REPRODUCING "EMERGENCE OF SHAPE BIAS IN CNNS THROUGH ACTIVATION SPARSITY"

Lianghong Ye, Michail Rontionov & Michalis Kontos
{l.ye,m.rontionov,m.kontos}@soton.ac.uk

1 INTRODUCTION

We have investigated and implemented work by Li et al. (2024), which aims to demonstrate how Convolutional Neural Network (CNN) sparsity can be used to control network bias towards shape or texture. Our approach comprised of selecting relevant figures that best describe the conclusion of the paper, implementing the proposed network amendments individually, reproducing the selected figures, and comparing with the original paper. We have selected Figures 2, 3 and 5, which showcase a good description of the original paper.

2 TEXTURE SYNTHESIS

To understand the information included in the Top-K responses in a particular layer, in this paper, the researchers compare the texture-synthesis output with and without Top-K filters. This paper is based on the texture-synthesis approach, proposed by Gatys et al. (2015), which characterise texture by the correlation between feature responses across multiple layers of the CNN.

2.1 ALGORITHM

The texture synthesiser model includes two components: texture analysis and texture generation. In the first part, the source image T, whose texture is wanted, is passed through a modified CNN, which acts as a feature extractor. An inference process is conducted and the feature maps, generated in each convolution layer i, are represented as $X_i(\cdot)$.

Texture should be *spatially independent*. To rule out spatial information from the synthesised image, a Gram matrix of layer activations $Gr(X_i(T))$ are used, which captures the correlation between features. Applying $Gr(\cdot)$ to target layers returns a set of target Gram matrices $\{G^1, G^2, ..., G^L\}$.

In the second part, texture is generated based on the target, achiev by optimising a white noise image I to match the Gram matrix specification of the source image T. The loss then consists of a difference between the target T and objective, and is optimised by `LBFGS` in the paper. The "Texture Synthesis (TS)" method can then be written: $I \leftarrow I - lr * (\frac{\partial}{\partial I} \sum_i^L [Gr(X_i(I)) - Gr(X_i(T))])$

The gradient of layer-wise loss with respect to input $\frac{\partial \mathcal{L}}{\partial I}$, therefore, can be computed using back-propagation, and used as the input for numerical optimisation method to update the input image I, which evolves over iterations to match texture of the source image.

The paper had another method of evaluating the shape v. bias hypotheses. The other method, deemed "Reconstruction" is similar to TS with one important difference: it does not calculate the gram matrix and instead acts on neuron activations directly. The optimisation for reconstruction is written as $I \leftarrow I - lr * (\frac{\partial}{\partial I} \sum_i^L [X_i(I) - \texttt{Mask}_i * X_i(T)]$.

The `Mask` vector is used to select which responses the network cares about: Top-K, non-Top-K, or all responses (identity). From this optimisation, I would resemble the original image, where Top-K responses emphasise structure, and non-Top-K responses emphasise texture.

2.2 IMPLEMENTATION

The first implementation was heavily based on the paper introducing texture synthesis Gatys et al. (2015). The original texture synthesis module used in *Emergence of Shape Bias* paper was built upon a pre-trained VGG16 network from a source which is no longer available, therefore we had to rewrite it. We took the VGG16 model with pre-trained weights from the `PyTorch` library, ignored

the FC layers, and replaced MaxPooling layers with AveragePooling as specified by Gatys et al. (2015). We did not apply weight rescaling, as per Petit (2019) it has not shown not bring significant improvements for the output. Due to seperation of work, we created two seperate programs to generate the desired figures, one for TS and the other for reconstruction.

To extract feature maps from output of conv layers and calculate the gradient, a `TextureSynthesizer` class was created, serving as an analyser layer that can be inserted to any part of the model. Initialisation of this class takes a Gram matrix as input and stores it as reference. It also has a forward method, but input is returned unchanged as output, therefore behaves as a simple hook. In the meantime the layer-wise loss E_l with top-k mask is calculated and stored by comparing the reference Gram matrix with the Gram matrix computed using activation of feature map given as input.

The first component of texture syntheses model, texture analyses, is carried out using the `analyze` function, where modified pre-trained model is loaded, source image is loaded as input, and an inference pass is carried out. Instances `TextureSynthesizer` is inserted to the models at user specified location and initialised with feature map as input. The generated texture image can therefore be found at input layer.

The reconstruction model used a different approach, and derived the code based on descriptions in the paper we are recreating Li et al. (2024). It was designed by introducing a seperate layer as a `torch.nn.Module`, and injecting it into the imported VGG16 model. The loss architecture was similar to the previous one: (1) a forward pass was made for the target image, (2) target activations were acquired with a forward hook and stored in the Top-K layers and then removed and (3) the losses of subsequent forward passes stored and later combined via the sum operation.

3 Synthesis Figure Recreation

3.1 Figure 2

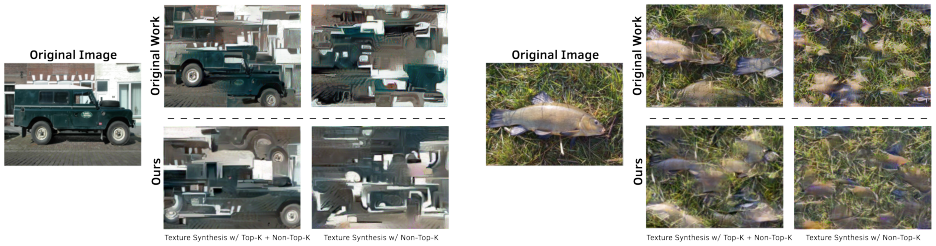

Figure 1: Re-creation of Figure 2 from Li et al. (2024), comparing original (upper) and ours (lower).

Figure 1 shows the original figure alongside our recreation. The goal of this image is to demonstrate which features the VGG16 network has learned to look for in input images, and proves that sparsity has a significant effect on where the network looks for correlations.

The left part of Figure 1 shows an image of a jeep. We see that in both the original work and our re-creation, local features such as wheels, windows, and the jeep's structure tend to group together. In the non-Top-K responses, we see that the features next to each other tend to have different colours with less structure, which hints that removing Top-K responses removes the structure encoding in the network. Similarly in the right part of 1, we have pictures of fish with their recreations in the same format. We see how the shape of a fish is much clearer in the left synthesis, with the right being much more random.

The difference between our and original work here is remarkably similar. As the networks used in the original paper and our work are slightly different due to versioning issues, the robustness of the hypothesis is proven. The synthesised images describe the same problem, we conclude that this is hypothesis is correct.

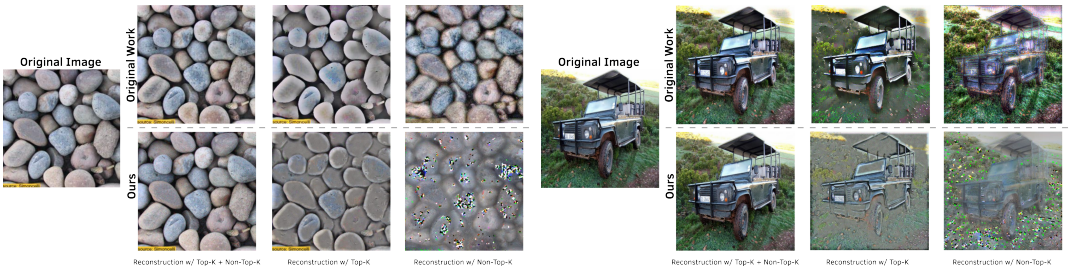

Figure 2: Re-creation of Figure 3 from Li et al. (2024), comparing original (upper) and ours (lower).

3.2 Figure 3

Figure 2 shows the reconstruction of the original images (left) by optimising input image I to match specific activations. This illustration demonstrates which parts of the image the network finds significant depending on the Top-K filter. The first column (Top-K + Non-Top-K) effectively serves as a control, as selecting both of the filters is the identity mask to make sure the filters are working correctly. There is a difference in the reconstructions between the original works and ours, however it is visible that they describe a similar phenomenon.

In the left part of Figure 2, an image of many intersecting rocks is used. The Top-K reconstruction in the original work shows contours of where the rocks are placed, with the middle part (the texture) less legible. The effect of the original work is a blur on the texture of the rocks, signifying that they are of little importance. In our reconstruction, we see that the effect is exaggerated with the texture being completely greyed out, meaning that Top-K responses of our network value texture very little. The Non-Top-K reconstructions, on the other hand, show an emphasis on the texture in both reconstructions. The right part of Figure 2 describes a similar situation, giving more evidence to the hypothesis. It is reasonable to say that our implementation of Top-K reconstruction acts as an edge-detector.

It is evident that our reconstruction with the original has much more noise associated with it. If we compare the output of our Top-K and non-Top-K reconstructions, we observe that texture details that were picked up by the Top-K filter show significant noise in the non-Top-K filter. This is likely due to implementation differences, as the implementation was based off the paper and not code. Nevertheless, if you look past the noise we can observe how similar the texture is on the non-noisy parts, with which we conclude that the two observations describe the phenomenon.

4 Benchmark vs Humans

The paper moves on to evaluate the effectiveness of Top-K activated CNNs with varying levels of sparsity in recognising shape biases, using a texture-shape cue conflict dataset proposed in previous research. Past studies have demonstrated that CNNs struggle with shape-based tasks compared to humans, who excel in such classifications. Integrating the Top-K operation into simple pre-trained models like AlexNet or VGG16 significantly enhances shape bias. With sparsity set at 10% and 20%, Top-K alone can rival or surpass the shape bias of state-of-the-art ViT models.

4.1 Implementation

The authors conducted their studies using a Python framework called modelvshuman from Geirhos et al. (2021), which made it easier to assess how humans and various vision models differ from one another in terms of texture bias. The experimental replication focused solely on AlexNet due to computational constraints, implementing Top-K sparsity levels of 50%, 40%, 30%, 20%, 10%, 5%, and the original unaltered model. This approach was chosen to reproduce results outlined in the paper. It is noted that the python framework includes a broader range of datasets and models for evaluation.

Moreover, a custom implementation of the top-K layers was implemented. This is to test the validity of the described steps taken along with the mathematical approach described. The new algorithm can

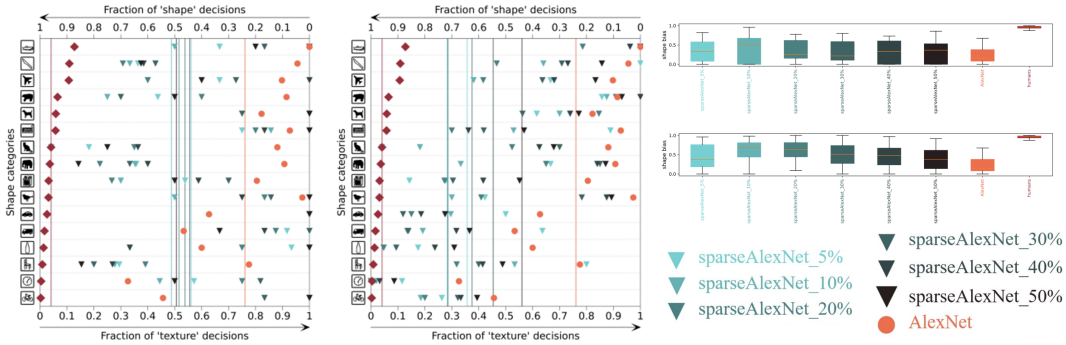

Figure 3: Re-creation of Figures 4 & 5 of the original paper. (Left) Using custom topK implementation. (Right) No modification done. (Top boxplot corresponds to left graph, and bottom to the right).

be found in the project's repository, named *topK_custom.py*. The place where top-K is implemented remains the same, but the way it is done was changed. To test this, a change in the *model_zoo.py* file should be made, to use the custom implementation.

4.2 Analysis

When employing the author's implementation without modifications, the replicated results closely resembled the original findings in terms of identifying which model performed best across different sparsity levels. However, there were slight discrepancies noted, which could be attributed to factors such as the randomisation process. This can be seen from figure 3, where the "no modification" graph and boxplot show the same results as Li et al. (2024).

The replication process involved a modification to the TopKLayer class within the specified directory (/modelvshuman/models/pytorch/topK/topK_custom_v1). This alteration led to notable deviations from the original results, characterised by overall inferior performance across all models, including those anticipated to exhibit the most favourable outcomes (10 and 20% top-K). It can be seen that all models with the implemented top-K layers are now closer together, at around the 0.5 mark. The models can now better recognise some categories situated towards the top of the graph but fail in the ones towards the bottom. Since the dataset being used replaces the texture of an image with the texture from other classes of images, it makes more sense for the resulting graph to spread out rather than favour one class over another.

5 Conclusion

In conclusion, we have described the original paper's goals, implemented the hypothesis and demonstrated the similarity between our implementation and the author's. We have shown that the original paper omits some implementation details, which is why we couldn't recreate the figures exactly, but have shown them come to the same conclusion.

References

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. *Advances in neural information processing systems*, 28, 2015.

Robert Geirhos, Kantharaju Narayanappa, Benjamin Mitzkus, Tizian Thieringer, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Partial success in closing the gap between human and machine vision. In *Advances in Neural Information Processing Systems 34*, 2021.

Tianqin Li, Ziqi Wen, Yangfan Li, and Tai Sing Lee. Emergence of shape bias in convolutional neural networks through activation sparsity. *Advances in Neural Information Processing Systems*, 36, 2024.

Romain Petit. Visual texture synthesis with convolutional networks, May 2019. URL `https://github.com/rpetit/texture-synthesis/blob/master/report.pdf`. [Online; accessed 15. May 2024].