Image Inpainting Using Mixed Convolutional Layers

Zhuang Zijun
zhuangzijun1996@gmail.com

Abstract

Recently[1], GAN based image inpainting again catches people's attention with a significant impovement in both clarity and realisticity. However, the newly proposed convolutional layers (which is the partial convolution in [11] and the gated convolution in [20]) have their own issues in either computational cost or the flexibility. In this project, we proposed a new insight of the these two state-of-art work by integrating the two kinds of convolutional layer efficiently. The result shows that our model has similar performance with those two and far better computational efficiency than gated convolution constructed model. We show qualitative and quantitative comparisons with these two methods to validate our approach.

1. Introduction

Image inpainting is the task to reconstruct lost or deteriorated part of images. The core challenge of image inpainting lies in synthesizing visually realistic and semantically plausible pixels for the missing regions that are coherent with existing ones as efficiently as possible. It has been applied in various kinds of applications including image editing, object removal, image lossy compression, high resolution research and many others. Traditional patch based methods iteratively search for the best fitting patches to reconstruct lost or deteriorated part of images. Another kinds of approaches attempt to transfer the inpainting problem to a learning-based optimization problem. However, deep generative models based on vanilla convolutional networks are naturally ill-fitted for image hole-filling because convolutional filters treat all input pixels as same valid ones. Vanilla convolutions apply same filters on all valid, invalid or mixed (on hole boundaries) pixels, leading to visual artifacts such as color discrepancy, blurriness and obvious edge responses surrounding holes. Liu G. *et al.* proposed partial convolution [11] where the convolution is masked and renormalized to enable conditional convolution only on valid pixels and then follow by a mask-update step to re-compute new mask layer by layer. However, in real world image inpainting, it seems not practicable enough as we actually do not know where mask is in most of the cases. Another state-of-art approch in terms of the task is gated convolution [20]. It dynamically learns a feature selection mechanism for each channel and each spatial location and got much more flexibility which makes the model extenable as well as adaptive but the price is a doubled parameter number and additional computational cost. In this way, in this project we are about to propose an improved model based on these two state-of-the-art image inpainting approaches by using both partial convolution and gated convolution to achieve a balance in terms of performance and cost.

In summary, our contributions can be summarized as follows:

– We propose a reasonable mechanism to integrate both partial convolution and gated convolution within a same network and achieve the state-of-art results.

– We also raise a question on whether we should project a binary mask into high dimension spaces and learn its high level feature which in some extent, can be transferred to the question that whether the principal 'deeper is better' fits simple classification tasks.

2. Related Work

In this section, we will discuss the existing works for image inpainting. They can roughly be divided into two classes. The first is patch based methods and the second class attempts to solve the inpainting problem through a learning-based approach.

2.1. Traditional patch based methods

Traditional patch based methods copies similar patches from the background regions into holes by iteratively searching. While these methods work well for stationary textures, they are limited for non-stationary data such as natural images as it ignores semantics. Simakov D. *et al.* proposed a bi-directional similarity based scheme in [15] to better model non-stationary visual data for re-targeting and inpainting applications. However, dense computation of patch similarity is a very expensive operation which prohibits the broad applications of this method. In order to

[1]This papaer is provided only for a better understand to the project and is strictly prohibit for **ANY** other possible usage.

address the problem, Barnes C. *et al.* proposed a fast nearest neighbor field algorithm called PatchMatch [1], shows significant practical value for image editing applications including inpainting. Patch based algorithms progressively extend pixels close to the hole boundaries based on low-level features (features of mean square difference on RGB space for example) to search and paste the most similar image patches thus are limited for non-stationary data.

2.2. Learning-based approaches

Recently, deep learning based approaches have emerged as a promising paradigm for image inpainting. Initial efforts to train convolutional neural networks for denoising and inpainting of small regions can be found in [8, 17]. In [13], Pathak D. *et al.* firstly proposed an approach to train deep neural networks for inpainting large holes. It is trained to complete center region of 64×64 in a 128×128 image, using both l_2 pixel-wise reconstruction loss and generative adversarial loss as the objective function. More recently, Iizuka *et al.* improved it by introducing both global and local discriminators as adversarial losses [5]. The global discriminator assesses whether completed image is coherent as a whole, while the local discriminator focus on small region centered at the generated region to enforce the local consistency. In addition, they also use dilated convolutions in inpainting network to replace channel-wise fully connected layer adopted in Context Encoders, both techniques are proved to be capable to increase receptive fields of output neurons.

Meanwhile, there has been several studies focusing on generative face inpainting. In [19], Yeh R. *et al.* proposed a searching approach, using the closest encoding in latent space of the corrupted image and decode to get completed image. Li Y. *et al.* introduced additional face parsing loss for face completion [10]. However, these methods typically rely heavily on post-processing steps such as image blending operation to enforce color coherency near the hole boundaries. Several works follow ideas from image stylization [2, 9] to formulate the inpainting as an optimization problem. Yang C. *et al.* proposed a multiscale neural patch synthesis approach base on joint optimization of image content and texture constraints, which not only preserves contextual structures but also produces high-frequency details by matching and adapting patches with the most similar mid-layer feature correlations of a deep classification network[18]. This approach shows promising visual results but is very slow due to the optimization process.

More recently, for capturing long-range spatial dependencies, contextual attention module [21] is proposed and integrated into networks to explicitly borrow information from distant spatial locations. Liu G. *et al.* proposed partial convolution [11] where the convolution is masked and re-normalized to utilize valid pixels only. It is then followed by a mask-update step to re-compute new masks layer by layer. Partial convolution improves the quality of inpainting on irregular masks and is able to robustly handle arbitrary hole shapes. To enable user-guided image inpainting, Yu J. *et al.* proposed gated convolution for image inpainting network. Instead of using hard masks updated with rules, gated convolution learn soft masks automatically from data. The proposed gated convolution enables network to learn a dynamic feature selection mechanism for each channel and each spatial location.

3. Approach

In this section, we will focus on our proposed model, as well as the convolutional layers we applied. Our proposed model uses gated convolution to learn the mask and stacked partial convolution and mask updating steps to perform image inpainting. We will first give the definition of the two convolutional layers and mask update mechanism, then the design of inpainting network, the loss function and free-form mask generation algorithm.

3.1. Partial Convolutional Layer

Partial convolutional layer is the convolutional layer proposed by Liu G. *et al.* in [11] refers to the partial convolution and mask update. Let W be the convolution filter weights and b its corresponding bias. X are the feature values (pixels values) for the current convolution (sliding) window and M is the corresponding binary mask. The partial convolution at every location can expressed as:

$$\hat{x} = \begin{cases} W^T(X \odot M)\frac{1}{\sum M} + b & \text{if} \sum M > 0 \\ 0 & \text{otherwize} \end{cases}$$

where $\odot$ denotes element-wise multiplication, which let output values depend only on the unmasked inputs. The scaling factor $\frac{1}{\sum M}$ applies appropriate scaling to adjust for the varying amount of valid (unmasked) inputs. After each partial convolution operation, we then update our mask. The unmasking rule is simple: if the convolution was able to condition its output on at least one valid input value, then we remove the mask for that location. This is expressed as:

$$\hat{m} = \begin{cases} 1 & \text{if} \sum M > 0 \\ 0 & \text{otherwize} \end{cases}$$

and can easily be implemented in any deep learning framework as part of the forward pass. With sufficient successive applications of the partial convolution layer, any mask will eventually be all ones, if the input contained any valid pixels.

3.2. Gated Convolution Layer

Another layer used is introduced by Yu J. *et al.* in [20]. Considering a convolutional layer in which a bank of filters are applied to the input feature map to produce an output featuremap. Instead of hard masks updated with rules, gated convolutions learn soft masks automatically from data. It can be expressed as:

$$Gating_{y,x} = \sum \sum W_g \cdot I$$

$$Feature_{y,x} = \sum \sum W_f \cdot I$$

$$O_{y,x} = \phi(Feature_{y,x}) \odot \sigma(Gating_{y,x})$$

where σ is sigmoid function thus the output gating values are between zeros and ones. ϕ can be any activation functions (e.g. ReLU or LeakyReLU).W_g and W_f are two different convolutional filters.

Gated convolution enables network to learn a dynamic feature selection mechanism for each channel and each spatial location. It is also capable to learn to select the feature maps not only according to backgrounds or masks but also considering semantic segmentation in some channels.

3.3. Network Architecture

While the two kinds of convolutional layers all achieved plausible results visually, there still remain some critical issues to be solved. For partial convolution, it works quite inflexible as we need to feed model the mask to get it work but for real world image inpainting, it seems not practicable enough as we actually do not know where mask is in most of the cases. On the other hand, gated convolution solve the problem by learning the mask using an isolated CNN (although the original paper train/test their model with a mask channel, it can work without an explicit knowledge of mask and have barely no damage to its performance), the price is a doubled parameter number and additional computational cost. Thus, we proposed a mixed usage of these two convolutional layers to attempt to solve the issues at some content.

The whole architecture is mainly alike the one used in [11], which in specific, is a UNet-like architecture proposed in [14]. We replace the first two layer from partial convolution to gated convolution to enable the model to learn the mask with minimal extra computational cost. What's more, unlike a normal vision classification task, a random mask has much weaker correlation over spatial thus there's actually meaningless (even may cause trouble in some cases) to extract high level feature of a mask. This let us to only focus on low level feature of mask and use partial convolution to fill the hole. In decoding stage, nearest neighbor upsampling is used. ReLU is used in the encoding stage and LeakyReLU with $alpha = 0.2$ is used between all decoding layers. Batch normalization layer [6] is used between each partial convolution layer and ReLU/LeakyReLU layer except the first and last convolutional layers. The encoder comprises seven convolutional layers with $stride = 2$. The kernel sizes are 7, 5, 5, 3, 3, 3 and 3. The channel sizes are 64, 128, 256, 512, 512, 512, and 512. The decoder include 7 upsampling layers, each with a factor of 2, and followed by a convolutional layer. The output channel for convolutional layers in the decoder are 512, 512, 512, 256, 128, 64, and 3. The skip links feeding the decoder stage concatenate both the feature maps and binary masks channel-wise before being input to the next convolution layer. The last convolution layer is also replaced by a gated convolutional layer and its input contains the concatenation of the original input image with hole. This makes it possible for the model to simply copy non-hole pixels. The whole network architecture can be checked in table.4. As for mask updating, the original gated convolution consider mask learning process as a pixel-wise classification problem thus $\sigma(Gating_{y,x}) > 0.5$ is used to explicitly give out the learned mask to be input of the followed by partial convolutional layer.

Table 1. Definitions of the Symbol Used in Total Loss Function

I_{in}	input image with hole
M	initial binary mask
I_{out}	network prediction output
I_{gt}	ground truth
I_{comp}	ouput with non-hole pixels set to ground truth
Ψ_n	activation map of images higher level feature
P	1-pixel dilation of the hole region
K_n	normalization factor
$\mathcal{G}(A)$	gram matrix of A ($A^T A$)

3.4. Loss Function

For simplicity, we use the same loss function as [11]. The whole loss function is determined by 6 terms. First, some symbol definition can be checked in table.1. The first per-pixel losses is defined as follow:

$$\mathcal{L}_{hole} = \|(1 - M) \odot (I_{out} - I_{gt})\|_1$$

$$\mathcal{L}_{valid} = \|M \odot (I_{out} - I_{gt})\|_1$$

These are the l_1 losses on the network output for the hole and the non-hole pixels respectively. Second, followed by [3], the perceptual losses introduced by Gatys *et al.* can be defined as:

$$\mathcal{L}_{perceptual} = \sum_{n=0}^{N-1} \|\Psi_n(I_{out}) - \Psi_n(I_{gt})\|_1 + \sum_{n=0}^{N-1} \|\Psi_n(I_{comp}) - \Psi_n(I_{gt})\|_1$$

The perceptual loss computes the l_1 distances between both I_{out} and I_{comp} after projecting these into high level feature spaces using an ImageNet-pretrained VGG-16 [16].

As mentioned in [7], perceptual loss is known to generate checkerboard artifacts. Thus, following style-loss and total variation loss [7] terms are included:

$$\mathcal{L}_{style_{out}} = \sum_{n=0}^{N-1} \|K_n \mathcal{G}(\Psi_n(I_{out})) - \mathcal{G}(\Psi_n(I_{gt}))\|_1$$

$$\mathcal{L}_{style_{comp}} = \sum_{n=0}^{N-1} \|K_n \mathcal{G}(\Psi_n(I_{comp})) - \mathcal{G}(\Psi_n(I_{gt}))\|_1$$

$$\begin{aligned}\mathcal{L}_{tv} = & \sum_{i,j \in P, i,j+1 \in P} \|I_{comp}^{i,j+1} - I_{comp}^{i,j}\|_1 \\ & + \sum_{i,j \in P, i+1,j \in P} \|I_{comp}^{i+1,j} - I_{comp}^{i,j}\|_1\end{aligned}$$

Finally, the total loss L_{total} is the combination of all the above loss functions:

$$\begin{aligned}\mathcal{L}_{total} = & a \cdot \mathcal{L}_{valid} + b \cdot \mathcal{L}_{hole} \\ & + c \cdot \mathcal{L}_{perceptual} + d \cdot (\mathcal{L}_{style_{out}} + \mathcal{L}_{style_{comp}}) + e \cdot \mathcal{L}_{tv}\end{aligned}$$

where $a \sim e$ are the result given in [11] determined by performing a hyperparameter search on 100 validation images and is set to $1, 6, 0.05, 120, 0.1$ respectively.

4. Experiments

All the trainings are done with PyTorch 0.4.1, CUDNN v7.0, CUDA v9.0 with a NVIDIA K80 graphic card. We use CelebA faces [12] as the dataset since it provides less variation thus is much easier for training. The original dataset contains 202,599 images of faces with 10,177 celebrity identities and here, we sample one eighth of each person in the dataset, and finally end up with around 20K images where one-tenth is used for testing. The weights initialization follows [4] by He K. *et al.* and Adam is used for optimization. The training images are resized to 256×256 (from an original size of 178×218) and the batch size is set to 16.

In order to use Batch Normalization in the presence of holes, we follow the procedure rasied in [11]. we first turn on Batch Normalization for the initial training using a learning rate of 0.0002. Then, we fine-tune using a learning rate of 0.00005 and freeze the Batch Normalization parameters in the encoder part of the network. We keep Batch Normalization enabled in the decoder. This not only avoids the incorrect mean and variance issues, but also helps us to achieve faster convergence. For each model, we do 100 epoches of training and 200 epoches of fine-tuning at a speed of 50 epoches per day. As for testing, it runs at around 0.05 seconds per image with a size of 256×256, on average.

4.1. Free-Form Masks Generation

To create contaminated image, we simply use the algorithm described in [20] to automatically generate random free-form masks on-the-fly. The full algorithm is listed as Algorithm.1. Some mask examples can be found in Figure.1.

Algorithm 1. Algorithm for sampling free-form training masks.
maxVertex, maxLength, maxBrushWidth, maxAngle are four hyperparameters to control the mask generation.

```
mask = zeros(imageHeight, imageWidth)
numVertex = random.uniform(maxVertex)
startX = random.uniform(imageWidth)
startY = random.uniform(imageHeight)
for $i = 0$ to numVertex do:
    angle = random.uniform(maxAngle)
    if $i\%2 == 0$:
        $angle = 2 \cdot \pi - angle$
    end if
    length = random.uniform(maxLength)
    brushWidth = random.uniform(maxBrushWidth)
    Draw line from point (startX, startY) with angle, length and brushWidth as line width.
    $startX = startX + length \cdot sin(angle)$
    $startY = stateY + length \cdot cos(angle)$
    Draw a circle at point (startX, startY) with radius as half of brushWidth.
end for
mask = random.flipLeftRight(mask)
mask = random.flipTopBottom(mask)
```

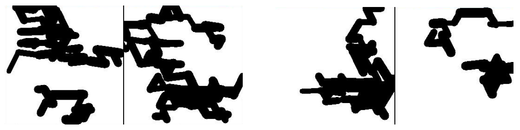
Figure 1. Sample free-form masks. The masks are generated under a parameter: $maxVertex = 20$, $maxLength = 50$, $maxBrushWidth = 20$, $maxAngle = 4.0$, $maxNum = 5$. Mask size = $(256, 256)$

4.2. Comparisons

We compare our method with 2 other methods under the same architecture and loss function as the official implemention and model of [20] has not been released yet:

– PConv: Method proposed by Liu G. *et al.*. All the convolutional layers are partial convolutional layer.

– GConv: Method proposed by Yu, J. *et al.*. All the convolutional layers are gated convolutional layer.

4.2.1 Qualitative Comparisons

Figure.2 shows the sample results on CelebA dataset. It can be seen that our model has exactly the same performance

Table 2. Results of mean l_1 error, mean l_2 error and tv loss on test images of CelebA with free-form masks.

Method	mean l_1 loss	mean l_2 loss	mean tv loss
PConv	2.77%	**1.04%**	**11.08%**
GConv	4.97%	2127%	14.03%
Ours	**2.63%**	1.07%	11.09%

as the other state-of-art models. Interestingly, for GConv model, we can see in Figure.3 there's a chance that it may mis-learn the mask and generate contaminating artifacts on none mask location. While the exact reason causes this may never known, there seems the over learning of mask maybe a cause of this weird phenomenon (statistically, we get around 1% of test result being contaminated by this random artifacts while all other two models have completely no problem). Since the mask itself actually have little correlation in high level, maybe we should consider whether it makes sense to learn a mask high level feature.

4.2.2 Quantitative comparisons

As mentioned in [11], there is no good numerical metric to evaluate image inpainting results due to it mostly depends on subjectiveness and there exists many possible solutions. Thus, we follows [20], reporting the l_1 loss, l_2 loss and tv loss on test set. Table.2 shows the comparison results. It can be seen that our method has similar performance with PConv network without the knowledge of mask and the huge gap on l_2 loss between GConv and the other two also shows the issues casued by mask over learning.

Also, the training efficiency is an important issue we should consider. Table.3 shows the parameters and training time for comparisons. As is illustrated above, the GConv network needs doubled parameters to work, while surprising only causes an extra 20% back propagation time. But Liu G. also mentioned in [11] that partial convolutional layer can be improved both in time and space using custom layers instead of a fixed convolution layer to perform mask updating. However, GConv actually runs the fastest when testing which tells that the partial convolutional layer really needs a speedup. Our approach has a good balance between computational cost and flexibility.

5. Conclusion

In this project, we proposed a fresh insight of the two state-of-art image inpainting networks by integrating the two kinds of convolutional layer efficiently. Our model can robustly handle arbitrary holes shapes at any possible location and the results are comparable with the two state-of-art approaches, in a more flexiable and computational efficient manner. Furthermore, we rasied the question on whether

Table 3. Parameters number and training time of three models. The training time is averaged using one testing epoch.

Method	parameter number	training time(per iter)	testing time(per img)
PConv	**25.78M**	**1.198s**	$\sim$0.06s
GConv	51.56M	$\sim$1.3s	**$\sim$0.05s**
Ours	26.00M	$\sim$1.4s	$\sim$0.055s

we should learn the mask high level feature. Around 1% of the testing results of GConv are contaminated by random artifacts on non-mask location, which the over learning procedure maybe the main cause.

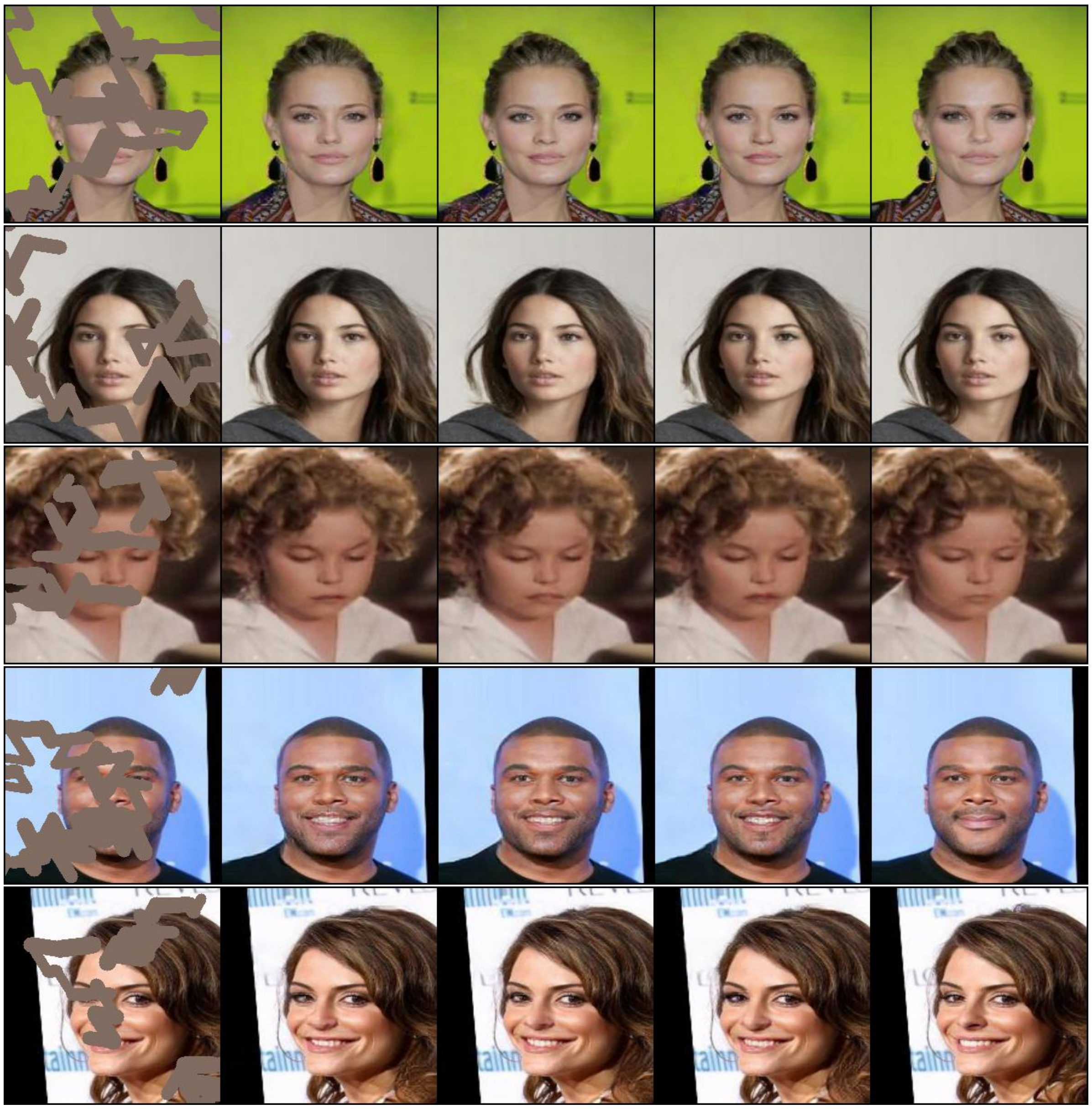
Figure 2. Sample testing results on CelebA images. From left to right: input, PConv, GConv, Ours and ground truth.

References

[1] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics (ToG)*, 28(3):24, 2009.

[2] T. Q. Chen and M. Schmidt. Fast patch-based style transfer of arbitrary style. *arXiv preprint arXiv:1612.04337*, 2016.

[3] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[5] S. Iizuka, E. Simo-Serra, and H. Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (TOG)*, 36(4):107, 2017.

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[7] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[8] R. Köhler, C. Schuler, B. Schölkopf, and S. Harmeling. Mask-specific inpainting with deep neural networks. In *German Conference on Pattern Recognition*, pages 523–534. Springer, 2014.

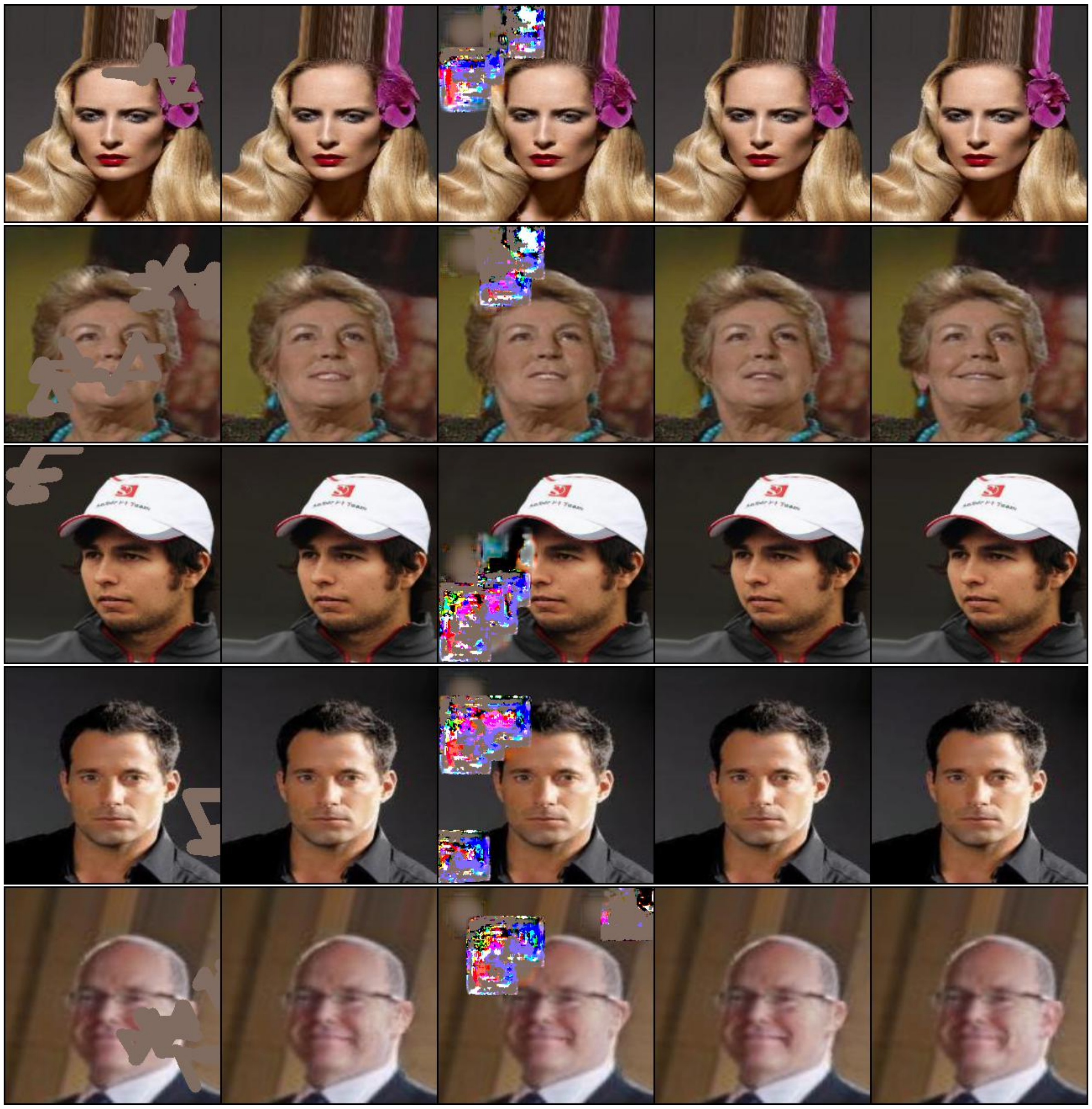

Figure 3. Another group testing results on CelebA images. We can see random artifacts generated by GConv network in non mask location while the other two remain clean.

[9] C. Li and M. Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2479–2486, 2016.

[10] Y. Li, S. Liu, J. Yang, and M.-H. Yang. Generative face completion. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5892–5900. IEEE, 2017.

[11] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro. Image inpainting for irregular holes using partial convolutions. *arXiv preprint arXiv:1804.07723*, 2018.

[12] Z. Liu, P. Luo, X. Wang, and X. Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15:2018, 2018.

[13] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[14] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[15] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[16] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[17] L. Xu, J. S. Ren, C. Liu, and J. Jia. Deep convolutional neural network for image deconvolution. In *Advances in Neural Information Processing Systems*, pages 1790–1798, 2014.

[18] C. Yang, X. Lu, Z. Lin, E. Shechtman, O. Wang, and H. Li. High-resolution image inpainting using multi-scale neural patch synthesis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, page 3, 2017.

[19] R. Yeh, C. Chen, T. Y. Lim, M. Hasegawa-Johnson, and M. N. Do. Semantic image inpainting with perceptual and contextual losses. arxiv preprint. *arXiv preprint arXiv:1607.07539*, 2, 2016.

[20] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Free-form image inpainting with gated convolution. *arXiv preprint arXiv:1806.03589*, 2018.

[21] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. Generative image inpainting with contextual attention. *arXiv preprint*, 2018.

Appendices

A. Details of Network Architecture

Table 4. Details of Network Architecture

Module Name	Module Name	#Filters/Channels	Stride/Up Factor	BatchNorm	Nonlinearity
GConv1	7×7	64	2	-	ReLU
GConv2	5×5	128	2	Y	ReLU
PConv3	5×5	256	2	Y	ReLU
PConv4	3×3	512	2	Y	ReLU
PConv5	3×3	512	2	Y	ReLU
PConv6	3×3	512	2	Y	ReLU
PConv7	3×3	512	2	Y	ReLU
NearestUpSample1		512	2	-	-
Concat1(w/ PConv6)		512+512		-	-
PConv8	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample2		512	2	-	-
Concat2(w/ PConv5)		512+512		-	-
PConv9	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample3		512	2	-	-
Concat3(w/ PConv4)		512+512		-	-
PConv10	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample4		512	2	-	-
Concat4(w/ PConv3)		512+512		-	-
PConv11	3×3	512	1	Y	LeakyReLU(0.2)
NearestUpSample5		512	2	-	-
Concat5(w/ PConv2)		512+256		-	-
PConv12	3×3	256	1	Y	LeakyReLU(0.2)
NearestUpSample6		512	2	-	-
Concat6(w/ PConv1)		256+128		-	-
PConv13	3×3	128	1	Y	LeakyReLU(0.2)
NearestUpSample7		512	2	-	-
Concat7(w/ Input)		64+3		-	-
GConv14	3×3	3	1	-	-

B. More Comparisons on Free-Form Masks

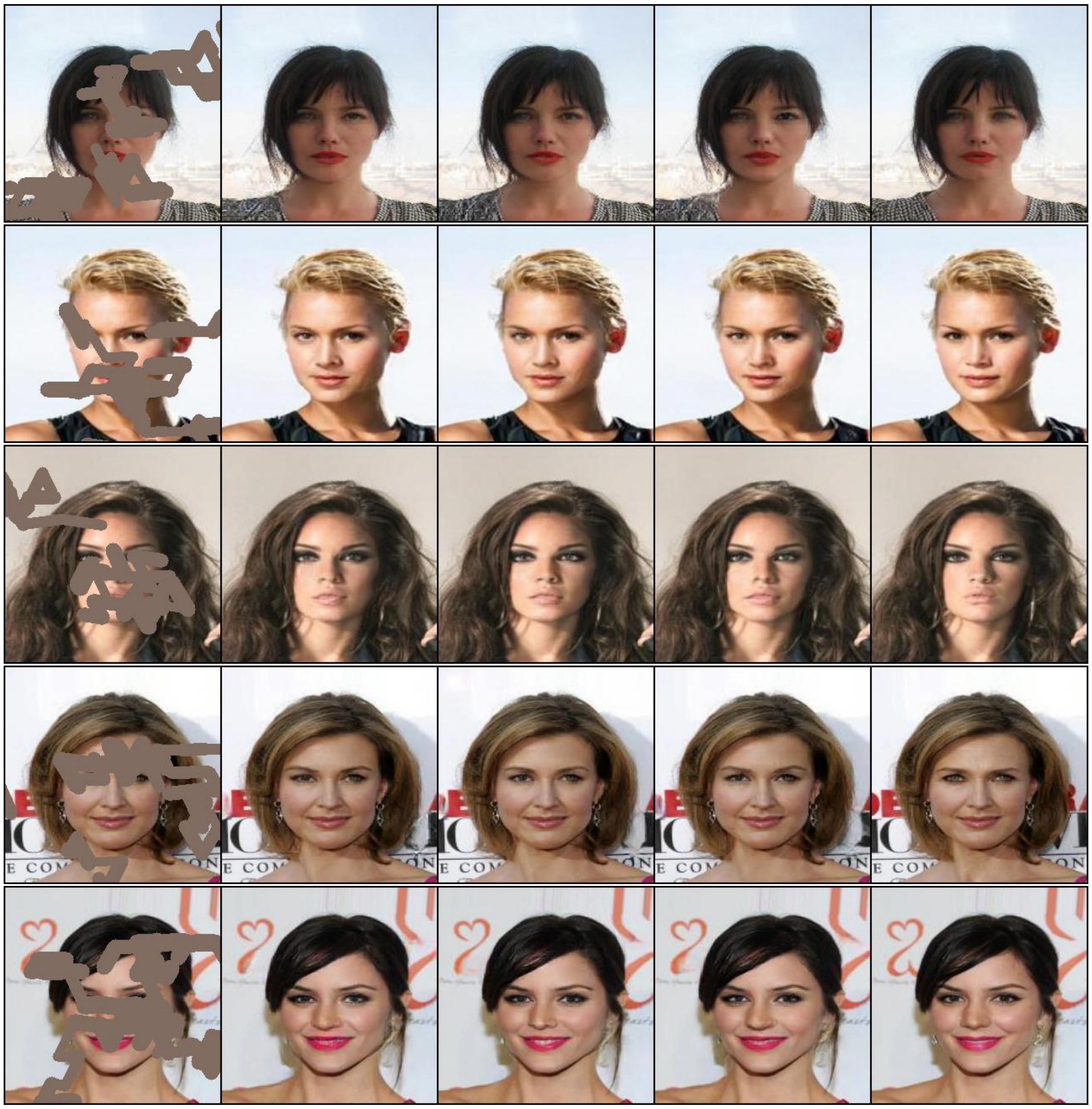

Figure 4. More Comparisons on Free-Form Masks. Still, from left to right is input, PConv, GConv, Ours and ground truth.

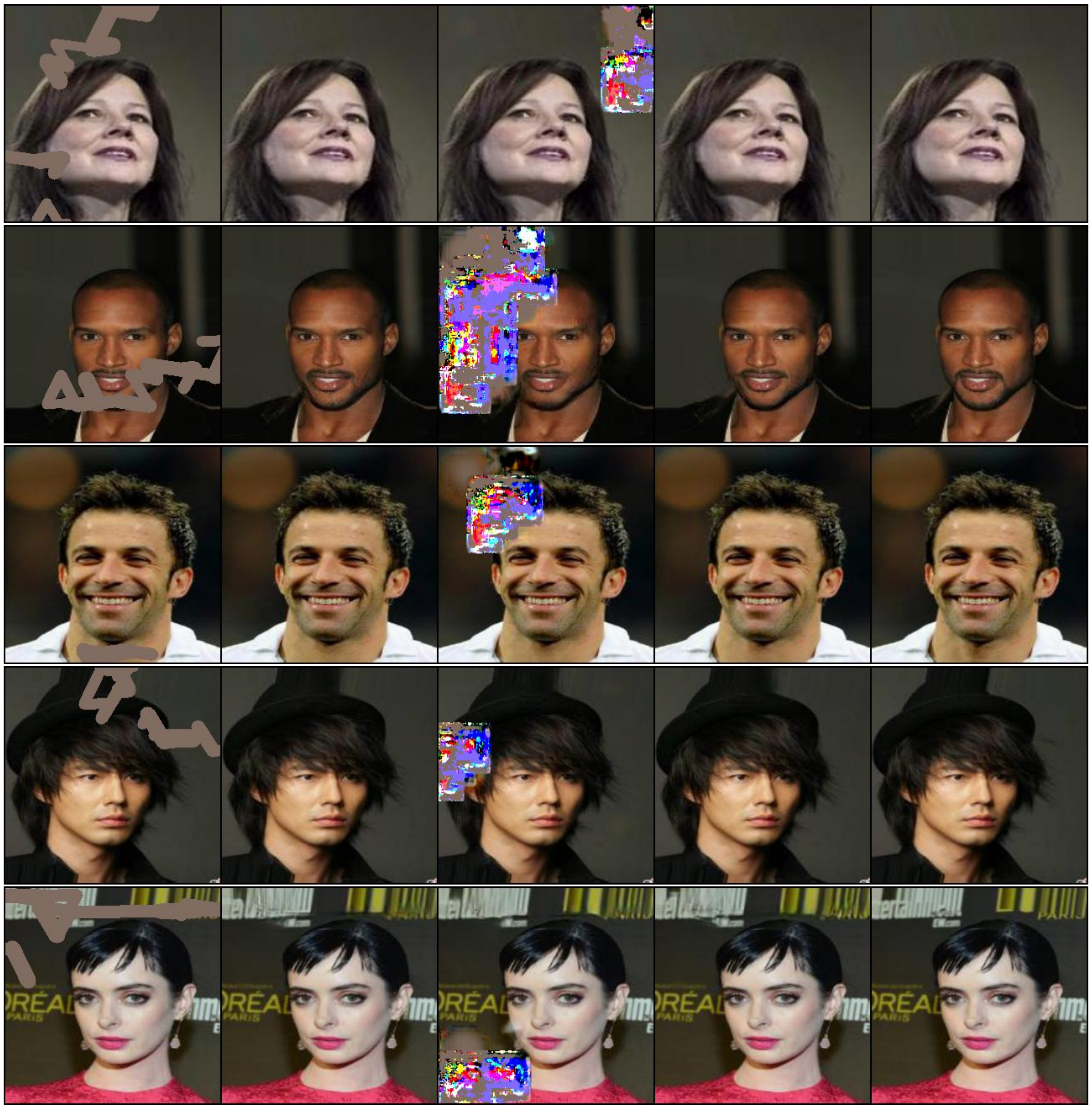
Figure 5. More contaminated outputs generated by GConv network

C. Training Process

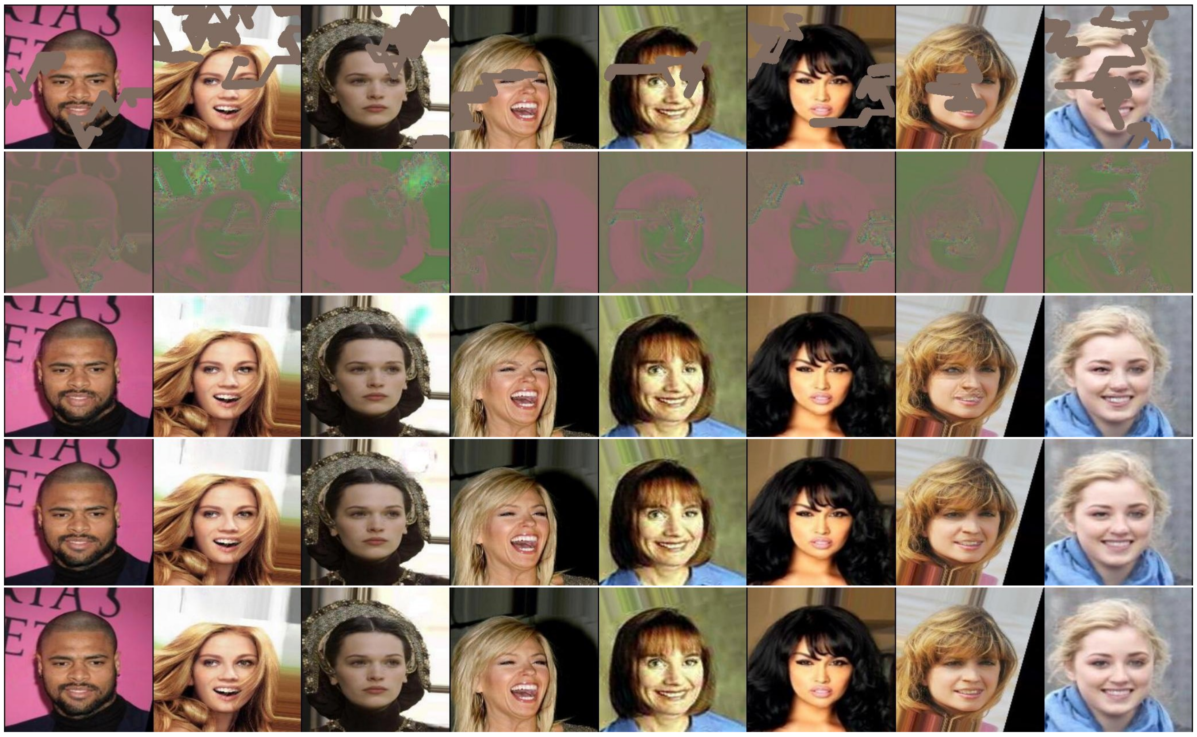
Figure 6. PConv validating results during training. When in validation, the mask is set to the same for the convenience of comparison. From top to bottom: input, initial weights result, after 20 epoches, after 100 epoches, final results

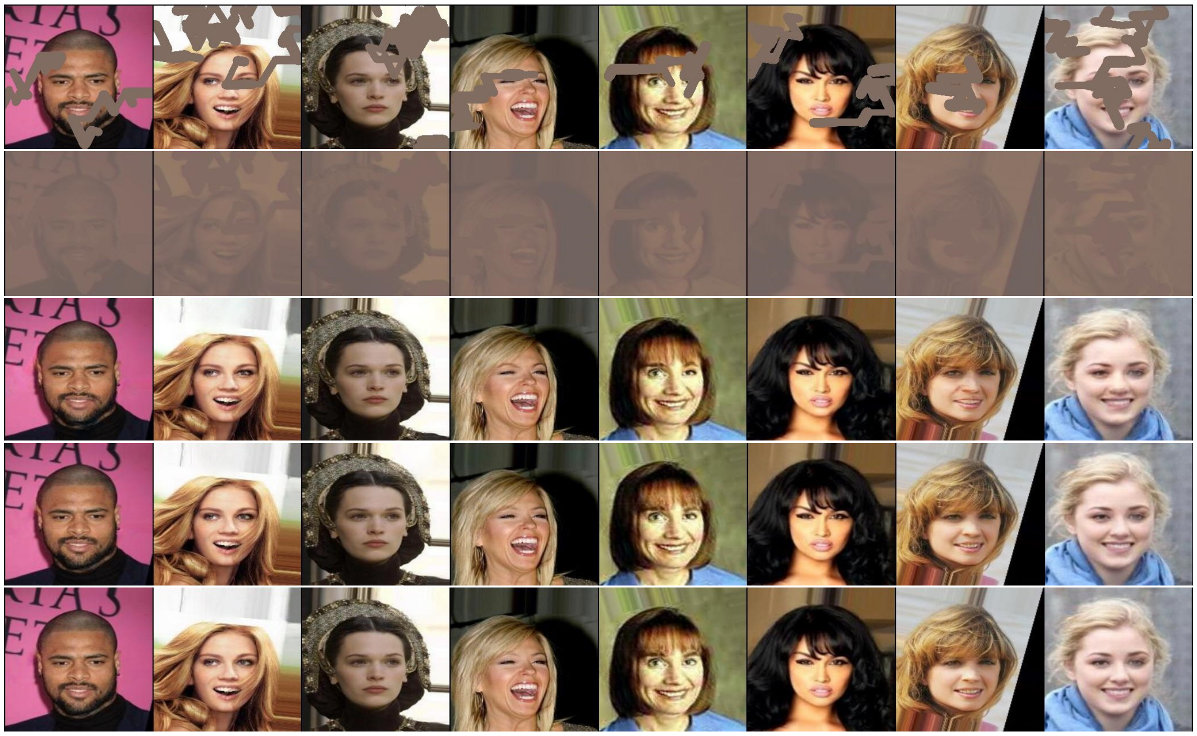
Figure 7. GConv validating results during training

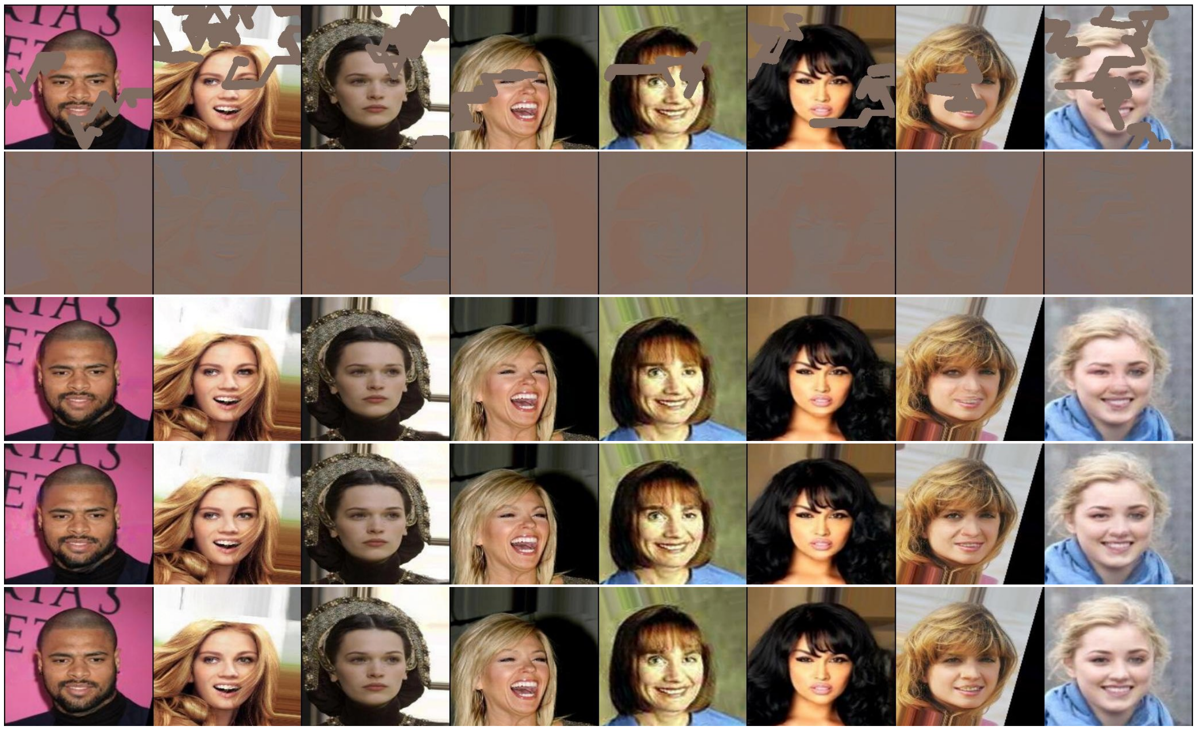
Figure 8. Ours validating results during training