Texture Synthesis with Recurrent Variational Auto-Encoder

Rohan Chandra Sachin Grover Kyungjun Lee Moustafa Meshry Ahmed Taha
{rohan,kjlee,mmeshry,ahmdtaha}@cs.umd.edu saching@umd.edu

Abstract

We propose a recurrent variational auto-encoder for texture synthesis. A *novel* loss function, FLTBNK, is used for training the texture synthesizer. It is rotational and partially color invariant loss function. Unlike L2 loss, FLTBNK explicitly models the correlation of color intensity between pixels. Our texture synthesizer [1] generates neighboring tiles to expand a sample texture and is evaluated using various texture patterns from Describable Textures Dataset (DTD). We perform both quantitative and qualitative experiments with various loss functions to evaluate the performance of our proposed loss function (FLTBNK) — a mini-human subject study is used for the qualitative evaluation.

1 Introduction

Texture synthesis is the process of generating a new texture given a texture sample. Mapping a texture onto surface, scene rendering, occlusion fill-in, lossy image, video compression, and foreground removal are applications for texture synthesis. In this project, DRAW network [1], initially proposed to generate MNIST dataset, is amended to generate textures. Generated tiles are constrained to both having the same texture and aligning smoothly with neighboring tiles. Our texture synthesis model can expand a sample texture and generate a new sample with user-defined dimensions. Figure 1 shows our pipeline.

We train DRAW to synthesize a texture and enforce smooth alignment between neighboring tiles. We propose a *novel* loss function, FLTBNKs, for training a generative texture network. It is evaluated against $L2$ loss, as a baseline, and the texture loss proposed by Gatys et al. [2]. As we sample texture tiles from DTD, DRAW, a recurrent variational auto-encoder, learns neighboring tiles. Multiple loss functions are evaluated for texture synthesis. In the deployment phase, the trained network generates the four neighboring tiles for a center tile — an initial sample texture. The generated tiles act as input in the next step to further expand the texture size.

[1] Code available at Github

2 Background

In this section, we describe the frameworks and the loss functions utilized by our project.

2.1 Deep Recurrent Attentive Writer (DRAW)

Deep Recurrent Attentive Writer [1] (DRAW), introduced by Google DeepMind, is a generative recurrent variational auto-encoder. Figure 2 shows DRAW network architecture. An encoder network determines a distribution over latent variables to capture input data salient information; a decoder network receives samples from the latent distribution and uses them to condition its own distribution over images. This operation is performed iteratively to update attention mechanism that selects "where to read", "where to write", and "what to write".

2.2 Texture Loss Functions

We evaluate four loss functions for texture synthesis. Traditional image loss functions such as L2 loss and cross-entropy are considered; yet, other specific texture loss functions are evaluated as well. Sections 2.2.1 and 2.2.2 describe filter bank loss and VGG loss – a recent loss function used for texture synthesis. The equations for these loss functions are summarized in table 1

2.2.1 Filter Bank

Varma et al. [3] propose a statistical approach for texture classification. Before deep neural networks made their mark, it was a very competitive classification approach. Despite being primitive, it is popular for its simplicity and accuracy. To generate a texture descriptor, a filter bank is applied on a texture image. The Leung-Malik (LM), shown in figure 3, is a commonly used filter bank. After filtering, a texture image becomes a L-dimensional image, where L is the number of filters. Each pixel, L-dimensional vector, gets classified to pre-trained cluster centers called textons. Using these textons, a texture image is represented by a texton histogram. We propose such concept for training our texture synthesis network.

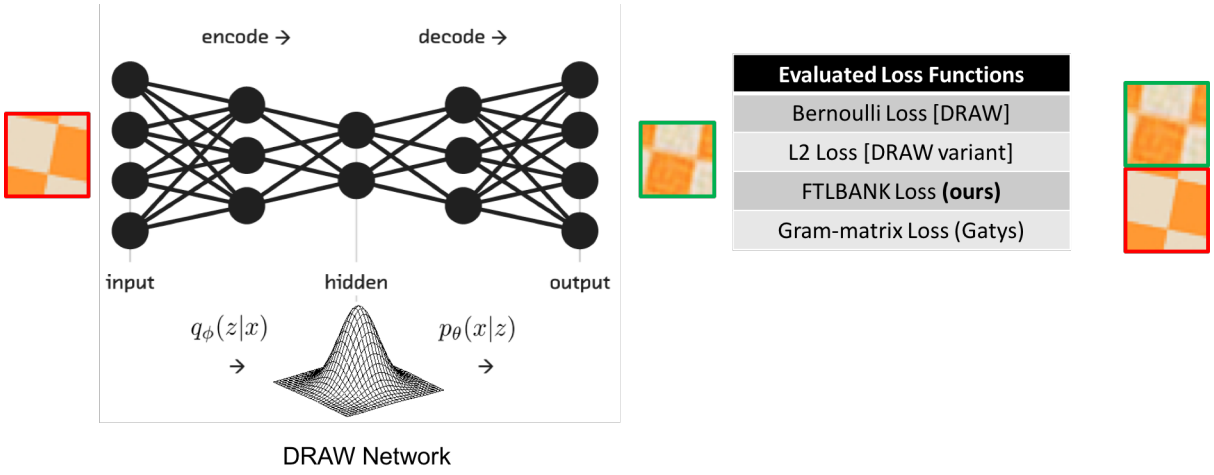


Figure 1: Network Pipeline. In the training phase, a texture and the corresponding neighbor tiles are sampled from DTD. Using center tile as input, DRAW network learns the neighbor tiles. Multiple loss functions are evaluated. In deployment phase, the center tiles is fed as input, and the generated neighbor tile is stitched to the input tile. To expand further, previously generated tiles are fed in DRAW as input.

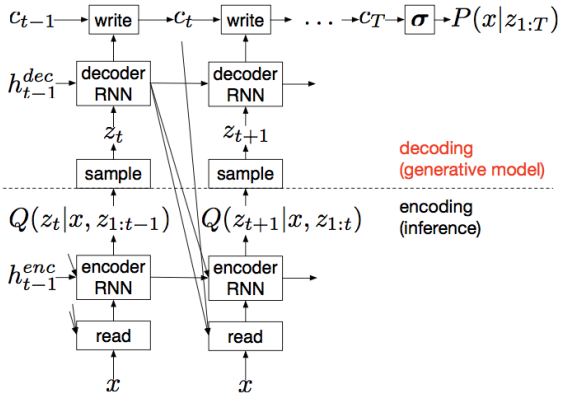


Figure 2: DRAW, a recurrent variational auto-encoder network.

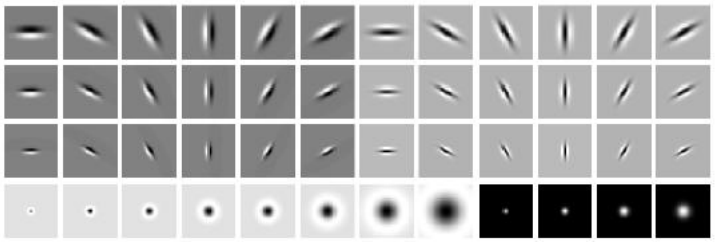


Figure 3: LM filters bank

2.2.2 VGG

Another recent loss function for texture generation is very deep convolutional networks (VGG). It is first introduced in a large-scale image classification work [4]. This texture loss function is proposed by Gatys at el. [2] for texture generation and adapted later for image style transfer. Basically, feature maps generated at different VGG-network layers, by similar textures, have high correlation. Thus, the correlation between input texture x and the synthesized image $\hat{x}$ is a quantitative metric. To elaborate, x and $\hat{x}$ are fed into the VGG-network independently, each producing their own set of feature maps, $F^l \in \mathbb{R}^{N_l \text{x} M_l}$ and $\hat{F}^l \in \mathbb{R}^{N_l \text{x} M_l}$, where each layer l has N_l distinct feature maps each of size M_l when vectorized. F^l_{jk} is the activation of the j^{th} filter at position k in layer l. The correlation between these feature maps are stored in a matrix G and $\hat{G}$ respectively. $G^l \in \mathbb{R}^{N_l \text{x} N_l}$ is defined as:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

The distance, between two textures, is the weighted sum of layer-wise distance.

$$D_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - \hat{G}^l_{ij})^2$$

$$\mathcal{D}^{total}(x, \hat{x}) = \sum_{l=0}^{L} w_l D_l$$

3 Approach

The DRAW [1] network is a variational generative model to generate images that cannot be distinguished from real data with the naked eye. Our project extends the application of the DRAW network to generate neighboring textures tiles from the DTD dataset [5]. Our approach is illustrated in figure 4, and table 1 shows all the reconstruction loss functions used for evaluation. The following subsections describe our approach details

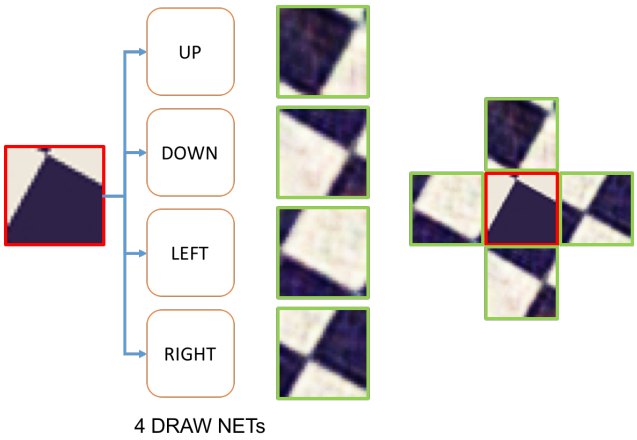


Figure 4: Four DRAW networks are trained independently to generate the four neighboring tiles. Generated tiles are stitched to the input center tile to output the final texture

Evaluated Reconstruction Loss Functions	
Cross Entropy	$y \log \hat{y} + (1-y) \log (1-\hat{y})$
L2	$(y-\hat{y})^2$
FB	$(LM(y) - LM(\hat{y}))^2$
VGG	$(1/(4N_l^2 M_l^2)) \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2$ $G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$

Table 1: Reconstruction loss functions evaluated for texture synthesis. In our experiments, different combinations are evaluated. Total network loss is reconstruction loss plus latent loss.

3.1 Texture Generation

To train our network, five regular texture patterns are selected from DTD. During each training epoch, tiles are sampled from these five texture patterns. The DRAW network is fed the center tile and evaluated using both the latent and reconstruction loss. Different combination of loss functions, shown in table 1, evaluates the reconstructed texture tile L^c. At the same time, a latent loss L^z evaluates the latent variable distribution.

We use a Gaussian latent variable distribution for two reasons. First, the gradient of a function of the samples with respect to the distribution parameters can be easily obtained using the so-called reparameterization trick [6, 7]. This makes it straightforward to back-propagate unbiased, low variance stochastic gradients of the loss function through the latent distribution. Since our latent prior is a standard Gaussian with mean zero and standard deviation one. The Kullback-Leibler divergence is computed by the closed form equation:

$$L^z = \sum_{t=1}^{T} KL(Q(Z_t|h_t^{enc})||P(Z_t)) \tag{1}$$

$$L^z = \frac{1}{2}\left(\sum_{t=1}^{T} \mu_t^2 + \sigma_t^2 - \log \sigma_t^2\right) - \frac{T}{2} \tag{2}$$

The original filter bank, proposed in [3], computes a histogram for a texture image. Histograms are not differentiable; so, back-propagation becomes infeasible. To overcome such obstacle, loss is computed directly from LM filter responses. Thus no clustering assignment or binning is performed. Throughout this report, FB refers to LM filter bank response; while FTLBNK refers to a combination of FB with other losses defined in section 3.2. The total loss L for the network is the sum of the reconstruction and latent losses $L = L^c + L^z$. Once the four networks are trained, neighboring tiles are generated by feeding center tile and stitching the tiles accordingly.

3.2 FLTBNK Loss Function

We propose a novel loss function, called FLTBNK. It utilize the idea that filter banks are rotationally and partially color invariant. The filter banks normalize the images and their filter responses. Thus, it is partially invariant to changes in illumination intensity. Neighboring pixels contribute to a pixel filter's response. Thus, unlike L2 loss, a pixel intensity is correlated with its neighbors. Such characteristics nominate FLTBNK loss function to train our texture generative network.

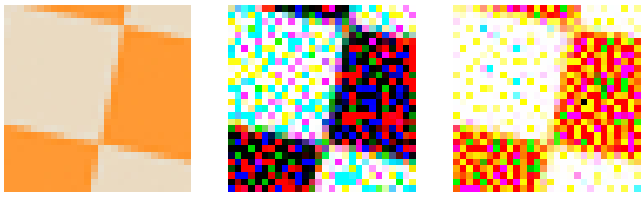
Texture 5(b) shows the noisy reconstructed texture image when using FB only. To address the color problem, we apply a color regularizer that optimize for the mean color within a tile; this generated texture 5(c). In texture 5(d), we reduce the noise by adding total variation loss to FB. To eliminate both color and noise problems, both total variation and color regularization are added to FB loss as shown in texture 5(e).

4 Experimental Results

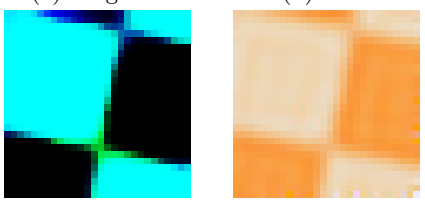
In this section, we show qualitative results of our model on two tasks: texture reconstruction and large texture generation from a small texture sample. Then, a small quantitative evaluation of the synthesized texture is provided. Finally, we show a human subject study to evaluate generated textures.

4.1 Input Texture Reconstruction

The first evaluation metric is sample texture reconstruction. Figure 6 shows reconstructed textures using four different losses: cross-entropy, L2, FTLBNK and VGG. It shows that corss-entropy, L2 or FTLBNK generates a blurry output. This happens because DRAW



(a) Original (b) FB (c) FB+col reg



(d) FB+TV (e) FB+ col reg +TV

Figure 5: Effect of different regularizations on the filter-bank loss. First row shows (a) input textures, (b)filter-bank loss with no regularization, and (c) filter-bank loss with a color regularizer respectively. Second row shows (d)filter-bank loss with total variation and (e) filter-bank loss, total variation with a color regularizer.

is a variational auto-encoder where the loss function is a variational upper bound on the log-likelihood of the data.

All the losses evaluated optimize the generated data likelihood using a latent representation $P_\theta(\tilde{X}|Z)$. In some sense we are averaging out and we are trying to fit a unimodal distribution to a multimodal solution which results in blurry images. On the other hand, the VGG loss didn't work well with our approach. One reason is that the VGG loss requires large input images, since the VGG architecture contains many pooling layers. To verify this reason, we ran a public implementation of [2], and we confirmed that it only works with large images, while it generates almost random noise with small inputs (such as 28x28), as shown in figure 7

4.2 Texture Generation

The second evaluation metric is sample texture expansion. Figure 8 shows qualitative output of generated textures using two different losses: L2 and FTLBNK. Generation using the L2 loss captures texture colors well, but it doesn't perform very well on more complicated textures. For example, the first sample of green texture contains many fine details, and the last sample texture contains curves instead of a simple chess-board tiles. In both cases, the L2 loss performs poorly. On the other hand, FTLBNK does a better job capturing the textures outlines, but it suffers from a color-problem. This is probably because the color regularization term from section 3.2 focuses largely on the mean color value for the red, green and blue channels independently and less on the correlation between different color channels.

To numerically evaluate generated textured without bias, generated texture is compared with original textures using two distance metric: VGG and texture histogram as originally proposed in [3].

Figure 9 shows the VGG and histogram distance between original and synthesized texture. The orange and green textures from figure 6, first and third columns, are used for evaluation. From the figure, we conclude that a network trained with VGG loss generates a texture with smaller VGG distance. On the other hand, a network trained with FB loss generates a texture with smaller histogram distance. A merit for histogram distance is using textons which are clusters trained one a texture dataset. Thus, we believe the histogram distance is more informative than the VGG distances based on solely a pair of images information.

4.3 Human Subjects Evaluation

To further evaluate the loss functions used to generated textures, we conducted a short human subject study. The study was designed as a survey asking human subjects to rate the quality of resemblance (5 point Likert Scale) between the original and generated textures for each loss function. All participants are machine learning students, thus, our study suffers some level of selection bias. Due to time constraints, we obtained 18 responses, but this is enough to run a statistical test. To avoid confirmation bias, the loss functions names are hidden in the survey. As the independent variable is categorical, and the dependent variable is ordinal (5 point Likert Scale), we perform the Kruskal-Wallis Test (non-parametric). The p-value is $<< 0.05$ and therefore we reject the null hypothesis that the medians of all groups (loss functions) are equal.

Figure 10 shows a summary of the responses. The key observations is that although traditional losses (cross-entropy and L2) have more 'Excellent' ratings, our proposed loss, FLTBNK, seems to get better overall ratings. FLTBNK gets many 'Very Good' ratings compared to other losses. We hypothesize that the results of traditional losses have high variance, while results of FLTBNK are more consistent, and so, we see that a majority of the responses to traditional losses are 'Acceptable', whereas for FLTBNK the majority of responses are 'Very Good'.

5 Related Works

Several recent work uses convolutional neural networks for texture synthesis. Gatys et al. [2] proposed a new texture synthesis model based on the feature spaces of convolutional neural networks. This model computes the Gram matrices on feature maps to transform textures into a feature space. However, they do not evaluate how suitable the Gram matrices are to recognize textures, which might lead to low accuracy of the model on certain textures. Also, this model can only

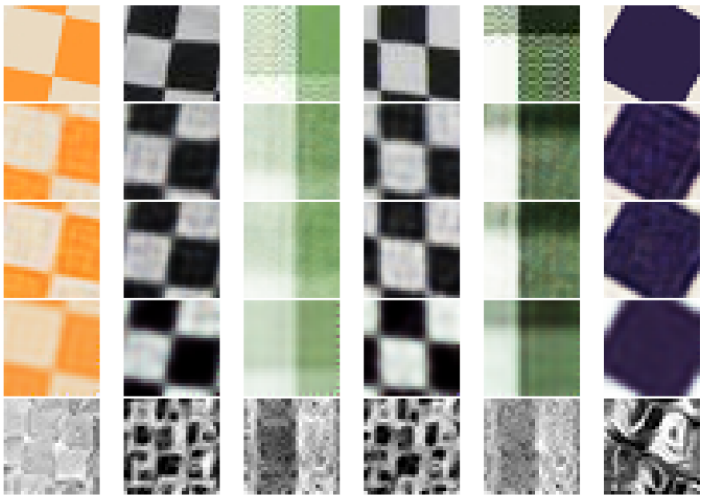


Figure 6: Sample texture reconstruction. First row shows the original input texture. The following rows show reconstructed textures using the cross-entropy, L2, FTLBNK and VGG losses respectively. Each tile is 28x28 pixels.

(a) Sample from original texture (b) Synthesized 28×28 texture (c) Synthesized 50×50 texture (d) Synthesized 80×80 texture (e) Synthesized 196×196 texture

Figure 7: Texture synthesis using an implementation of Gatys *et al.* [2]. Results show that VGG loss needs large input images to work properly.

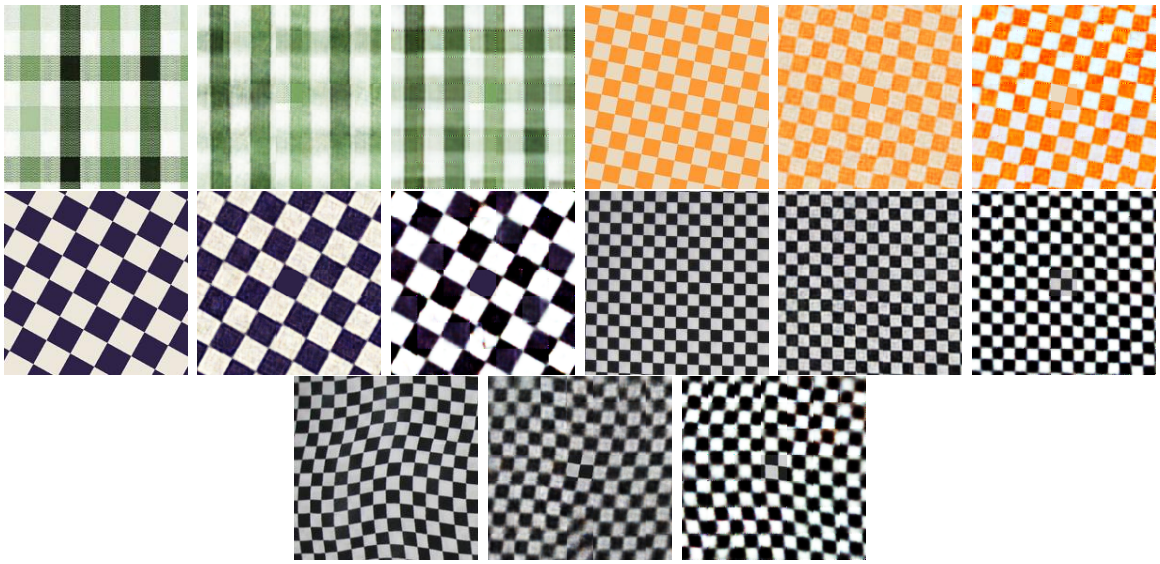


Figure 8: Generating texture. Input is only a 28x28 center tile, while the output size is 196x196. For each image, the left: the original texture; middle: generated texture using L2 loss; right: generated texture using filter-bank loss

generate a textures of same size as input texture, while our model can expand a textures using a small sample. Yet, it is able to generate more challenging textures than our model.

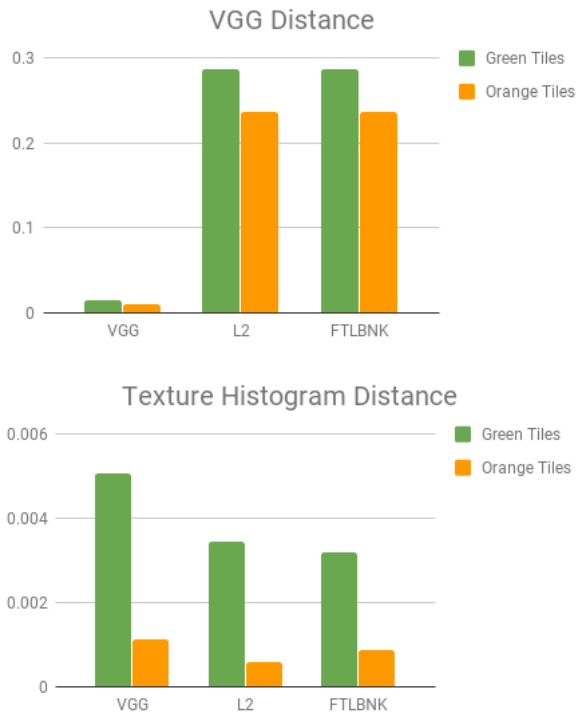


Figure 9: Distance between original texture and synthesized texture using both VGG and histogram distances. The orange and green textures from figure 6 are used for evaluation.

While Gatys et al. model requires a lot of memory to process the inputs, a new model proposed by Ulyanov et al. [8] alleviate the memory overheads. The new model reduces memory overhead during training by putting computational burden to a learning stage, while maintaining the same accuracy. Yet, this memory-less model does not expand a small texture sample, which our model does.

To avoid deep neural network inherently high computational costs, one research work on texture synthesis introduced computational cost reduction using Markovian generative adversarial networks [9]. This model utilizes adversarial generative networks in a Markovian setting to capture the feature statistics. As this model improves the performance and is able to generate continuous images with an image of a small fixed size, it would be interesting to compare our model with this model in terms of the synthesized image quality. Evaluation between our proposed model and the Markovian generative adversarial model is a worth-noting extension missing due to time constraints.

6 Conclusion

We use DRAW, a variational generative model, for texture synthesize. Our model expands a small sample texture by generating neighboring tiles. A novel texture loss function, FLTBNK, is proposed to impose color correlation between generated pixels. A qualitative study, with 18 human subjects, shows that a combination of FLTBNK, variational loss and a color regularize is competitive as l2 loss. While exact color matching is attractive attribute for human subjects, it is weakly required in texture context. Thus, FLTBNK, a combination of filter bank response, color variational loss and a color regularizer is more intuitive for texture generation.

References

[1] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1462–1471, 2015.

[2] L. Gatys, A. S. Ecker, and M. Bethge, "Texture synthesis using convolutional neural networks," in *Advances in Neural Information Processing Systems*, pp. 262–270, 2015.

[3] M. Varma and A. Zisserman, "A statistical approach to texture classification from single images," *International Journal of Computer Vision*, 2005.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[5] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, , and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[6] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International Conference on Machine Learning*, pp. 1278–1286, 2014.

[7] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2015.

[8] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. S. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images.," in *ICML*, pp. 1349–1357, 2016.

[9] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *European Conference on Computer Vision*, pp. 702–716, Springer, 2016.

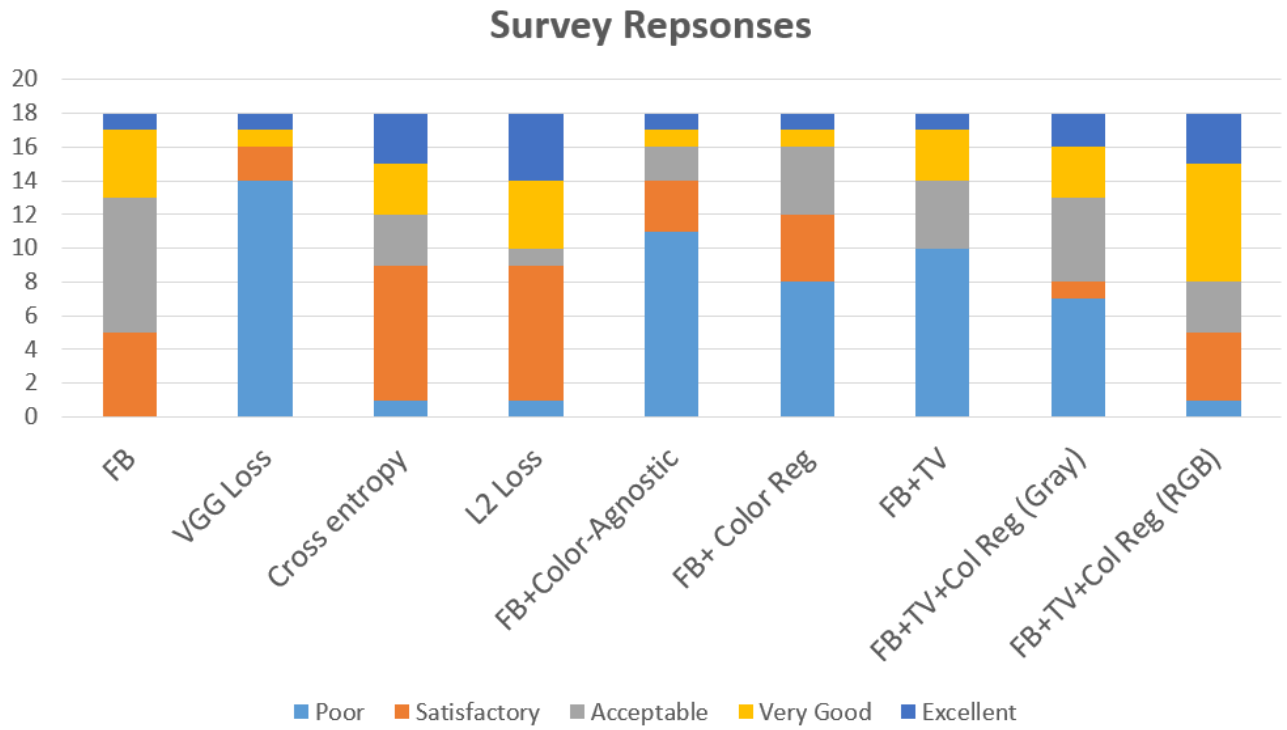


Figure 10: Ratings from 18 participants on a 5-point scale. Participants prefer traditional losses, like cross entropy and L2, outputs in terms of excellent rating. Yet, FLTBNK loss, last column, gets better overall ratings. FLTBNK gets many 'Very Good' ratings compared to other losses.