ALADIN: All Layer Adaptive Instance Normalization for Fine-grained Style Similarity

1. Further detail on BAM-FG Dataset

We propose BAM-FG, a novel dataset of 2.62 million digital artworks in 310K project groups sampled from Behance.net – a creative portfolio website. The dataset will be released upon publication of this work. We gather annotation over 1.62M of these images as discussed in the main paper to create BAM-FG-C_N from 135K projects sampled at random from BAM-FG. We provide further detail on the crowd-annotation task that cleans this data using workers (turkers) on Amazon Mechnical Turk (AMT).

The subjective nature of artistic style required the task to be designed around getting consensus from multiple annotators of the same input. Each project was presented as an interactive mood-board to turkers, and they were asked to select the largest group of images containing a similar style, if one was present. Fig. 2 (left) shows an example mood-board. Turkers were trained using some example groups and were instructed to select based on appearances rather than semantics.

We gathered the grouping from this and four other turkers for each moodboard, and the graph in Fig. 2 (right) visualizes the post-processing algorithm. The values for pairwise edges between images were aggregated in an affinity matrix, which was then thresholded at the five consensus levels C_N.

An item-wise list of similarities is first created from the affinity matrix to cluster the remaining connections into groups. The list items are iterated, and the sample index pairs are added to an array of sets, where each set is a new group. On each iteration, error checking is performed to ensure that no data index is contained by multiple sets. If this happens, the two sets are merged. This ensures that a group is created from samples, even if not all samples within the group are connected with the same strength.

Further to the group co-membership information in the dataset, this strategy implicitly offers information on the strength of group co-membership an image has to other images at different consensus levels.

The lower consensus levels reflect less strict similarities but contain the highest average number of images per group. Similarly, the higher consensus levels reflect strong similarities between images, but at the cost of lower average image counts per group. Fig. 1 shows some example style groups from the first consensus level C_1. Figure 5 shows some examples of groups before and after cleaning. The choice of 5 turkers was determined from preliminary experiments as the appropriate balance between cost and consensus signal.

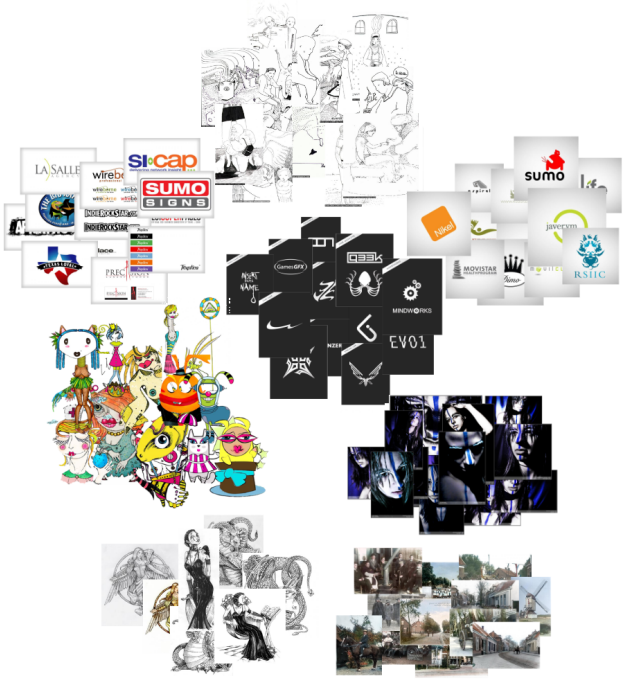

Figure 1. Example style groups, at C_1

We verify the assumption that the implicit style groupings are grounded in human-judged style similarity via responses from our image retrieval user study. We check the user responses for retrieved images in the same group as the query and calculate the frequency with which they selected these as relevant. We average accuracy of 87.73% at $C_N = 5$.

The experimental set-up allows the cleaning of weak annotations within a single project and does not cover the merging of multiple style groups into one.

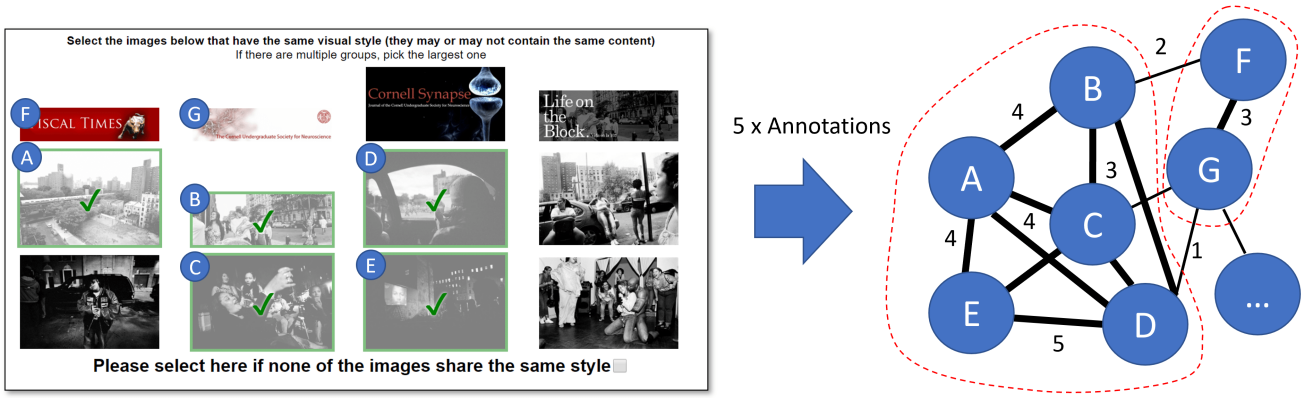

Figure 2. Crowd-annotation (cleaning) of the BAM-FG-C_N dataset. Left: Representative task showed to turkers – the largest style-coherent subset of images in the project group is annotated (green check marks). Right: Given such annotation for 5 turkers, a graph is constructed with nodes as images and edges weighted to indicate the number of times a given image pair had been co-selected by a turker. The graph is processed to identify style groupings (sub-graphs) at different levels of consensus (thresholds on the edge weights).

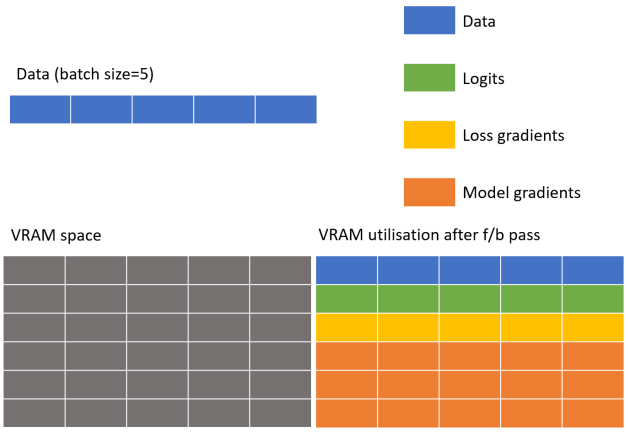

Figure 3. VRAM usage of a batch using regular contrastive learning. Given some data samples (blue), the VRAM space (grey) is taken up by it, the output logits (green), the loss gradients (yellow), and the gradients of the entire model (orange) Each block represents a visualisation of GPU memory usage.

2. Logit accumulation

As per the wider contrastive learning literature, large batch sizes are important for driving a strong signal for learning. We report our experiments for batch size of 1024, where the gain plateaus (with some example lower batch sizes of 512 and 256 dropping from 56.89 to 30.4 and 29.5 respectively). High batch sizes push GPU VRAM requirements past the capacity of regular hardware.

Figures 3 and 4 show a visualisation of the logit accumulation algorithm described in the paper for large-batch contrastive learning under constrained VRAM. Figure 3 shows the traditional forward and backward passes of data through a model, using the following process:

- 1. Use model to generate logits
- 2. Use logits to compute contrastive loss across all batch items
- 3. Backpropagate the gradients through the logits, and then the model layers

Figure 4 shows the proposed logit accumulation process, following these steps:

- Step 1. Split the data into smaller chunks
- Step 2. Use the model to generate the logits in inference mode (no model gradients). The data and the logits here take up the VRAM.
- Step 3. Compute contrastive loss using the concatenated logits. Backpropagate the loss, but keep the gradients at the logits. The logits and the loss gradients here take up the VRAM.
- Step 4. For each data split, re-compute the logits using the model, this time keeping the model gradients. The VRAM is now taken up by the data, the loss gradients and finally, the model gradients.
- Step 5. Without computing any further loss, use the existing logits' gradients and pass them down the model layers. These last two steps function as traditional VRAM usage, except the loss, which is not computed again. Instead, the gradients from the previously computed loss (in chunks) now continue to propagate through the model, where there are now gradients for the model parameters.

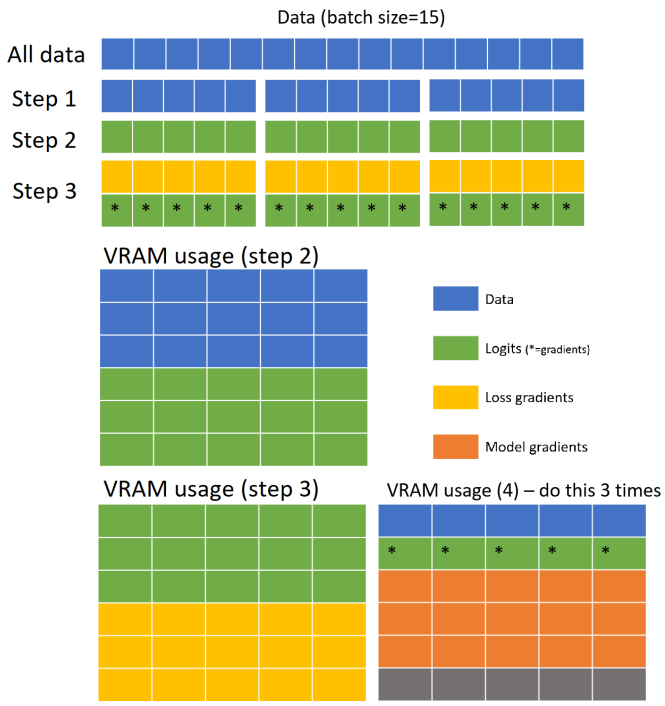

Figure 4. VRAM usage using proposed logit accumulation method for contrastive training. Visualisation for an example where a target batch size of 15 samples is broken up into 3 chunks of 5 samples. The loss is computed across all 15 sets of logits, with the rest of the operations (inference and backpropagation) being executed in smaller chunks that fit into VRAM. Each block represents a visualisation of GPU memory usage.

3. Further examples of image retrieval results

Figs 6 and 7 show further image retrieval results using ALADIN-L. The searches are performed over the holdout 79k BAM-FG C_3 image test set. The left-most images are the search query images, and the following images are retrieved results in order of closest match.

Figs 8, 9, 10, 11 and 12 show very large scale image retrieval results of ALADIN-L over a 150M image corpus of artwork on Behance.net.

4. Video: Web-scale style retrieval demo

We include a video demonstration of large scale image retrieval using ALADIN-L in the supplementary materials. The video showcases the capability of the model on a web-scale image corpus (over 150 million images).

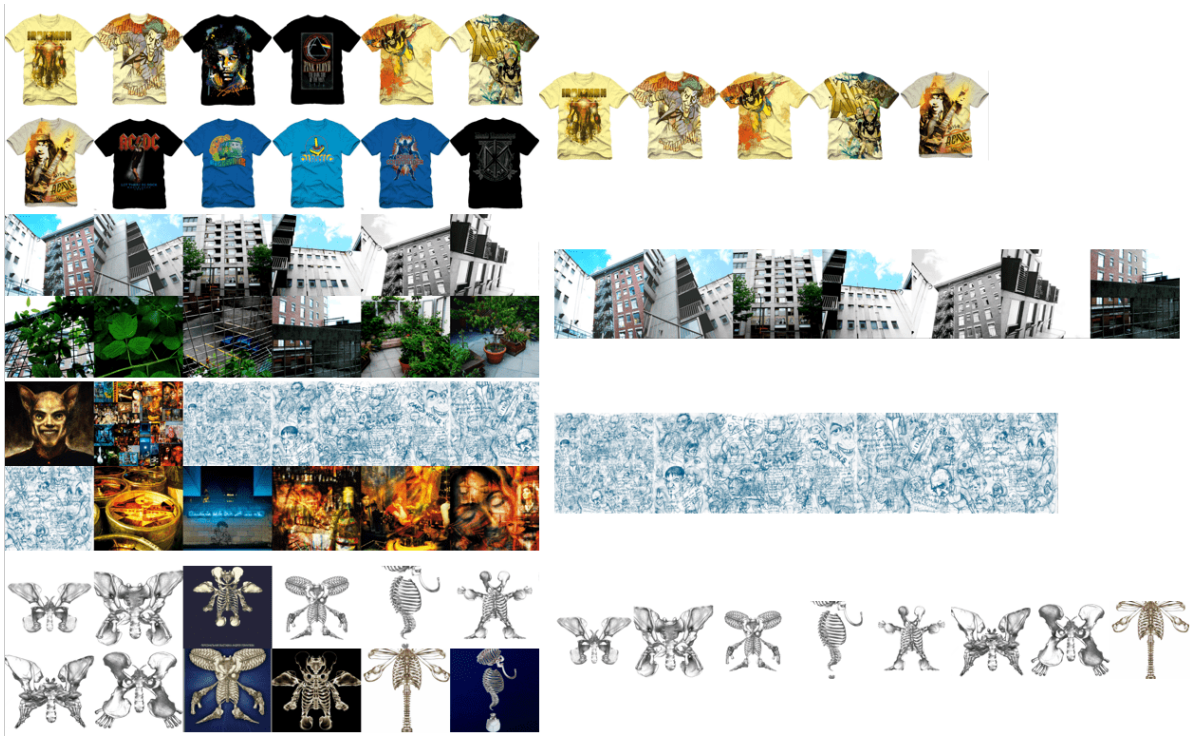

Figure 5. Example projects before and after the cleaning process.

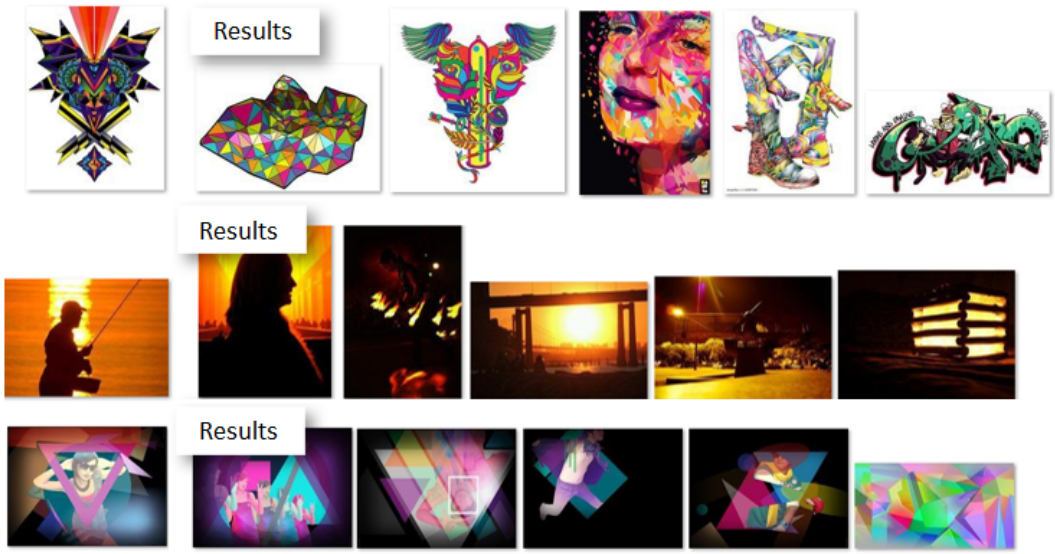

Figure 6. ALADIN image retrieval results over the test corpus

Figure 7. ALADIN image retrieval results over the test corpus

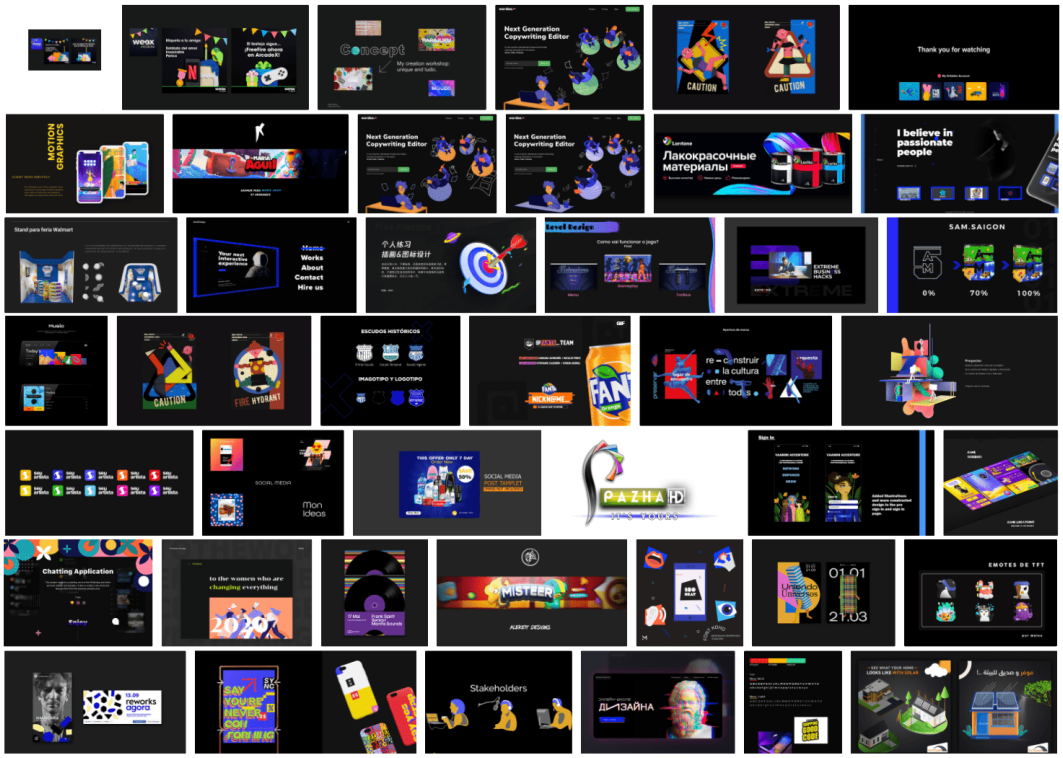

Figure 8. ALADIN image retrieval results over a 150M image corpus. The top-left-most image is the query image

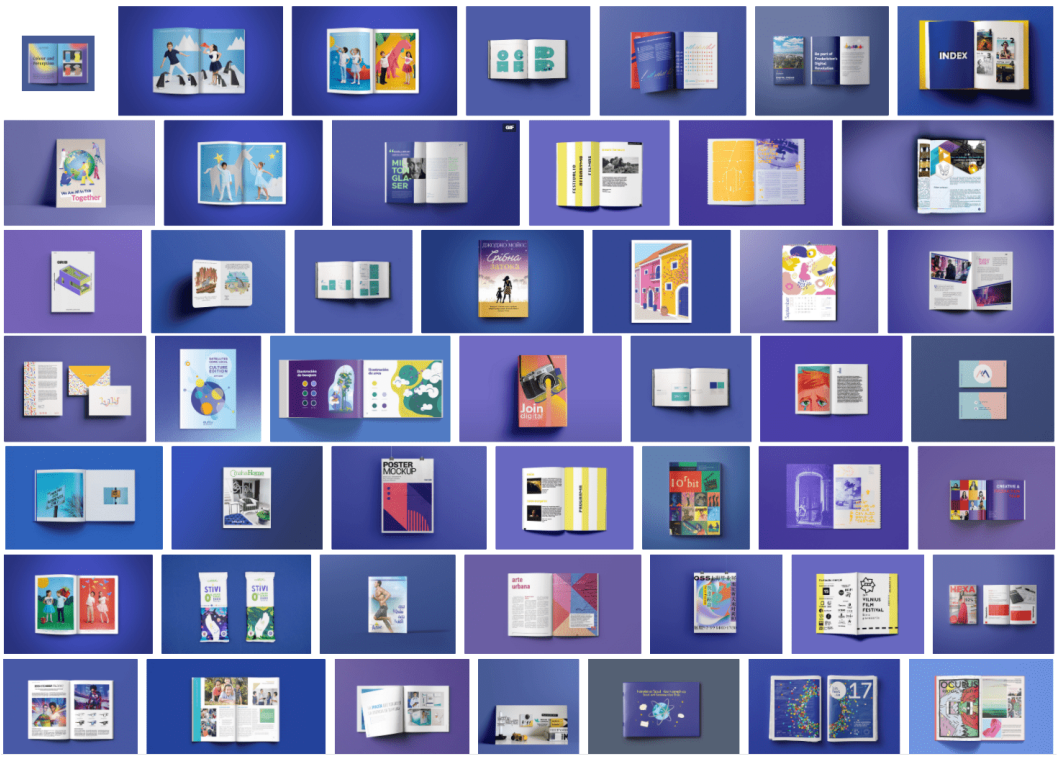

Figure 9. ALADIN image retrieval results over a 150M image corpus. The top-left-most image is the query image

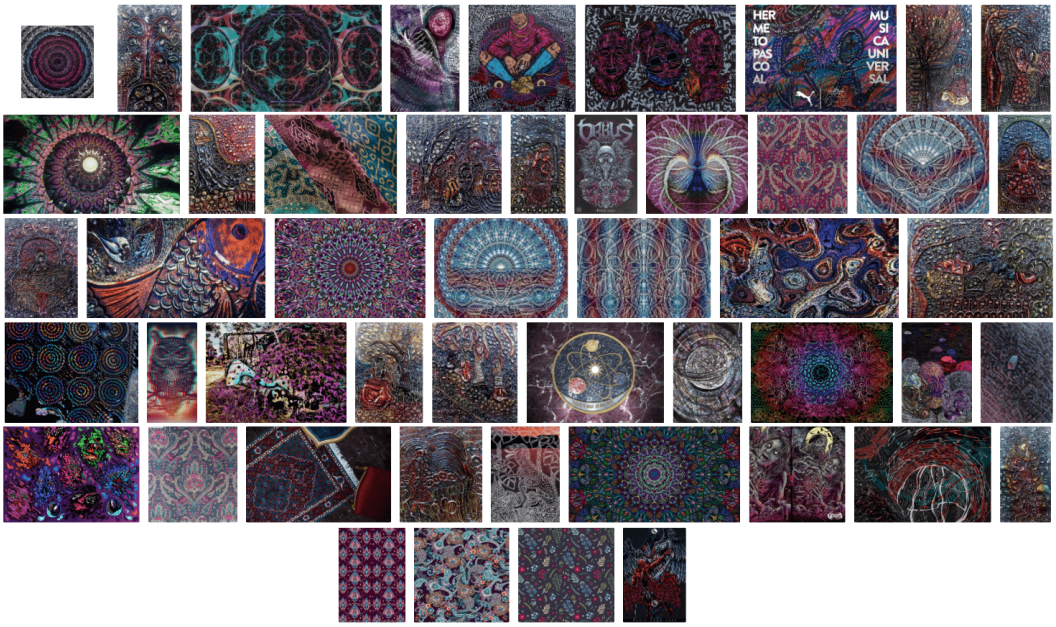

Figure 10. ALADIN image retrieval results over a 150M image corpus. The top-left-most image is the query image

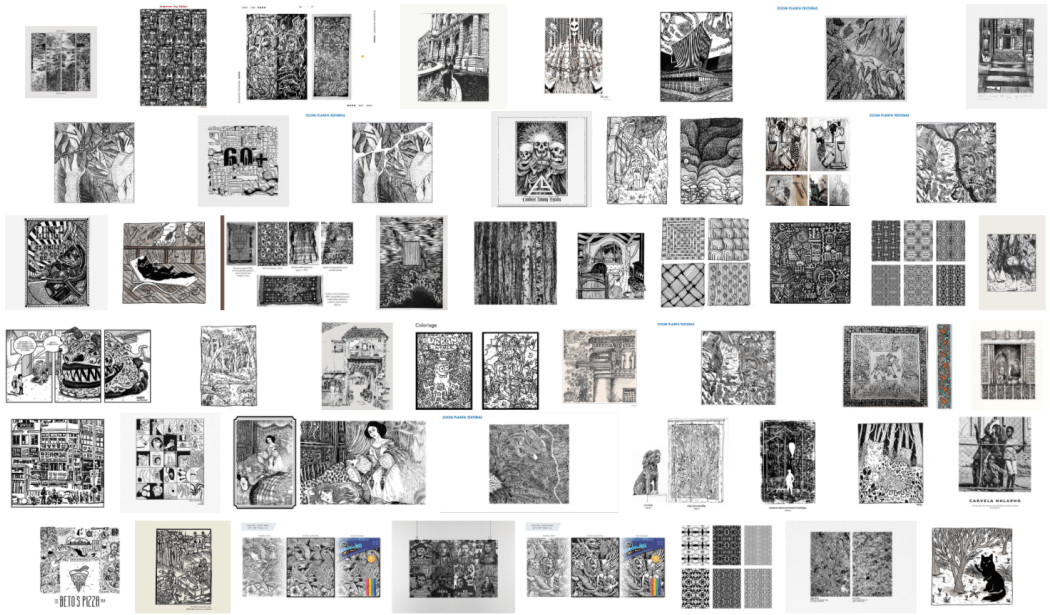

Figure 11. ALADIN image retrieval results over a 150M image corpus. The top-left-most image is the query image

Figure 12. ALADIN image retrieval results over a 150M image corpus. The top-left-most image is the query image