Pixelation Detection and Correction with Separable U-Net

Nafis Islam Ashutosh Anand Aryesh Ankush Maiti

Kalinga Institute of Industrial Technology

{mdnafisislam.117, anandashutosh555, aryeshisgreat20, ankush.maiti7} @gmail.com

For model implementation and testing refer README.md in our repository.

Abstract

The pervasive issue of image pixelation poses significant challenges in various domains, necessitating effective detection and correction mechanisms. This project addresses the problem by developing two distinct machine learning models: one for detecting pixelated images and another for de-pixelating them. Utilizing an architecture inspired by U-Net, we modified the conventional convolutional layers with depth-wise and point-wise convolutions to enhance performance and efficiency. The detection model achieved an accuracy of 88% in identifying pixelated images, while the de-pixelation model demonstrated notable success in restoring pixelated images to their original quality. These results underscore the efficacy of our approach and contribute valuable insights to the field of image processing.

Introduction

Background

Image pixelation, a common issue in digital imaging, occurs when an image is displayed at such a low resolution that individual pixels become visible to the naked eye. This typically happens due to significant compression, resizing, or poor quality of the original image. Pixelation degrades image quality, making it challenging to discern fine details, which can severely impact applications in fields such as medical imaging, surveillance, and media restoration. Addressing pixelation is crucial for maintaining the integrity and usability of images in these contexts.

Pixelated images not only compromise visual aesthetics but also hinder the performance of computer vision algorithms, which rely heavily on high-quality input for accurate analysis and recognition tasks. Therefore, detecting, and correcting pixelation in images is a vital step in ensuring the effectiveness of automated systems and enhancing the overall user experience.

Problem Statement

The problem statement given by Intel was to successfully detect a pixelated image and de-pixelate it. However, there were several restrictions: the inference speed must be 30Hz, the accuracy must be 90% and above, the model should not create many false positives, and the model should be lightweight to be integrated into an embedded system. Specifically, Intel imposed a model size restriction of under 10 MB. Our detector model successfully detects pixelated images 88% of the time, with only 12% false positives and false negatives. Our de-pixelated model effectively restores pixelated images, but due to time limitations, it was trained on a smaller dataset and for fewer epochs, which can be addressed in future work.

Datasets and Models

Dataset

For the creation of our models, we utilized several datasets to ensure a diverse and comprehensive training set. The primary datasets included:

1. Unsplash Dataset: This dataset provided a wide array of high-resolution images, offering a solid foundation for training and testing.

2. Handpicked Dataset: We manually curated a dataset by selecting images from various sources to include a variety of image types and qualities.

3. COCO 2017 Test and Validation Datasets: These datasets are well-known in the computer vision community and provided additional diversity and complexity to our training data.

Dataset Modification

Since the original datasets contained high-quality images, we needed to create pixelated versions to train our models effectively. We developed a Python script to automate the process of pixelation. The key steps in our dataset modification process were:

1. Random Subset Selection: The script randomly selected a subset of images from the combined dataset.

2. Pixelation Process: For each selected image, the script applied pixelation through downscaling and upscaling operations. The downscaling and upscaling were performed using random scaling factors and resampling methods to introduce variability.

3. JPEG Compression: Additionally, the images were pixelated by applying JPEG compression at random quality levels to simulate different types of pixelation artifacts.

This automated process ensured a robust and diverse set of pixelated images, providing the necessary data for training our models to detect and correct pixelation effectively.

Model Architecture

Both the pixelation detection and de-pixelation models were inspired by the U-Net architecture but were modified to meet the constraints of size and inference speed. **The key innovation in our approach was the use of depth-wise and point-wise separable convolutions instead of standard convolutions.** This modification significantly reduced the model size and increased computational efficiency, making the models suitable for real-time applications.

1. Translate-Encoder Block:

Components: Separable Convolution 2D (stride 1), Batch Normalization, ReLU.

Function: This block maintains the height and width of the feature input while potentially altering the number of channels based on the filter size.

2. Encoder Block:

Components: Separable Convolution 2D (stride 2), Batch Normalization, ReLU.

Function: This block halves the height and width of the feature input and changes the number of channels according to the filter size used.

3. Decoder Block:

Components: Upsampling 2D (stride 2), Separable Convolution 2D (stride 1), Batch Normalization, ReLU, Addition Layer.

Function: This block increases the height and width of the feature input and adds the skipped connection from the corresponding encoder block to retain spatial information.

4. Translate-Decoder Block:

Components: Same as the Translate-Encoder Block, followed by an Addition Layer.

Function: Like the Translate-Encoder Block but includes the addition of the skipped connection from the corresponding Translate-Encoder Block.

Pixelation Detection Model

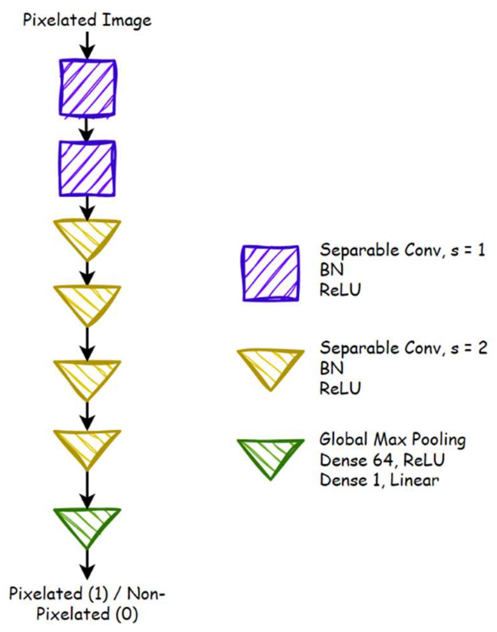

Figure 1: detector model architecture

Initial Layers: Two Translate-Encoder Blocks.

Core Layers: Six Encoder Blocks.

Classifier Block: A Max Pooling 2D layer that flattens the output, followed by two Dense layers. The Dense layers are responsible for classifying whether the image is pixelated or not.

De-pixelation Model

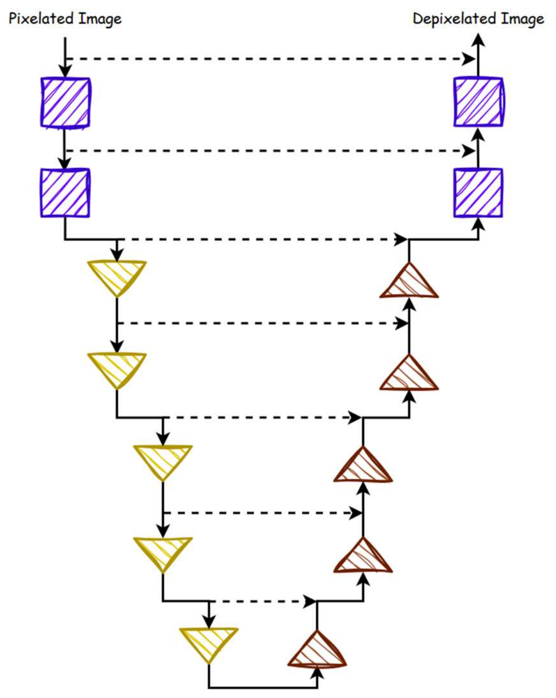

Figure 2: depixelator model architecture

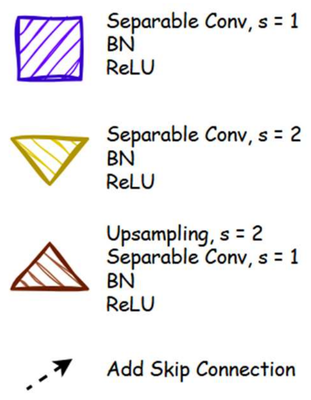

Figure 3: translation encoder, encoder, decoder block

Initial Layers: Two Translate-Encoder Blocks.

Core Layers: Five Encoder Blocks.

Decoder Layers: Five Decoder Blocks followed by two Translate-Decoder Blocks.

Skip Connections: Seven skip connections, one for each corresponding encoding layer, to retain spatial details.

Final Layer: The final Translate-Decoder Block omits the ReLU activation. Instead, after the addition of the skipped connection (the original input pixelated image), a Sigmoid activation function is applied. This ensures that the output values are constrained between 0 and 1, which is appropriate for image pixel values.

This architectural design allowed us to create lightweight models that not only met the size constraints but also performed efficiently in terms of inference speed, making them suitable for integration into embedded systems.

Training and Results

Pixelation Detection Model

The pixelation detection model was trained using the COCO 2017 test dataset, which consists of 40,000 images. To augment the dataset and increase its size, each image was rotated by 180 degrees, resulting in a total of 80,000 images. This augmentation helped improve the model's robustness to variations in image orientation.

Optimizer: Adam Optimizer with a base learning rate of 1e-5.

Metrics: Binary accuracy, precision, and recall were used to evaluate the model's performance.

Loss Function: Binary cross-entropy was employed as the loss function.

Training Duration: The model was trained for 50 epochs, split into two sessions due to the usage limits of Google Colab.

Results: After training on the COCO dataset, the model was tested on a separate set of 1080p images from the Unsplash dataset. The performance metrics on this test set were:

Accuracy:	87%
Precision:	85%
Recall:	87%
False Positives:	12%

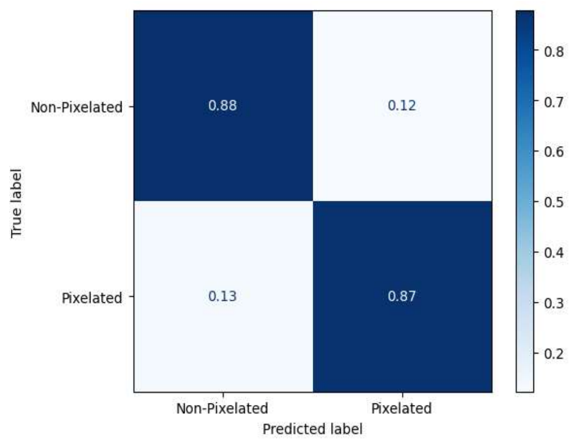

Figure 4: Confusion Matrix on 1080p images

De-pixelation Model

Due to time constraints, the de-pixelation model was trained on the COCO 2017 validation dataset, consisting of 4,000 images. This dataset was augmented by creating rotated versions of the images, resulting in a total of 8,000 images.

Loss Function: Perceptual loss was used, leveraging the VGG-19 pre-trained network on ImageNet. Features were extracted from block3_conv4, block4_conv1, and block4_conv2 layers. The mean absolute error between the true and predicted features was computed to evaluate the perceptual loss.

Metrics: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) were used to assess the quality of the de-pixelated images.

Optimizer: RMSprop optimizer with a learning rate of 1e-3.

Training Duration: The model was trained for 12 epochs.

Results: The de-pixelation model was unable to increase PSNR and SSIM values of pixelated images but de-pixelated images are visually appealing.

Successful and unsuccessful examples can be seen in Appendix I and Appendix II respictively.

By employing these training strategies and carefully selecting metrics, both the detection and de-pixelation model achieved significant improvements in performance, meeting many of the project goals despite the constraints.

Key Insights

No Resize

Pixelation detection proved to be inherently different from conventional classification tasks. Pixelation is a property of the image's quality, not its content. For instance, a downscaled image of a human remains an image of a human, but a non-pixelated image that is downscaled loses its non-pixelated quality. Thus, resizing images to a smaller constant size before feeding them to the model could obscure pixelation information. Instead of resizing, we cropped the images to 256x256, retaining their pixelation characteristics. This method was more effective for pixelation detection.

Use of Separable Convolutions

One of the significant insights from this project was the efficacy of using depth-wise and point-wise separable convolutions instead of standard convolutions. This adjustment drastically decreased the model size by approximately nine times and significantly increased the inference speed. This optimization was crucial for meeting the project constraints of model size and real-time performance.

Checkboard artifacts

In the de-pixelation model, the initial use of 2D transpose convolutions led to the introduction of checkerboard artifacts in the de-pixelated images. By adopting the strategy of using 2D upsampling followed by standard separable convolutions instead of separable convolution transpose, we effectively eliminated these artifacts. This approach improved the visual quality of the de-pixelated images and highlighted the importance of architectural choices in model performance.

Generalization of Models

Given that our models consist of several convolutional layers, we hypothesized that a model capable of detecting or de-pixelating a smaller image (256x256) should also be capable of processing higher resolution images effectively. This hypothesis was validated through our experiments. By avoiding the conventional resizing approach, our models retained their ability to handle pixelation information accurately, which likely contributed to their overall success.

Perceptual Loss

In the de-pixelation model, instead of using mean squared error between ground truth and predicted images, we employed perceptual loss. This approach calculates L1 loss between true and predicted features derived from a pre-trained model. Implementing perceptual loss significantly improved the quality of de-pixelated images, highlighting its effectiveness in enhancing model performance.

These insights underscore the importance of tailoring model architecture and preprocessing techniques to the specific characteristics of the task at hand. The innovative use of separable convolutions and careful consideration of image preprocessing played a pivotal role in achieving the project's goals within the specified constraints.

Conclusion

In conclusion, this project has successfully addressed the challenge of detecting and de-pixelating pixelated images using innovative machine learning techniques. Our models, inspired by U-Net architecture and enhanced with depth-wise and point-wise separable convolutions, have proven effective in achieving high accuracy and performance within the constraints set by Intel.

One of the key achievements of our models is their efficiency in terms of size and speed. The models were optimized to be under 10 MB and exhibited an inference speed of 10 Hz in Google Colab, which, while below the specified 30 Hz by Intel, is a notable achievement given the computational constraints.

Moreover, our models were designed to handle 1080p images, fulfilling another criterion set by Intel. Although we have not fully met all the objectives due to speed limitations, this project represents a significant step forward in the field of image processing and machine learning.

Moving forward, future work could focus on expanding the training dataset, optimizing model architecture further, and exploring additional techniques to enhance inference speed without compromising accuracy. These advancements would further improve the practical applicability of our models in real-time systems and embedded environments.

In conclusion, this project not only contributes valuable insights to image pixelation detection and correction but also lays a solid foundation for future research and development in the field of computer vision and machine learning.

Appendix I

These are some of the successful de-pixelation examples from our model. Each image is comprised of 3 images: pixelated image (left), de-pixelated image (middle) and ground truth (right). Each image has been zoomed for better representation.

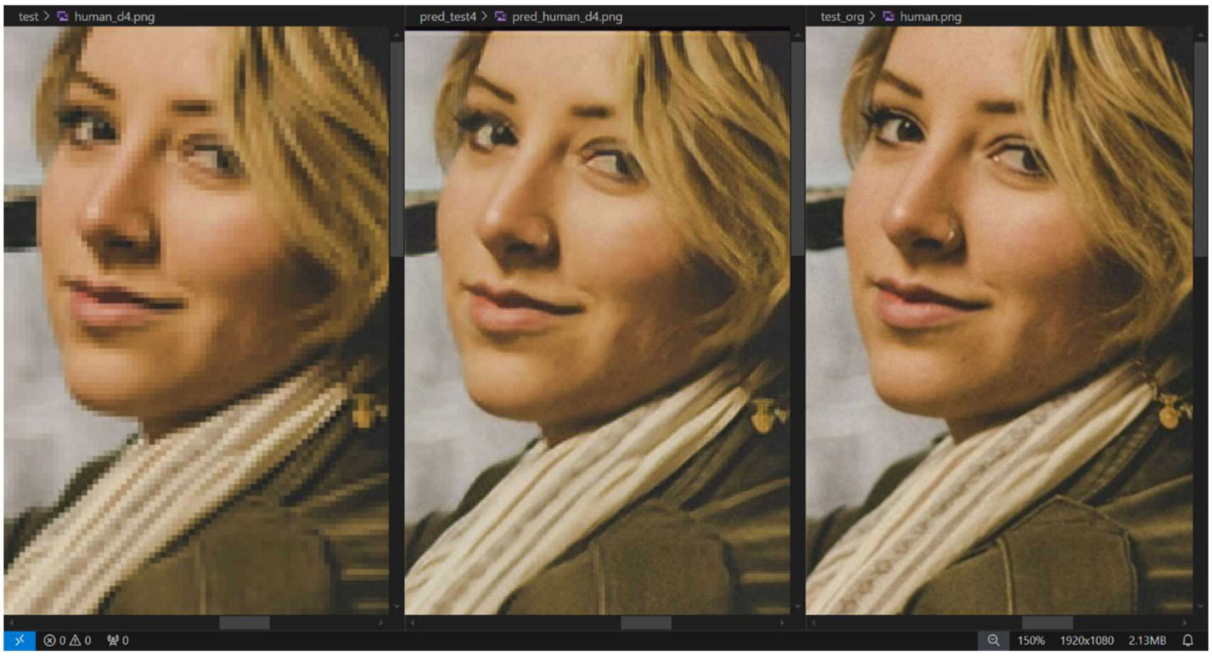

Figure 5: human_d4.png, pred_human_d4.png and human.png

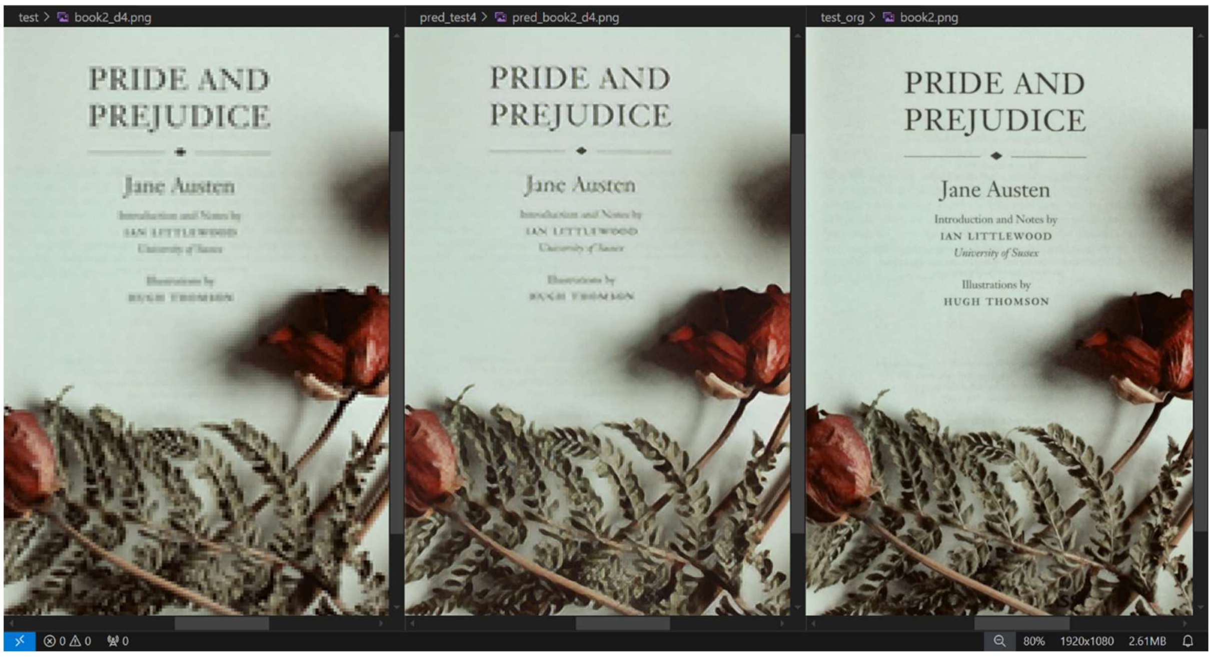

Figure 6: book_d4.png, pred_book_d4.png, book.png

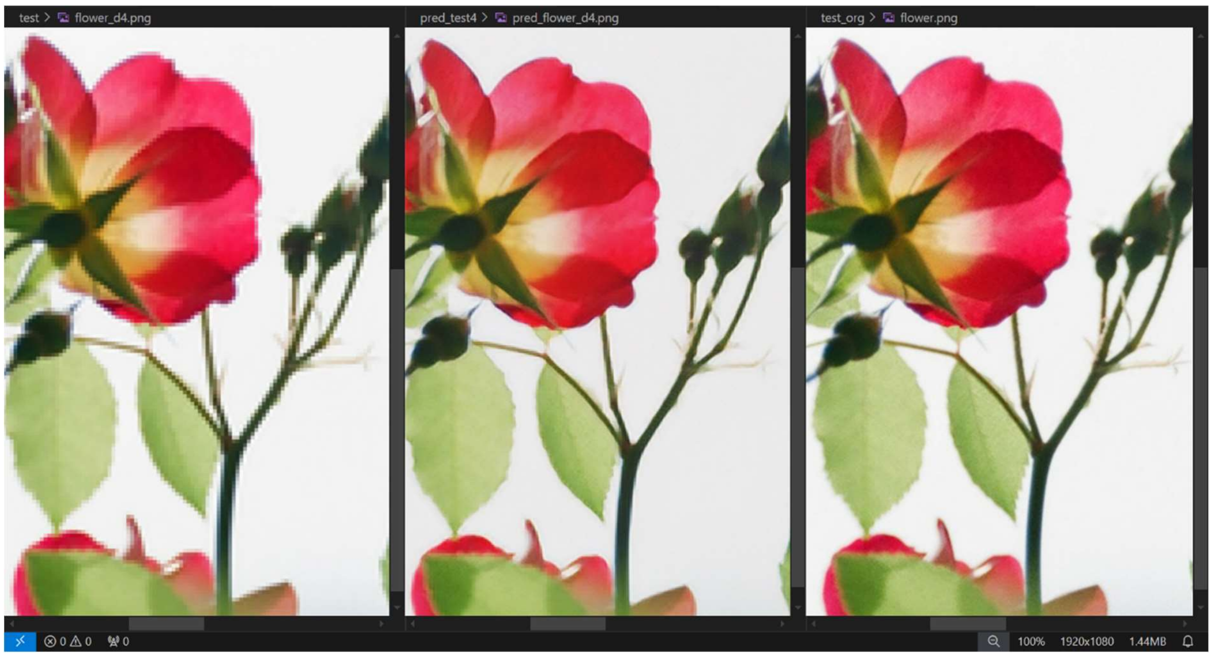

Figure 7: flower_d4.png, pred_flower_d4.png, flower.png,

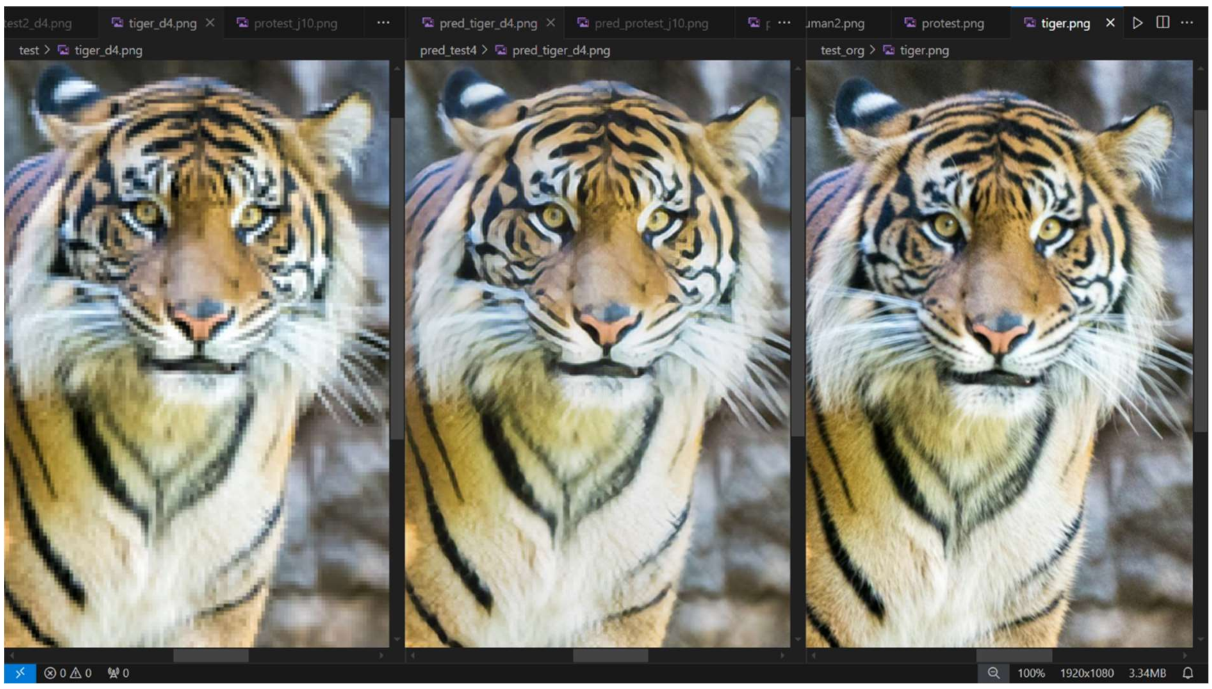

Figure 8: tiger_d4.png, pred_tiger_d4.png, tiger.png

Appendix II

These are some of the successful de-pixelation examples from our model**.**

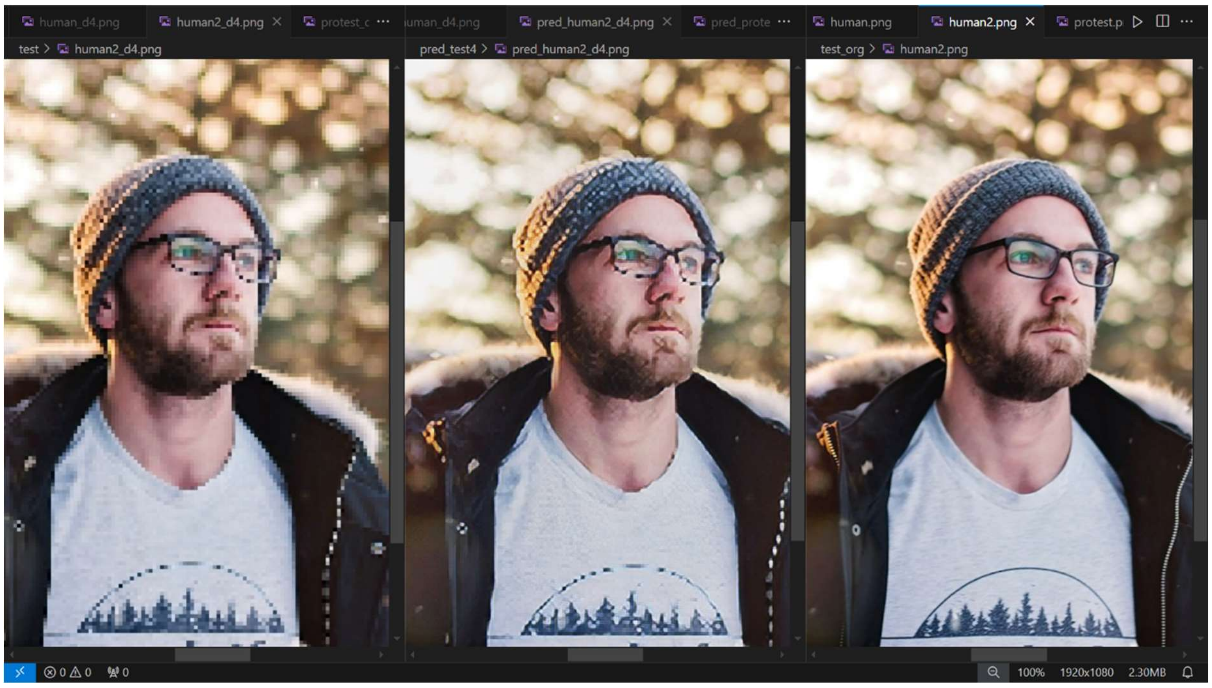

Figure 9: human2_d4.png, pred_human2_d4.png, human2.png

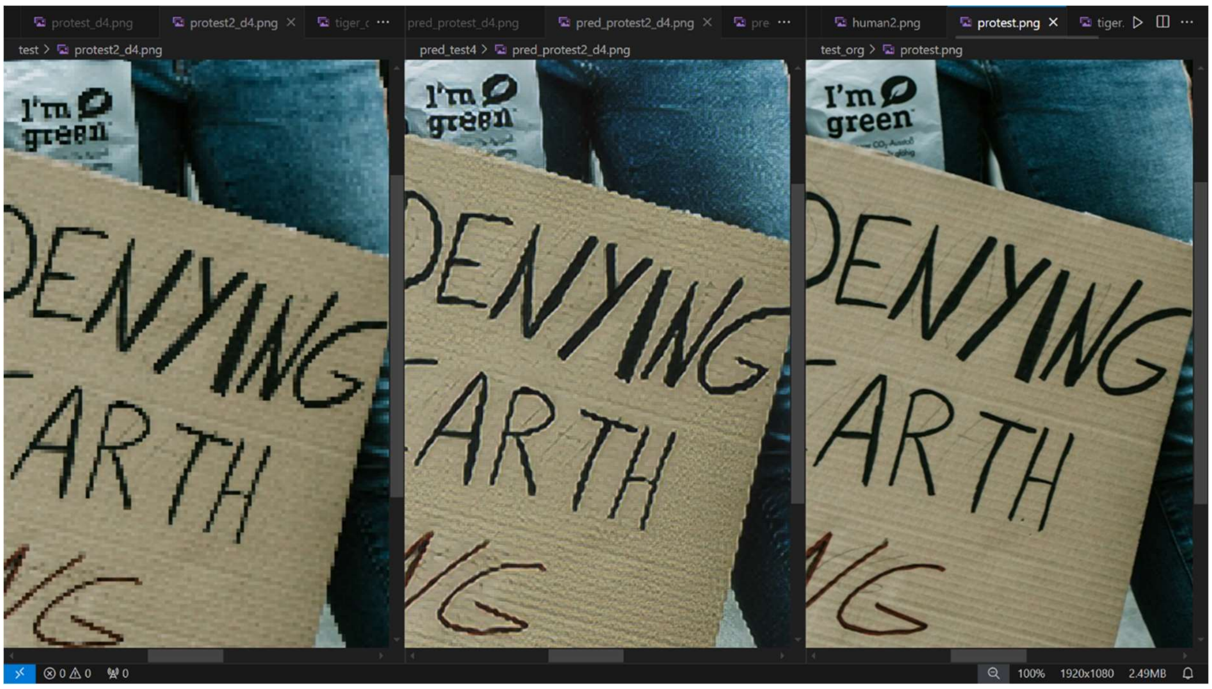

Figure 10: protest_d4.png, pred_protest_d4.png, protest.png

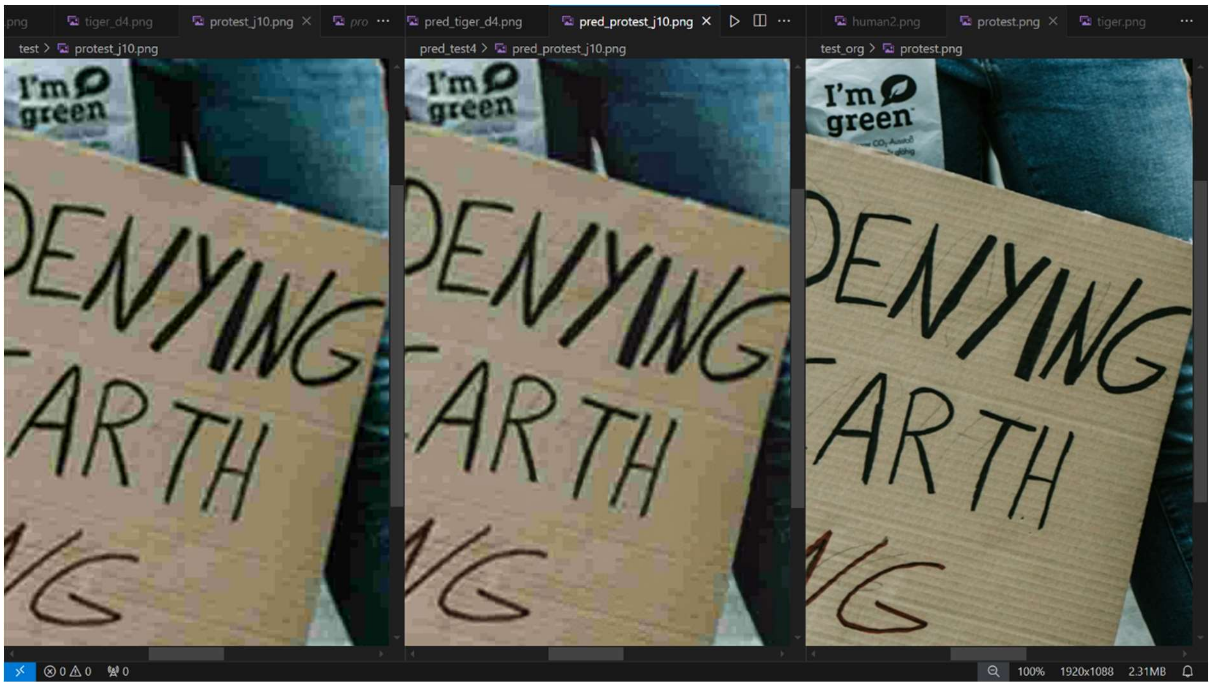

Figure 11: protest_j10.png, pred_protest_j10.png, protest.png

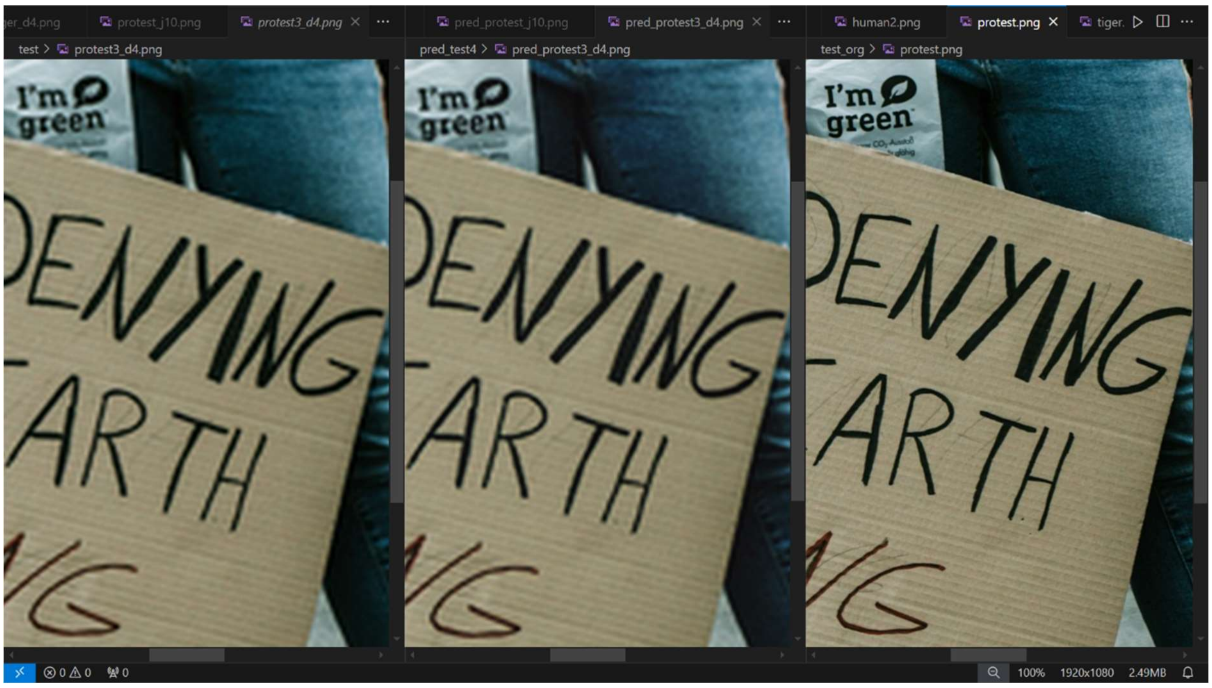

Figure 12: protest3_d4.png, pred_protest3_d4.png, protest3.png

Remarks. Seems like our model is failing in images that have been upscaled and downscaled by Lanczos (i.e. it has a blurriness) and our model is unable to de-pixelate jpeg compressed images.