Image Super-Resolution

Preda Alexandru-Florin

1. Introduction

Image Super-Resolution is a fundamental problem in computer vision that aims to reconstruct a high-resolution (HR) image from a single low-resolution (LR) input. Super-Resolution has an important role in many practical applications, including video games, medical imaging, remote sensing, video enhancement, and compression.

In this project, the goal is to perform 4x super-resolution on RGB images, increasing their spatial resolution from 32x32 pixels to 128x128 pixels. The problem is formulated as a supervised task, where paired LR and HR images are provided during training. Model performance is evaluated using the Mean Squared Error (MSE).

Two different deep learning architectures are implemented and evaluated. The first model serves as a baseline that refines bicubic upsampling using a shallow convolutional network. The second and main model is a deep residual CNN that learns rich feature representations at low resolution and progressively upsamples them to the target resolution. Various training hyperparameters are explored to improve accuracy.

2. Dataset and Preprocessing

The dataset consists of paired low-resolution and high-resolution RGB images. The training set contains 7,000 image pairs, the validation set contains 1,250 image pairs, and the test set consists of 500 low-resolution images without ground-truth targets. Each low-resolution image has a spatial resolution of 32x32 pixels, while the corresponding high-resolution image has a resoltuion of 128x128, corresponding to a 4x upscaling factor.

All images are loaded in RGB format and converted to tensors using ***ToTensor()*** transformation, which scales pixels intensities to the range [0, 1]. No additional normalization (e.g., mean or standard deviation normalization) is applied.

To reduce generalization and reduce overfitting, joint data augmentation is applied during training. Specifically, random rotations by multiples of 90 degrees are used. These geometric transformations are applied consistenly to both the low-resolution and high-resolution images to preserve pixel correspondence.

3. Deep Learning Architectures

In this project, I implemented and evaluated two different convolutional neural network (CNN) architectures. The first model serves as a simple baseline, while the second model is a deeper residual network designed to achieve higher reconstruction accuracy. Both models take a low-resolution RGB image of size 32x32 as input and produce a super-resolved RGB image of size 128x128 as output.

3.1 Baseline Model: Bicubic Refinement Network

The baseline model follows a straightforward approach. First, the low-resolution is upsampled to the target resolution using bicubic interpolation, a fixed and non-learnable operation. This upsampled image is then passed through a superficial convolutional neural network that predicts a correction to refine the interpolated result.

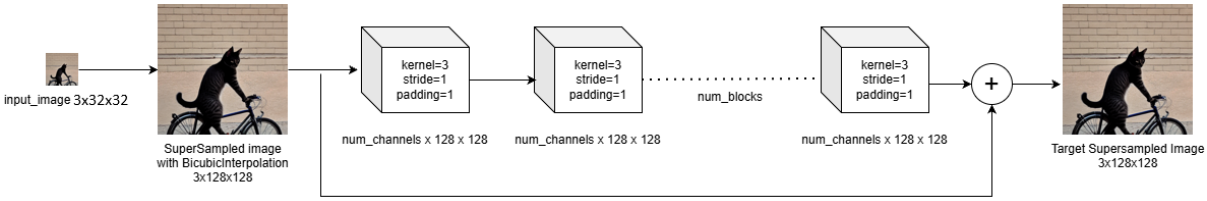

Fig. 1 Bicubic Refiner Architecture

3.2 Performance Model: Deep Residual CNN

The performance model is a deep residual convolutional neural network. Unlike the baseline approach, this architecture extracts its own features at low-resolution scale, which is both computationally efficient and performs better.

The network starts with a convolutional layer that maps the RGB input image to a higher-dimensional feature space. These features are processed by a sequence of residual blocks. Each residual block consists of two convolutional layers and a nonlinear activation (LeakyReLU), with a skip connection that preserves the input features.

After residual blocks, the network increases the spatial resolution using two upsampling blocks, each doubling the height and width of the feature maps. This results in the final representation at the target resolution of 128x128. A final convolutional layer maps these features back to tree channels to produce the super-sampled RGB image.

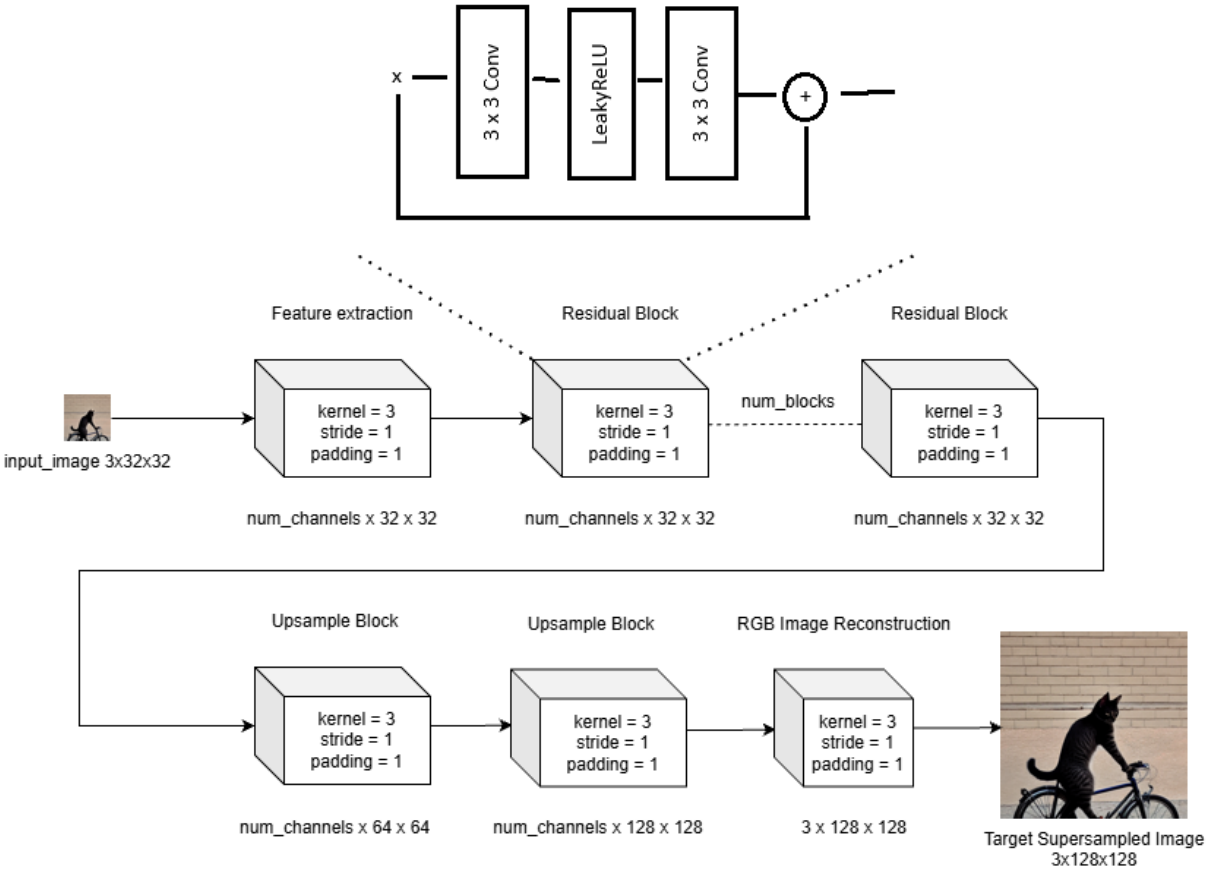

Fig. 2 Deep Residual CNN Architecture

4. Training Setup and Hyperparameter Tuning

4.1 Training Setup

All models were trained using the Mean Squared Error (MSE) loss function, and optimization was performed using the Adam optimizer.

To improve training efficiency, I used mixed-precision training with PyTorch Automatic Mixed Precision (AMP). The forward and backward passes were executed in half-precision where possible, while model parameters were maintained in a single precision. I used gradient clipping to limit the norm of gradients and improve the stability of the networks.

A cosine annealing learning rate scheduler was used to gradually reduce the learning rate. Early stopping based on the validation MSE was used to prevent overfitting and to reduce unnecessary computation.

4.2 Hyperparameter Tuning

To optimize the performance of each model, I focused on tuning tree key hyperparameters: learnin rate (lr), number of blocks (num_blocks) and number of channels (channels). The range of values tested are:

- Number of blocks: 8, 16 and 24
- Number of channels: 64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 712
- Learning rate: 1e-4, 3e-5, 5e-5

NAME 0 visualized	RUNTIME	CHANNELS	ETA_MIN	LR	NUM_BLOCKS	TRAIN/MSE	VAL/MS	VAL/PSNR	VAL/SSIM
plain_b24_c612	3h 26m 36s	612	1.0000e-7	0.0001	24	0.0040442	0.0049377	23.95097	0.7987
plain_b24_c548	2h 58m 16s	548	1.0000e-7	0.0001	24	0.0040667	0.0049556	23.93134	0.79838
plain_b24_c712_lr1e-4_e75	3h 55m 21s	712	0.000001	0.0001	24	0.0040242	0.0049609	23.93095	0.79836
plain_b24_c484	2h 19m 55s	484	1.0000e-7	0.0001	24	0.0041293	0.0049727	23.91911	0.79772
plain_b24_c712_lr7e-5_e75	4h 2m 23s	712	1.0000e-7	0.000075	24	0.0041991	0.0049908	23.8977	0.79691
plain_b16_c612	2h 41m 3s	612	1.0000e-7	0.0001	16	0.0041208	0.0049912	23.90152	0.79721
plain_b16_c548	2h 18m 52s	548	1.0000e-7	0.0001	16	0.0041447	0.0050028	23.88848	0.79677
plain_b16_c484	1h 47m 40s	484	1.0000e-7	0.0001	16	0.004298	0.0050182	23.87293	0.79595
plain_b24_c548	2h 18m 27s	548	1.0000e-7	0.0001	24	0.0044234	0.0050503	23.84261	0.79494
plain_b24_c712_lr5e-5_e75	4h 36m 59s	712	1.0000e-7	0.00005	24	0.0043964	0.005075	23.82427	0.79417
plain_b24_c484	1h 52m 34s	484	1.0000e-7	0.0001	24	0.0045119	0.0050811	23.81985	0.79402
plain_b8_c612	1h 54m 9s	612	1.0000e-7	0.0001	8	0.0043537	0.0050877	23.81542	0.79401
plain_b16_c548	1h 46m 7s	548	1.0000e-7	0.0001	16	0.0045368	0.0050957	23.80695	0.79334
plain_b8_c548	1h 38m 14s	548	1.0000e-7	0.0001	8	0.0044239	0.0050977	23.80179	0.79333
plain_b24_c420	1h 26m 16s	420	1.0000e-7	0.0001	24	0.004624	0.0051129	23.7895	0.79268
plain_b8_c896	1h 37m 57s	896	1.0000e-7	0.0001	8	0.0050318	0.0051184	23.77929	0.79162
plain_b16_c484	1h 26m 39s	484	1.0000e-7	0.0001	16	0.0046658	0.0051256	23.77439	0.79217
plain_b8_c484	1h 21m 51s	484	1.0000e-7	0.0001	8	0.0044828	0.005126	23.77803	0.79262
plain_b16_c420	1h 8m 22s	420	1.0000e-7	0.0001	16	0.0047584	0.0051656	23.74036	0.79085
plain_b8_c548	1h 17m 3s	548	1.0000e-7	0.0001	8	0.0048073	0.0052098	23.70215	0.78932

Fig. 3 Hyperparameter Tuning for Deep Residual CNN Network

The same approach was used to find the best hyperparameters for Deep Residual CNN Network and for Bicubic Refinement Network.

NAME 0 visualized	RUNTIME	CHANNELS	ETA_MIN	LR	NUM_BLOCKS	TRAIN/MSE	VAL/MSE	VAL/PSNR	VAL/SSIM
refiner_b16_c192	1h 43m 9s	192	0.000001	0.0001	16	0.0056239	0.0056823	23.29778	0.77344
refiner_b16_c256	2h 32m 43s	256	0.000001	0.0001	16	0.0057022	0.0057416	23.24432	0.77137
refiner_b16_c128	54m 21s	128	0.000001	0.0001	16	0.0059511	0.0059194	23.10406	0.76518
refiner_b8_c256	53m 35s	256	0.000001	0.0001	8	0.0058764	0.0059209	23.1051	0.76713
refiner_b8_c192	48m 31s	192	0.000001	0.0001	8	0.0059909	0.0059828	23.06085	0.76488
refiner_b8_c128	25m 37s	128	0.000001	0.0001	8	0.0062434	0.0061877	22.90475	0.75802
refiner_b8_c64	11m 9s	64	0.000001	0.0001	8	0.0066728	0.0065692	22.62245	0.74444
refiner_b24_c64	4m 50s	64	0.000001	0.0001	24	0.009531	0.0093338	21.05107	0.65997
refiner_b24_c192	19m 1s	192	0.000001	0.0001	24	0.009531	0.0093338	21.05107	0.65997
refiner_b24_c128	12m 37s	128	0.000001	0.0001	24	0.009531	0.0093338	21.05106	0.65997
refiner_b16_c64	3m 48s	64	0.000001	0.0001	16	0.009531	0.0093339	21.05106	0.65997

Fig. 4 Hyperparameter Tuning for Bicubic Residual Network

5. Evaluation Metrics and Results

5.1 Evaluation Metrics

I evaluated the model's performance using tree metrics: Mean Squared Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM).

Mean Squared Error (MSE) measures the average squared difference between the predicted and ground-truth pixel intensities. The lower the MSE value, the more accurate the reconstruction is.

Peak Signal-to-Noise Ratio (PSNR) is derived from MSE and is expressed in decibels. Higher PSNR values correspond to better reconstruction quality.

Structural Similarity Index Measure (SSIM) evaluates the image quality by considering luminance, contrast and structural information.

5.2 Results

The performance of the proposed deep residual model is compared against the baseline bicubic refinement network on the validation set

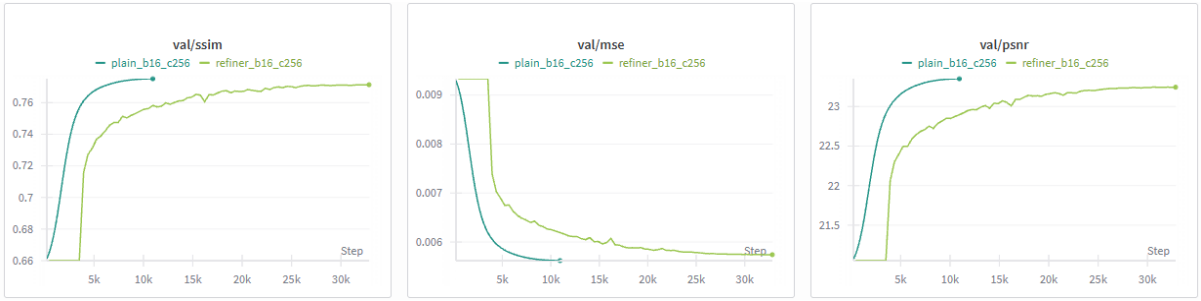

Fig. 5 Comparison results between Bicubic Refinement Network and Deep Residual Network

The results show that the proposed deep residual model has better performance compared to the baseline approach across all evaluation metrics using the same number of blocks and channels. It demonstrates the effectiveness of learning feature representations at low resolution and progressively upsampling them to the target resolution.

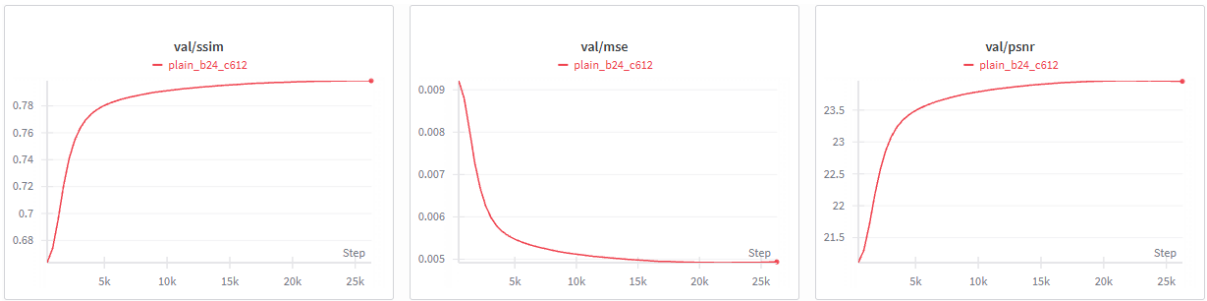

Fig. 6 Training progress of the most performance model

As seen in Fig. 6, the best performance model was the Deep Residual Network, with 24 blocks and 612 channels. It achieved **0.0049192** MSE score on validation dataset, **23.95 PSNR** and **0.79828** SSIM**.**

5.3 Impact of Hyperparameters

Analyzing the results of hyperparameter tuning, they indicates that increasing the number of feature channels leads to a consistent improvement in validation MSE, while increasing the depth provides smaller incremental gains beyond a certain point.

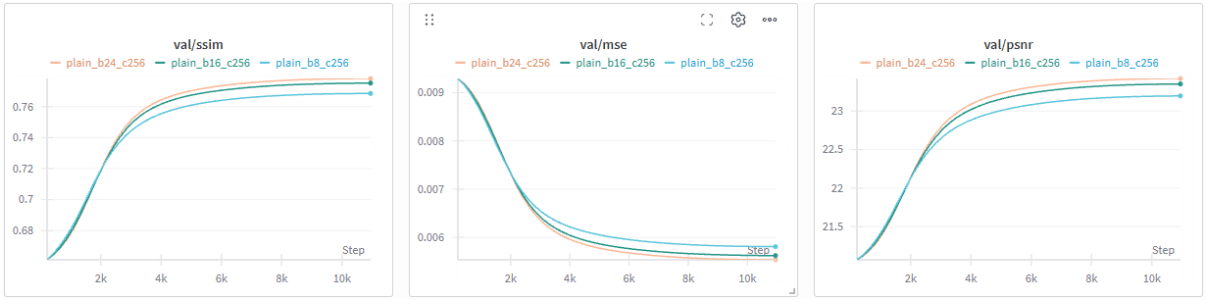

Fig. 7 Deep Residual Network trained on 256 channels with different number of blocks

We can observe a bigger improvement from 8 number of blocks to 16 number of blocks and a smaller improvement increasing to 24 number of blocks.

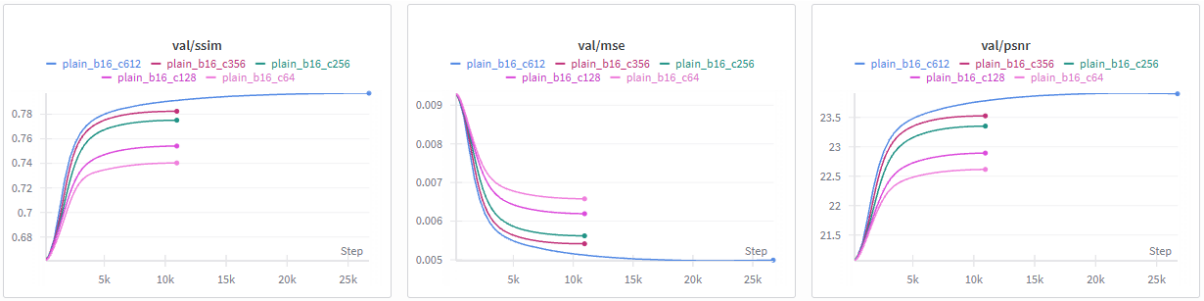

Fig. 8 Deep Residual Network trained on 16 number of blocks and different number of channels

We ca observe that increasing the number of channels, all validation metrics increases in proportion with the number of channels.

6. Conclusion

In this project, I implemented and compared two deep learning architectures Deep Residual Network and Bicubic Refinement Network for supersampling 32x32 images to 128x128. Each model was built from scratch using PyTorch. The models were trained and optimized through custom hyperparameter tuning, which systematically selected the best configurations based on validation loss.

Among the two architectures, Deep residual Network achieved the best results, with an **0.0049192** MSE, **23.95 PSNR** and **0.79828** SSIM on validation dataset.

For future improvements, using pretrained models and adding more channels and/or number of blocks could further enhance performance.