RCAI: Machine Learning on Resources Constrained Devices

A presentation by
Jaideep Singh Heer (M20CS056)

Problem definition

Perform **Neural Architecture Search (NAS)** on a UNet neural network (**SR3**) to optimize for **minimal FLOPs and inference latency**; while working in a denoising diffusion generative framework (**DDPM**).

The task is image super-resolution on **DIV2K** dataset for **64x64->128x128** patches.

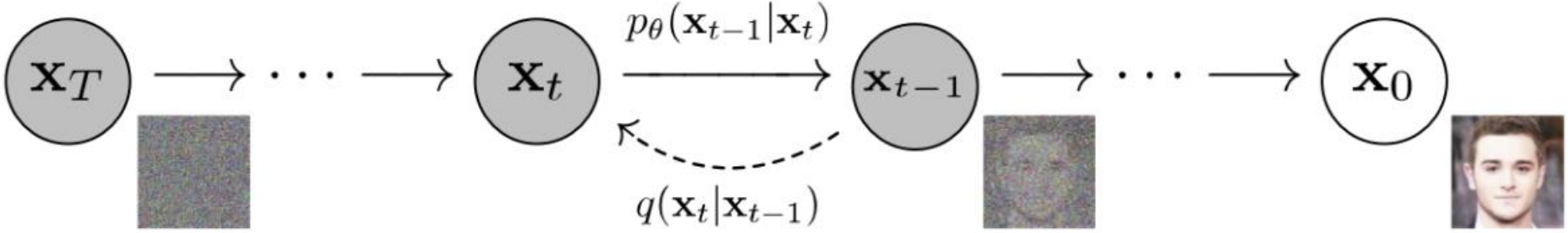


Figure 3.1: Forward and reverse diffusion process [4]

Problem definition

Inspired by the paper [Image Super-Resolution via Iterative Refinement](), this work adds NAS to the proposed SR3 UNet-type neural network, for reducing the FLOPs and latency of the super-resolution process.

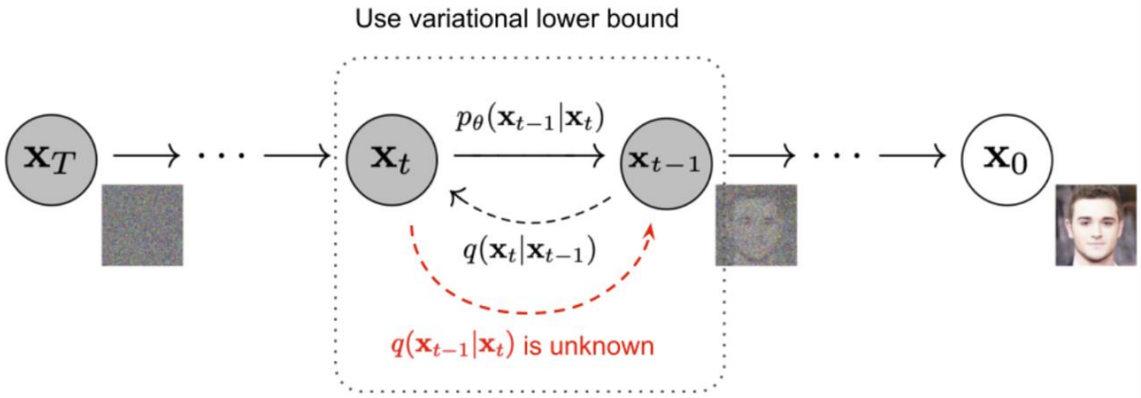


Fig. 2. The Markov chain of forward (reverse) diffusion process of generating a sample by slowly adding (removing) noise. (Image source: Ho et al. 2020 with a few additional annotations)

Source: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Here, the DDPM process is modeled as a Markov chain of **T** diffusion steps.

Here, **q** and **p** are forward and reverse diffusion transitions that add or remove noise to the image respectively.

The neural network SR3 is trained to predict the reverse transition **q** to re-generate the original image from pure Gaussian noise.

Problem definition

In the DDPM framework, the forward diffusion step **q** adds Gaussian noise to the image at step **t-1** to get the noisy image at step **t**. This noise is governed by hyper-parameters β_1 to β_T (inference schedule) for each diffusion step. When $\beta_{1:T}$ are fixed, the forward process can be reduced to a single formula,

$$
\begin{aligned}
q(\boldsymbol{y}_{1:T} \mid \boldsymbol{y}_0) &= \prod_{t=1}^{T} q(\boldsymbol{y}_t \mid \boldsymbol{y}_{t-1}) \,, \\
q(\boldsymbol{y}_t \mid \boldsymbol{y}_{t-1}) &= \mathcal{N}(\boldsymbol{y}_t \mid \sqrt{\alpha_t}\, \boldsymbol{y}_{t-1}, (1-\alpha_t)\boldsymbol{I}) \\
q(\boldsymbol{y}_t \mid \boldsymbol{y}_0) &= \mathcal{N}(\boldsymbol{y}_t \mid \sqrt{\gamma_t}\, \boldsymbol{y}_0, (1-\gamma_t)\boldsymbol{I}) \\
\text{using the notation } \alpha_t &:= 1 - \beta_t \text{ and } \gamma_t = \prod_{i=1}^{t} \alpha_i
\end{aligned}
$$

Source: [Denoising Diffusion Probabilistic Models]

Problem definition

For the reverse diffusion process, one can simply rearrange the forward diffusion equation to get the reverse diffusion equation. However notice that the reverse diffusion requires the original image x_0.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}),$$

$$\text{where} \quad \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad \text{and} \quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

$$\text{using the notation } \alpha_t := 1 - \beta_t \text{ and } \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

Source: [Denoising Diffusion Probabilistic Models]

To overcome this issue, the DDPM paper authors propose using a neural network f_θ to predict the Gaussian noise that is added/subtracted in transition t -> t-1.

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \tilde{\boldsymbol{\mu}}_t\left(\mathbf{x}_t, \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t} f_\theta(\mathbf{x}_t))\right) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\mathbf{x}_t, t)\right)$$

Problem definition

Applying the previous slides' formulae, the DDPM training and inference algorithms become as follows.

Algorithm 1 Training a denoising model f_θ

1: **repeat**
2: $(\boldsymbol{x}, \boldsymbol{y}_0) \sim p(\boldsymbol{x}, \boldsymbol{y})$
3: $\gamma \sim p(\gamma)$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take a gradient descent step on
$\nabla_\theta \left\| f_\theta(\boldsymbol{x}, \sqrt{\gamma}\boldsymbol{y}_0 + \sqrt{1-\gamma}\boldsymbol{\epsilon}, \gamma) - \boldsymbol{\epsilon} \right\|_p^p$
6: **until** converged

Source: Image Super-Resolution via Iterative Refinement

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} f_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Source: Denoising Diffusion Probabilistic Models

Note that here the noisy image is: $\sqrt{\gamma}\boldsymbol{y}_0 + \sqrt{1-\gamma}\boldsymbol{\epsilon}$

Methodology

The model f_θ is a UNet architecture with skip connections which is given a 6x128x128 tensor and the γ noise level. The input tensor is the low-resolution 64x64 image upscaled to 128x128 using bicubic upscaling and concatenated with pure Gaussian noise at the channel dimension in the first iteration.

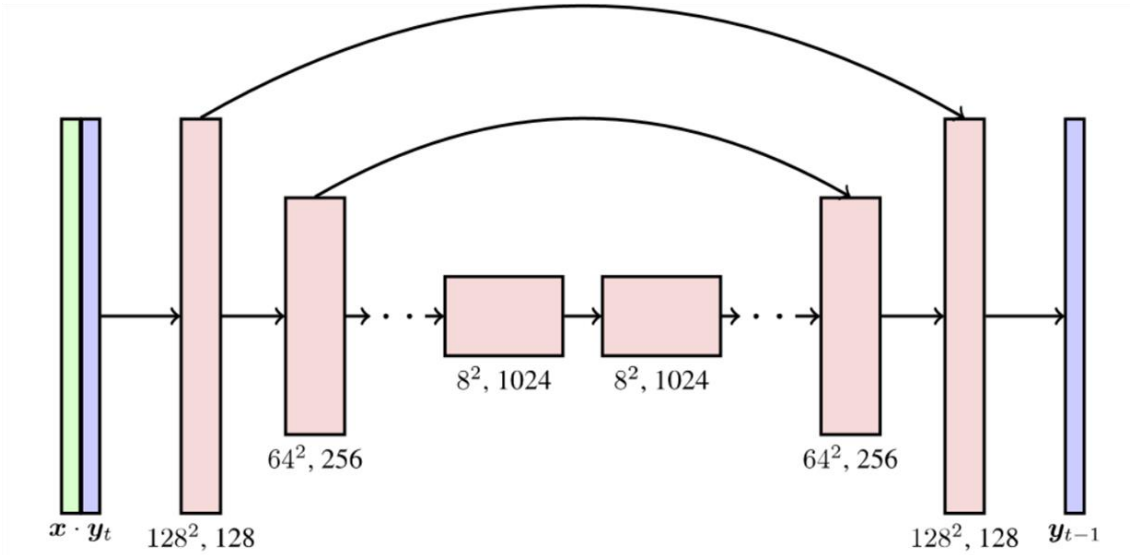
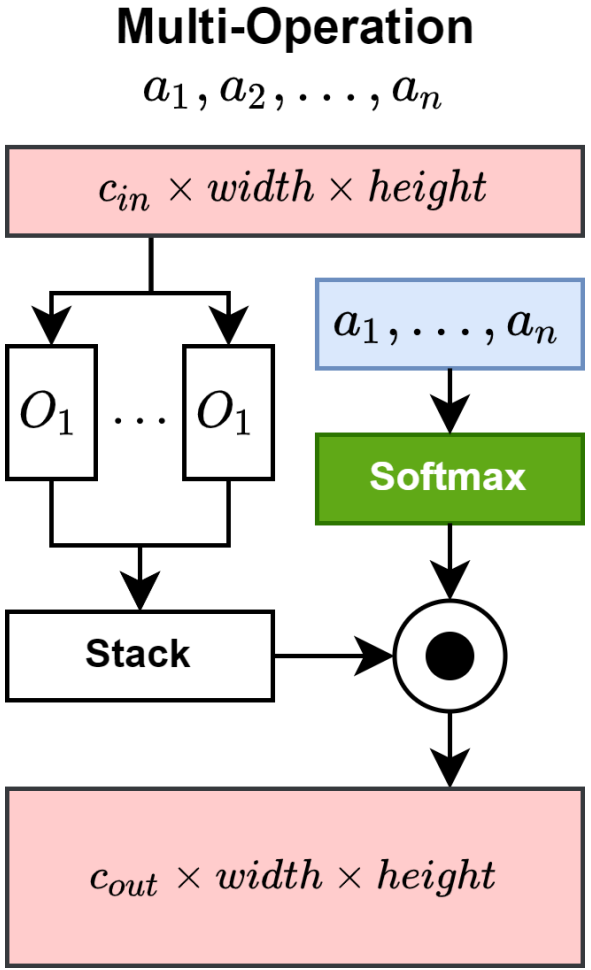


Figure 4.1: UNet model with skip connections showing image-size, channels per UNet block and input x and y_t concatenated to predict y_{t-1} [4]

Source: [Image Super-Resolution via Iterative Refinement]

Methodology

To apply NAS to the UNet model, a **DARTS** like approach is adopted. A Multi-Operation module performs different operations on the same input and scales their output using **Softmax** of its own learnable parameters. There parameters are called the network's **architecture parameters** since they determine which operation's output is dominant in the final output.



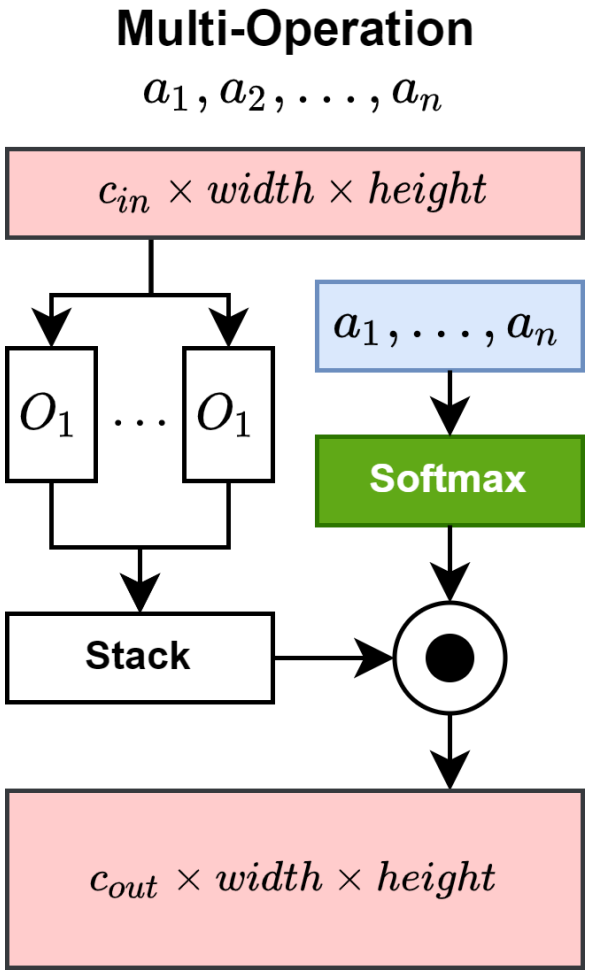
Methodology

The neural Architecture Search (NAS) is performed by first replacing some of the SR3 UNet's blocks by Multi-Operations containing multiple operation blocks to create a **supernet**.

Thi supernet is then trained to convergence and **sub-network** is extract from it by replacing all multi-operation blocks with its corresponding operation block that has the largest Softmax value of the architecture parameters.

This approach is equivalent to the DARTS sub-network extraction approach.

The sub-network is then fine tuned on the training data to obtain the final searched neural network.



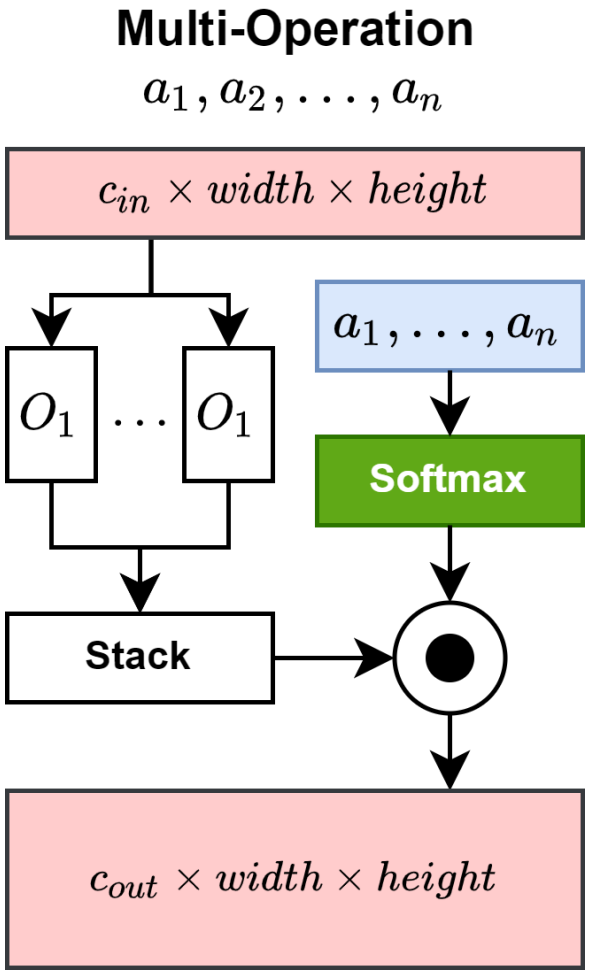
Methodology

The neural Architecture Search (NAS) is performed by first replacing some of the SR3 UNet's blocks by Multi-Operations containing multiple operation blocks to create a **supernet**.

Thi supernet is then trained to convergence and **sub-network** is extract from it by replacing all multi-operation blocks with its corresponding operation block that has the largest Softmax value of the architecture parameters.

This approach is equivalent to the DARTS sub-network extraction approach.

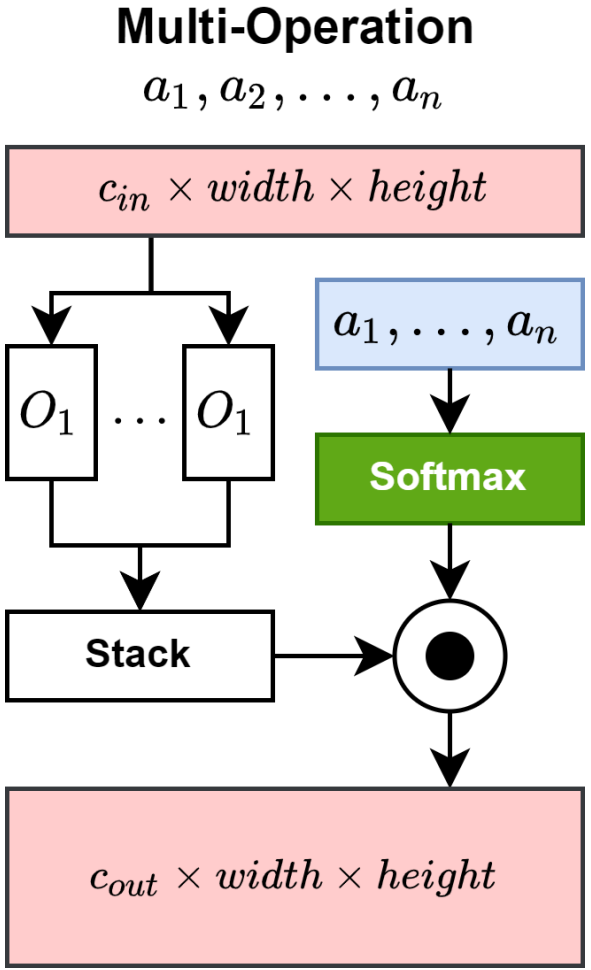
The sub-network is then fine tuned on the training data to obtain the final searched neural network.



Methodology (NAS Search Space)

This work focuses on comparing the resulting models searched using three different NAS techniques by using the following activation functions on the architecture parameters,

- Simple softmax
- Softmax with temperature
- Gumbel-Softmax



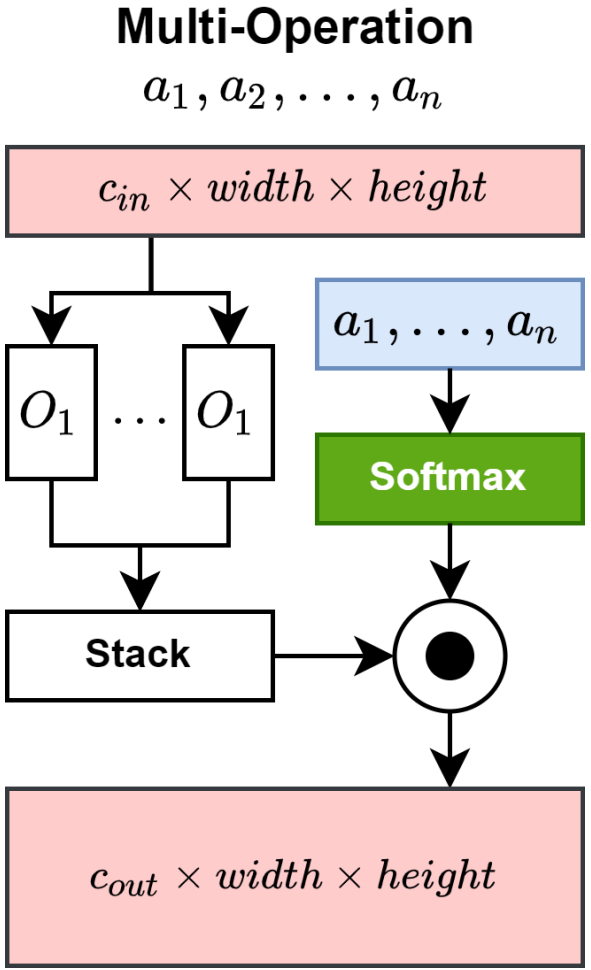
Methodology (NAS FLOPs/latency)

To optimize for FLOPs and latency, this work implements custom modules to allow propagating the FLOPs and latency costs of every module/operation used in the UNet.

This propagation is done by simply accumulating (by addition) these metrics as they are propagated through the network.

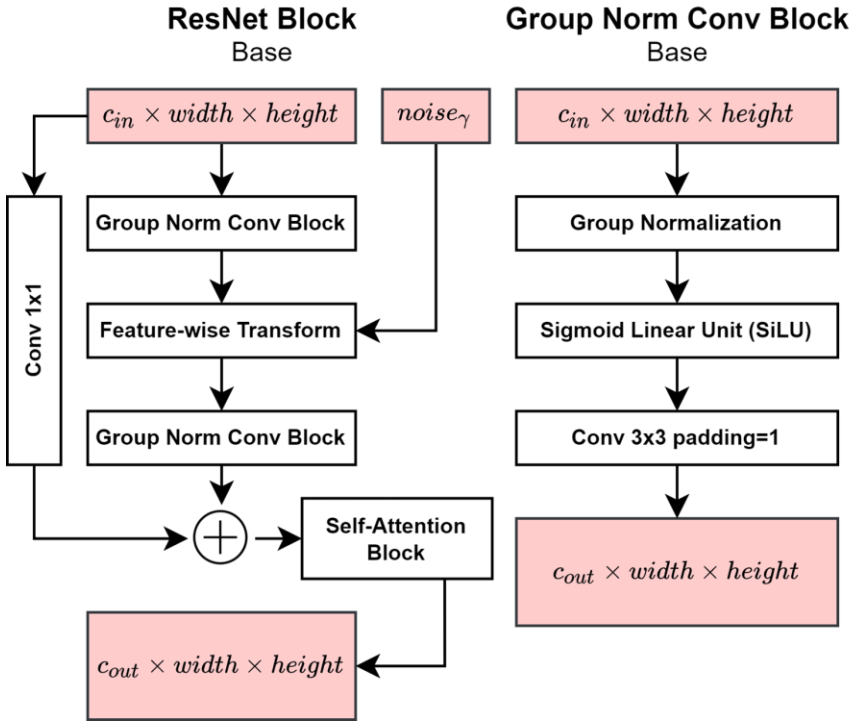
The Multi-Operation block however scales these metrics using the architecture parameters allowing the backpropagation process to use these propagated metrics to perform gradient descent on the architecture parameters only.

This method is inspired by: [AutoGAN-Distiller: Searching to Compress Generative Adversarial Networks]



Methodology (UNet)

Every UNet layer has sequential ResNet blocks. This image shows the structure of the ResNet blocks and the Group-normalisation Convolution blocks inside them.

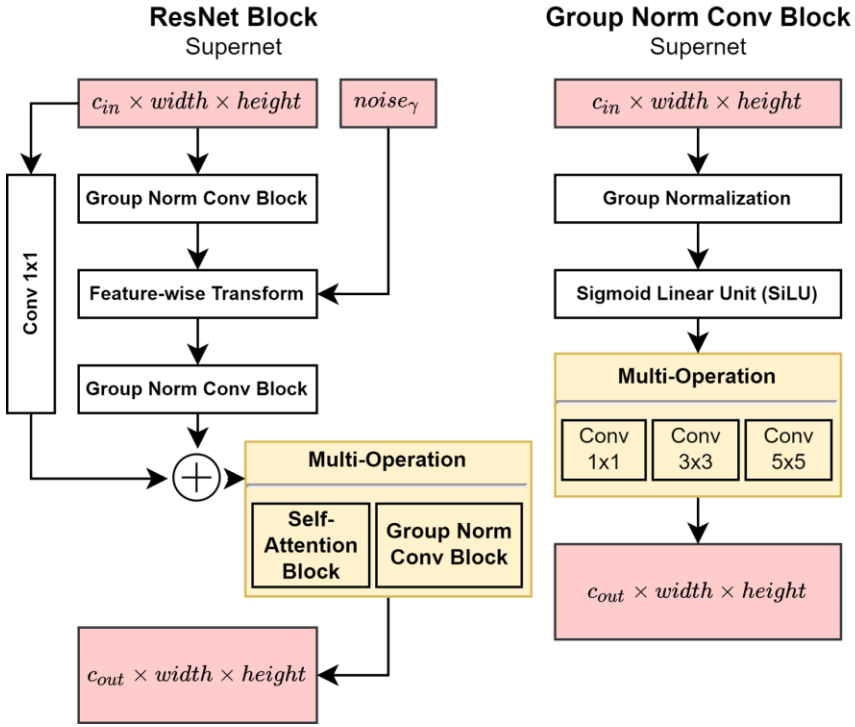


Methodology (NAS + UNet)

To apply NAS to UNet, the following UNet blocks are replaced by Multi-Operation blocks.

Since each Group-norm Conv. block has 3 operations and a ResNet block can have a Group-norm Conv. block, the total no. of possible sub-networks for a ResNet layer with attention becomes 3*3*(1+3) = 34.

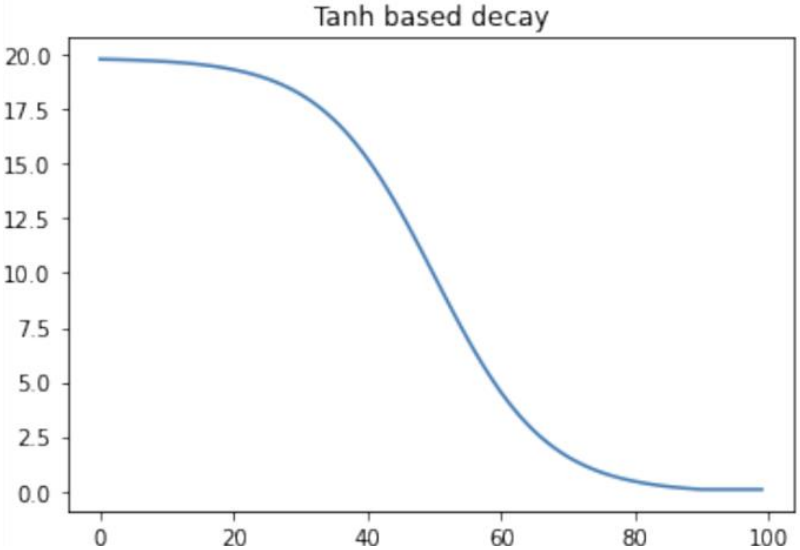
If self-attention is disabled then the ResNet block has 3*3 = 9 sub-networks.



Methodology (NAS + Temperature decay)

Both Softmax with temperature and Gumbel-Softmax have a property called temperature. This property affects the behaviour of the softmax function in a large extent and generally makes the softmax output more categorical as the temperature drops.

This work, therefore implements a temperature decay function to allow the softmax functions to aggressively prefer operations as the training reaches its end.



$$temperature(x) = \left(1 - tanh(\frac{3 * (x - \frac{width}{2})}{\frac{width}{2}})\right) * \frac{max - min}{2} - min$$

Training Hyper-Parameters

Table 4.1: Training pipeline common hparams

Item	hparam	Value
Model training engine	lr	3e-06
Architecture training engine	lr	3e-05
Architecture training engine	temperature	Eq. 8 create-tanh-decay(max=20,min=0.08)
Architecture training engine	flops-loss-scaling	5e-17
Architecture training engine	latency-loss-scaling	5e-5
All train engines	optimizer	torch.optim.Adam
All train engines	loss	L1Loss(reduction=mean)
All engines	T	2000
All engines	$\beta_1, ..., \beta_T$	torch.linspace(start=1e-06, end=0.01, steps=2000)
Data-loader	batch-size	8
Data-loader	crop-patch-size	128
Data-loader	up-scaling	bi-cubic
Data-loader	dataset	DIV2K-unknownx2
Task	super-resolution	64x64 to 128x128
Model	start-channels	64
Model	channel-multipliers	1, 2, 4, 8, 16
Model	attention-from-depth	3
Model	resnet-blocks-per-unet-layer	1

Experimental findings

The following table shows the results of the three models found by NAS in comparison to the base SR3 model, all trained with the sams hyper-parameters from the last slide.

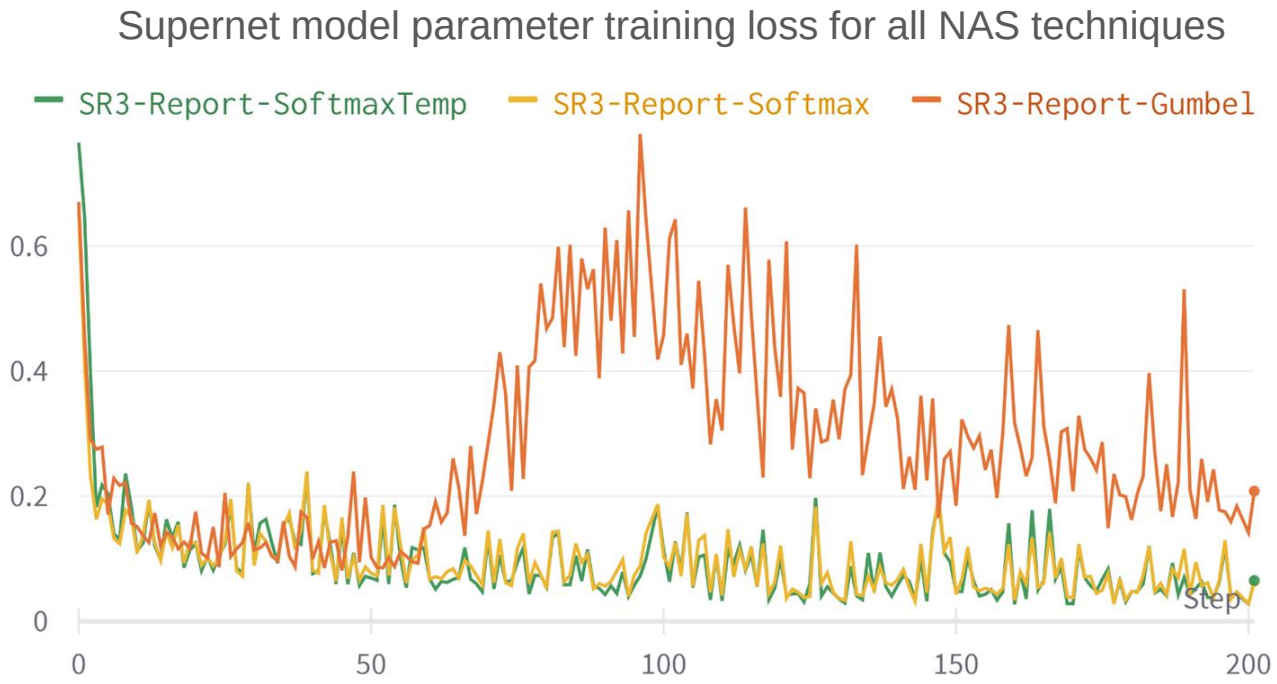
It shows that Simple softmax provides a good tradeoff between FLOPs/latency and number of parameters.

The table also shows a discrepancy between FID and PSNR scores which can be explored in further studies.

Table 5.1: NAS Model comparison

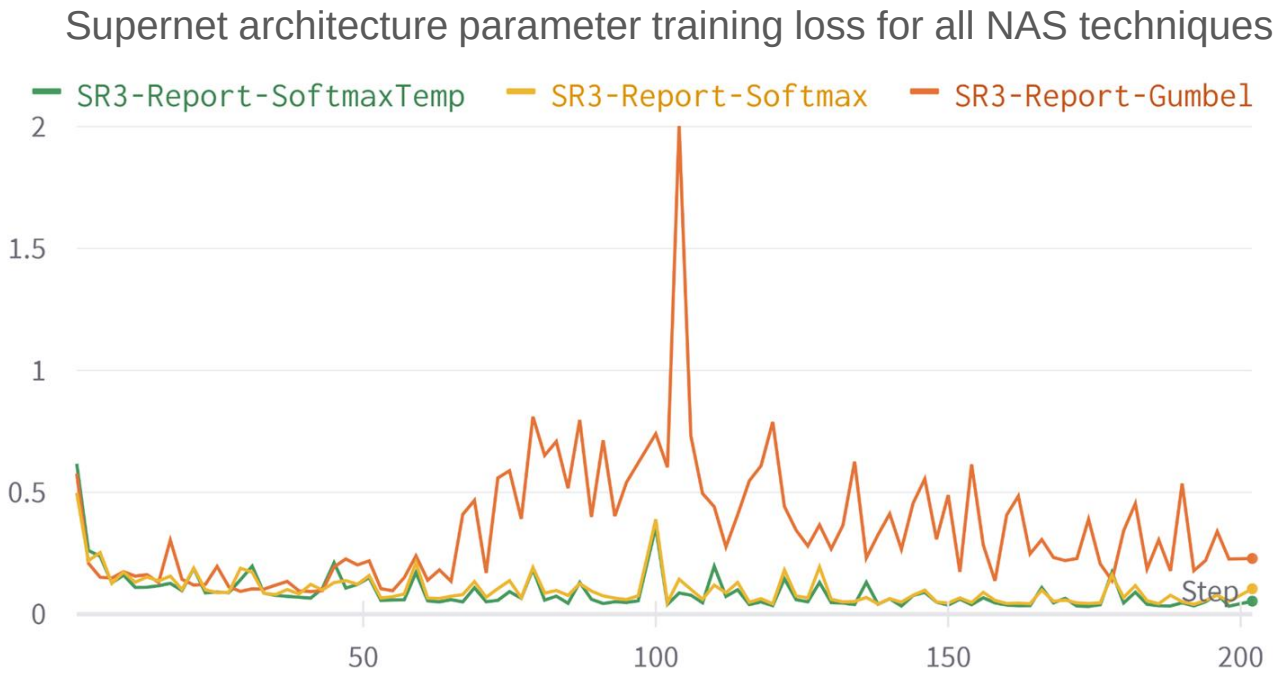
Model	NAS-Softmax	Terra FLOPs ↓	Latency ↓	FID ↓	PSNR ↑	No. params ↓
Base SR3	-	140.32	31.23	0.54	20.84	155,147,779
NAS Supernet SR3	-	841.44	326.11	-	-	475,298,249
NAS Subnet SR3	Simple Softmax	134.92	42.62	0.60	20.24	133,100,803
NAS Subnet SR3	Temperature Softmax	213.39	63.91	0.44	13.41	147,061,507
NAS Subnet SR3	Gumbel-Softmax	184.62	78.35	0.32	10.75	110,851,843

Experimental findings (Plots)



Here we see that Gumbel-Softmax NAS becomes unstable as the function temperature drops, while Softmax with temperature NAS performs similar to Softmax NAS.

Experimental findings (Plots)



Again, the plots look similar for Gumbel-Softmax NAS as the temperature drops.

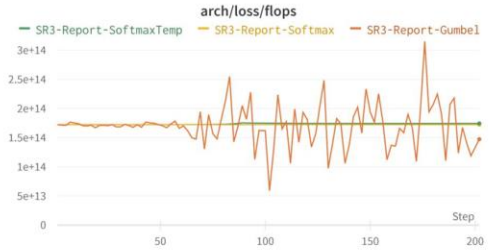
Experimental findings (Plots)

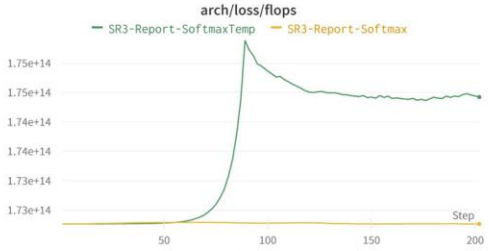
Propagated FLOPs and latency values for all NAS techniques.

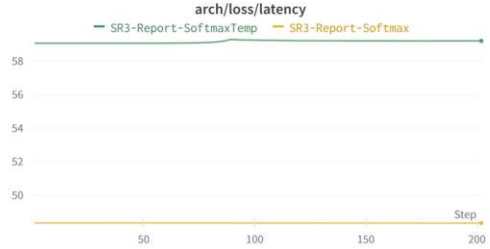
- Gumbel-Softmax (Orange)
- Softmax with temperature (Green)
- Simple softmax (yellow)

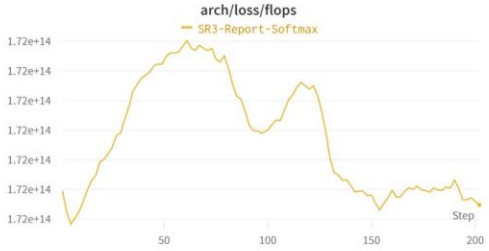
These plots show that the propagated metrics also start varying greatly as the temperature drops for Gumbel-Softmax.

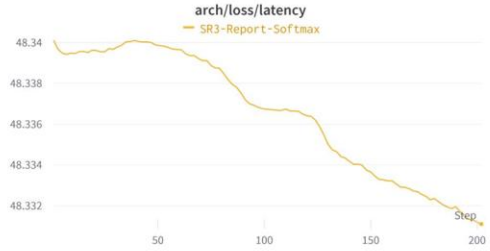
This means that the Gumbel distribution sampling plays a bigger role in the Gumbel-Softmax output as the temperature drops thus introducing more randomness in the operation selection.

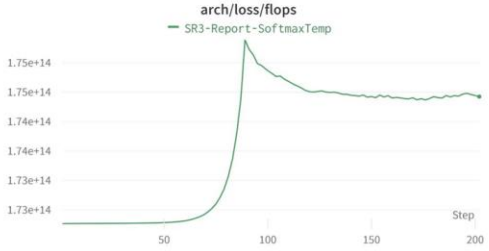


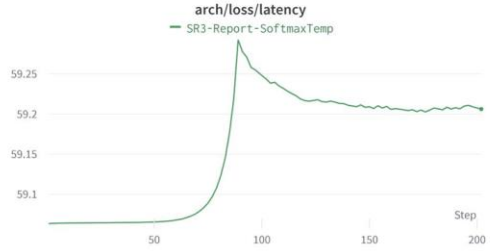






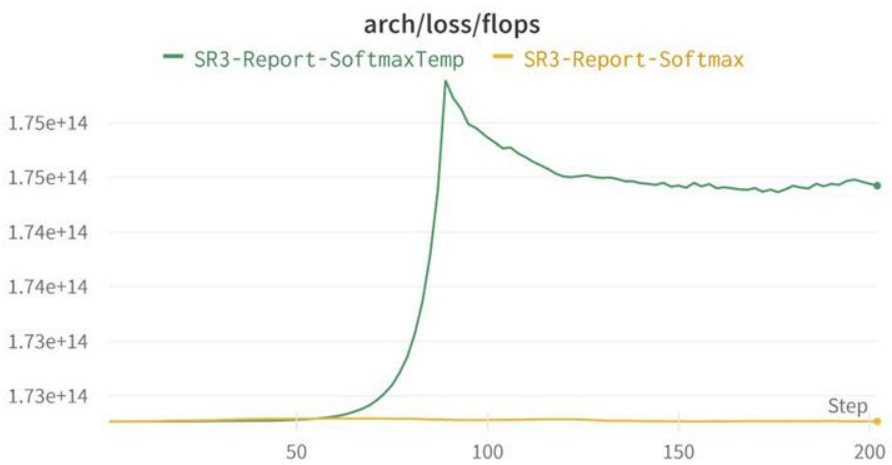


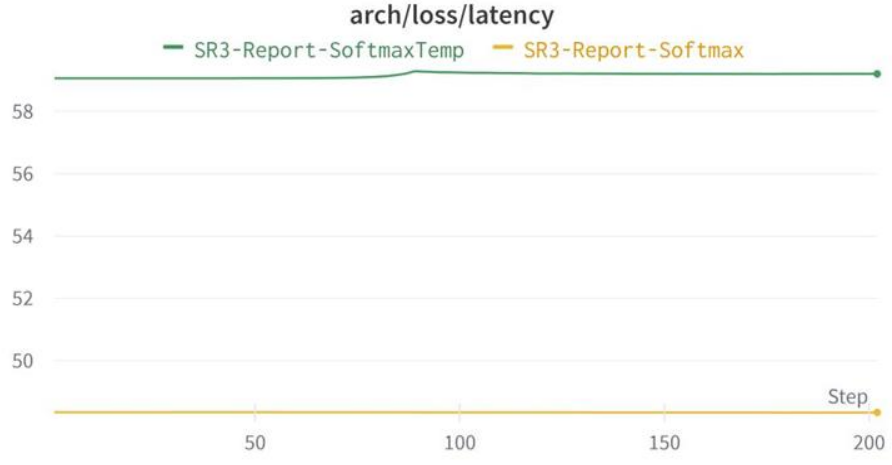




Experimental findings (Plots)

Supernet training FLOPs/latency loss

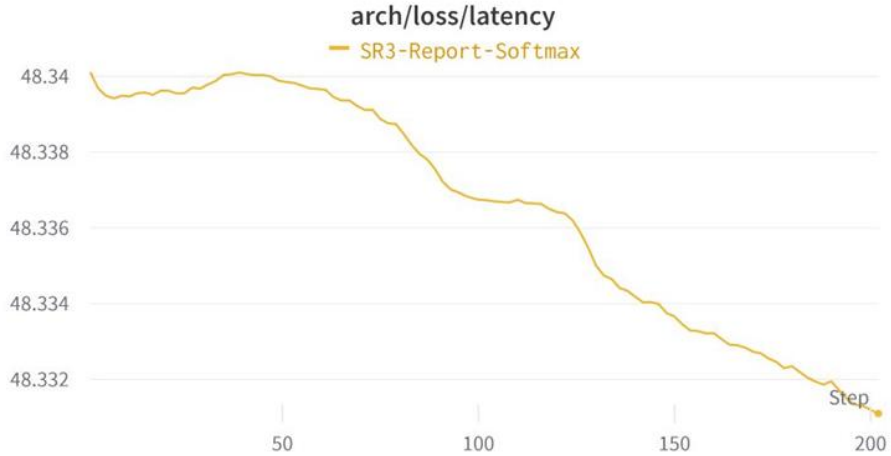




The above plot shows the behaviour of Softmax with temperature as the temperature drops. We see that the FLOPs value is low at first due to the function's output being close to uniformly distributed at high temperatures. However as the temperature drops, the function becomes more categorical and eventually stagnates at very low temperatures where changes in architecture parameters cause minimal change in function output.

Experimental findings (Plots)

Supernet training FLOPs/latency loss



The above plot shows the behaviour of Simple softmax which is independent of the temperature decay. We see that the FLOPs value is low at first due to the function's output being close to uniformly distributed due to equal architecture parameters. Subsequently, the FLOPs value rises as its weight in comparison to the L1 loss is low. As the weighted FLOPs becomes comparable to the L1 loss the NAS algorithm starts optimising for FLOPs as well.

Experimental findings (Images - Base)

These are the test images upscaled using the Base SR3 model.

Row 1 = Initial Gaussian noise

Row 2 = Image after 500 denoising diffusion steps

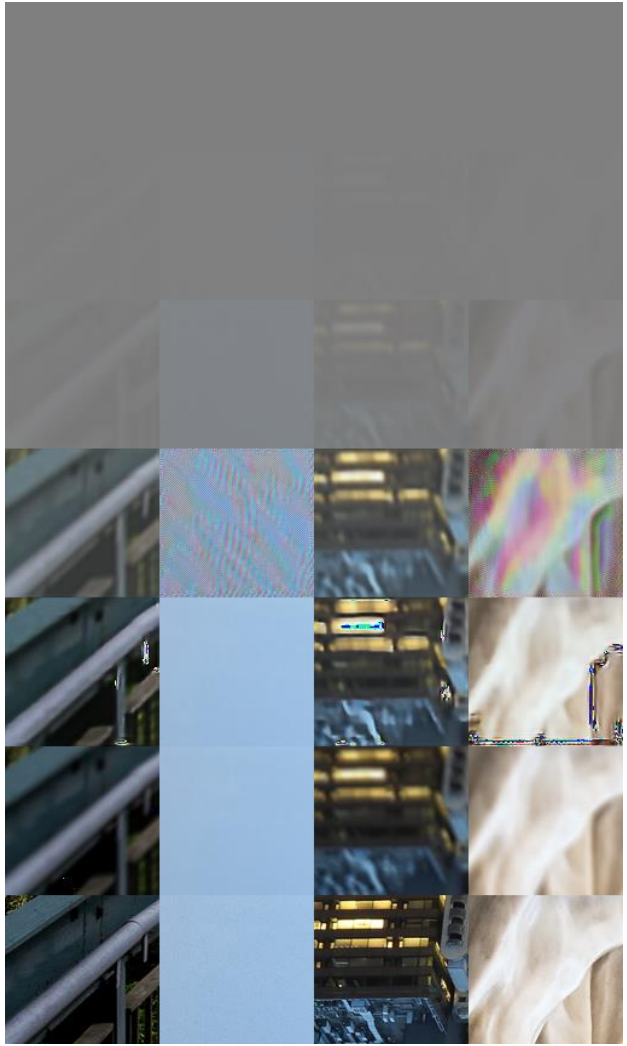
Row 3 = Image after 1000 denoising diffusion steps

Row 4 = Image after 1500 denoising diffusion steps

Row 5 = Final generated image (2000 steps)

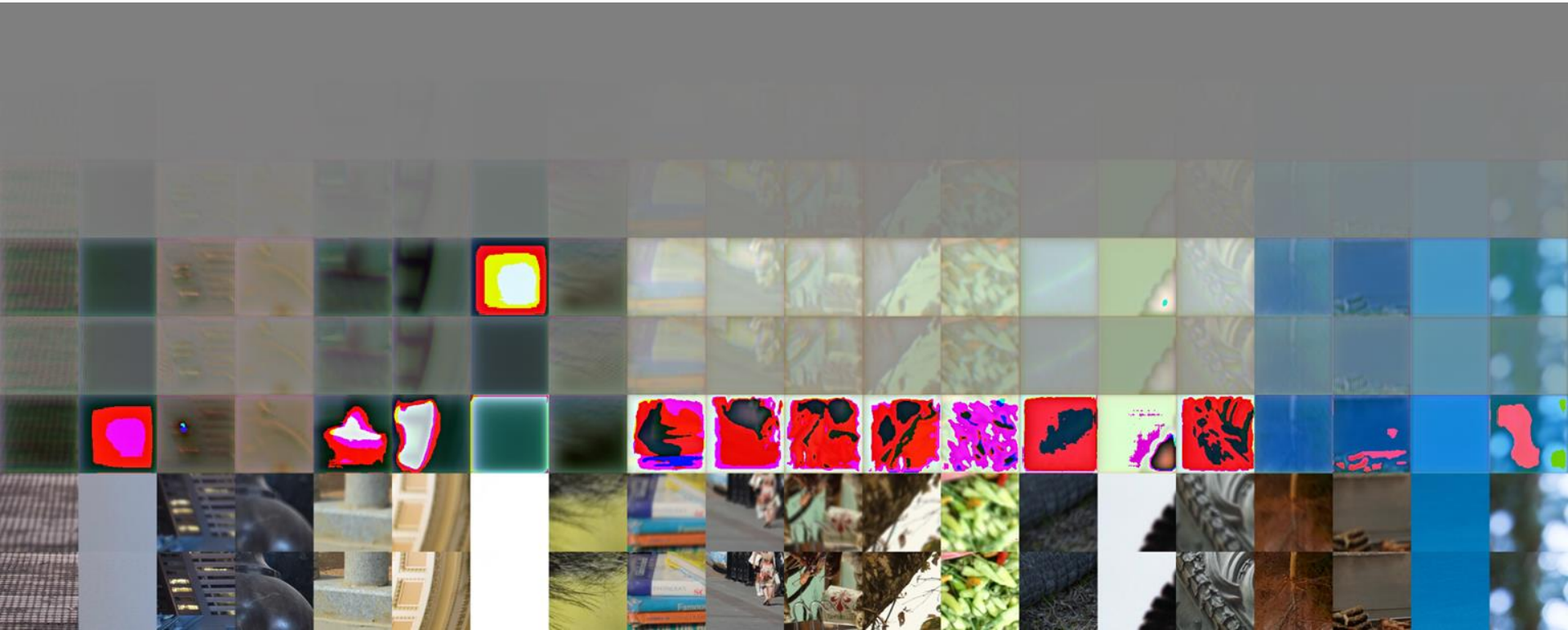
Row 6 = Input low-resolution image

Row 7 = Target high-resolution image



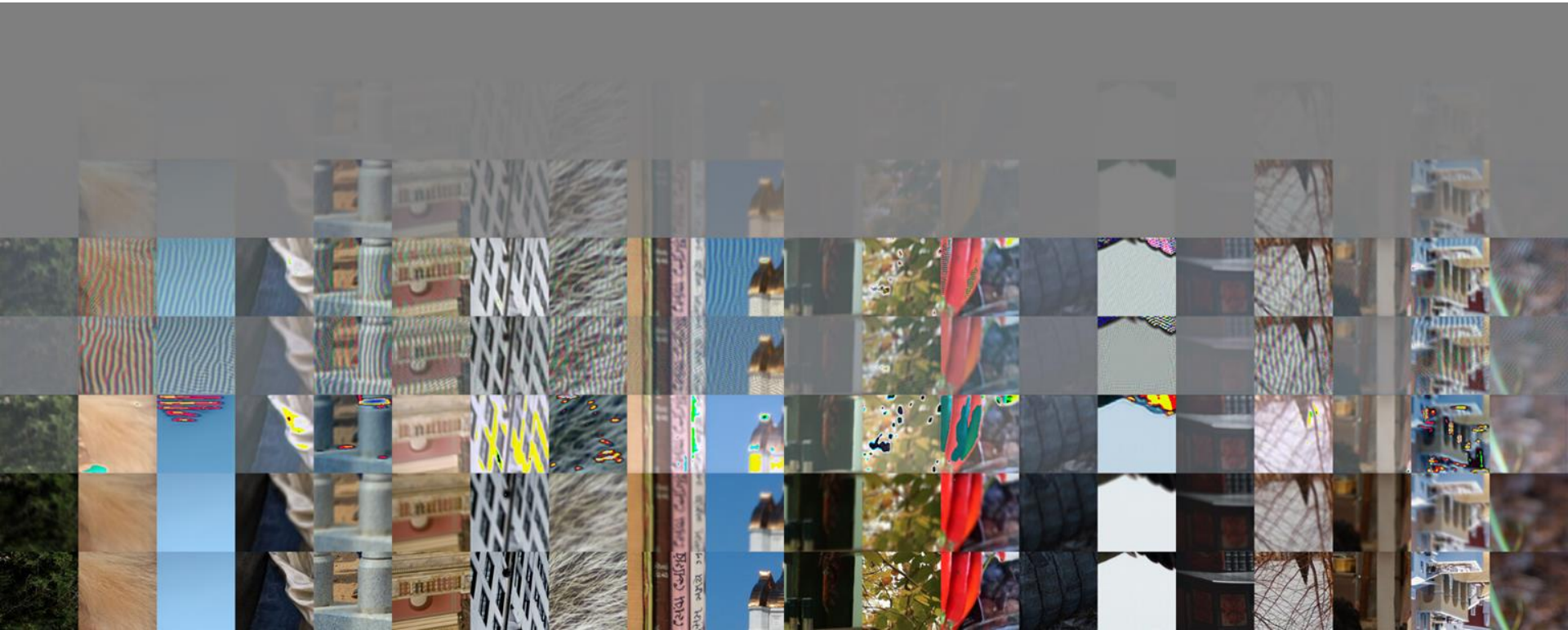
Experimental findings (Images - Gumbel-Softmax)

These are the test images upscaled using Gumbel-Softmax.



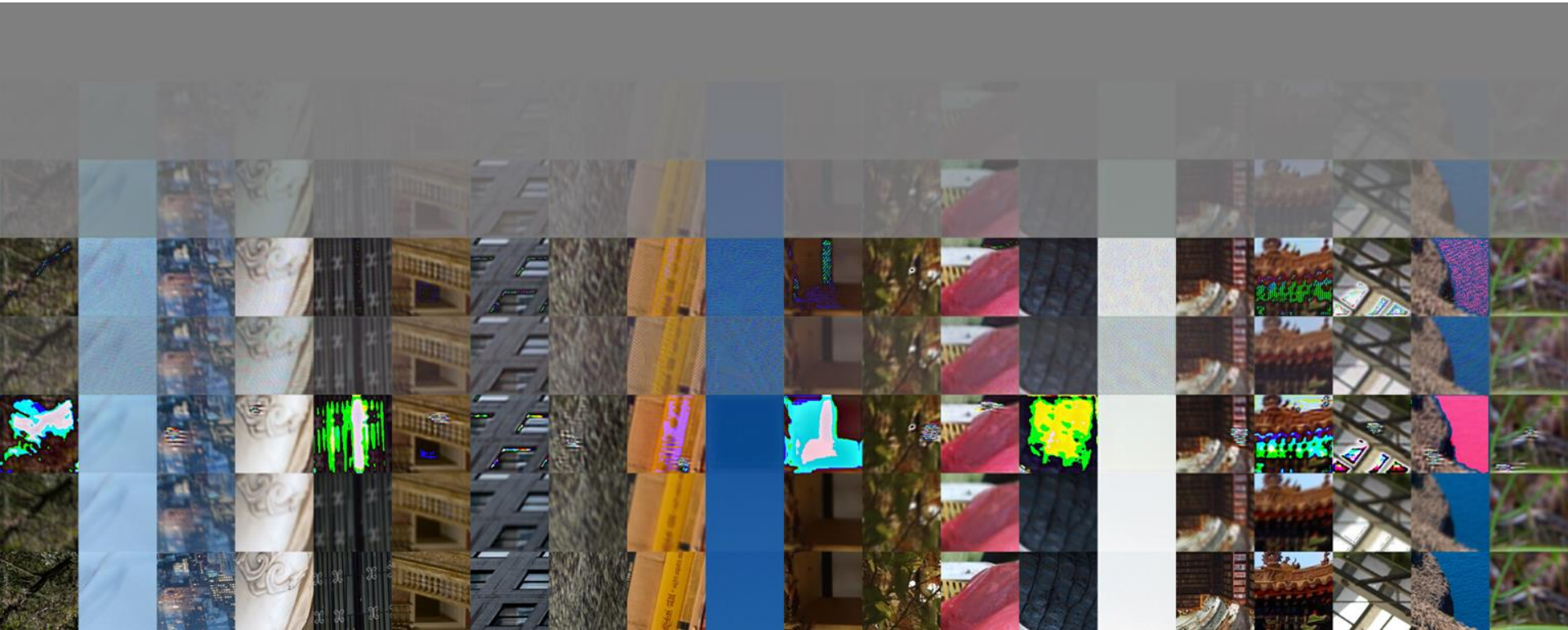
Experimental findings (Images - Softmax w/ temperature)

These are the test images upscaled using Softmax with temperature.



Experimental findings (Images - Simple softmax)

These are the test images upscaled using Simple softmax.



Thank You