Names:

Sindhu (MD24S001)

Bhargava Satya Nunna(BT24S007)

Shivam Chaudhari(BT24S003)

Data Augmentation

ImageDataGenerator and Random Cropping

This part had the biggest challenge. Since we're using a very small set for training, it is imparative to have good image augmentations. Keras' ImageDataGenerator class comes with different functionalities but it lacks an important one: random cropping. Since the images have different sizes, it is not optimal to resize them to a fixed size that would result in deformation and degrading images. I dug into Keras' source code and found that a function called load_img referenced as keras.preprocessing.image.image.load_img takes care of loading images and then immediately resizing them to a specifed size. So defined my own function that performs random cropping and overrided it with the original function.

Requirements

- Python >= 3.5
- Keras >= 2.2.4
- PIL
- Numpy
- Matplotlib
- Jupyter Notebook

Model Architecture

The architecture somewhat resembles U-Net, an encoder-decoder network with skip connections between mirrored layers in the encoder and decoder stacks. The network consists of convolutional, dropout and batch normalization layers LeakyReLU used as the activation function. One of notable differences between this model and UNet is that it uses strided convolutions and transposed convolutions instead of pooling and upsampling layers.

The architecture you're looking at is a DnCNN (Denoising Convolutional Neural Network) model, which is specifically designed for image denoising tasks. Here are some resources and key points to help you understand and get more information about the DnCNN architecture:

The foundational work on DnCNN was presented in the paper titled "DnCNN: Beyond a Gaussian Denoiser." You can find it here: DnCNN Paper. This paper explains the architecture, the training process, and various results.

Key Components of DnCNN:

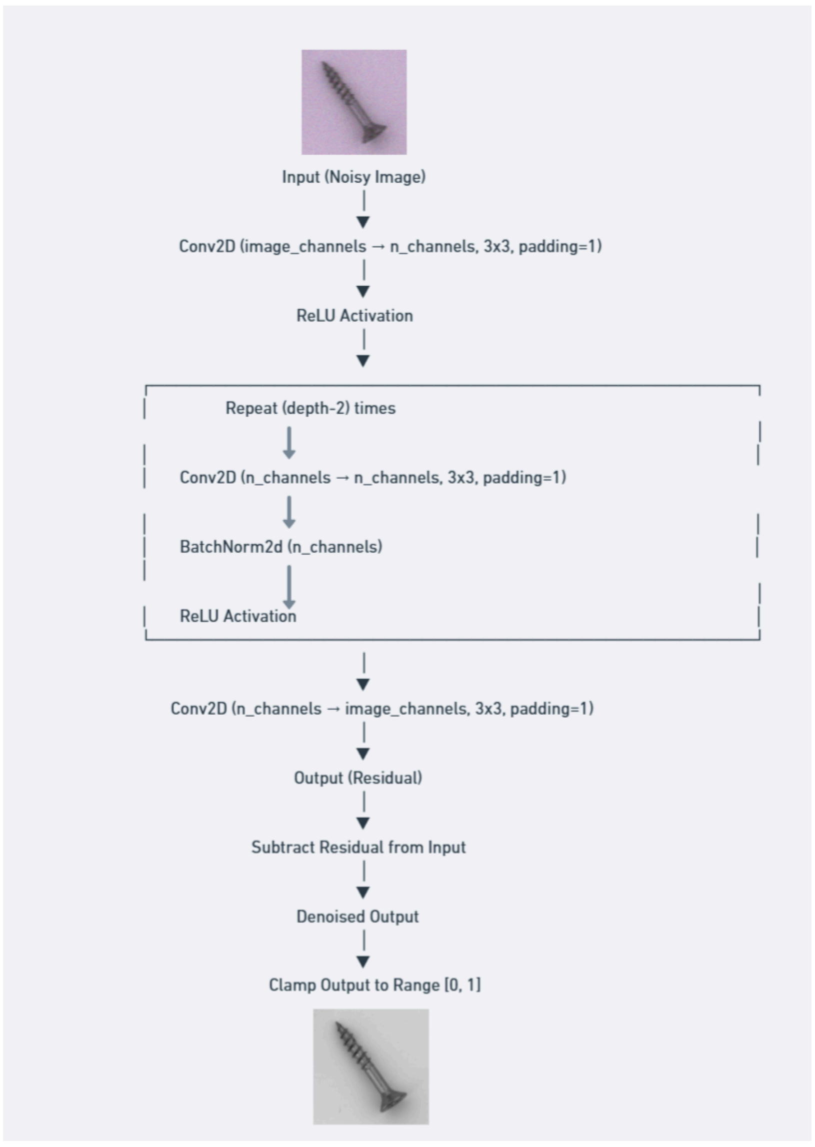
Convolutional Layers: The model consists of multiple convolutional layers to learn the mapping from noisy images to clean images.

Residual Learning: DnCNN employs residual learning, where the output is the noise that needs to be subtracted from the input image.

Batch Normalization: This is used between the convolutional layers to stabilize and accelerate the training process.

ReLU Activation: The model uses the ReLU activation function to introduce non-linearity.

Implementations: You can find various implementations of DnCNN on GitHub, which can help you understand how the architecture is applied in practice. Here are a few notable repositories:


Input (Noisy Image)
Conv2D (image_channels → n_channels, 3x3, padding=1)
ReLU Activation
Repeat (depth-2) times
Conv2D (n_channels → n_channels, 3x3, padding=1)
BatchNorm2d (n_channels)
ReLU Activation
Conv2D (n_channels → image_channels, 3x3, padding=1)
Output (Residual)
Subtract Residual from Input
Denoised Output
Clamp Output to Range [0, 1]

Github link

GitHub: https://github.com/nbs19/AutoAE_Denoising/tree/main

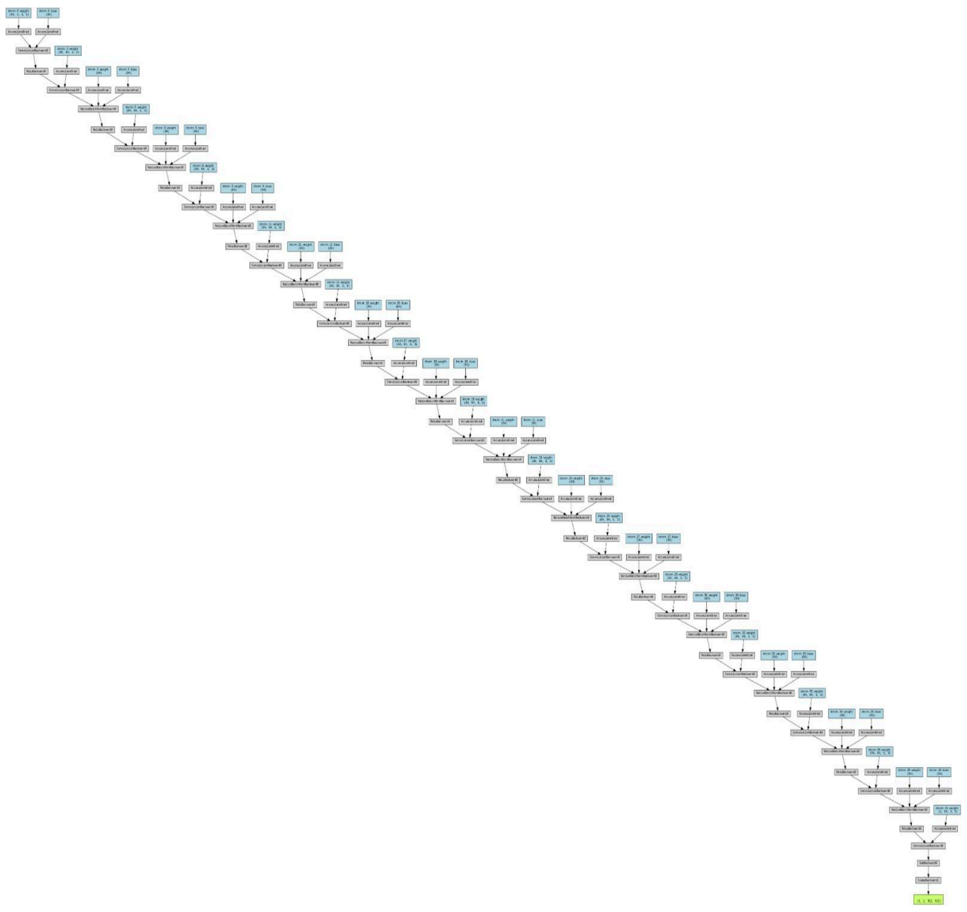
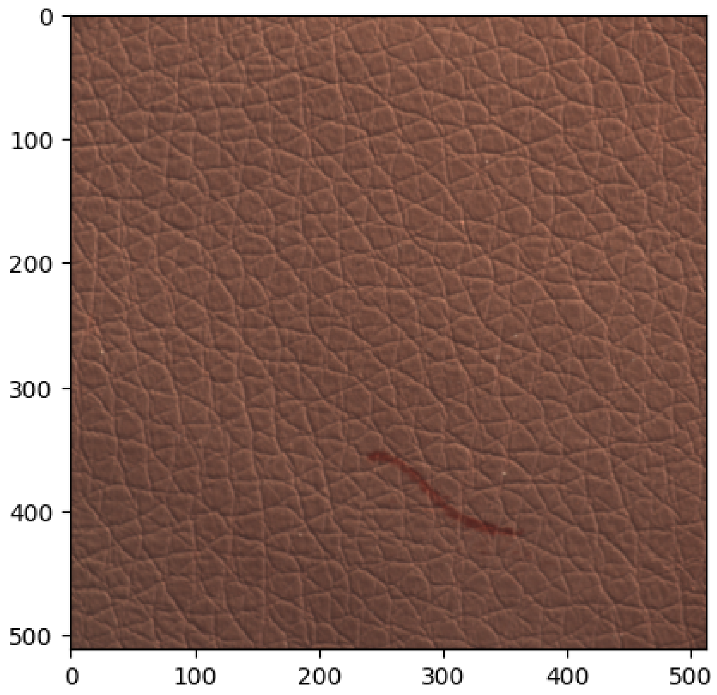


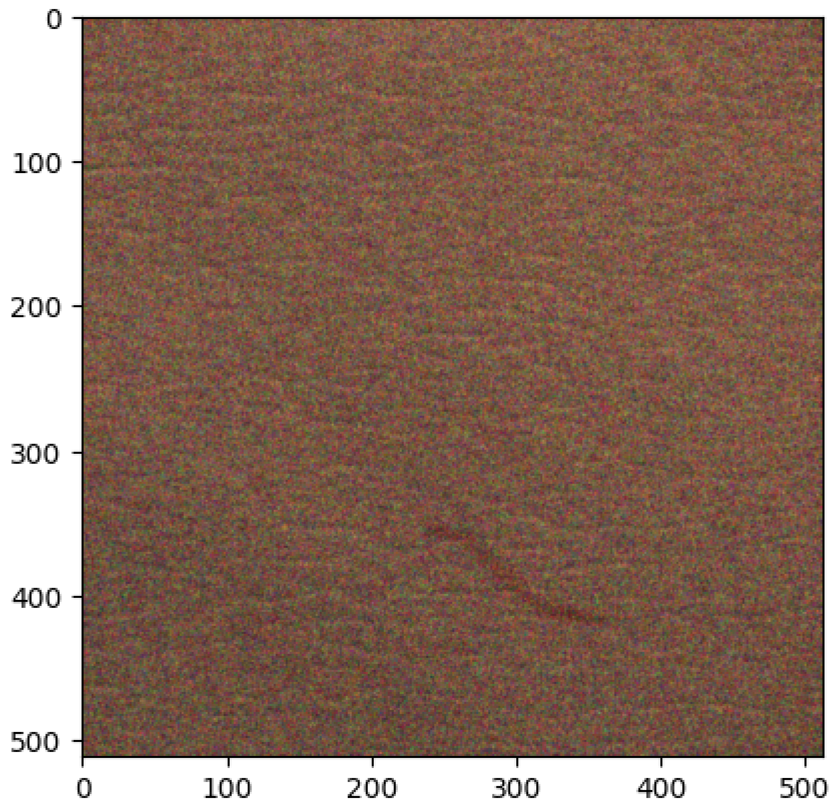
Image architecture generated using Torchviz library

Sample Image:

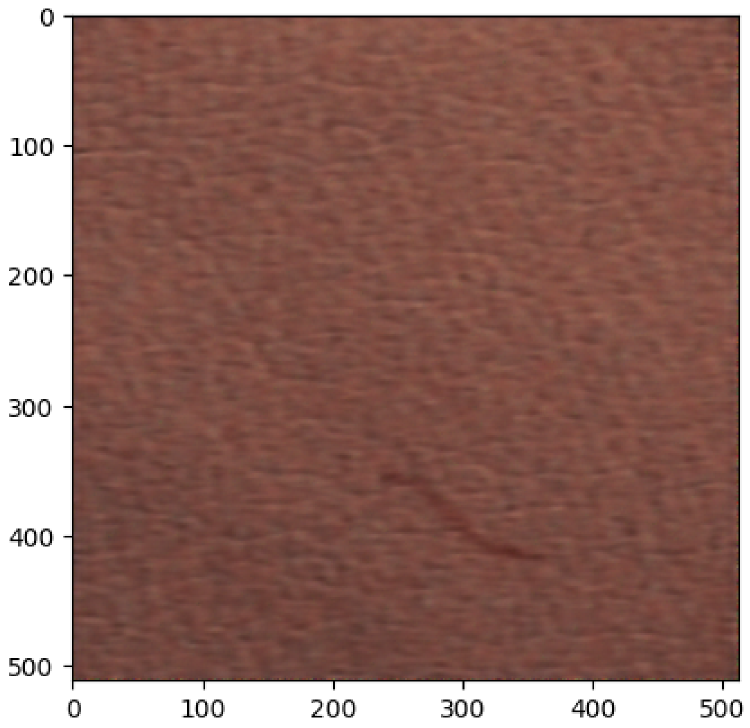
Ground Truth:



Denoised Image:



Denoised generated image:



Results:

Avg PSNR Value for validation: 28.3

Avg SSIM Value for validations 0.76

Class Wise PSNR:

```
psnr for bottle is  26.731958389282227
psnr for cable is  25.60096549987793
psnr for capsule is  27.70871925354004
psnr for carpet is  20.63148307800293
psnr for grid is  24.275985717773438
psnr for hazelnut is  30.310144424438477
psnr for leather is  27.201934814453125
psnr for metal_nut is  27.034360885620117
psnr for pill is  29.53520965576172
psnr for screw is  26.993492126464844
psnr for tile is  24.234760284423828
psnr for toothbrush is  29.037992477416992
psnr for transistor is  28.25482177734375
psnr for wood is  26.24666976928711
psnr for zipper is  28.36189079284668
```

END