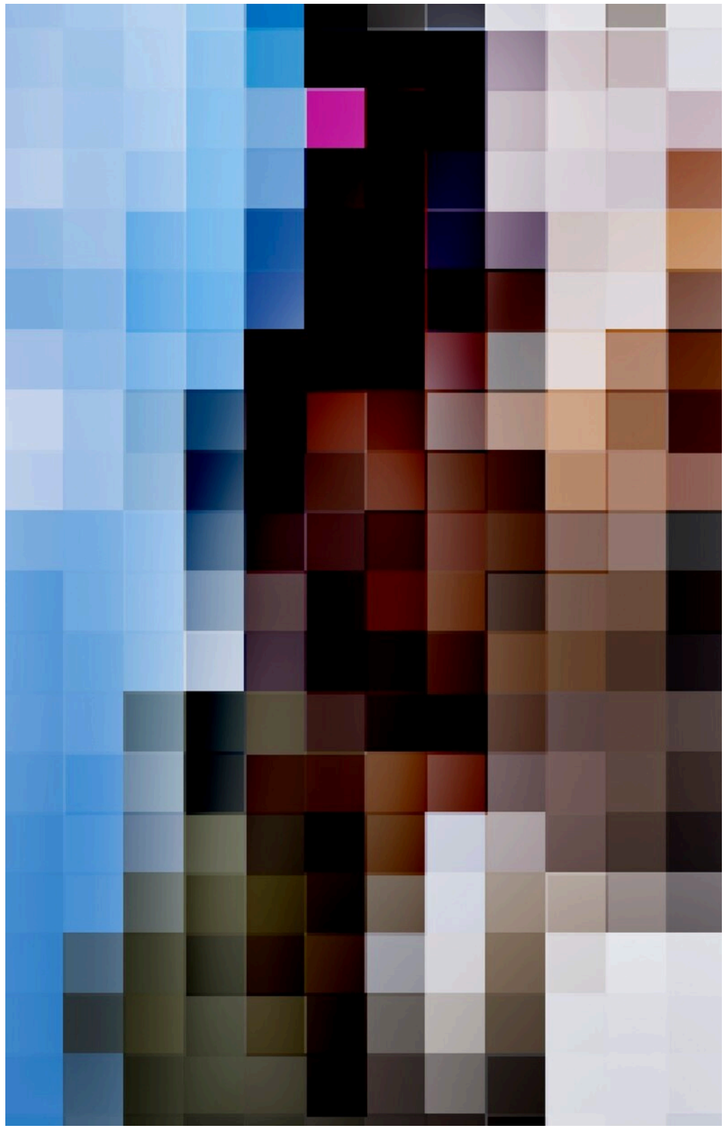


Detecting Pixelated Image And Its Correcting

Brief on problem statement and solution

The problem statement our team got is detection of pixels in images and its correction. This is a significant and interesting problem statement due to ML and DL models being able to work with images so well in today's time but there are few or none such model that could detect and fix pixelation in images.

Our team started with understanding different artifacts of an image such as noise, blur, pixelation, etc, and tried to understand what pixelation of an image actually means. After we figured this out we tried to find certain metrics that defines such artifacts in an image such as dct, hog features etc, and set a threshold to find what makes an image pixelated on these features we created a classification ML model that works as a detection model.

After this we went on to the second part of the problem statement which is to fix the pixelated image. For this we had to do research on the current top notch super resolution models which are CNN and GAN based and tried to understand how these architectures are going to help with our problem. To our understanding these models were made for task such as upsampling a lower resolution image to a much higher resolution image which loosely aligned to what we were trying to do, further we studied some state of the art GAN models and made our own model based on GAN to fix pixelation of image, and after that we trained the model and the next few slides has some more explanation of both the ML and GAN DL model.

Pixelation Detection Model

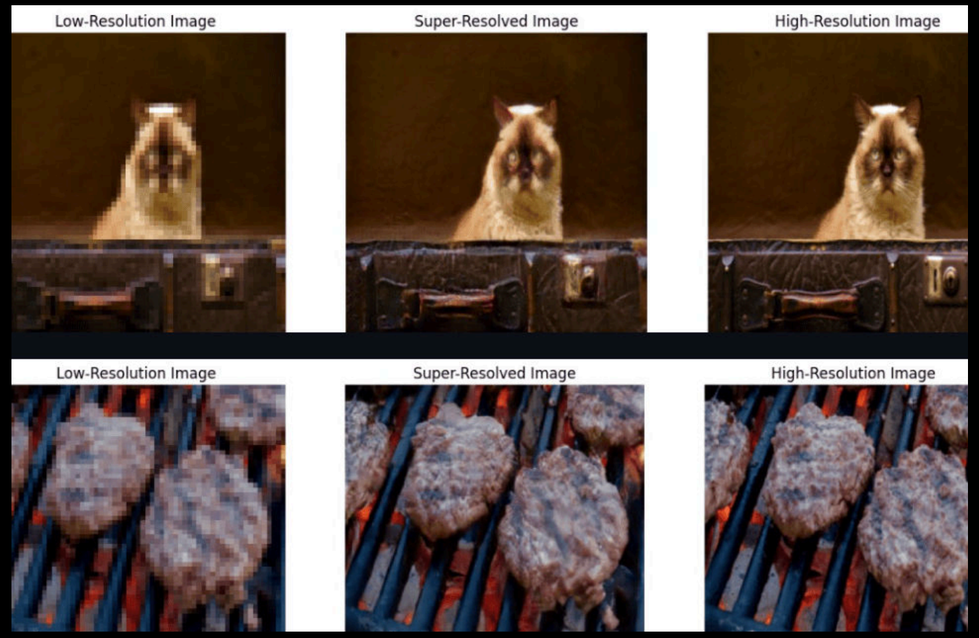
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier

estimators = [
    ('rf', RandomForestClassifier(...)),
    ('xgb', XGBClassifier(...)),
    ('hclf', HistGradientBoostingClassifier(...))
]

stack = StackingClassifier(
    estimators=estimators,
    final_estimator=MLPClassifier(random_state=42),
    cv=5
)
```

Metric	Proposed Method on Flickr2K +Div2k
Precision	0.99
Recall	0.99
F1 Score	0.99
Accuracy	0.99
FPS	300
Model Size	17.2 MB

Our image processing script utilizes advanced techniques to extract detailed features from images by analyzing patches systematically. Each patch undergoes rigorous analysis: Canny edge detection computes edge density, while Discrete Cosine Transform (DCT) captures block-like patterns via the first 8x8 coefficients. Histograms normalize pixel intensities across patches, while Histogram of Oriented Gradients (HOG) precisely maps textural details. Fast Fourier Transform (FFT) analyzes frequency components, extracting initial 8x8 coefficients to provide insights into image structure and content. These methods collectively offer a comprehensive understanding of image characteristics and variations. The script aggregates these patch-level features across the entire image, facilitating a holistic grasp of its intricate visual details and structural composition. This approach enhances our ability to analyze and interpret complex image data effectively. For classification tasks, we've implemented a stacking classifier that combines Random Forest, XGBoost, and HistGradientBoosting models, with a final Multi-Layer Perceptron (MLP) as the ultimate estimator. Leveraging 5-fold cross-validation ensures robustness and generalizability of the model. In evaluation, our classifier achieves a commendable [insert accuracy] accuracy on the test dataset. Detailed performance metrics including precision, recall, and F1-score highlight the model's efficacy in both feature extraction and classification tasks. Overall, our script not only excels in extracting fine-grained image features using patch analysis but also demonstrates strong classification performance. This dual capability makes it a powerful tool for applications requiring nuanced image understanding and accurate categorization.

Pixelation Correction Model

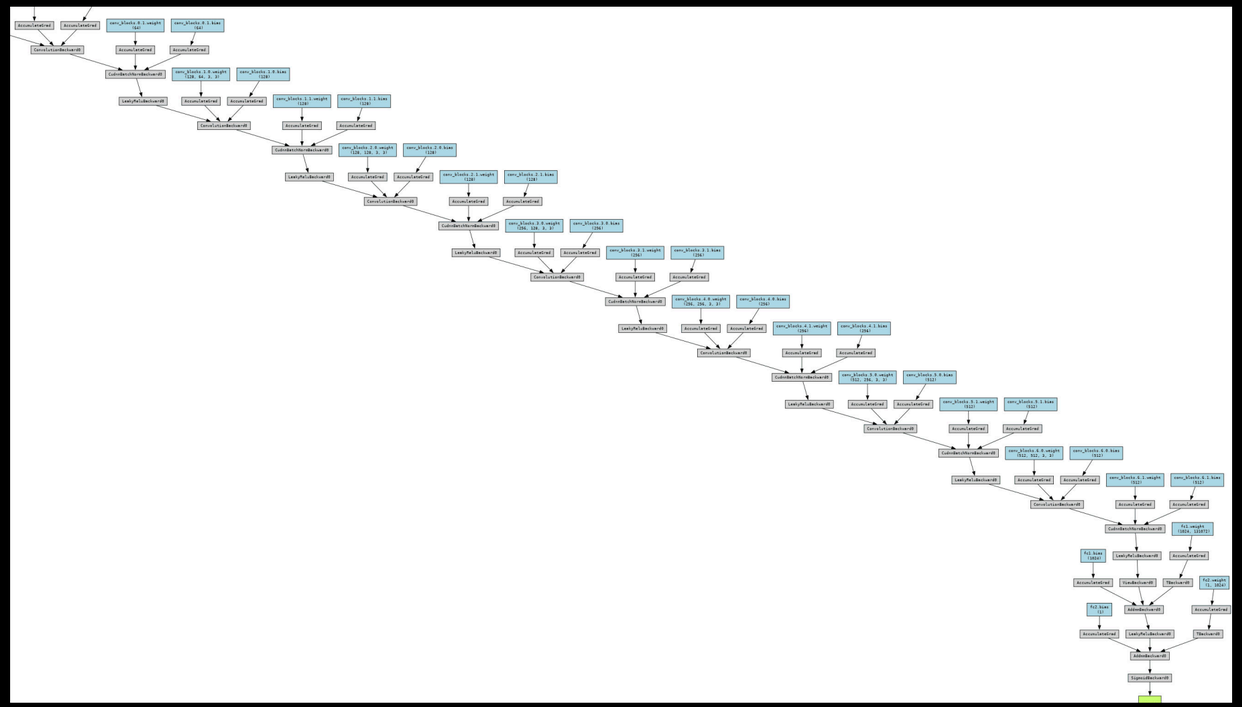
Metric	Proposed Model	SRGAN	EDSR	FSRCNN
PSNR	26.556 dB	29.99 dB	31.78 dB	30.52 dB
SSIM	0.7690	0.8176	0.8895	0.8548
LPIPS	0.3064	0.1118	0.1922	0.2013
Speed (FPS)	79	12	16	188
Model Size (MB)	6.01	5.874	5.789	0.049



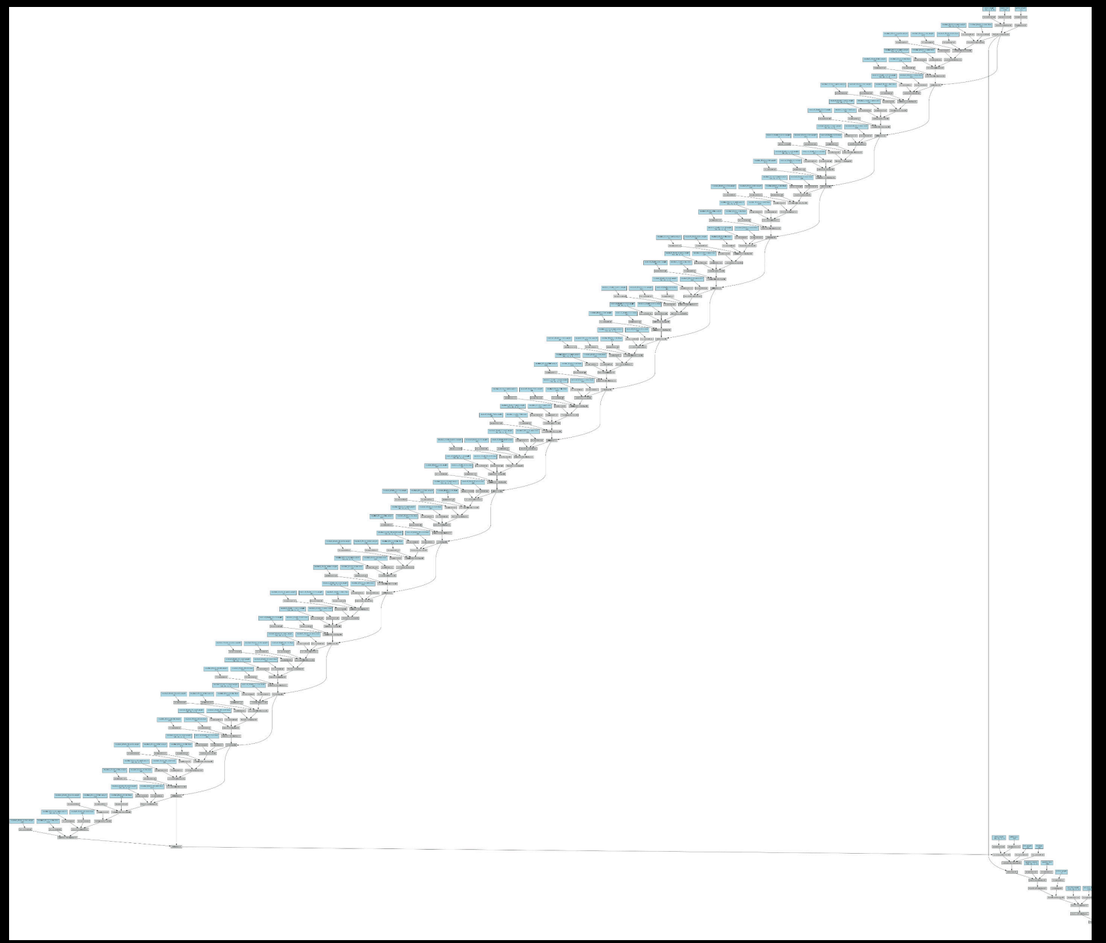
In our SRGAN (Super-Resolution Generative Adversarial Network) model, the Discriminator plays a critical role in distinguishing between genuine high-resolution images and those synthesized by the Generator. The architecture begins with a series of convolutional layers designed to progressively extract hierarchical features from the input images. Starting with 64 filters and a 3x3 kernel size, these layers increase in depth to 128, 256, and 512 filters, each followed by Leaky ReLU activations to introduce non-linearity and enhance feature representation. Batch normalization layers are incorporated to stabilize training and accelerate convergence.To capture both local and global image features effectively, the Discriminator employs strided convolutions for downsampling, reducing spatial dimensions while enriching feature depth. This hierarchical feature extraction process ensures that the Discriminator can discern intricate details in both real and generated images.Following convolutional feature extraction, fully connected layers further refine the learned features. A dense layer with 1024 units and Leaky ReLU activation refines the feature representation, preparing it for the final decision-making stage. The Discriminator concludes with a single-unit output layer activated by a Sigmoid function. This output layer outputs a probability score, indicating whether the input image is classified as real or fake.During training, the Discriminator is trained to minimize the Binary Cross-Entropy (BCE) loss function, optimizing its ability to distinguish between real and generated images effectively. This adversarial training process ensures that the Generator continually improves in producing high-quality, perceptually realistic high-resolution images. Overall, our Discriminator network enhances the SRGAN model's capability to deliver superior image restoration and enhancement, showcasing its potential in various applications requiring enhanced visual fidelity and realism.

Correction Model Architechture

Discriminator Architecture



Generator Architecture



Technologies/Modules used

torch==2.1.0
torchvision==0.16.0
pillow==10.0.1
numpy==1.26.0
opencv-python==4.9.0.80
scikit-learn==1.3.2
matplotlib==3.8.2
tqdm==4.66.1
joblib==1.4.2
scikit-image==0.24.0
xgboost==2.1.0
scipy==1.13.1
threadpoolctl==3.5.0

Subjective Analysis

We had 5 participants. Participants score each image restoration on a scale of 1 to 5. We choose 10 images not used in the training dataset. Each participant scores 10 images restored by our model.

Image	Participant 1		Participant 2		Participant 3	
	Participant 4		Participant 5		Average Score	
1	4	3	4	5	4	4.0
2	4	4	3	4	4	3.8
3	3	4	4	3	4	3.6
4	4	3	4	5	4	4.0
5	4	4	3	3	4	3.6
6	3	4	3	4	5	3.8
7	4	3	4	5	3	3.8
8	3	4	4	4	3	3.6
9	4	4	4	3	3	3.6
10	4	4	4	5	3	4.0

Calculate the overall MOS for our model: MOS= Sum of Average Scores for All Images/Number of Images

MOS= 38.2/10

=3.82

Contributors

B. Lawrence Sharma: Led the research and development of the detection and correction model. Played a key role in testing and deriving inferences from the models while also handling model training and evaluation.

Nilotpal Kashyap: Managed version control and conducted extensive research on the models. Responsible for metrics calculation, model testing, and dataset discovery and analysis.

Madhav Punj: Focused on researching and analyzing various detection models, contributing significantly to the model selection process.

Harsh Shandilya: Specialized in researching and analyzing super-resolution models, aiding in the enhancement of image quality.

Conclusion

Conclusion The implementation provided combines two key functionalities: detecting pixelation in an image and enhancing its resolution using a Super-Resolution Generative Adversarial Network (SRGAN). Key Components: Pixelation Detection: This part of the code uses a machine learning model to detect if an image is pixelated or not. It involves: Extracting various features from image patches, such as edge detection, DCT features, histogram of pixel intensities, HOG for texture, and FFT for frequency analysis. Aggregating these features to form a comprehensive feature vector for the entire image. Using a GAN model for Super Resolution: If the image is found to be pixelated by the detection model, the GAN model is used to enhance its resolution and de-pixelate itThe generator model consists of an initial convolutional layer, multiple residual blocks, upsampling layers using PixelShuffle, and a final convolutional layer.The generated image is then denormalized, saved, and displayed using matplotlib. Our Model is extremely efficient and lightweight with a mere size of 30mb (detection+correction) making it perfect for embedded systems with respect to other pre-existing models which may give a better result at times but are extremly heavy and memory inefficient.

Thanks!

Team name: On Top
Intel Unnati 2024