Texture-based Error Analysis for Image Super-Resolution

Salma Abdel Magid[1] Zudi Lin[1] Donglai Wei[2] Yulun Zhang[3] Jinjin Gu[4] Hanspeter Pfister[1]
[1]Harvard University [2]Boston College [3]ETH Zürich [4]The University of Sydney

Abstract

Evaluation practices for image super-resolution (SR) use a single-value metric, the PSNR or SSIM, to determine model performance. This provides little insight into the source of errors and model behavior. Therefore, it is beneficial to move beyond the conventional approach and reconceptualize evaluation with interpretability as our main priority. We focus on a thorough error analysis from a variety of perspectives. Our key contribution is to leverage a texture classifier, which enables us to assign patches with semantic labels, to identify the source of SR errors both globally and locally. We then use this to determine (a) the semantic alignment of SR datasets, (b) how SR models perform on each label, (c) to what extent high-resolution (HR) and SR patches semantically correspond, and more. Through these different angles, we are able to highlight potential pitfalls and blindspots. Our overall investigation highlights numerous unexpected insights. We hope this work serves as an initial step for debugging blackbox SR networks.

1. Introduction

The standard practice of training and evaluating image SR models has not changed drastically in recent years. Neural networks are optimized on the Diverse 2K (DIV2K) [40] HR images. The models are then evaluated on five benchmark datasets: Set5 [3], Set14 [45], B100 [28], Urban100 [15], and Manga109 [29] with two metrics, the peak-signal-to-noise ratio (PSNR) and the structural similarity index (SSIM) [43]. This protocol ultimately shapes the performance of the network and provides a means by which models are compared against each other.

We note several limitations to this standard approach. First, a single value is used to represent the model's performance on an entire dataset. There is immense variability within a single image, let alone an entire dataset. Using only the PSNR does little to showcase the true performance or indicate to the user where and how the model fails. Second, because we do not have a concrete description of the datasets, these performance results might be misleading. For example, models might perform well on patches that are not reflective of the dataset's overall content. Additionally, the datasets have not been probed, and it is unclear whether these are appropriate choices for the task, whether that be training or evaluation.

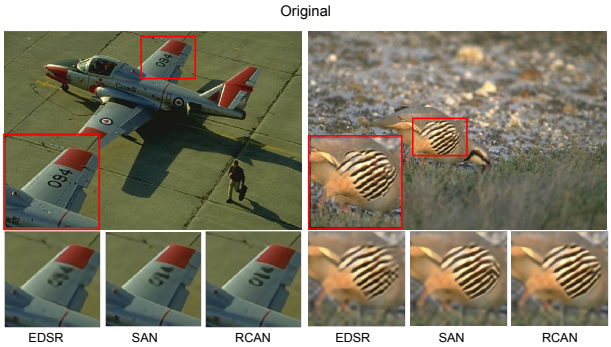

Figure 1. Visual examples of errors from state-of-the-art SR models. Some reconstructions completely change the semantic meaning of a patch. In the bottom left row, the digits "094" on the airplane wing have been obscured or changed to a different digit. Especially at inference time, it is critical to detect such mistakes.

Moreover, during inference time, we do not have access to the original HR image. It has been demonstrated that neural networks make overly confident predictions [14]. Even more troubling, SR networks frequently change the semantic meaning of images. Figure 1 shows the original HR image alongside several reconstructions by popular SR models. In the top left figure, the airplane wing contains the digits "094". However, in the reconstructions by all three SR models, the numbers have been obscured or transformed to a different number. Similarly, in the following image of the bird, the orientation, width, and number of lines have changed. These types of errors are significant as they can alter the semantic meaning of the image. The importance of catching and understanding such errors is compounded in safety-critical domains such as biomedicine. For example, Weigert *et al*. [44] introduced CARE networks to restore fluorescence microscopy data. They showed that important biological structures can appear and disappear randomly, significantly influencing downstream analysis.

To begin to address these concerns, we take a closer look at the error sources for each step of the SR framework. The

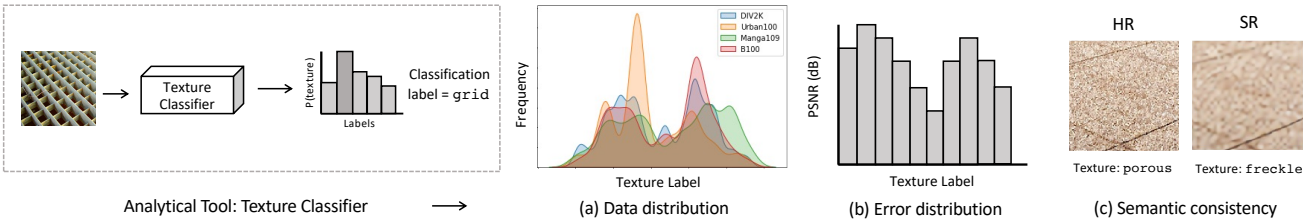

Figure 2. Overview of our texture-based error analysis framework for image SR. After training a texture classification model, we can analyze the source of SR errors from multiple perspectives, including **(a)** different texture distributions of datasets, **(b)** SR performance *vs*. texture classes, and **(c)** texture alignment between HR and SR pairs. This approach provides insights that the PSNR/SSIM alone cannot.

guiding principle to our analysis is a human-centric one. That is, we want to use language which is intuitive, easily recognizable, and sufficiently expressive. Our investigation is designed through the lens of textures and patterns. The analysis, as described in Figure 2, is conducted by partitioning the natural image manifold into semantically meaningful groups (*e.g.*, `grid`, `striped`, `polka-dotted`, etc.). Our experiments address the following questions:

- What type of semantic information is encoded in each SR dataset, and to what extent do training and evaluation sets align?
- Where do model failures occur relative to these semantic groups? What is the nature of those images?
- Do image SR models preserve the semantic meaning of patches?

This work results in the following main insights and contributions: **(1)** We provide a comprehensive semantic profile of each dataset, which is valuable to assess the diversity of the datasets and capture the degree of semantic alignment between them. We find that the datasets in the SR framework are biased towards certain semantic groups. **(2)** We verify that the semantic class label can serve as a proxy for the difficulty of a patch. This equips us with a meaningful way to think and communicate about the data. **(3)** The labels the SR models perform the best on are not the labels that reflect the content of the entire dataset. This motivates us to consider alternative methods of evaluation beyond PSNR. **(4)** For some semantic labels, there is a surprisingly small benefit from deep learning, as simple bicubic upsampling can even outperform deep models. **(5)** We propose a simple metric to curate a training set strategically. We demonstrate that training on only those patches of interest can accelerate training, even when these patches comprise only 20 percent of the original training data. **(6)** SR models surprisingly alter the semantic meaning of patches.

2. Related Work

Image Super-Resolution. The image SR problem is usually framed as a dense regression task with the goal of learning a functional mapping between the low-resolution (LR) image and its high-resolution (HR) counterpart. There are a variety of approaches to the task. Models can be residual-based [18, 23, 47, 48], generative [11, 17, 21, 42], or even recently, transformer-based [4, 22, 24]. The first method that used deep learning for this task was SRCNN [8] with a small three-layer architecture. Many works have made several attempts at widening or deepening SR networks. To overcome the vanishing gradient problem, EDSR [23] adopted the residual learning approach. Due to its success and popular code-base, the research community has focused heavily on building upon EDSR. One way is by integrating different channel and spatial attention mechanisms. For example, RCAN [47], DFSA [26], SAN [7], CSNLN [32], and NLSN [31] all incorporate an attention mechanism. SAN, CSNLN, and NLSN models rely on non-local attention to capture global semantic relationships and long-range feature correlations. We focus on these types of models for our analysis. The main motivation of our work is to study the behaviors of SR models so we can design better and more safe architectures. For this, we turn to interpretability.

Interpretability. Machine learning interpretability is a growing field aimed at understanding how models make predictions (*e.g.*, network inspection [27]). Zhou *et al*. [49] proposed *network dissection* to quantify the interpretability of nodes. They found that certain nodes are responsible for specific semantic concepts like “plane” and “lamp”. Aside from the model, one can also better understand the data. For example, influence functions determine which training points contributed to a specific prediction [19]. Other interpretability methods [25, 35–39] consider the input instead of the model or data. Their goal is to quantify which input features affect the output, known as *attribution maps*. LAM [12] adopts the idea to image SR, which identifies the importance of each pixel in the input LR image with respect to the SR image. Another way to explain model predictions is by using counterfactuals [10, 41]. This is done by changing the input feature values of an instance, then analyzing how the prediction changes. The methods discussed in this section (except for LAM) are intended for image classification and are not directly applicable to image SR. The main

challenge is that the SR modeling task is fundamentally different from image classification. We hope to bridge the divide by conducting interpretable analysis for image SR. To our knowledge, we are the first to analyze the data and models with respect to semantic labels. Moreover, since we are introducing a classifier into the SR framework, many of the interpretability tools can potentially also be used. We want to note that the concept of incorporating classifiers into the SR framework is not new. For example, perceptual loss [9, 16] uses a classifier to minimize the difference between deep features of the ground truth and the prediction. Thus, it is valuable to leverage classification models for SR, especially those trained on relevant datasets.

3. Method

Our goal is a more rigorous understanding of the potential error sources in the SR framework, enabled by a *texture* classifier (Figure 2). Prior to the error analysis, we first study the dataset and textures from a complexity perspective. Once we understand the textures, we proceed to the error analysis. In the first stage, we determine the semantic content of each dataset in terms of textures and to what extent they align. We then check how SR models perform on each texture. Then, we consider how deep learning compares to traditional methods on these textures. This is followed by qualitative and quantitative assessments of how labels can be used to determine how SR models change the semantic meaning of a patch. In this section, we first describe how the texture classifier is trained and sanity checked.

To motivate our use of textures, consider some alternatives. One might argue the usage of some features more technically precise and less fluid in interpretation, such as frequency coefficients. This would entail representing the image patches in terms of Fourier coefficients. But how human-understandable is this? What does a frequency coefficient *actually* mean? It involves discussing terms such as magnitude, phase, and sometimes complex numbers. Texture labels enable us to circumvent these problems.

3.1. Texture Classifier

Training Data. We use the Describable Textures Dataset (DTD) [5] to train the texture classifier. DTD is a curated assortment of textural images. The most important quality of this dataset is that the labels are human-centric. The dataset consists of 5,640 images belonging to 47 texture categories. Each category has 120 images. The image sizes vary from 300×300 to 640×640.

Architecture. In general, the convolution layers of a classifier extract features locally by using a sliding window. This results in feature maps which preserve the relative spatial arrangement of input images. These ordered feature maps are then fed into a fully connected layer for classification. Although this works well for general image classification, it is not ideal for texture recognition since the representation needs to be spatially invariant. Zhang *et al*. [46] propose an end-to-end framework that includes an encoding for an orderless representation. The architecture of the texture classifier is a deep *texture encoding network* (DeepTEN) [46]. The backbone network is ResNet50 [13]. DeepTEN suits our use case particularly well due to the proposed residual encoding layer built on top of the backbone network.

Implementation Details. We train the DeepTEN model on both the training and validation sets of DTD [5]. Achieving reasonable accuracy for the texture classifier is non-trivial. Our hyper-parameter tuning experiments resulted in the following settings. We use inputs of size 224×224 with a batch size of 64. We train the model for 600 epochs with an initial learning rate of 0.01 (decrease by $10\times$ every 150 epochs), momentum of 0.9 and weight decay of 10^{-6}. The data is augmented using RandAugment [6] with $N = 1, M = 1$ and with additional random crops and horizontal and vertical flips. Inputs are normalized using the ImageNet statistics. The standard multi-class cross entropy (CE) loss is used to train the network. We achieved a training accuracy of 97% and testing accuracy of 72%.

3.2. Texture Embedding

We check the learned representations of the classifier through manifold visualization. Simply clustering the original images produces poor results and does not provide human understandable labels. Instead, we extract intermediate representations from the trained classifier. In this way, we are able to study the manifold from a bird's eye view with respect to interpretable and pertinent labels.

For automatic labelling, we randomly extract 50 patches of size 128×128 from each image in a given SR dataset. These steps are carried out as described to mimic the SR training procedure. Since there are 800 images in the DIV2K training set, the resulting number of points is 40,000. The texture classifier is applied to each patch to retrieve (1) the intermediate embedding and (2) the texture classification label. The intermediate embedding is extracted from the penultimate layer of the classifier with a dimension of 4,096. Thus, the data used for the clustering is of size 40,000×4,096. We use the standard K-Means clustering algorithm and set the number of clusters to 15 for a reasonable level of granularity. The Uniform Manifold Approximation and Projection (UMAP) [30] algorithm is used to reduce the dimension from 4,096 to 2 for visualization.

4. Experiments

4.1. Data and Texture Understanding

Method 1: PSNR *vs*. Entropy. The goal is to understand the dataset content through complexity analysis and to bet-

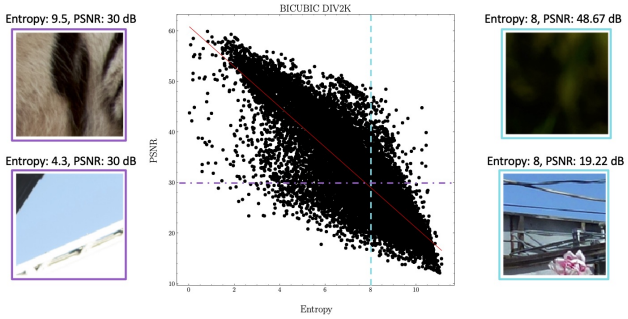

Figure 3. The entropy of the gradients *vs*. the bicubic PSNR of DIV2K patches. The best fit line is shown in red, indicating a strong inverse relationship. As entropy increases, the PSNR generally decreases. Sample patches along the purple and blue dashed lines demonstrate an interesting behavior. When holding one variable constant, the other variable can have a large spread in values.

ter understand the defining characteristics. Several metrics have been proposed with the aim of capturing complexity. In a related image enhancement field, [33] propose the PatchSNR, or the "Signal-to-Noise" ratio within a patch. The PatchSNR is defined as the square root of the variance of the clean patch divided by the variance of the noise. They demonstrated that certain patches have a preference to internal or external denoising using this metric. Alternatively, a recent work [20], proposes ClassSR in which they frame this as an image classification task. They train a shallow network to classify the complexity of a patch. The ground truth is determined by the PSNR of MSRResNet [42] and split into three classes: `easy`, `medium`, and `hard`. When proposing DIV2K, Agustsson *et al*. [40] examined the entropy of images *vs*. the bicubic PSNR. The entropy is an indicator of the amount of information present in the image per image pixel [1]. We show below that contrary to Agustsson *et al*. [40], entropy can not always be used to predict the PSNR, *i.e*., entropy serves as a proxy for the complexity. Following the logic of [40], Figure 3 shows the entropy of the gradients *vs*. the bicubic PSNR ($\times 4$) of DIV2K patches. The best fit line is shown in red, indicating a strong inverse relationship. From the plot, we can see that as entropy increases, the PSNR generally decreases. However, for a single entropy value, we can have a large spread of different PSNR values. Consider the blue dashed line at entropy $= 8$. We show two patches along this line that have the same entropy, yet the PSNR of one is more than twice that of the other. Likewise, for a single PSNR value of 30 dB (indicated by the dashed purple line), we have entirely different entropy values. This means that neither the PSNR alone nor the entropy of the gradients alone are effective measures of the perceptual complexity and difficulty.

Method 2: PSNR *vs*. Entropy by Texture Labels. To gain a better understanding of what the textures are capturing

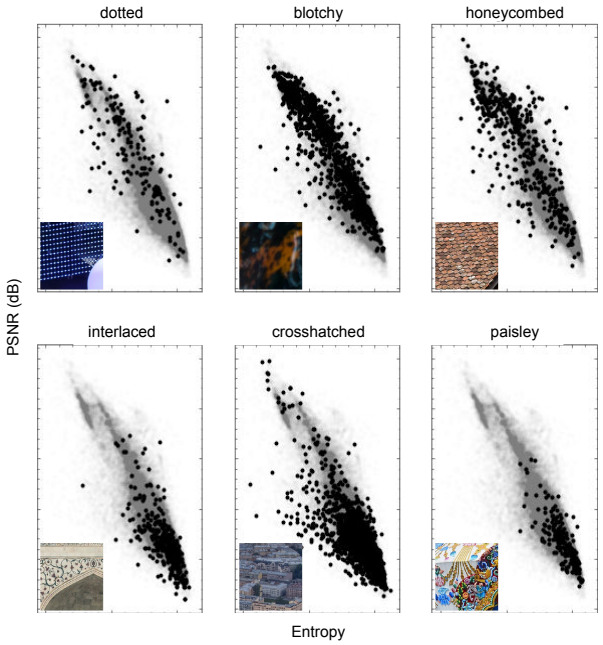

Figure 4. Top 3 and bottom 3 performing labels with respect to bicubic PSNR performance. The bottom 3 labels generally have high entropy and low PSNR. Black points belong to the label of interest, while points in gray correspond to the remaining points.

relative to complexity, we conduct the same analysis again, but this time, relative to the texture labels. Figure 4 shows the top 3 and bottom 3 labels for the same data of Figure 3. With this view, we are able to assess the textures and understand what complexity they correspond to. It is interesting to note that for the bottom 3 labels, they are generally condensed in the right quadrant, where patches have a low PSNR and a high entropy. On the other hand, the top 3 labels are distributed across the spread of entropy and PSNR values, indicating a varied complexity. We also note there is some redundancy among these top 3 labels. Since they span generally the entire plot, they contain redundant patterns. Although this method helps us understand the textures in a non-biased way, we still need a better picture of how the textures are related and how the aforementioned redundancy manifests itself. For this, we turn to grouping the patch texture embeddings via clustering.

Method 3: Texture Feature Manifold. Finally, to understand the relationship between textures, we visualize the manifold they span after projecting the patches to the embedding space of the texture classifier (outputs of the penultimate layer). Figure 5 shows the manifold spanned by DIV2K's clusters. In the bottom plot, points are colored based on their cluster label. There is a rich and varied complexity across and within clusters. Due to this inter- and intra-cluster variability, we show the same points, but instead colored based on their classification labels in the top

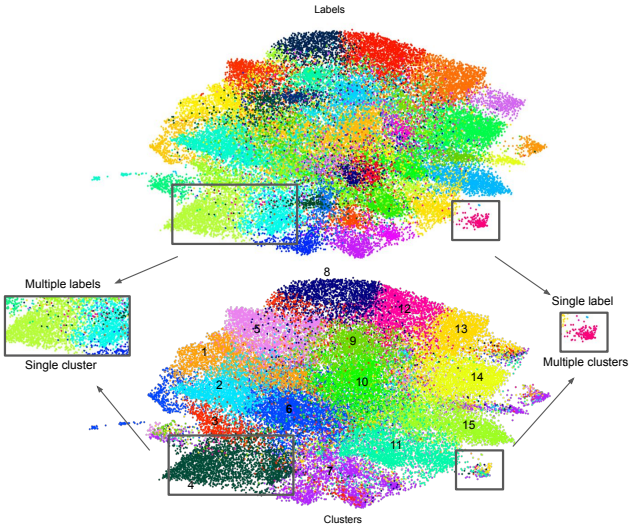

Figure 5. UMAP of DIV2K [40] training patches projected to the *texture classifier* embedding space (outputs of the penultimate layer). In the top visualization, points are colored based on their classification label predicted by the texture classifier with 47 labels. In the bottom visualization, points are colored based on embedding-space clustering, with 15 clusters. The boxes demonstrate how a single cluster (bottom left) can span several classes and how a single label (top right) can contain multiple clusters.

Figure 6. Sample DIV2K HR patches from each embedding cluster along with the cluster's most frequent texture classification label. Best viewed in color and zoomed in.

plot. Sample patches in each cluster along with the cluster's top label are shown in Figure 6 for reference.

We find interesting semantic correlations which make the description even more granular and expressive. Although the labels in Figure 6 are the most frequent among patches in that cluster, some can be highly correlated with multiple semantically related labels (i.e. `grid` and `lined` or `cobwebbed` and `veined` 3rd column, 4th row in the figure). This behavior is further illustrated in the manifold visualization, in which we see several classes overlapping. In the boxed regions to the left of Figure 5, we demonstrate how cluster #4 (dark green) contains several texture classes. Interestingly enough, we also note that the opposite behavior can occur. In the top right bounding box, points correspond to a single label (dark pink). However, those points belong to multiple clusters. This results in two main takeaways. First, there must be underlying common features which cause points to belong to multiple clusters and labels simultaneously. It is useful for us then to learn what these features are and how they express themselves. Second, and at the core of our message: multiple perspectives should be considered when evaluating SR models.

Discussion. Our analysis so far has resulted in a very important takeaway. There is significant redundancy among labels and clusters in terms of complexity. Given this redundancy, we conjecture as to whether an SR model needs the entire DIV2K training set. We noted that the most difficult labels and patches occurred in the bottom right quadrant (high entropy, low psnr). Those patches roughly correspond to points where the ratio of entropy to PSNR is 0.4.

To determine whether this metric is useful for capturing the redundant features, we train a model (EDSR) from scratch on all patches (indicated by "all" in Figure 7) and on patches whose ratio of entropy to PSNR is greater than or equal to 0.4 (indicated by "high" in Figure 7). For Figure 7 (left), we see significant differences in performance when training on high complexity patches using our metric (blue) vs. when training on all patches (green). This means that there is useful and generalizable information encoded in these patches, and it is sufficient to train on those alone. These patches comprise *only* 20 percent of the training data.

By focusing resources on these fewer patches, we can achieve better and faster results. Therefore, throwing more data at the problem is not necessarily always the answer. Patches cannot just be sampled at random, they need to be carefully chosen. This may be related to the tendency of neural networks to learn generalizable features first [2, 34]. These generalizable features often consist of simple and low-frequency/entropy patterns. In other words, SR models take advantage of patterns shared by multiple training examples. When 80% of the data consists of lower complexity patches, it is expected that learning stagnates for more difficult patches.

In Figure 7 (right), we show the performance when evaluating on all patches vs. high patches for Urban100. Regardless of what we are evaluating on, training on high patches is better. We also note the large difference in performance between evaluating on all patches (top) and high patches (bottom). These insights can inform our decisions when studying these models and the data used to train them.

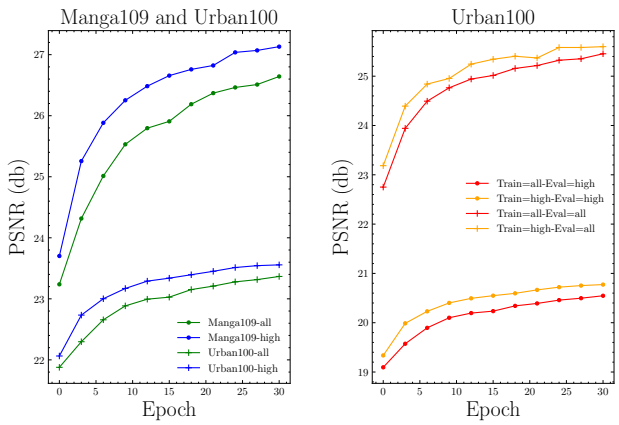

Figure 7. We train EDSR [23] from scratch on **all** patches and on patches whose ratio of entropy to PSNR is greater than or equal to 0.4 (indicated by **high**). The results show significant differences in performance when training on high complexity patches using our metric *vs*. when training on all patches on two benchmarks.

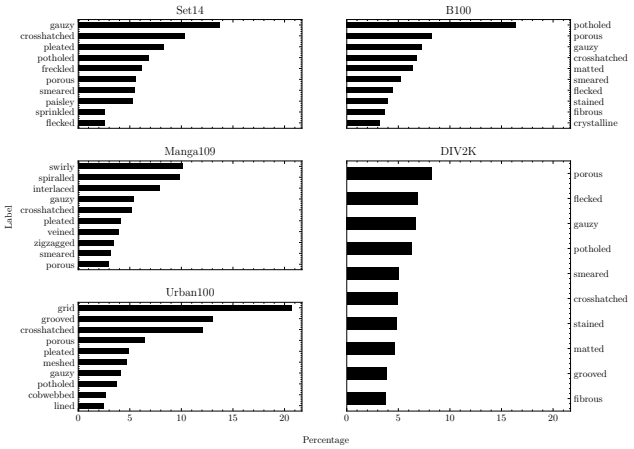

Figure 8. Percentage of top-10 texture labels for each dataset.

4.2. SR Error Analysis

4.2.1 Data Distribution by Texture Labels

Figure 8 quantifies the percentage of each label for each dataset. This distribution can be viewed as a semantic profile that gives a meaningful overview of the types of images in a dataset. We can use this to not only capture this information, but also as a unique fingerprint, which can be used to determine if there is a potential domain shift from the training data and evaluation data. For DIV2K, we see that about 30% of the training data consists of `porous`, `flecked` patches, and `gauzy`. Sample patches with these labels are visualized in Figure 6. These may correspond to backgrounds, or out of focus image patches, like those found in "water" or "sky" superpixels.

For Urban100, a large share of the patches are assigned the class labels of `grid`, `crosshatched`, and `grooved`. The Urban100 dataset contains images of urban environments, with buildings and windows, which generally exhibit very grid-like structures. The images all contain repetitive patterns and high self-similarity. The label distribution succinctly summarizes these semantically related dataset characteristics. Similarly, B100 mostly consists of `potholed`, `porous`, and `gauzy` patches. B100 covers a large variety of real-life scenes.

The distribution of Manga109 labels is of particular interest. It is dominated by two closely related labels: `swirly` and `spiralled` followed by `interlaced`. None of these top 3 labels appear in DIV2K's top 10 labels. Manga109's semantic profile does not exhibit similar class label distributions as the other evaluation datasets. This behavior can potentially be attributed to the fact that Manga109 consists of images drawn by artists which follow an animated style. They are not natural images; their internal image statistics and priors are different. Although this is the case, SR performance on Manga109 is usually higher since the image complexity is low. Evaluation on this dataset can be a misleading reflection of the model's performance. We note how our semantic profile was able to detect this deviation. This can be a valuable method to capture the degree of semantic alignment with the training data, and can flag serious problems like potential domain shift. Additionally, this method can be used to assess the diversity of the dataset.

4.2.2 Error Distribution by Texture Labels

Potential Pitfall: The semantic groups that models perform the best on are not representative of the dataset. Now that we have an overview of the types of patches within each dataset, we want to evaluate how deep learning SR models perform on each of these semantic groups. Figure 9 shows the average PSNR for each benchmark dataset for the top and bottom 5 labels, separated by a red line. The white text over each bar indicates the absolute difference between the SR model and traditional bicubic upsampling.

There are several insights to be extracted from this figure. First, some class labels are consistently difficult, or consistently easy across datasets and SR models. Consider, for example, the `fibrous` class label. It appears in the bottom 5 for each dataset and each SR model. Likewise, `bubbly` and `polka-dotted` commonly appear in the top 5 performing labels. Because of this consistent behavior, we can verify that the class label can serve as a proxy for the difficulty of a patch. This can be useful since it gives us a more meaningful way to think and communicate about the data. For example, if we are to curate a new challenging dataset, we can collect images using the worst performing class labels. Second, we noted in the previous section what

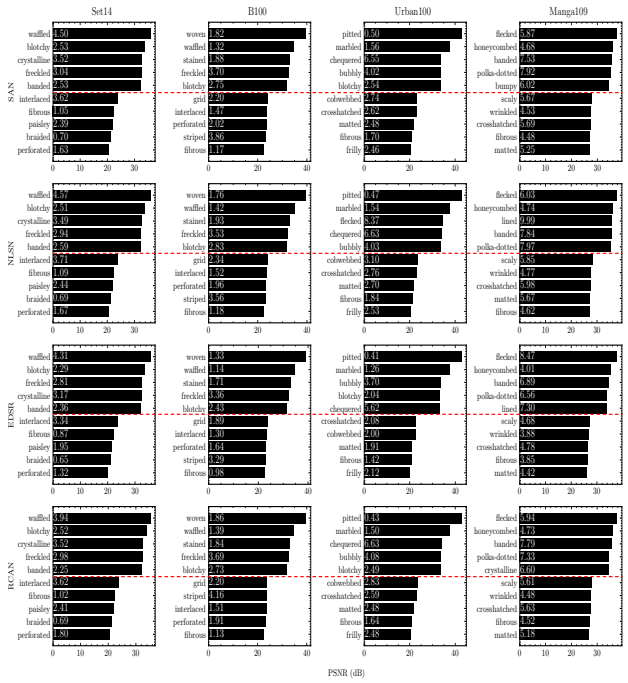

Figure 9. Average performance of each model on the top and bottom 5 labels separated by a red line. The absolute delta between deep learning and bicubic is indicated in white over each bar.

the most frequent labels are for each dataset (in Figure 8). This distribution gives us a description of the dataset and an indication of which labels reflect its contents the most. It is concerning then that Figure 9 does not show these patches in the top 5 performing labels. The labels the model performs the best on are *not* the labels that reflect the content of the dataset. For example, `crosshatched` is the 3rd most frequent label of Urban100, but is always in the worst performing labels across all SR models. This is even more motivation to couple these two overviews together when considering model performance. They give a more granular and holistic reflection of the performance. We might attribute the consistent appearance of certain labels in the bottom 5 for some datasets to the fact they are not common in the training data. If the model did not encounter enough of this type of patch, performance will be low.

Potential Pitfall: The difference between bicubic and deep learning can be surprisingly small. We must consider how bicubic upsampling performs relative to each model. We use deep learning (DL) to achieve gains over traditional, non DL based methods. State-of-the-art methods, such as EDSR [23] and RCAN [47] use roughly 43 M and 15 M parameters, respectively. Additionally, these models encounter thousands of image patches during training. Considering the amount of data and parameters DL models can utilize, it is thus expected that the margin between DL and traditional methods will be vast.

Model	Set14	B100	Urban100	Manga109
Bicubic	20.2	26.1	31.6	31.9
EDSR [23]	33.4	30.9	47.1	52.4
RCAN [47]	37.4	32.9	50.1	53.2
NLSN [31]	37.9	34.4	50.1	54.2
SAN [7]	35.1	32.6	48.6	52.6

Table 1. Percentage (%) of reconstructed SR patches with the same texture labels as their HR ground truth counterparts.

We find that the margin is surprisingly small. Figure 9 illustrates the average PSNR for each of the top and bottom 5 labels. The white text over each bar indicates the difference between bicubic and DL. Consider the bottom 5 labels in each graph. These labels should indicate the most challenging cases. For these difficult patches, both bicubic and DL perform poorly, as expected. What is less expected is that there is little gain from using DL. For example, for the hardest patches of Set14, using EDSR only provided a 0.65 dB increase in the PSNR. Similarly, for the easiest patch of Urban100, using EDSR only provided a 0.41 dB increase.

This is counter-intuitive since we use DL models to achieve superior performance to traditional methods. It may be beneficial then to consider a mixture of experts. By doing so, we can save the computational resources on specific patches where the margin between bicubic interpolation and DL models is small, irrespective of whether it was a difficult or easy patch. It can also be useful to propose a direct comparison against bicubic in this fashion to determine the exact extent of the performance gain. Admittedly, the significance of the performance gain is application dependent (and label dependent). Having 0.5 dB margin on a "sky" patch is not a cause for concern in natural images, but what could a 0.5 dB difference indicate in a biomedical setting? DL might not always be the most efficient option. Our technique enables us to ask these types of questions and genuinely critique the true performance gain of using DL.

4.2.3 Semantic Consistency by Texture Labels

We have discussed in great detail the types of errors image SR models make by using texture classification labels and proposed a simple metric that can help us quantify the patch complexity. There is an opportunity to dig even further given access to these rich semantic groups.

We saw qualitatively in Figure 1 that some types of SR errors are rather serious, as they visually change the meaning of an image. Our assumption–or rather our hope–is that the SR image should maintain to some extent the overall semantics of its HR ground-truth. One way to evaluate this is by considering the classification accuracy before and after reconstructing the LR. In Table 1 we show the percentage

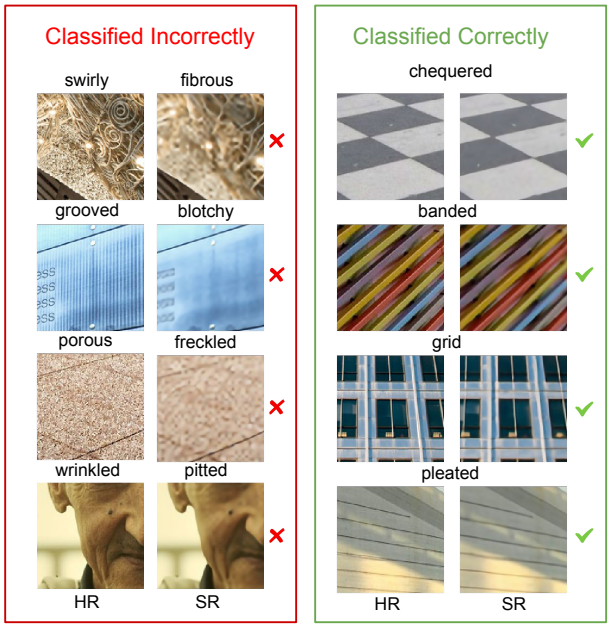

Figure 10. The predicted class label of sample patches. Patches with the correct classification maintain their semantic meaning. On the other hand, patches that are misclassified after applying the SR model can lose valuable semantic information.

of SR patches that have the same texture classification label as their associated HR counterparts. The results across all datasets and models in Table 1 are surprising. The lowest accuracy (among image SR models) of 30.9% occurs when using EDSR and evaluating on patches from B100. This means that EDSR changes the semantic label of more than half the patches in this dataset. The top accuracy across all models and datasets is around 54%.

Another surprising finding is that the bicubic classification accuracy is extremely low. This is yet another reason which motivates this type of analysis from a different perspective. When we only consider PSNR values, we can be misled as to what the true nature of the model performance is. The findings from this table open up opportunities for thinking about what it is that we want to evaluate, and more importantly, prioritize during training.

In addition to quantifying this accuracy, we also show qualitative examples of patches along with their predicted label (Figure 10). For patches whose class label changes after applying the SR model, we can clearly see that the SR model has changed the semantic meaning of the patch. For example, in the first incorrectly classified patch, the swirls on the right are completely missing from the SR. Instead, there are long, thread-like lines. Hence, the new label, `fibrous`. Similarly, in the next example, the HR patch is classified as `grooved` since there are repetitive vertical lines. In the SR patch, those lines are smudged, resulting in the new label, `blotchy`. An intriguing example occurs in the last patch. The HR patch is classified as `wrinkled`, and although some wrinkles remain in the SR patch, the new label is `pitted`. The wrinkles on the upper cheekbone are blurred and cause the misclassification. For correctly classified patches, we see very little qualitative differences between the HR and SR. To catch these types of mistakes, we would need to tediously inspect all the fine details of SR patches and compare them against the ground truth. The texture classifier can be used to streamline this process by limiting the search space to patches whose SR label does not match the HR label. It also help us gauge the seriousness of misclassifications. Perhaps `banded` to `lined` is not as dire as `striped` to `smeared`.

4.3. Limitations

There are a few caveats to our analysis. As usual, the results depend on the choice of hyper-parameters to some extent. For instance, there is a trade-off between patch size and accuracy. A larger patch size would incorporate more patterns. The patch size can also affect the analysis concerning the difference between bicubic and deep learning. Another problem is the accuracy of the texture classifier. We achieved an accuracy of 72% on the test set of DTD after careful hyper-parameter tuning. The generalization capability needs to be improved, perhaps with stronger data augmentations and regularizations. One approach would be to follow the same data collection scheme as DTD (using keywords/labels to scrape images from search engines) and use more images per label and more labels. Possible explanations for why the SR changes the classification label despite the images seeming perceptually identical is of interest and this behavior can certainly skew the results.

5. Conclusion

In this work, we take a first step towards reconsidering the SR evaluation. The analysis is indeed conducted relative to the classes in the texture dataset. This means that the general approach is task and application agnostic. The approach can be applied to any image enhancement method (de-blurring, restoration, etc.) and any application (biomedical, natural, etc.). The "texture" dataset can be defined and specified based on the application. For example, one application can be SR models that operate on images of faces. Our approach may detect biased models by identifying which classes the model fails on (*e.g.*, sensitive attributes such as gender, race, and age); information the PSNR and SSIM alone could not provide.

Acknowledgements

This work has been partially supported by NSF grant NCS-FO-2124179 and NIH grant 5U54CA225088-03.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *CVPR Workshops*, pages 126–135, 2017. 4

[2] Devansh Arpit, Stanisaw Jastrzebski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *ICML*, 2017. 5

[3] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *BMVC*, 2012. 1

[4] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. Pre-trained image processing transformer. In *CVPR*, 2021. 2

[5] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *CVPR*, pages 3606–3613, 2014. 3

[6] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020. 3

[7] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. Second-order attention network for single image super-resolution. In *CVPR*, 2019. 2, 7

[8] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *TPAMI*, 2016. 2

[9] Muhammad Waleed Gondal, Bernhard Schölkopf, and Michael Hirsch. The unreasonable effectiveness of texture transfer for single image super-resolution. In *European Conference on Computer Vision*, pages 80–97. Springer, 2018. 3

[10] Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual visual explanations. In *ICML*, 2019. 2

[11] Jinjin Gu, Haoming Cai, Haoyu Chen, Xiaoxing Ye, Jimmy Ren, and Chao Dong. Image quality assessment for perceptual image restoration: A new dataset, benchmark and metric. *arXiv preprint arXiv:2011.15002*, 2020. 2

[12] Jinjin Gu and Chao Dong. Interpreting super-resolution networks with local attribution maps. In *CVPR*, 2021. 2

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3

[14] Matthias Hein, Maksym Andriushchenko, and Julian Bitterwolf. Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 41–50, 2019. 1

[15] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *CVPR*, 2015. 1

[16] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 3

[17] Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018. 2

[18] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *CVPR*, 2016. 2

[19] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017. 2

[20] Xiangtao Kong, Hengyuan Zhao, Yu Qiao, and Chao Dong. Classsr: A general framework to accelerate super-resolution networks by data characteristic. In *CVPR*, 2021. 4

[21] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017. 2

[22] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. Swinir: Image restoration using swin transformer. In *ICCV*, 2021. 2

[23] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *CVPRW*, 2017. 2, 6, 7

[24] Zhisheng Lu, Hong Liu, Juncheng Li, and Linlin Zhang. Efficient transformer for single image super-resolution. *arXiv preprint arXiv:2108.11084*, 2021. 2

[25] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NeurIPS*, 2017. 2

[26] Salma Abdel Magid, Yulun Zhang, Donglai Wei, Won-Dong Jang, Zudi Lin, Yun Fu, and Hanspeter Pfister. Dynamic high-pass filtering and multi-spectral attention for image super-resolution. In *ICCV*, 2021. 2

[27] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. *IJCV*, 2016. 2

[28] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *ICCV*, 2001. 1

[29] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 2017. 1

[30] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. 3

[31] Yiqun Mei, Yuchen Fan, and Yuqian Zhou. Image super-resolution with non-local sparse attention. In *CVPR*, 2021. 2, 7

[32] Yiqun Mei, Yuchen Fan, Yuqian Zhou, Lichao Huang, Thomas S Huang, and Humphrey Shi. Image super-resolution with cross-scale non-local attention and exhaustive self-exemplars mining. In *CVPR*, 2020. 2

[33] Inbar Mosseri, Maria Zontak, and Michal Irani. Combining the power of internal and external denoising. In *ICCP*, 2013. 4

[34] Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *ICML*, 2019. 5

[35] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, pages 9243–9252, 2020. 2

[36] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*, 2017. 2

[37] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 2

[38] J Springenberg, Alexey Dosovitskiy, Thomas Brox, and M Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015. 2

[39] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *ICML*, 2017. 2

[40] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, Lei Zhang, Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, Kyoung Mu Lee, et al. Ntire 2017 challenge on single image super-resolution: Methods and results. In *CVPRW*, 2017. 1, 4, 5

[41] Sandra Wachter, Brent Mittelstadt, and Chris Russell. Counterfactual explanations without opening the black box: Automated decisions and the gdpr. *Harv. JL & Tech.*, 31:841, 2017. 2

[42] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *ECCVW*, 2018. 2, 4

[43] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *TIP*, 2004. 1

[44] Martin Weigert, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, et al. Content-aware image restoration: pushing the limits of fluorescence microscopy. *Nature methods*, 2018. 1

[45] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *Proc. 7th Int. Conf. Curves Surf.*, 2010. 1

[46] Hang Zhang, Jia Xue, and Kristin Dana. Deep ten: Texture encoding network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 708–717, 2017. 3

[47] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun Fu. Image super-resolution using very deep residual channel attention networks. In *ECCV*, 2018. 2, 7

[48] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *CVPR*, 2018. 2

[49] Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. Interpreting deep visual representations via network dissection. *TPAMI*, 2018. 2