SRCNN Residual Family – Architecture Overview
===

This document summarizes the three scale-agnostic super-resolution refinement networks used in your project.
All models are fully convolutional and input-size agnostic: you can feed any H×W image or patch; the output
will have the same spatial size. The 128×128 size you saw in a printout was just an example input used for
a quick shape check. For training, we recommend patch-based sampling (e.g., 192×192 HR patches).

Design goals implemented:
- Residual learning with residual blocks (3×3) and residual-scaling (0.1) for stable deep training.
- Larger receptive field at the head (9×9) followed by 3×3 residual trunk.
- No BatchNorm (preserves range and fine details), PReLU activations.
- Optional lightweight channel attention (SE) in medium/high, applied every 2 blocks.
- Global skip connection (input→output) and optional clamp to [0,1].
- Architecture independent of scale factor (×2/×3/×4/×6...), matching your evaluation protocol.

Model Variants (Depth is the main complexity driver)

- SRCNN_low:
  - Channels: 64; Residual Blocks: 4; SE: none; Residual scaling: 0.1
  - Purpose: lightweight baseline, fast and stable on smaller datasets.

- SRCNN_medium:
  - Channels: 64; Residual Blocks: 10; SE: every 2 blocks; Residual scaling: 0.1
  - Purpose: balanced capacity vs. speed, a good mainline model for comparisons.

- SRCNN_high:
  - Channels: 64; Residual Blocks: 20; SE: every 2 blocks; Residual scaling: 0.1
  - Purpose: deeper trunk for harder scales or more texture recovery; still scale-agnostic.

Parameter counts (trainable):
- SRCNN_low   : 313,103 parameters
- SRCNN_medium : 759,523 parameters
- SRCNN_high   : 1,501,623 parameters

I/O Behavior and Scale-Agnostic Training

- The networks do not change the resolution: output H×W equals input H×W.
- To train for different upscale factors while keeping architectures identical:
  1) For each factor s, generate LR images by mod-cropping HR to be divisible by s, then downscale by s (bicubic) and bicubic-up back to HR size.
  2) Train the network to map bicubic-upscaled inputs to HR targets (residual refinement).
- Training patches: 192×192 (divisible by 2/3/4/6) is recommended.
- Evaluation on full images: apply mod-crop per scale so that dimensions are divisible by s.

Residual Block (used in all trunks)

- Structure: Conv(3×3, C) → PReLU(C) → Conv(3×3, C) → (optional SE) → Residual Add with scaling 0.1
- Residual scaling stabilizes optimization in deeper stacks and helps prevent exploding gradients.
- SE (Squeeze-and-Excitation) is applied in medium/high variants every second block.

Head & Tail

- Head: Conv(9×9, 3→64) + PReLU(64). A large kernel gives a strong initial receptive field.
- Tail: Conv(3×3, 64→3). Projects features back to RGB.
- Global skip: Conv(1×1, 3→3), added to the tail output, encourages residual learning in RGB space.

Training & Evaluation Tips

- Loss: L1 or Charbonnier for fidelity; optionally combine with perceptual or LPIPS for qualitative metrics.
- Optimizer: Adam/AdamW, initial LR 1e-4 with cosine decay; use gradient clipping (e.g., 1.0) and EMA of weights.
- Data: strong patch augmentation (flip/rotate), patch sizes 128–192; ensure reproducible bicubic interpolation.
- Metrics: PSNR/SSIM on Y-channel; report LPIPS for perceptual quality.
- Avoid BatchNorm; it can harm SR detail. Prefer PReLU/LeakyReLU activations.