Image Super-Resolution via Iterative Refinement

Research Questions, Results & Discussion

Authors:
Maxim Chanturiay, Ofir Nahshon, Rashan Shkeer

Table of Contents

Introduction: Overview

This is the 2nd presentation, which presents research questions, methods, results and discussion for our project throughout the course “Deep Learning”, Afeka College of Engineering.

The 1st presentation is available at part_01_concepts_and_existing_solutions.pdf.

Our code, assets and other implementation related artefacts are hosted at the Github project https://github.com/maxim-chn/deep-learning

Introduction: Question at hand

We have decided to tackle the following research question:

Is Automated Process to Generate Train Dataset Applicable

For Models that Perform Image Super-Resolution?

Introduction: Question at hand (*continued*)

Motivation:

- Little room for improvement in terms of model capabilities on test data (explained in detail at our 1st presentation)
- Reduce time spent on manual image preparation
- Reduce cost on equipment like camera and software editing
- Reduce storage costs
- Improve diversity in workable data beyond ImageNet, Flicker-Faces HQ and similar counterparts

Introduction: Methodology

Each model is evaluated using the same test dataset, resulting in the following metrics:

Qualitative Assessments: Human evaluators review and compare images to determine their realism and aesthetic appeal.

Quantitative Metrics:

-Mean Squared Error (MSE)

-Peak Signal-to-Noise Ratio (PSNR)

-Structural Similarity Index (SSIM)

Mean Squared Error (MSE):

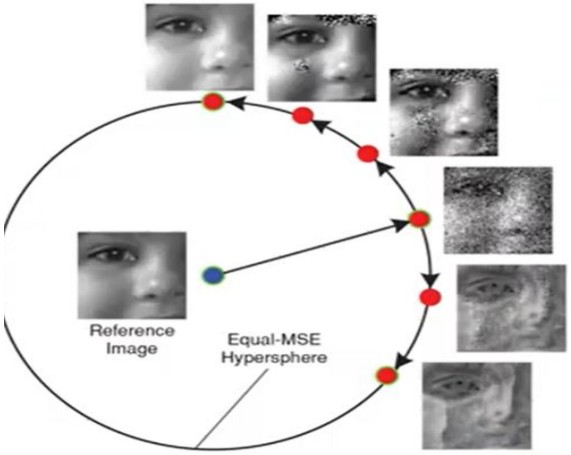
Purpose: Quantifies the difference between generated images and target images.

How It Works: Computes the average squared difference between corresponding pixels.

Pros: Simple and interpretable.

Cons: Sensitive to outliers and does not always

$$MSE = \left(\sum_{m,n} [I_1(i,j) - I_2(i,j)]^2 / (m \times n) \right)$$



I_1	I_2	$I_1(i,j) - I_2(i,j)$	$(I_1(i,j) - I_2(i,j))^2$
5	8	3	9
2	9	7	49
7	4	-3	9
1	2	1	1
4	1	-3	9
8	5	-3	9
9	7	-2	4
0	4	4	16
3	3	0	0

Summation 106

MSE 11.8

Peak Signal-to-Noise Ratio (PSNR):

Purpose: Measures the quality of generated images by comparing them to reference images.

How It Works: Measures the ratio between the maximum possible power of a signal and the power of corrupting noise.

Pros: Simple to compute and widely used in image processing.

Cons: Does not always align with perceptual quality and can be less effective for diverse datasets.

$$PSNR = 10 \log_{10}(R^2 / MSE)$$

- $R = 255$
- Expressed as db

Structural Similarity Index (SSIM):

- **Purpose**: Measures the similarity between generated and real images based on structural information.
- **How It Works**: Evaluates luminance, contrast, and structure similarities between images.
- **Pros**: Captures perceptual quality and structural information.
- **Cons**: Primarily used for image quality but may not reflect diversity. **-1<=SSIM<=1**

Three Key Components:

- **Luminance**: Measures the brightness of the images.
- **Contrast**: Compares the contrast of the images.
- **Structure**: Compares the patterns of pixel intensities,
- capturing texture and structural information.

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

In python Library:scikit-image

```
# Compute SSIM

ssim_index, ssim_map = ssim(image1, image2, full=True)
```

Introduction: Methodology (*continued*)

Our implementation involves several distinct training datasets:

1. The "Original" dataset, located at Kaggle.
2. The "Divide-and-Conquer" dataset, created through an automated process from the "Core" images.
3. The "Noise-Framed" dataset, developed by further improving the "Divide-and-Conquer" dataset.

Each dataset is used to train a distinct model, with no differences in architecture or underlying implementation technology. In total, three distinct models were trained, differing only in the training dataset used.

"Core" images: Randomly sampled from the MSCOCO, DIV2K, and BSD500 datasets.

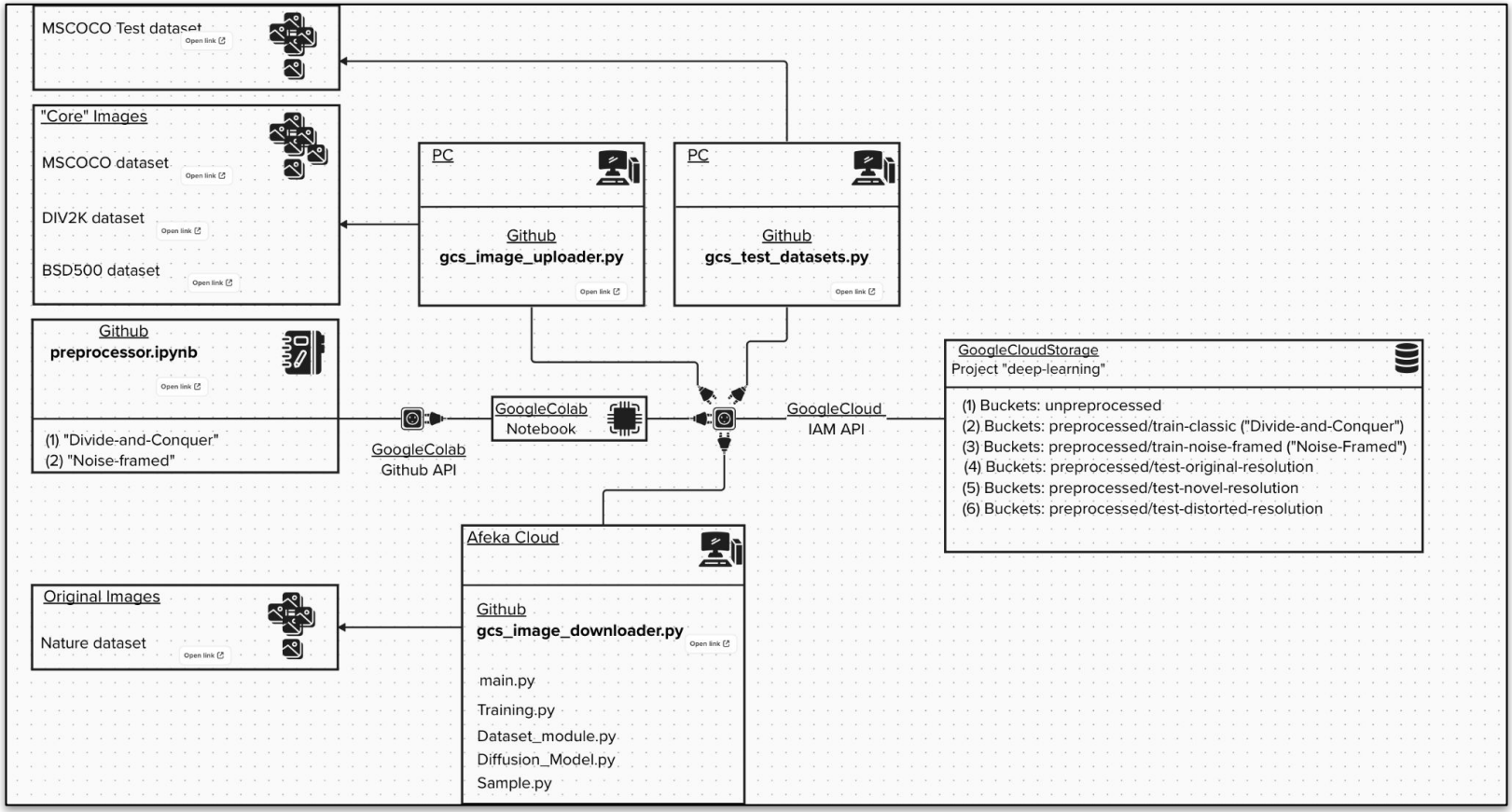
Introduction: Results and Conclusion

We use the models' metrics to compare their performance, operating on the assumption that higher or "better" metrics indicate a model that performs more effectively in the task of image super-resolution.

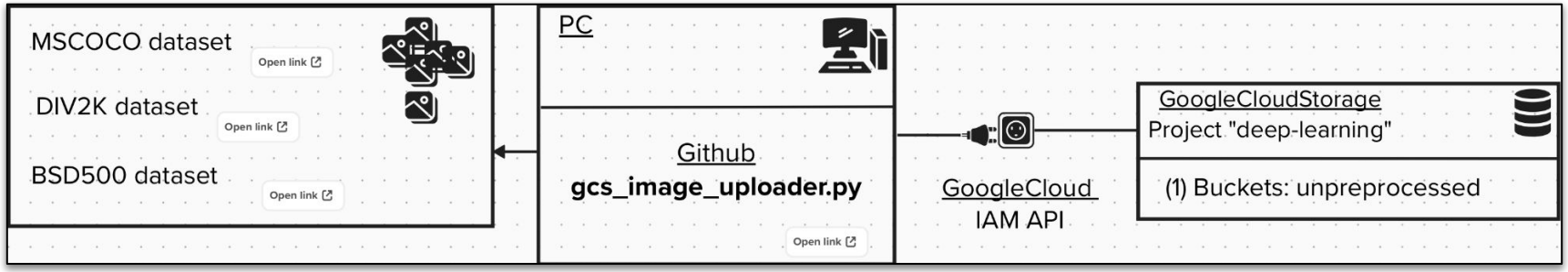
In other words, the key factor in answering our research question is whether a model trained on an automatically prepared dataset outperforms a model trained on a traditionally prepared dataset.

Or, at the very least, does such a model perform equally well?

Methodology: Implementation Overview



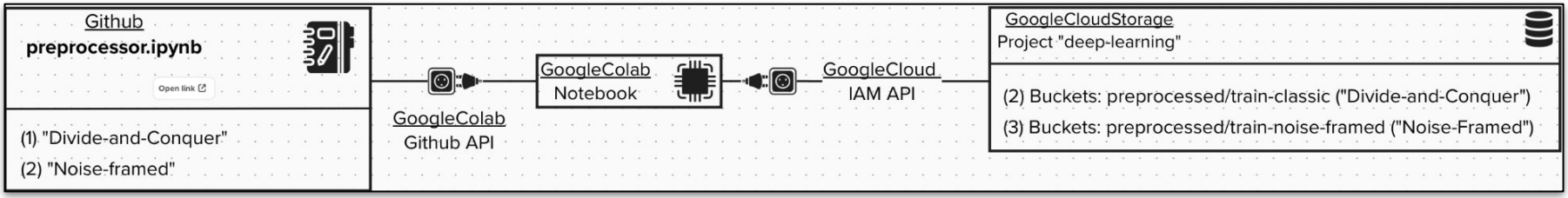
Methodology: “Core” Images



“Core” images were randomly sampled from various training data sources: MSCOCO, DIV2K, BSD500. There was no specific preference for any particular source, aside from ease of access and downloading. The underlying assumption was that "core" images are abundant across the Internet, reducing the need to invest significant effort into acquiring them.

What was the main risk?
Selecting a significant portion of images that are equal to or smaller than the model's required resolution, which would essentially leave no room for splitting such images into "pieces."

Methodology: Automated Image Generation



Our initial observation was the abundance of readily available images across the Internet that exceeded the model's requirement of 128x128 pixels. This led to the creation of the "Divide-and-Conquer" concept. The idea was simple: why not crop each 1920x1080 "core" image into "pieces" of 128x128 pixels?

The second approach, "Noise-framed," emerged quickly after that. Once we reviewed the initial output data from "Divide-and-Conquer," the issue was apparent enough to warrant a solution, which then became an integral part of our methodology.

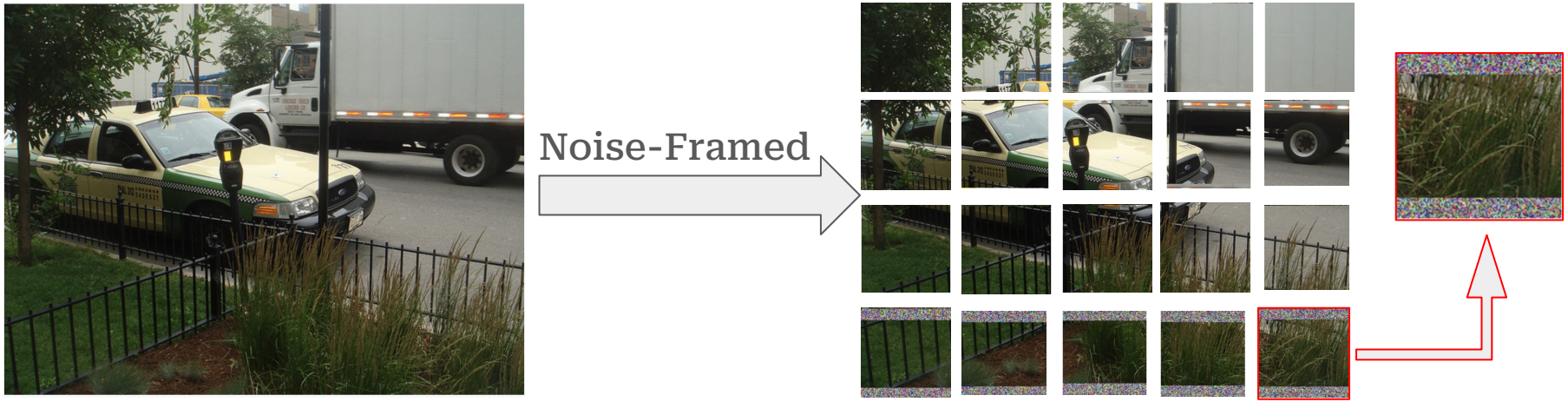
Methodology: Automated Image Generation (*Continued*)

Divide-and-Conquer

At first glance, the “Divide-and-Conquer” approach seemed promising. We were able to divide a single, random image into dozens of high-quality “pieces.”

However, after repeating the process multiple times, a rather obvious issue emerged: what should be done with the “leftovers”?

Methodology: Automated Image Generation (*Continued*)



Given the nature of the breakthrough we covered during our first presentation, Gaussian distribution seemed like a suitable source for generating simulated data that wouldn't introduce significant bias or variability. The simulated data in question is random “noise,” limited to RGB values.

This “noise” is used to frame images until they reach the model’s required resolution.

Methodology: Dataset Preparation

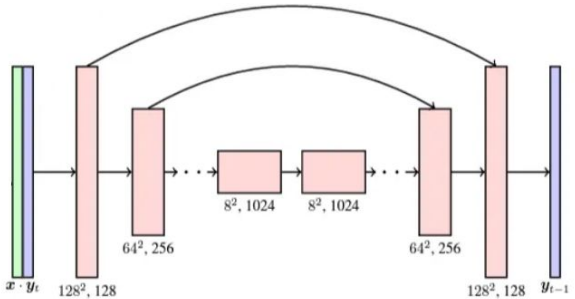
- **Data Collection**: We gathered a diverse set of 100,000 high-resolution images, which were then downscaled to create corresponding low-resolution images. This paired data forms the foundation of our training dataset.

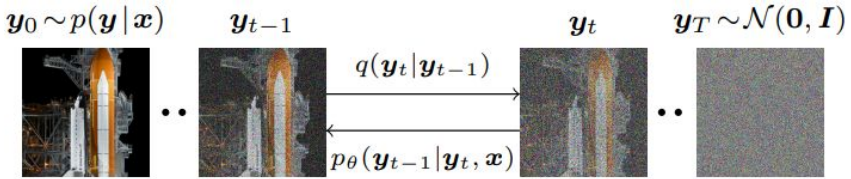
128X128 -> 32X32

- **Data Augmentation**: The dataset is loaded using a custom SRDataset class, which applies transformations such as random flips, color jittering, and sharpness adjustments. These transformations help augment the data and improve the model's ability to generalize.

Methodology: Model Architecture

- This model is based on a U-Net architecture designed to process images through a series of encoder and decoder blocks, along with attention mechanisms that help the model focus on important features.





- A key component of our model is the **Diffusion Process**, where noise is progressively added to images during training, and the model learns to denoise them, effectively super-resolving the input images.

Methodology: Model Training - Training Process

The training loop runs for a specified number of epochs (20 in this case). During each epoch, the model processes batches of images (batch size = 16), learning to predict and remove the noise added to the high-resolution images. The entire process took approximately 10 hours.

For each batch, the model:

1. **Adds Noise:** Noise is added to the HR images based on the current time step. This simulates the process of image degradation.
2. **Upsamples LR Images:** The LR images are upsampled to match the size of the HR images (128x128).
3. **Concatenates Inputs:** The upsampled LR images and noisy HR images are concatenated and passed through the model.
4. **Predicts Noise:** The model predicts the noise present in the images.
5. **Computes Loss:** The mean squared error (MSE) loss is computed between the predicted noise and the actual noise added to the HR images. This loss measures how well the model can reverse the noise process.
6. **Backpropagation:** The loss is backpropagated, and the model's parameters are updated using the Adam optimizer.

Methodology: Saving and Evaluation

1. After every 500 steps within an epoch, the model generates a super-resolved image from a test LR image and saves it to track the model's progress.
2. At the end of each epoch, the model's state is saved as a checkpoint. This allows the training process to be resumed if interrupted and facilitates model evaluation at different stages of training.
3. After training is complete, the final model checkpoint is saved. This model can then be used to generate high-resolution images from low-resolution inputs.

EPOCH 0

EPOCH 5

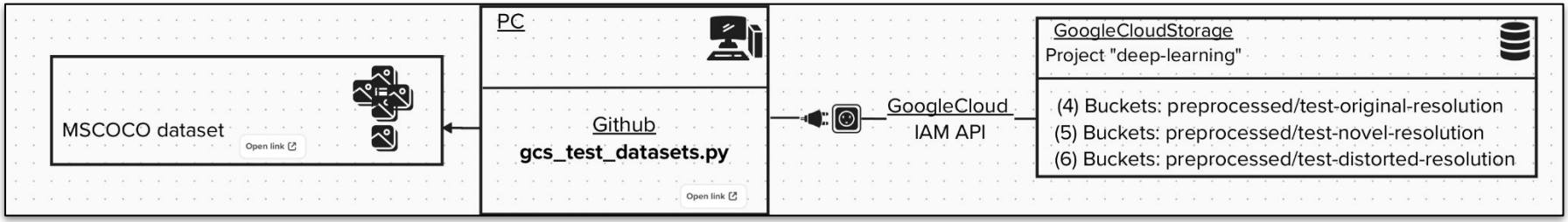
EPOCH 10

EPOCH 15

EPOCH 19

Original

Methodology: Tests



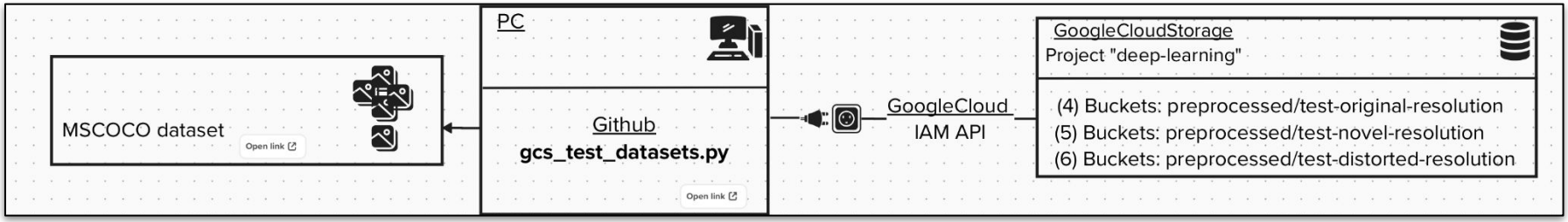
The test images were manually picked from test dataset source: MSCOCO. We made sure that none of the images were used in our train dataset preparation.

The test images were passed through an automated script that divide them into 3 categories:

- “Original”
- “Novel”
- “Distorted”

Every model that we trained was tested on these 3 categories of test dataset.
This was crucial to ensure that the only difference between the models was the train dataset.

Methodology: Tests (*Continued*)



"Original" test images
These are images of 128x128 that are shrunk to 32x32.

"Novel" test images
These are images of 256x256 that are shrunk to 32x32.
The original paper, "Image Super-Resolution via Iterative Refinement", did not provide insight about model's performance that has to perform super-resolution to a target that it was not trained for.

"Distorted" test images
An image of 128x128 was used to compare model's output for the same image that is "blurred".

The last 2 categories demonstrate how potent "generative" side of model is. Given its nature, we expected the model to tackle tasks that are a bit "outside" of the original definition.

Methodology: Tests (*Continued*)

We test a pre-trained image super-resolution model that converts low-resolution images (32x32 pixels) into high-resolution images (128x128 pixels) using the following process:

1. **Model Loading:** Each of the three models is loaded as described.
2. **Dataset Preparation:** A test dataset is loaded, including both low-resolution and high-resolution images.
3. **Image Generation:** The model processes each low-resolution image to generate a higher-resolution version.
4. **Quality Evaluation:** The generated images are compared with the original high-resolution images using metrics like PSNR, SSIM, and MSE to assess image quality.
5. **Results:** The generated images and evaluation metrics are saved, providing a summary of the model's performance.

Results & Conclusion: “Core” Images

Our end goal for a train dataset was 100,000 images. Hence, we settled on acquiring 50,000 core images.

Why 50,000 “core” images?
x2 storage improvement is the least rate worth researching.

Work hours spent
We’ve opted to download “core” images manually. Which took us approximately 5 hours.
The main bottleneck was requirement to download .zip directories as a whole, without an option to sample first.

Results & Conclusion: “Divide-and-Conquer” Train Dataset

Just under 50,000 “core” images were required to generate 100,000 train images.

Storage improvement
x2 improvement. In other words, our approach provides a significant reduction in the storage’s cost.

Effort improvement
It took around 700 min, or 11.6 hours, for the automated process to execute. Adding that to download time, we get 16.6 hours all in all.

In other words, our approach reduces total time invested in train dataset preparation from magnitude of days to hours.

Results & Conclusion: “Noise-framed” Train Dataset

Just under 6,000 “core” images were required to generate 100,000 train images.

Storage improvement
x10 improvement!

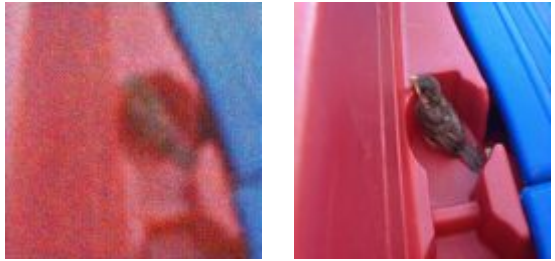
Effort improvement
It took around 500 min, or 8.3 hours, for the automated process to execute. Adding that to download time, we get 13.3 hours all in all.

Results & Conclusion: Models Performance

Model Matrice	Original Train Dataset	“Divide-and-Conquer” Train Dataset	“Noise-framed” Train Dataset
PSNR	18.9183	15.8555	19.0904
SSIM	0.5126	0.3439	0.5323
MSE	97.1121	103.7820	96.4605

Original Train Dataset

“Divide-and-Conquer” Train Dataset



“Noise-framed” Train Dataset

Results & Conclusion: 256X256 pixels and Augmentation

The model was unable to accurately reproduce images that had a resolution different from the one it was trained on or images that were distorted.

Original Image

Degraded Image

Generated Image

256X256 pixels

Results & Conclusion: Final Notes

We believe that our results provided significant proof that train dataset can be generated automatically for models that perform image super-resolution.

Our vision for future work involves further improvement in such processes. For example, usage of existing models that generate images from the “ground-up” instead of reliance on “core” images.

In terms of model improvement, we see 2 interesting questions:

- Should a model of “generative” family tackle super-resolution to size beyond its training objective?
- Should a model of “generative” family tackle image distortion, such as “blur”?

Our intuition was a positive answer to both of the questions, however, the results told us otherwise. Further work is needed to provide a decisive conclusion here.

Q&A

The End