Image Enhancement using Single Image Super Resolution (SISR)

Presenter: Daniel A. Masters, org 9358

Project Mentor: Tian J. Ma, org 6321

Sandia National Laboratories

Presenter Contact: damaste@sandia.gov

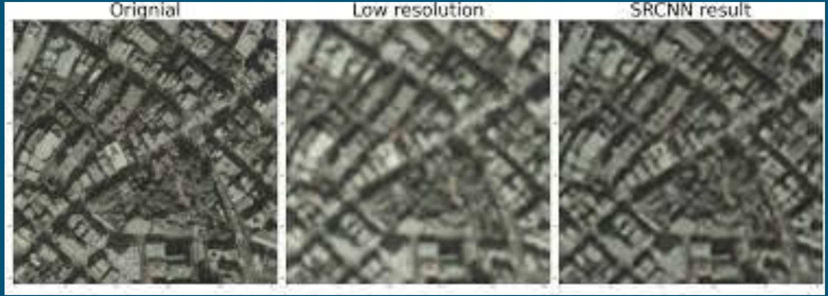
U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Date: 08/18/2023

Introduction

Goal: Developed Single Image Super Resolution (SISR) techniques to transform low-resolution satellite images into high-resolution



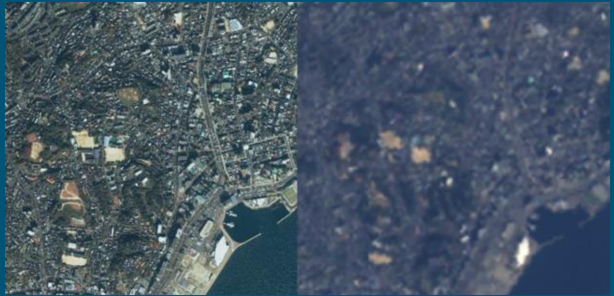
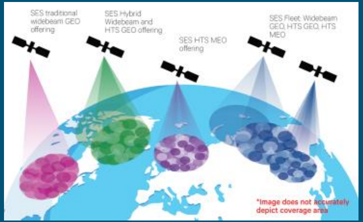
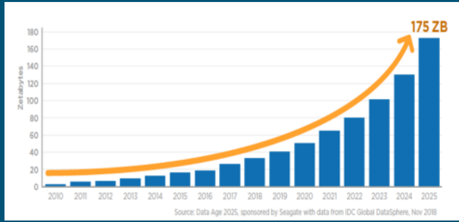


Figure 8: DIV2K learning (left) and the development formula for quadruple super-resolution images.

Motivation





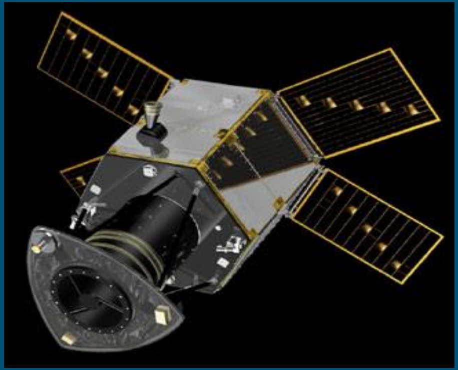
High-spatial-resolution remote sensing images vs Low-spatial-resolution remote sensing images:

- High-spatial-resolution images are more computationally expensive and requires more storage
- High-spatial-resolution images typically have smaller areas of coverage due to a decreased Field-of-View (FOV)

Transforming low-spatial resolution images into high-spatial-resolution enables us to overcome these challenges

Dataset

- Ultra High Definition (UHD) 3840x2160 video that was acquired by the International Space Station



Preprocessing

- The video was converted into 1000+ images (each frame is an .png image file)
- Used sliding window technique to get image patches of 960x540 resolution (cropping was performed on original image to get this resolution)
- Diagram of sliding window:

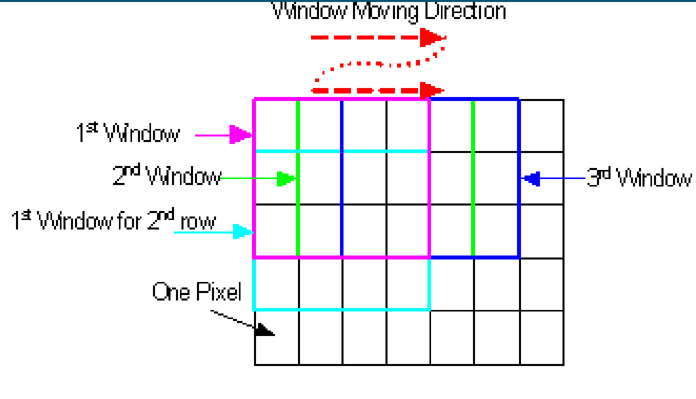


Image Credit: Miriam Leeser and Haiqian Yu

Training, Validation and Test Sets

- The training, validation and test sets will have 520 total random images
- Each of the 520 images is a 960x540 image patch resized to 320x180 and blurred using OpenCV's Gaussian Blur function
- The training, validation and test split is 60%, 20% and 20% respectively
- Hardware and memory limitations of our current PC have severely reduced the amount of usable image data on Jupyter Notebooks
  - PC specs: -CPU: i9-10850k -GPU: RTX-2080 -Memory: 32 GB DDR5 3600 Mhz
- Ideally, the goal would have been to split my training, validation and test data across all 13,000+ patches to get the most accurate model

Methodology: Transfer Learning

- Significant hardware and time constraints led us to transfer learning
- Transfer learning is a Machine Learning technique that involves taking a pre-trained model and fine-tuning it for a different task
- The pre-trained model selected was Enhanced Deep Residual Network for Single Image Super-Resolution (EDSR)
- EDSR is a type of deep learning model with the goal of generating a high-resolution image given low-resolution input which aligns with the project's goal
- This model is trained using any type of image data and ouputs an enhanced version 4x the input's resolution
- https://huggingface.co/eugenesiow/edsr-base

Hugging Face

Pre-trained Model Changes

- Since the pre-trained model was trained using unrelated images, re-training the model was necessary to get SR results that make sense in the context of our project
- Tailoring the TrainingArguments enabled the model to adjust to the specifications of our data
- The model was trained using the training set and evaluated using the validation set for 50 epochs
- Setup:

```
hf_edsr_model = EdsrModel.from_pretrained('eugenesiow/edsr-base', scale= 4)

if torch.cuda.is_available():
    torch.cuda.empty_cache()

train_args = TrainingArguments(
    output_dir = results_path,
    num_train_epochs = 50,
    per_device_train_batch_size = 1
)

class CustomTrainer(Trainer):
    def save_model(self, output_dir=None):
        if isinstance(self.model, nn.DataParallel):
            scale = self.model.module.config.scale
        else:
            scale = self.model.config.scale

trainer = CustomTrainer(
    model=hf_edsr_model,
    args=train_args,
    train_dataset=train_set,
    eval_dataset=valid_set
)
trainer.train()
```

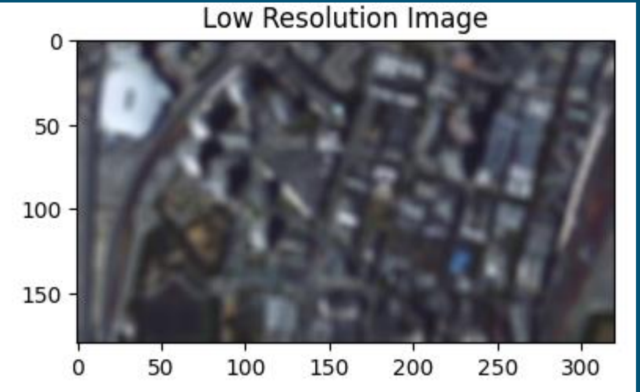
Training Evaluation

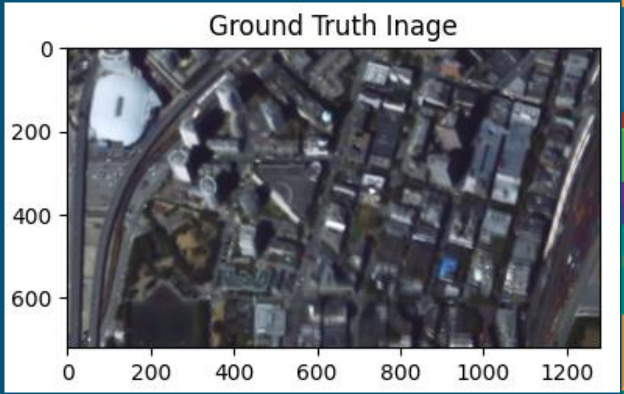
- Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM)
- There is no parameter or function to enable Early Stopping for the model, so monitoring was done for 50 epochs
- The evaluation score metrics fluctuated, but never had decreasing scores for 5 epochs in a row (threshold for determining overfitting)

```
epoch: 0/49: 100%   312/312 [00:32<00:00, 9.39it/s, loss=0.010219]
 scale:4      eval psnr: 44.45     ssim: 0.9669
 best epoch: 0, psnr: 44.453194, ssim: 0.966925
epoch: 1/49: 100%   312/312 [00:32<00:00, 9.52it/s, loss=0.007572]
 scale:4      eval psnr: 45.36     ssim: 0.9703
 best epoch: 1, psnr: 45.357117, ssim: 0.970326
epoch: 2/49: 100%   312/312 [00:32<00:00, 9.55it/s, loss=0.006856]
 scale:4      eval psnr: 47.23     ssim: 0.9777
 best epoch: 2, psnr: 47.233200, ssim: 0.977708
epoch: 3/49: 100%   312/312 [00:32<00:00, 9.62it/s, loss=0.006430]
```

Model Evaluation

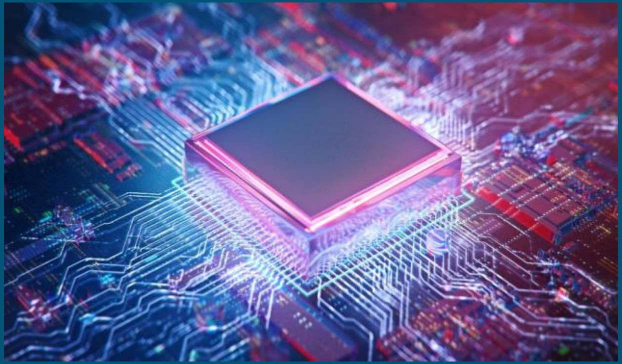
- After the 50 epochs have completed, the test dataset is fed into the model
- PSNR and SSIM scores are collected based on the super-resolved image and the ground-truth images
- The average PSNR score is 39.66 and the average SSIM score is .973, both are high scores and show that the super-resolved image is high quality and similar to its ground truth image





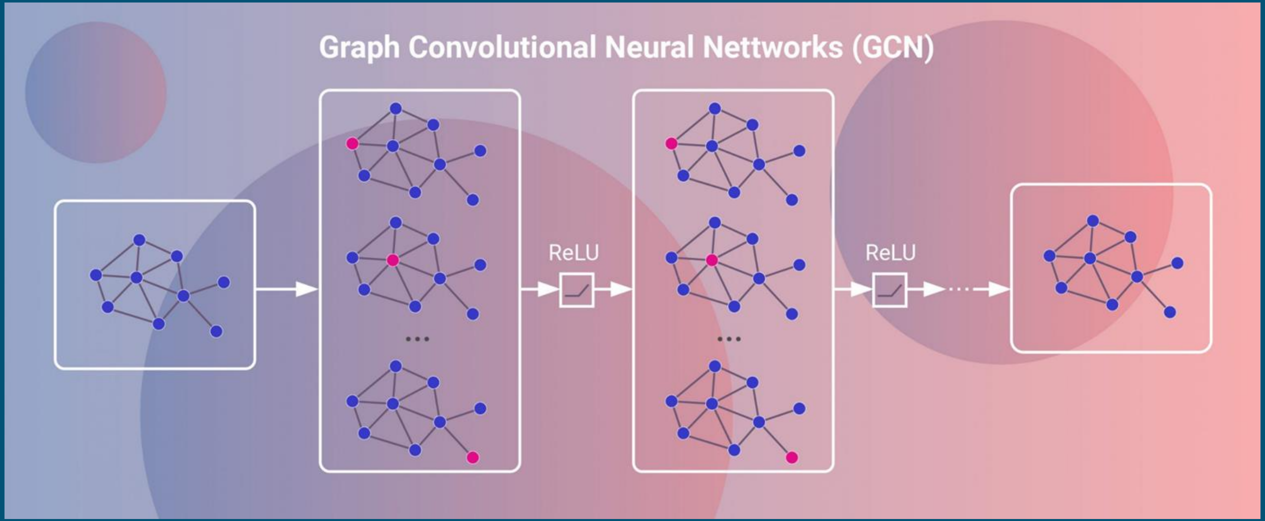
Conclusion

- Cropped and resized images were used to train the EDSR model due to memory limitations
  - With better hardware we expect to achieve similar performance on larger sized images
- The pre-trained EDSR model produced promising results
  - 4x resolution of low-resolution images
  - High average PSNR score (high quality)
  - High average SSIM score (minimal loss of information)



Future Work

- Develop a customized approach to SISR for satellite images using a Graph Neural Network (GNN)
  - A GNN is useful for highlighting spatial and relational structures which are especially important features when dealing with satellite imagery
  - GNNs concentrate computation efforts on graph structure, which can offer higher efficiency versus Deep Learning models that process entire grids of pixels



Back Up Slides

Using CNN for Feature Extraction

-defined the CNN architecture for feature extraction:

```
#defining CNN architecture for feature extraction
cnn_model = Sequential()
#output will have 32 feature maps, kernel size = 3x3, input shape = 128x128, padding meaning same out dimen as input dimen
cnn_model.add(Conv2D(16, (3,3), input_shape = (patch_size[0], patch_size[1], 3), padding='same', activation = 'relu'))
#add max pooling to reduce the number of parameters, identifies the essential feautes while discarding irrelevant features
cnn_model.add(MaxPooling2D(pool_size=(2,2)))
cnn_model.add(Conv2D(32, (3,3), activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))
cnn_model.add(Flatten())
```

-This will produce features which are high-level representations of the LR patches provided by the sliding window function. These extracted features will be used in conjunction with an adjacency matrix in the Graph Convolutional Network (GCN)

-The total number of features extracted was 30,752

Creating the Graph

-The graph will be created using an adjacency matrix

-Each row represents a node and each node represents an image patch in this adjacency matrix

-Each row will start off with 500 zeroes; ones will be inserted in the column whose index matches that of its nearest neighbor

-This process will be repeated 500 times (until all patches are accounted for)

-Found the five nearest neighbor patches for each patch using SSIM to determine the similarity between each patch

-This process was computationally expensive and required parallel processing using a Nvidia GPU and Nvidia's CUDA to speed up the process

```
For patch #0 the neighbors are [455, 927, 72, 670]
For patch #1 the neighbors are [361, 467, 797, 482]
For patch #2 the neighbors are [281, 212, 174, 372]
For patch #3 the neighbors are [974, 195, 677, 476]
For patch #4 the neighbors are [603, 632, 723, 195]
For patch #5 the neighbors are [847, 48, 622, 792]
For patch #6 the neighbors are [715, 47, 930, 852]
For patch #7 the neighbors are [755, 988, 472, 964]
For patch #8 the neighbors are [487, 163, 835, 896]
For patch #9 the neighbors are [471, 510, 599, 645]
For patch #10 the neighbors are [933, 84, 301, 802]
For patch #11 the neighbors are [403, 884, 585, 686]
For patch #12 the neighbors are [842, 707, 872, 708]
For patch #13 the neighbors are [69, 793, 251, 851]
For patch #14 the neighbors are [196, 430, 40, 896]
For patch #15 the neighbors are [856, 692, 641, 988]
For patch #16 the neighbors are [798, 225, 453, 353]
For patch #17 the neighbors are [960, 105, 504, 381]
For patch #18 the neighbors are [282, 469, 112, 159]
```

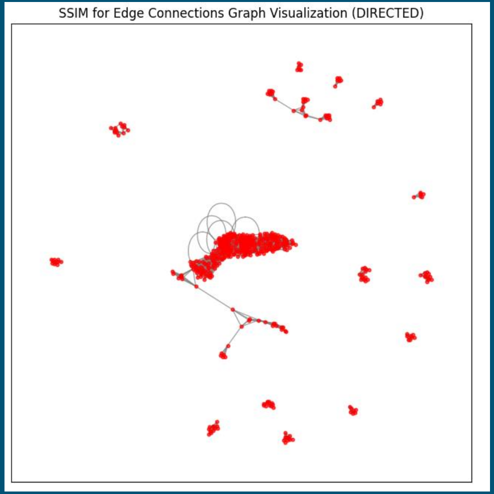
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0]
```

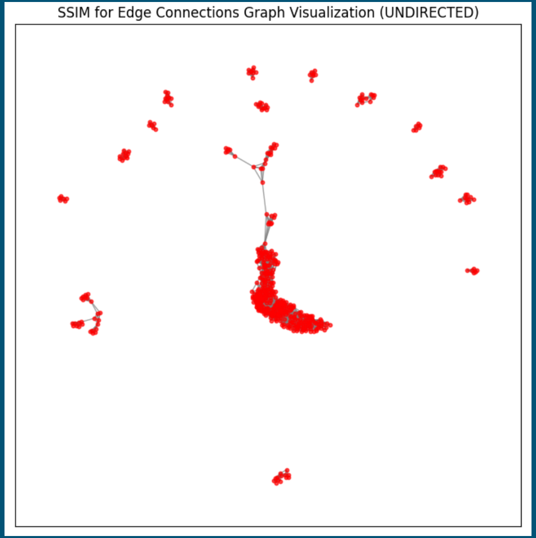
Establishing Connections

-Red = Nodes, Gray = Connections

-It is important to make sure that when edge connections are established between nodes, it is done so in an undirected fashion. For example if node A is a nearest neighbor to node B then node B is also a nearest neighbor to node A

-Note the difference in the visualizations above





PCA

-Since 30,000+ features is a lot for a GCN to handle computationally, PCA was applied to reduce the dimensionality while retaining important information

-The dimensionality was reduced to about 250 components and was fed to the GCN along with the adjacency matrix

```
desired_comps = 250
pca = PCA(n_components = desired_comps)
flat_feats = features.reshape(features.shape[0], -1)
pca_feats = pca.fit_transform(flat_feats)
print(pca_feats)
```

```
[[-1.3200360e+02  6.9363153e+02 -2.5443048e+02 ... -2.4106460e+00
   1.4968892e+00 -3.1187487e+00]
 [-9.2295490e+02 -2.0063677e+01 -1.7901289e-01 ...  1.1567141e+00
   4.0291435e-01  7.6119912e-01]
 [-7.6337604e+02 -3.7152846e+00 -4.8653989e+00 ...  3.0643148e+00
  -1.2383454e+00 -3.8060191e+00]
 ...
 [ 1.3654423e+03  2.6666302e+02 -3.7248633e+02 ...  3.3632192e-01
  -1.2024555e-01 -2.5555024e-01]
 [-9.0698688e+02 -2.7368935e+01 -1.7750207e+00 ...  1.6094997e+00
  -3.6126205e-01 -1.2243750e+00]
 [-8.6935815e+02 -7.4822607e+00  5.0226059e+00 ...  1.4625258e+00
  -7.3219371e-01  2.6475701e-01]]
```

Creating the GCN

-The adjacency matrix produced by KNN using SSIM as well as the features extracted from the CNN are used as input for the GCN

```
class GCN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(GCN, self).__init__()
        self.layer1 = nn.Linear(input_dim, hidden_dim)
        self.layer2 = nn.Linear(hidden_dim, hidden_dim) # 2nd GCN layer (hidden to hidden)
        self.layer3 = nn.Linear(hidden_dim, output_dim)
        self.skip1 = nn.Linear(input_dim, hidden_dim) # skip connection for layer 1
        self.skip2 = nn.Linear(hidden_dim, hidden_dim) # skip connection for layer 2

    def forward(self, feats, adj_matrix):
        x1 = F.relu(self.layer1(feats) + self.skip1(feats)) # apply layer and add skip connection
        x1_agg = torch.matmul(adj_matrix, x1)

        x2 = F.relu(self.layer2(x1_agg) + self.skip2(x1_agg)) # apply second layer and add skip connection
        x2_agg = torch.matmul(adj_matrix, x2)

        out = self.layer3(x2_agg)
        return out
```

-This outputs Pytorch tensors which are a dense representation of the extracted features and adjacency matrix. This can now be passed through a decoder network to produce HR images

Upscaling the GCN Output with a Decoder

-Now using the output from the GCN feed it through a decoder called UpsampleNet

-Feed UpsampleNet LR patches and their corresponding ground truth images

-Make sure to train several and adjust number of epochs based on the amount of image data that is provided to the model

-Remember to use a validation set to adjust the model's hyperparameters

```
class UpsampleNet(nn.Module):
    def __init__(self):
        super(UpsampleNet, self).__init__()

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(3, 64, kernel_size=4, stride=2, padding=1),  # to 256x256
            nn.ReLU(),
            nn.Conv2d(64, 32, kernel_size=3, stride=1, padding=1),  # Keeping 256x256
            nn.ReLU(),
            nn.Conv2d(32, 3, kernel_size=3, stride=1, padding=1),  # Keeping 256x256 but changing to 3 channels
            nn.Sigmoid()
        )

    def forward(self, x):
        return self.decoder(x)
```

Evaluation of the Output

-Similar to what was done in the pre-trained model, input the test dataset into model

-After it outputs the super-resolved images, compare this to the ground truth images using average SSIM and PSNR scores

```
psnr_avg = sum(psnr_total) / len(psnr_total)
ssim_avg = sum(ssim_total) / len(ssim_total)
print('The average psnr score is: '+str(psnr_avg))
print('The average ssim score is: '+str(ssim_avg))
```

```
The average psnr score is: 39.66362904869935
The average ssim score is: 0.9727798800497115
```

Notes for Project Replication

-Developing a deep-learning architecture for SR requires state-of–the-art hardware, a deep understanding DL/CV and ample time for training large datasets

-One must also be well-versed with parallel-processing libraries and CUDA compatibility (i.e. Tensorflow and Pytorch GPU capabilities) to maximize performance