Cascade Net

Jonathan Blanchette

September 9, 2016

1 Data

1.1 Negative Example Database for validation

The negative patches for validation comes from AFWL database. I removed face annotation with black patches from the AFWL database.

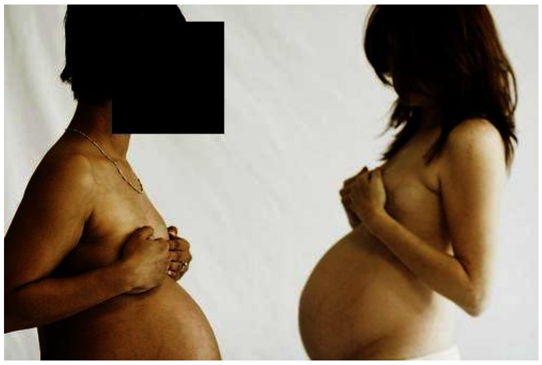

Figure 1: Cropped out faces

I went through the list of ~20k images about 4 times to remove all images that still contained a recognisable face so that I'm left with the images that are uniquely annotated. Using the names of the cleaned up cropped-out faces database Using SQL, a list of bounding boxes per image was done, and I've made three validation sets of images that contained only one face.(just to keep the validation simple).

1.2 Preprocessing

The test set and training set are split randomly and with a ratio of 1:10. The negative examples are taken from the PASCAL dataset (which needed also some cleaning). The negative patches are picked randomly from the PASCAL dataset with a minimum patchsize of 40. The data is normalised with the training data mean and standard deviation. If needed, the images are converted to grayscale or RGB to fit the model. The test and training patches are preprocessed according to [1, 2]. For grayscale images, no random lighting/contrast variations are added during training.

1.3 Collecting data for subsequent cascade levels.

I've used AFWL validation subsets to identify false positives as negative training patches for subsequent nets. The false positives are stored in a folder for use in the training of a subsequent cascade level. This method proved to be very slow and yielded highly varying results when validating on fddb.

2 Architectures

Design considerations

The number of multiplications in one convolutional layer is determined by a concave function:

$$\#mul = D_I h_f^2 D_O \left(\left\lfloor \frac{h_I + 2P - h_f}{s} \right\rfloor + 1 \right)^2 \approx D_I h_f^2 D_O \left(\frac{h_I + 2P - h_f}{s} + 1 \right)^2$$

The filter size that yields the maximum is:

$$\frac{\partial \sqrt{\#mul}}{\partial h_f} \equiv 0 \rightarrow h_f = \frac{h_I + 2P + s}{2}$$

Note that #mul decreases slower if h_f is chosen large (more like a fully connected) than if we pick it small compared to $h_f = \frac{h_I+2P+s}{s^2 D_I+1}$. So, intuitively picking smaller filters is better for reducing the multiplications.

The memory use of a convolutional layer is the sum of the memory required to store the parameters and the memory required to store the output. This is a concave function given by:

$$Memory = D_O\left(h_f^2 D_I + 1 + \left(\left\lfloor\frac{h_I + 2P - h_f}{s}\right\rfloor + 1\right)^2\right)$$

$$\approx D_O\left(h_f^2 D_I + 1 + \left(\frac{h_I + 2P - h_f}{s} + 1\right)^2\right)$$

The filter size that yields the minimum is:

$$\frac{\partial Memory}{\partial h_f} \equiv 0 \rightarrow h_f = \frac{h_I + 2P + s}{s^2 D_I + 1}$$

We can't simultaneously maximize $\#mul$ and minimize $Memory$ except if $D_I = 1$ and $s = 1$. If we wish to use max-pooling after a convolutional layer then it might be desirable to have a large output(by making h_f smaller).

First Convolutional net

The network structure of the 20 net is:

$Input(ich \times 20 \times 20) \implies Conv(ich \rightarrow 32, 3 \times 3, 1, 1) \Rightarrow Maxpool(3 \times 3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(32 \rightarrow 32, 3 \times 3, 1, 1) \Rightarrow ReLU \Rightarrow Maxpool(3 \times 3, 2, 2, 1, 1) \Rightarrow Conv(32 \rightarrow 32, 4 \times 4, 1, 1) \Rightarrow ReLU \Rightarrow FullyConnected(32 \rightarrow 2) \Rightarrow LogSoftmax$

The newtork takes an input patch of dimension $(ichx20x20)$. The following table is it's complexity analysis:

Layer	Out. dim.	#parameters	#Operations	Memory use
$Conv(ich \rightarrow 32, 3 \times 3, 1, 1)$	$O(32x18x18)$	$32 \cdot (ich \cdot 9 + 1) = \begin{cases} 320 \\ 896 \end{cases}$	$32 \cdot ich \cdot 3^2 \cdot 18^2 = \begin{cases} 93312 \\ 279936 \end{cases}$	$32 \cdot (ich \cdot 9 + 1) + 32 \cdot 18^2 = \begin{cases} 10688 \\ 11264 \end{cases}$
$ReLU$	$O(32x18x18)$	0		
$Maxpool(3 \times 3, 2, 2, 1, 1)$	$O(32x9x9)$	0		
$Conv(32 \rightarrow 32, 3 \times 3, 1, 1)$	$O(32x7x7)$	$32 \cdot (32 \cdot 9 + 1) = 9248$	$32^2 \cdot 3^2 \cdot 7^2 = 451584$	$32 \cdot (32 \cdot 9 + 1) + 32 \cdot 7^2 = 10816$
$Maxpool(3 \times 3, 2, 2, 1, 1)$	$O(32x4x4)$	0		
$ReLU$	$O(32x4x4)$	0		
$Conv(32 \rightarrow 32, 4 \times 4, 1, 1)$	$O(32x1x1)$	$32 \cdot (32 \cdot 16 + 1) = 16416$	$32^2 \cdot 4^2 = 16384$	$32 \cdot (32 \cdot 4^2 + 1) + 32 = 16448$
$ReLU$	$O(32x1x1)$	0		
$FullyConnected(32 \rightarrow 2)$	$O(2)$	$2 \cdot (32 + 1) = 66$	64	
$LogSoftmax$	$O(2)$	0		

Table 1: Net #1 patch analysis (20net)

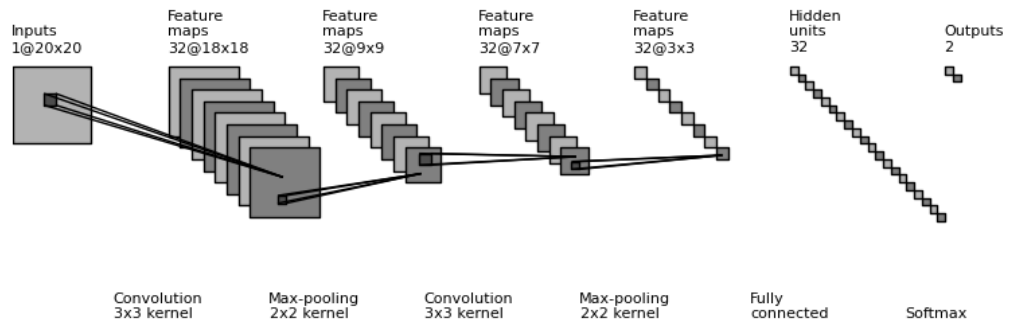

Figure 2: Gray 20-net

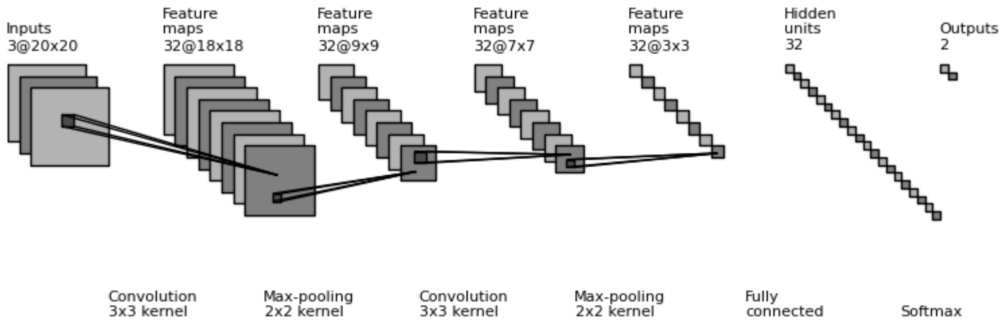

Figure 3: RGB 20-net

The network structure of the 12 net is:

$Input(ich \times 12 \times 12) \implies Conv(ich \rightarrow 16, 3 \times 3, 1, 1) \Rightarrow Maxpool(3 \times 3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(16 \rightarrow 16, 5 \times 5, 1, 1) \Rightarrow ReLU \Rightarrow FullyConnected(16 \rightarrow 2) \Rightarrow LogSoftmax$

The newtork takes an input patch of dimension $(ichx12x12)$. The following table is it's complexity analysis:

Layer	Out.Dim.	#parameters	#Operations	Memory use
$Conv(ich \to 16, 3\times3, 1, 1)$	$O(16x10x10)$	$16 \cdot (ich \cdot 9 + 1) = \begin{cases} 160 \\ 448 \end{cases}$	$16 \cdot ich \cdot 3^2 \cdot 10^2 = \begin{cases} 14400 \\ 43200 \end{cases}$	$16 \cdot (ich \cdot 9 + 1) + 16 \cdot 10^2 = \begin{cases} 1760 \\ 2048 \end{cases}$
$ReLU$	$O(16x10x10)$	0		
$Maxpool(3\times3, 2, 2, 1, 1)$	$O(16x5x5)$	0		
$Conv(16 \to 16, 5\times5, 1, 1)$	$O(16x1x1)$	$16 \cdot (16 \cdot 5^2 + 1) = 6416$	$16^2 \cdot 5^2 = 6400$	$16 \cdot (16 \cdot 5^2 + 1) + 16 = 6432$
$ReLU$	$O(16x1x1)$	0		
$FullyConnected(16 \to 2)$	$O(2)$	$2 \cdot (16 + 1) = 34$	16	
$LogSoftmax$	$O(2)$	0		

Table 2: Net #1 patch analysis (12net)

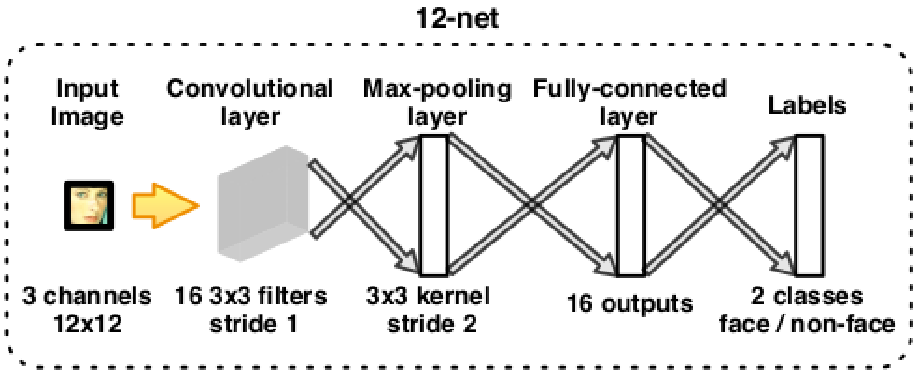

Figure 4: RGB-12net

First Calibration net

Here is the FIRST network structure (20 net)of the CNN model :

$Input(ich \times 12 \times 12) \Longrightarrow Conv(ich \to 16, 3\times3, 1, 1) \Rightarrow Maxpool(3\times3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(16 \to 128, 5\times5, 1, 1) \Rightarrow ReLU \Rightarrow FullyConnected(128 \to 45) \Rightarrow LogSoftmax$

Second Convolutional net

Here is the Second network structure (48 net)of the CNN model:

$Input(ich \times 48 \times 48) \Longrightarrow Conv(ich \to 64, 5\times5) \Rightarrow Maxpool(3\times3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(64 \to 64, 5\times5, 1, 1) \Rightarrow Maxpool(3\times3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(64 \to 128, 9\times9, 1, 1) \Rightarrow ReLU \Rightarrow Dropout(\frac{1}{2}) \Rightarrow FullyConnected(128 \to 2) \Rightarrow LogSoftmax$

Second Calibration net

Here is the Second network structure (48 net)of the CNN model:

$Input(ich \times 48 \times 48) \implies Conv(ich \rightarrow 64, 5 \times 5) \Rightarrow Maxpool(3 \times 3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(64 \rightarrow 64, 5 \times 5, 1, 1) \Rightarrow ReLU \Rightarrow Conv(64 \rightarrow 256, 18 \times 18, 1, 1) \Rightarrow ReLU \Rightarrow FullyConnected(256 \rightarrow 45) \Rightarrow LogSoftmax$

2.1 Alternative to fully connected layer

Since a fully connected layer uses a lot of bandwidth, I've tested a new architecture replacing the fully connected layer by the layer structure presented in [3,4].

$Input(ich \times 12 \times 12) \implies Conv(ich \rightarrow 16, 3 \times 3, 1, 1) \Rightarrow Maxpool(3 \times 3, 2, 2, 1, 1) \Rightarrow ReLU \Rightarrow Conv(16 \rightarrow 128, 5 \times 5, 1, 1) \Rightarrow ReLU \Rightarrow ConvolutionalMap(1to1, 4 \times 4) \Rightarrow LogSoftmax$

The grayscale model with the alleviated fully-connected yielded a slightly poorer performance. The parameter drop from FC to a partial FC is:

$$\frac{\#parameters_{PFC}}{\#parameters_{FC}} = \frac{D \cdot (H \cdot W + 1)}{D \cdot (D \cdot H \cdot W + 1)} = \mathcal{O}(D^{-1})$$

In our design D=32 so we use about 32 times less parameters if we use a one to one receptive field for the convolutional hidden units.

3 Results

The grayscale models are slightly better than the RGB based ones for the validation sets (training sets). This was also veryfied qualitatively and on the fddb benchmark. I got a below State of the Art performance for a 2-level Cascade. The Grayscale model is better than RGB one. Testing different models on fddb revealed that performance depended highly on the quality of the training dataset.

3.1 FDDB graphic

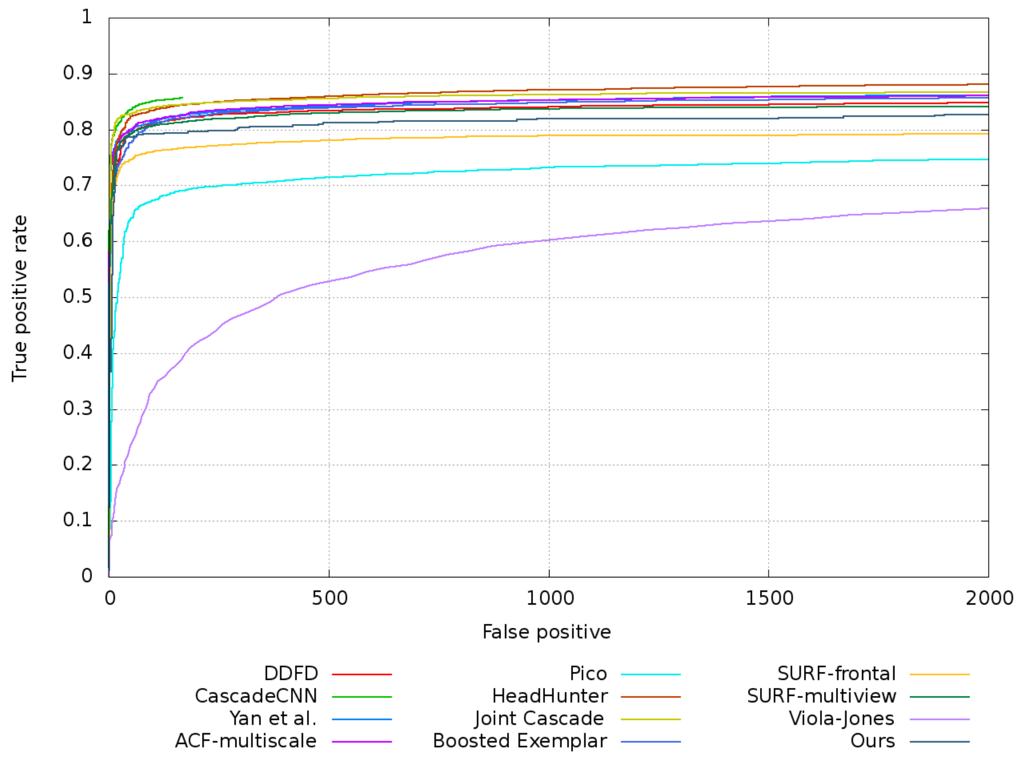

Figure 5: Gray

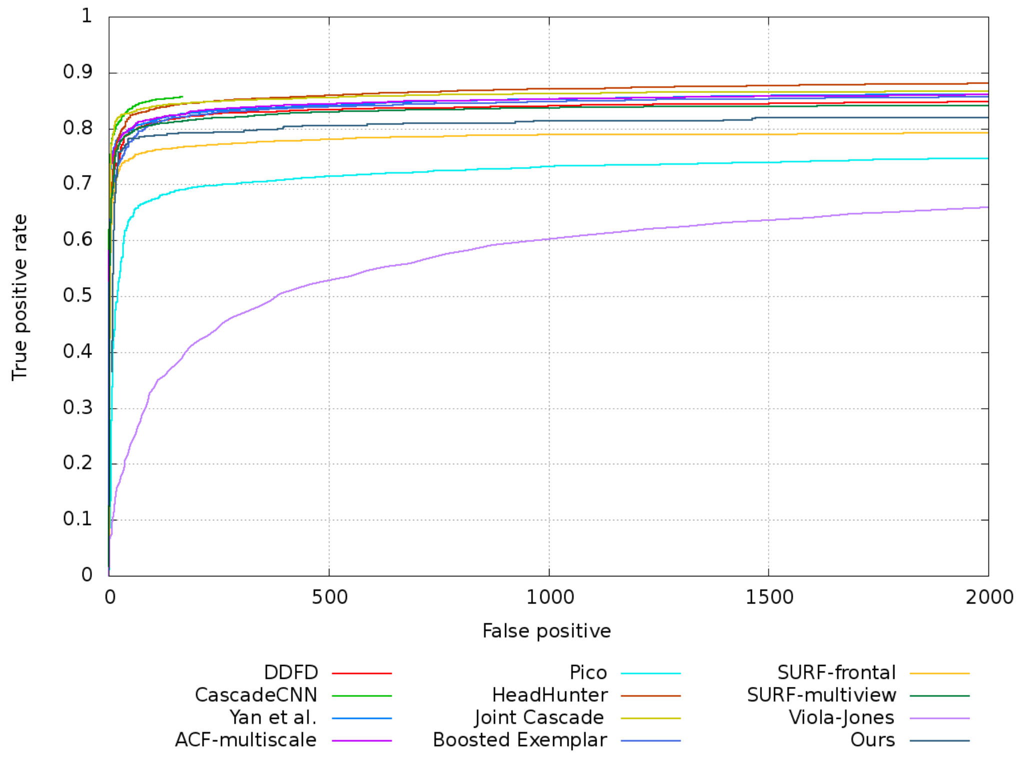

Figure 6: Grayscale with alleviated FC

As one can see, there is not much difference of performance (only a slight decrease for lower bandwidth FC). However, I will not hide the fact that the NMS caused a tremendous problem in the above ROC curve. There is always an offset (although very small) in the False Positive axis that occurs whenever we use NMS. This is probably because NMS can randomly picks the false positive patch if other false-positive detection windows have the same score. Intuitively, to remedy this situation, we would need false positive patches to have lower scores in such a way that the nms doesn't get confused. This means that the problem might be the training of the first classifier. There are many alternatives to NMS, but theses should be tested in future works.

3.2 Validation on AFWL

Three sets of images containing exactly one face were made to test the performance curves for the Cascades so that the parameters could be tuned optimally as to maximise the true negative detection rate while keeping the highest recall rate possible. Having done the framework for this, it should be straightforward to make

3.3 Qualitative results

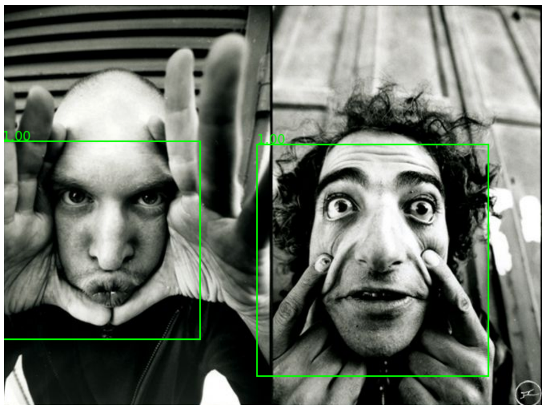

Figure 7: Grayscale detection results

4 Future works

The findings that the slight performance drop for a dramatic decrease in parameters prooves that it is possible to use the partial FC in real-time systems. In the article [4] , it is claimed that the system perform very well for at least twice the testing speed when using the partial FC. The framework used for the validation and the training technique used in [4] motivates to modify my training procedure in such a way of using the cleaned up AFWL database for the negative sampling. This makes it also easier to collect hard negative data for the subsequent cascades. However a possible downside is that the training procedure should be slower for the false negatives are cropped out from images(and resized) during training.

4.1 Training technique

More diverse background content can be added if we include the cleaned AFWL dataset background patches that has no (or tiny) IoU with the face. In this way we could reduce the amount of false positives after the first cascade.

4.2 Part Based models

In [5], the authors used heatmaps of face attributes in order to detect a face. Their dataset is available online. It would be interresting to see if we could use this in cascade layer 2 to 3.

4.3 NMS alternatives

In [6], the authors compared average NMS with hard IoU NMS. The conclusion is that if the best threshold for the NMS techniques are picked, then overall the average NMS is better. There is also in [4], they use a decision rule for the detection windows instead of the NMS between Cascade classifiers.

The NMS problem is akin to a clustering problem. This is typically an unsupervised learning problem. Maybe it is possible to use modern neural networks to have a good clustering algorithm.

References

[1] A. Krizhevsky, I. Sutskever, & G. E. Hinton *ImageNet Classification with Deep Convolutional Neural Networks.* In: NIPS(2012)

[2] A. G. Howard, *Some Improvements on Deep Convolutional Neural Network Based Image Classification.* In: CoRR abs/1312.5402 (2013)

[3] C. Garcia and M. Delakis, *Convolutional face finder: a neural architecture for fast and robust face detection.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 11, pp. 1408-1423, (Nov. 2004)

[4] I. Kalinovskii, V. Spitsyn, *Compact Convolutional Neural Network Cascade for Face Detection.* In: arXiv:1508.01292, (2015)

[5] S. Yang, P. Luo,C. C. Loy,X. Tang, *From Facial Parts Responses to Face Detection: A Deep Learning Approach.* In: arXiv:1509.06451, (2015)

[6] S. S. Farfade, M. Saberian,C. L. Li, *Multi-view Face Detection Using Deep Convolutional Neural Networks.* In: arXiv:1502.02766, (2015)

[7] H. Li, Z. Lin, X. Shen, J. Brandt, G. Hua; IN: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2015)