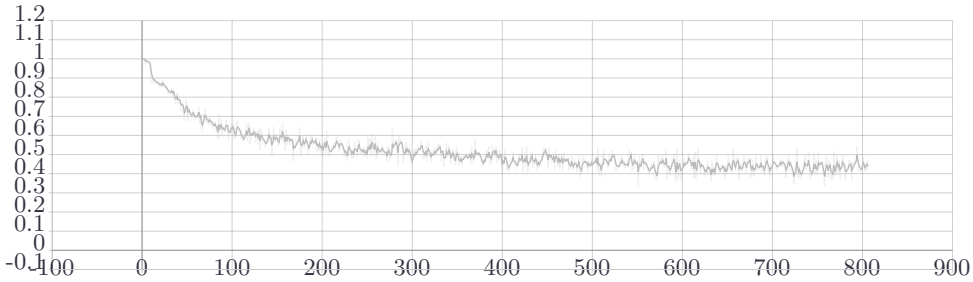

Fig. 15. Validation loss during training for the model using regular convolutions and euclidean distance squared

The models using the cosine distance converged earlier, but had worse metrics. This is visible when comparing image 15 to image 16, and image 17 to image 18.

As stated previously, the depthwise separable models might fall into local minimums. This happened initially during triplet training, so to avoid this, batch-normalization[3] and the leaky-ReLU activation function were used. This fixed the issue and made the models converge in fewer epochs.

Best Models Comparison

The models using the euclidean distance squared as their distance function were selected for a deeper comparison.

For tables 4 and 5, all convolutional layers have a stride of 1x1 and are padded with zeros. For table 4, those layers also use the ReLU activation function. All

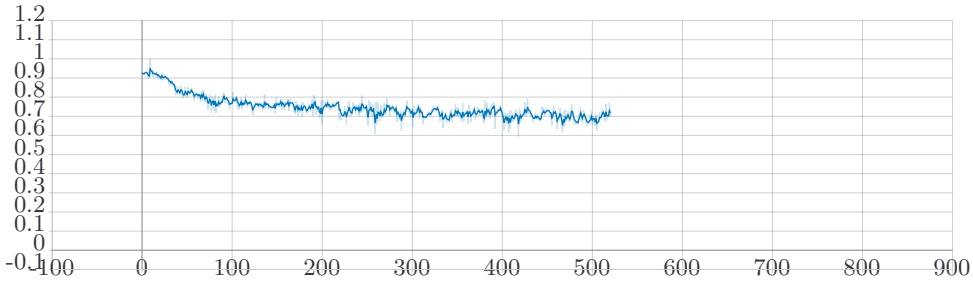

Fig. 16. Validation loss during training for the model using regular convolutions and cosine distance

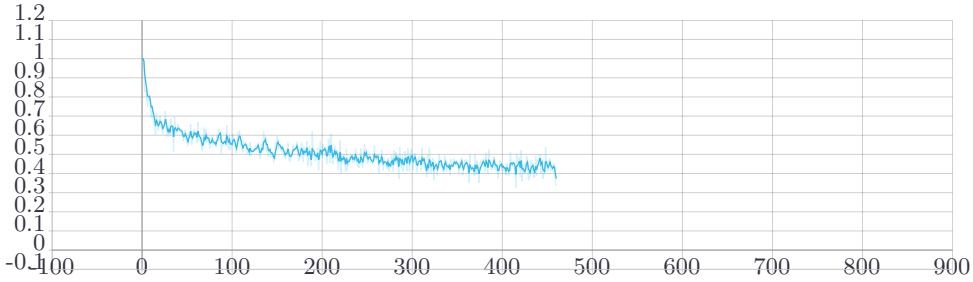

Fig. 17. Validation loss during training for the model using depthwise separable convolutions and euclidean distance squared

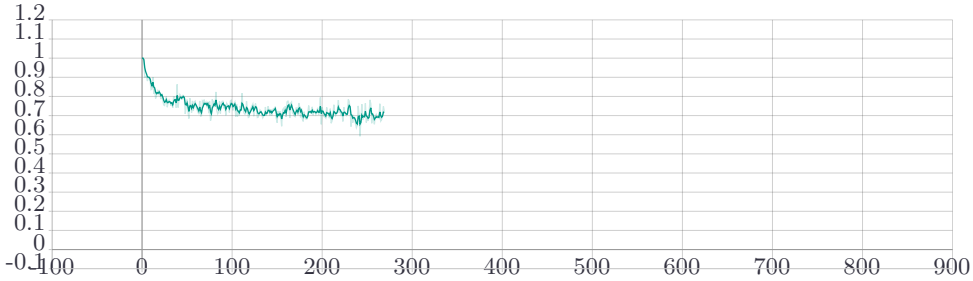

Fig. 18. Validation loss during training for the model using depthwise separable convolutions and cosine distance

Table 4. Architecture of the feature extraction model using regular convolutions

Layer Type	*Shape*	*Input Shape*	*Number of Weights*
2D Convolution	64x3x3	224x224x3	1792
2D Convolution	64x3x3	224x224x64	36928
2D Max Pooling	None	224x224x64	0
2D Convolution	128x3x3	112x112x64	73856
2D Convolution	128x3x3	112x112x128	147584
2D Max Pooling	None	112x112x128	0
2D Convolution	256x3x3	56x56x128	295168
2D Convolution	256x3x3	56x56x256	590080
2D Convolution	256x3x3	56x56x256	590080
2D Max Pooling	None	56x56x256	0
2D Convolution	512x3x3	28x28x256	1180160
2D Convolution	512x3x3	28x28x512	2359808
2D Convolution	512x3x3	28x28x512	2359808
2D Max Pooling	None	28x28x512	0
2D Convolution	512x3x3	14x14x512	2359808
2D Convolution	512x3x3	14x14x512	2359808
2D Convolution	512x3x3	14x14x512	2359808
2D Max Pooling	None	14x14x512	0
Flatten	None	7x7x512	0
Dense	4096	25088	102764544
Dropout	None	4096	0
Dense	4096	4096	16781312
Dropout	None	4096	0
L2 Normalization	None	4096	0
Total			134260544

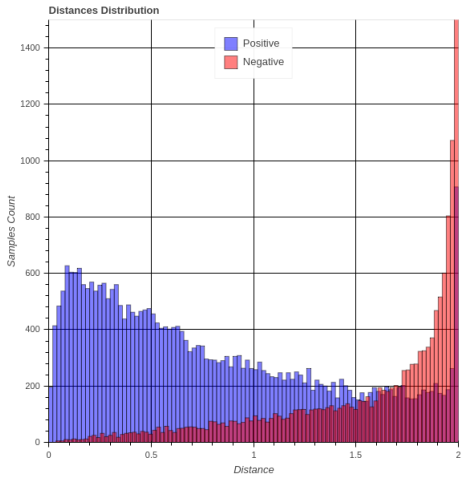

Fig. 19. Distribution of the distances between feature vectors for the model using regular convolutions

Table 5. Architecture of the feature extraction model using depthwise separable convolutions

Layer Type	*Shape*	*Input Shape*	*Number of Weights*
2D Separable Convolution	64x3x3	224x224x3	283
Batch Normalization	None	224x224x64	256
Leaky ReLU	None	224x224x64	0
2D Separable Convolution	64x3x3	224x224x64	4736
Batch Normalization	None	224x224x64	256
Leaky ReLU	None	224x224x64	0
2D Max Pooling	None	224x224x64	0
2D Separable Convolution	128x3x3	112x112x64	8896
Batch Normalization	None	112x112x128	512
Leaky ReLU	None	112x112x128	0
2D Separable Convolution	128x3x3	112x112x128	17664
Batch Normalization	None	112x112x128	512
Leaky ReLU	None	112x112x128	0
2D Max Pooling	None	112x112x128	0
2D Separable Convolution	256x3x3	56x56x128	34176
Batch Normalization	None	56x56x256	1024
Leaky ReLU	None	56x56x256	0
2D Separable Convolution	256x3x3	56x56x256	68096
Batch Normalization	None	56x56x256	1024
Leaky ReLU	None	56x56x256	0
2D Separable Convolution	256x3x3	56x56x256	68096
Batch Normalization	None	56x56x256	1024
Leaky ReLU	None	56x56x256	0
2D Max Pooling	None	56x56x256	0
2D Separable Convolution	512x3x3	28x28x256	133888
Batch Normalization	None	28x28x512	2048
Leaky ReLU	None	28x28x512	0
2D Separable Convolution	512x3x3	28x28x512	267264
Batch Normalization	None	28x28x512	2048
Leaky ReLU	None	28x28x512	0
2D Separable Convolution	512x3x3	28x28x512	267264
Batch Normalization	None	28x28x512	2048
Leaky ReLU	None	28x28x512	0
2D Max Pooling	None	28x28x512	0
2D Separable Convolution	512x3x3	14x14x512	267264
Batch Normalization	None	14x14x512	2048
Leaky ReLU	None	14x14x512	0
2D Separable Convolution	512x3x3	14x14x512	267264
Batch Normalization	None	14x14x512	2048
Leaky ReLU	None	14x14x512	0
2D Separable Convolution	512x3x3	14x14x512	267264
Batch Normalization	None	14x14x512	2048
Leaky ReLU	None	14x14x512	0
2D Max Pooling	None	14x14x512	0
Flatten	None	7x7x512	0
Dense	4096	25088	102764544
Dropout	None	4096	0
Dense	4096	4096	16781312
Dropout	None	4096	0
L2 Normalization	None	4096	0
Total			121234907

Table 6. Metrics comparison between the feature extraction models

Convolution Type	*Triplet Loss Mean*	*Positive Distance Mean*	*Negative Distance Mean*	*Inference Time (s)*
Regular	0.289	0.807	1.8098	143.1887
Separable	0.2308	0.6143	1.7366	78.1521
Convolution Type	*Triplet Loss STD*	*Positive Distance STD*	*Negative Distance STD*	
Regular	0.4458	0.5822	0.3765	
Separable	0.4159	0.5179	0.4329	

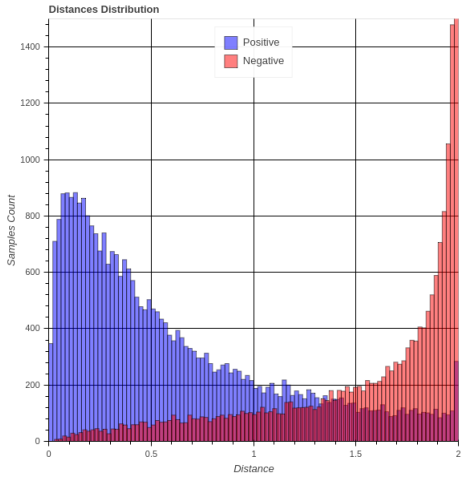

Fig. 20. Distribution of the distances between feature vectors for the model using depthwise separable convolutions

max-pooling layers have a stride of 2x2 and a size of 2x2. All dense layers use the ReLU activation function. All dropout layers have a rate of 0.5. For table 6, the inference time was calculated on 500 batches of 16 images, with processing limited to a single CPU core. For graphs 19 and 20, the samples were extracted from 2000 batches of 16 image triples, and the y-axis was capped at 1500. The negative distribution exceeded this value by a very wide margin in its last bin.

Using the distributions shown in graphs 19 and 20, a threshold was determined and a classifier created. It classifies 2 vectors with a distance between them greater or equal to the threshold as belonging to different people, otherwise belonging to the same person. The metrics for such classifier are shown in table 7. The metrics in it were calculated on 2000 batches of 16 image pairs.

Table 7. Metrics comparison between the classification models

Convolution Type	*Binary Accuracy*	*Precision*	*Recall*	
Regular	0.8441	0.847	0.8398	
Separable	0.8576	0.8598	0.8546	
Convolution Type	*True Negatives*	*False Positives*	*False Negatives*	*True Positives*
Regular	27145	4855	5125	26875
Separable	27541	4459	4652	27348

5 Conclusions

As seen in tables 3, 6 and 7, depthwise separable convolutions provide a significant reduction in costs with a very low loss in metrics. For both bounding box detection and face verification, the models using depthwise separable convolutions are the ideal choice for real-world applications, due to their higher ratio of results to computational cost.

It is evident however the benefits of including batch-normalization layers and activation functions without flat horizontal regions for those types of models. The several models that didn't converge at all on the segmentation aspect, compared to the fast conversion on the verification aspect show this. In that regard, the regular model on the verification step didn't include such layers, and would probably also benefit from including them.

Also, on the verification step, the metrics were good, but not excellent. The triplet loss did generate a separable distribution between positive and negative pairs, but there is still a high overlap. Tables 6 and images 19 and 20 show that, while the negative class has tight distribution, the positive class has a much broader one, and this results in a higher false negatives count.

Other techniques would probably be able to improve the verification aspect. Techniques that could be attempted include:

- Utilizing a bigger α (margin) for the triplet loss, which would make for a tighter positive distribution, since more distance would be needed for a pair to fall on the easy category.
- Pre-training the model on a face classification problem.
- Aligning the faces in the images through the use of detected landmarks, so that rotation and translation would become less relevant.
- For the comparison between 2 feature vectors, utilize a more complex classifier than a simple threshold on the distance between them.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
3. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc. (2012), http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
5. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV) (December 2015)
6. Moindrot, O.: Triplet loss and online triplet mining in tensorflow (2018), https://omoindrot.github.io/triplet-loss
7. Parkhi, O.M., Vedaldi, A., Zisserman, A.: Deep face recognition. In: British Machine Vision Conference (2015)
8. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
9. Schroff, F., Kalenichenko, D., Philbin, J.: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823 (2015)
10. Sifre, L., Mallat, S.: Rigid-motion scattering for image classification. Ph. D. thesis (2014)
11. Tsang, S.H.: Review: Mobilenetv1 — depthwise separable convolution (light weight model) (2018), https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69