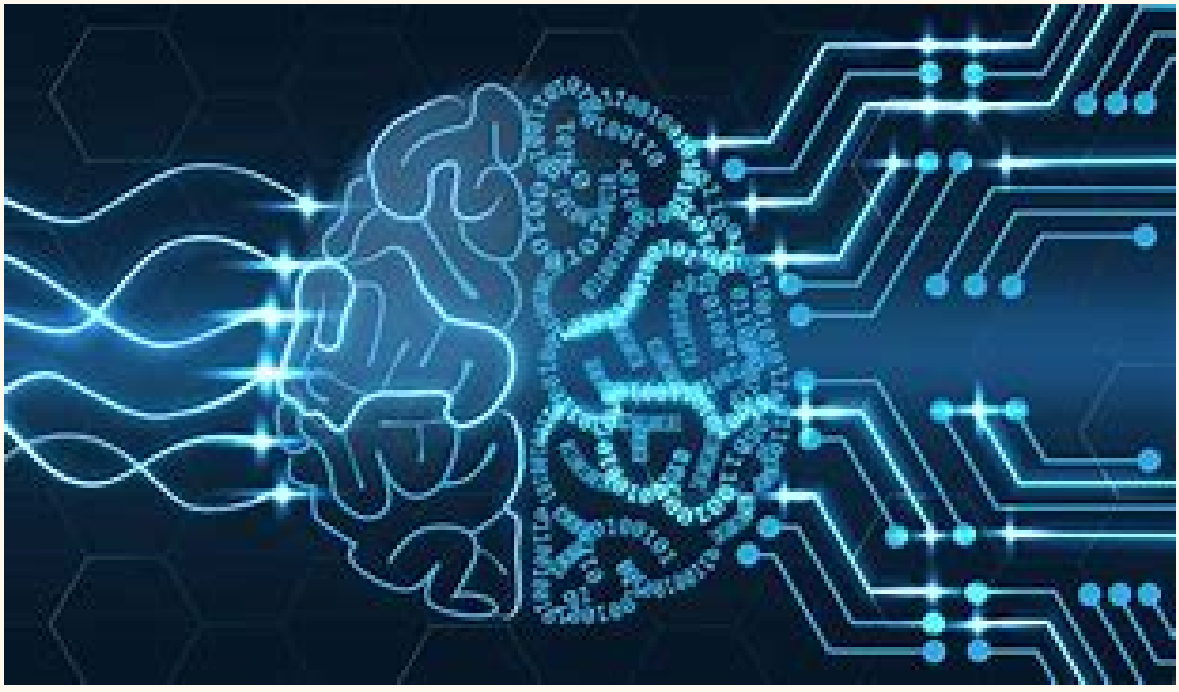
SUPER RESOLUTION using DEEP LEARNING with NEURAL ARCHITECTURE SEARCH (NAS) and HYPER PARAMETER TUNING

By

Muhammad FaziL K

CS20B1012

Indian Institute of Information Technology, Design & Manufacturing, Kancheepuram



ABSTRACT

This project delves into super-resolution, aiming to boost image quality through Deep Learning methods. Specifically, it explores the power of Neural Architecture Search (NAS) and Hyperparameter Tuning using the Neural Network Intelligence (NNI) framework. Our aim is to effectively utilize these technologies to find the best performing architecture and hyperparameters for the task of Super Resolution

The primary goal of Super Resolution is to generate high-resolution images from their low-resolution counterparts, a task critical in various applications such as medical imaging, surveillance, and digital photography. Deep learning techniques, particularly convolutional neural networks and generative adversarial networks, have shown remarkable success in super resolution by learning complex mappings between low and high-resolution image spaces.

This process of training super resolution models involves leveraging large datasets, optimizing network architectures through techniques like Neural Architecture Search (NAS), and fine-tuning hyperparameters to achieve optimal results.

In the experimentation phase, I tried to implement NAS on both Convolutional Neural Networks and Generative Advisory Networks and later proceeded with GAN after being enlightened about the advantage of GANs over CNNs for the specific task.

INTRODUCTION

Problem Statement

Develop a deep learning model capable of enhancing the resolution of images, transforming low-resolution images into high-resolution counterparts. Utilize Neural Architecture Search (NAS) for optimizing the architecture and conduct hyperparameter tuning to improve the model's performance

Objectives

Data Set Preparation

- Data Acquisition : Ensure Diversity
- Data Augmentation
- Down Sampling

Model Development

- Choose the Architecture Type
- Find the Best Architecture Using NAS
- Use Hyper Parameter Tuning to Get the Best Parameters for Training
- Train the Model on the Full Dataset leveraging the Acquired Knowledge

Evaluation

- Evaluate the Model Using Appropriate Metrics such as PSNR or SSIM
- Conduct a Detailed Analysis on Performance, Strength and Weaknesses

Literature Review

The literature review provides an overview of key models and methodologies in Image Super-Resolution (SR) task

SRCNN (Super-Resolution Convolutional Neural Network):

Introduced a fully convolutional network with a focus on simplicity and processing speed. Its success set the stage for subsequent developments in SR technology.

SRGAN (Super-Resolution Generative Adversarial Network):

A groundbreaking model. It introduced the concept of Generative Adversarial Networks (GANs) to SR. Utilizing a deep residual network with skip connections, SRGAN aimed at achieving photo-realistic super-resolved images at high upscaling factors.

ESPCN (Efficient Sub-Pixel Convolutional Network):

ESPCN addressed computational challenges by employing post-upsampling. The model efficiently learned the upscaling operation, contributing to improved performance compared to traditional SR methods.

LapSRN (Laplacian Pyramid Super-Resolution Network):

LapSRN adopted a progressive upsampling strategy, reconstructing sub-band residuals of high-resolution images. This approach aimed to enhance the quality of super-resolved images, particularly in cases with significant upscaling factors.

SwinIR (Swin Transformer for Image Reconstruction):

SwinIR represented a recent advancement by leveraging the Swin Transformer network. Integrating both Convolutional Neural Networks (CNNs) and Transformers, SwinIR achieved state-of-the-art results with notably fewer parameters, showcasing efficiency in SR tasks.

Evaluation Metric

Peak Signal to Noise Ratio (PSNR) : Quantifies image reconstruction quality in Image Super-Resolution. Calculated using Mean Squared Error, higher PSNR values indicate better fidelity between the original and super-resolved images, commonly measured in decibels (dB).

The idea range is **20~50 db**

METHODOLOGY

Data Preparation

- The training data was prepared by combining multiple datasets from different sources (mostly from Kaggle) to ensure quality and diversity
- More than 1000 images were collected and I handpicked a set of diverse ones with good quality
- Augmentation : I used random cropping as the method for Augmentation. Because it can make the model robust to positioning and context which leads to avoiding overfitting and improving generalization
- Another idea of augmentation idea was flipping the image, but had to avoid that due to low improvement of model to requirement of computational resources ratio
- Next step was downsampling the data. The entire dataset was downsampled by a scale of **4** with the help of a simple python script
- Then the data was Split into Train / Valid & Test.
    - Train : Used for training
    - Valid : Used for computing the metrics
    - Test : Used to manually test the quality

- The final dataset contained :
    - Train : 840 images
    - Valid : 248 images
    - Test : 112 images

Choosing the Model

- Super Resolution task is of two types
    - i) Single- Image : only one LR image is available, which needs to be mapped to its high-resolution counterpart
    - ii) Multi- Image : Multiple LR images of the same scene or object are available, which are all used to map to a single HR image.

  Our focus is on Super Resolution using Single Image

- Many popular architecture were there to begin with as I mentioned in the literature review
    - i) SRCNN
    - ii) SRGAN
    - iii) ESPCN
    - iv) LapSRN
    - v) SwinIR
    - And a lot of others...

In Single-Image SR methods, since the amount of input information available is low, fake patterns may emerge in the reconstructed HR image, which has no discernible link to the context of the original image. GAN is a good solution to this problem.

Also other advantages like Perceptual Loss Function, Photo Realistic Image creation etc. made me choose GAN as the model to train.

Neural Architecture Search on SRGAN

- The first step was to take a small sample dataset (**of size 32**) from the training images to make the NAS process considerably faster
- The next and most important step in implementing NAS was defining the search space

My search space for NAS was as follows :

```
{
 "num_block": {"_type": "choice", "_value": [8, 16]},
 "num_discriminator_blocks": {"_type": "choice", "_value": [3, 4]},
 "conv_kernel_size": {"_type": "choice", "_value": [3, 5]},
 "generator_depth": {"_type": "choice", "_value": [64, 128]},
 "discriminator_depth": {"_type": "choice", "_value": [64, 128]},
 "generator_activation": {"_type": "choice", "_value":
      ["prelu", "relu"]},
 "discriminator_activation": {"_type": "choice", "_value":
      ["leakyrelu"]}
  }
```

- Num_blocks : Number of blocks in the Generative Networks. The search space allows the NAS algorithm to explore architectures with either 8 or 16 blocks.
- Num_discriminator_blocks : Number of blocks in the Discriminator Network.The search space allows the NAS algorithm to explore architectures with either 3 or 4 blocks.
- Conv_kernal_size : Size of the convolutional kernel. Allows 3 or 5
- Generator_depth : Depth or number of filter in the Generative network
- Discriminator_depth : Depth or number of filter in the Discriminator network
- Generator_activation : Activation function used in the Generative network
- Discriminator _activation : Activation function used in the Discriminator network

These parameters were chosen according to standard and intuition .

Why PRelu/ Relu & LeakyRelu for networks ?

ReLU and its variants, like PReLU, introduce non-linearity into the model without suffering from the vanishing gradient problem, which is common with other activation functions like sigmoid or tanh. This allows for better backpropagation of errors and hence, better learning. So they are used in the generative network.

On the other hand, the discriminator network is typically best suited with LeakyReLU, which is used to fix the "**dying ReLU**" problem where a large gradient update can cause a neuron to become inactive and remain inactive irrespective of the input. By allowing small negative values when the input is less than zero, LeakyReLU provides a way for the gradient to backpropagate even for negative input values.

- Now the next step was to create a configuration file which we run to do the NAS.
- After that I edited the original SRGAN training code to accommodate the search space arguments.

  The modified code for NAS is well commented to show the differences.

- The tool that I leveraged for NAS was [Mircosoft NNI]

- I used the commands :

  ```
  tuner_params = nni.get_next_parameter()
  ```

  To inject the next set of parameters to the network and

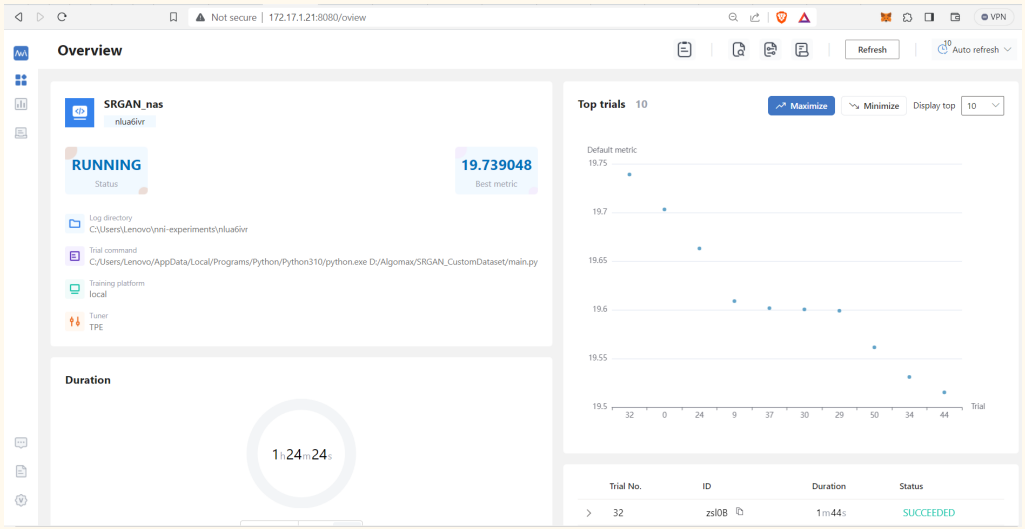
  ```
  nni.report_final_result(avg_psnr_pretrain)
  ```

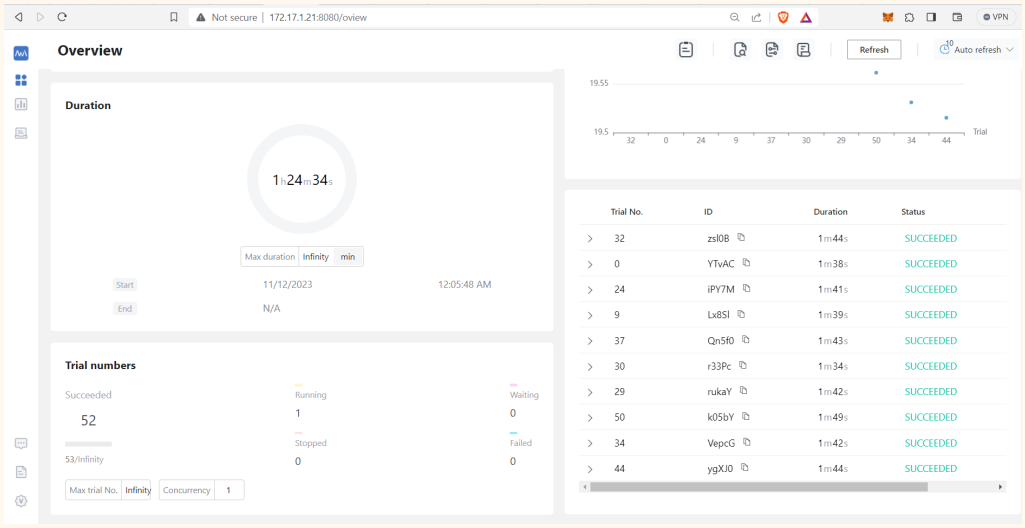
  To report PSNR (Peak Signal to Noise Ratio) as the metric to compare architectures

- Command to begin NAS is :

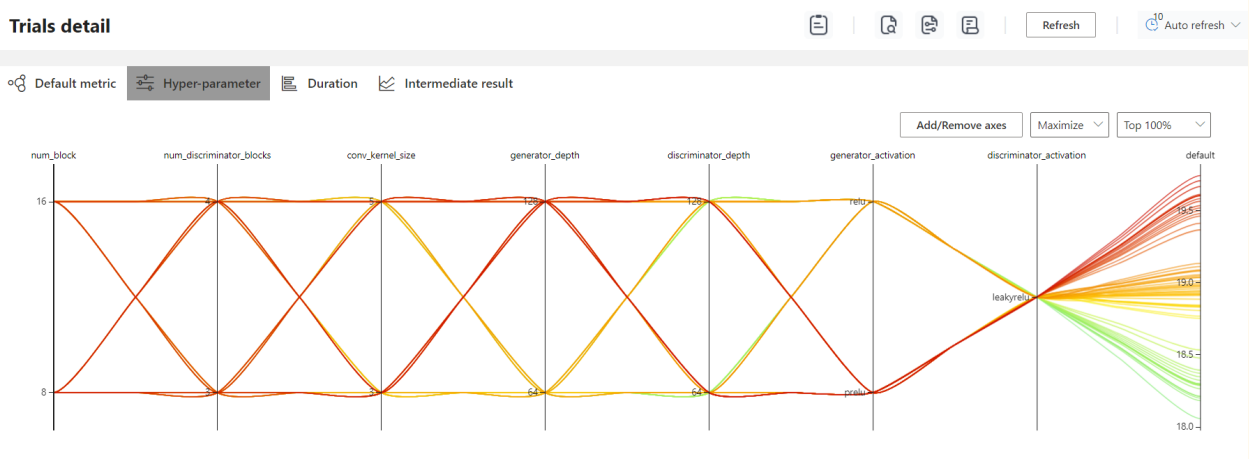
  ```
  nnictl create --config D:\Algomax\SRGAN_NAS\config.yaml
  ```

The NAS environment looks like this :





- Each architecture was compared based on the PSNR metric



- Here is the list of top 10 architectures

S.No	Exp No.	No. G Blocks	No. D Blocks	Conv Kernel	G Depth	D Depth	G_act	D_act	PSNR
1	32-zsl0B	8	4	3	128	128	prelu	leakyrelu	19.739
2	0-YTvAC	8	3	3	128	64	prelu	leakyrelu	19.703
3	24-iPY7M	8	3	5	128	64	prelu	leakyrelu	19.663
4	9-Lx8Sl	8	3	3	128	128	prelu	leakyrelu	19.608
5	37-Qn5f0	8	4	5	128	64	prelu	leakyrelu	19.601

6	30-r33Pc	8	4	3	128	64	prelu	leakyrelu	19.600
7	29-rukaY	16	3	5	128	64	prelu	leakyrelu	19.599
8	61-tdN1V	16	4	5	128	64	prelu	leakyrelu	19.578
9	50-k05bY	16	3	5	128	128	prelu	leakyrelu	19.561
10	34-VepcG	16	4	3	128	64	prelu	leakyrelu	19.531

- The low PSNR values were due to the small size of sample dataset used for NAS.

The final choice of architecture was :

- Convolution Window Size : 3
- No. of Generator Blocks : 8
- No. of Discriminator Blocks : 4
- Generator Depth : 128
- Discriminator Depth : 128
- Generator Activation Function : PRelu
- Discriminator Activation Function : LeakyRelu

Since I found out the best suited architecture of Super Resolution after this NAS step, I moved onto Hyper Parameter Tuning

- All details and log files of NAS is available here
-

Hyper Parameter Optimization

- The parameters I took into consideration were:
  - Batch Size
  - Number of Epochs
  - Learning Rate

-

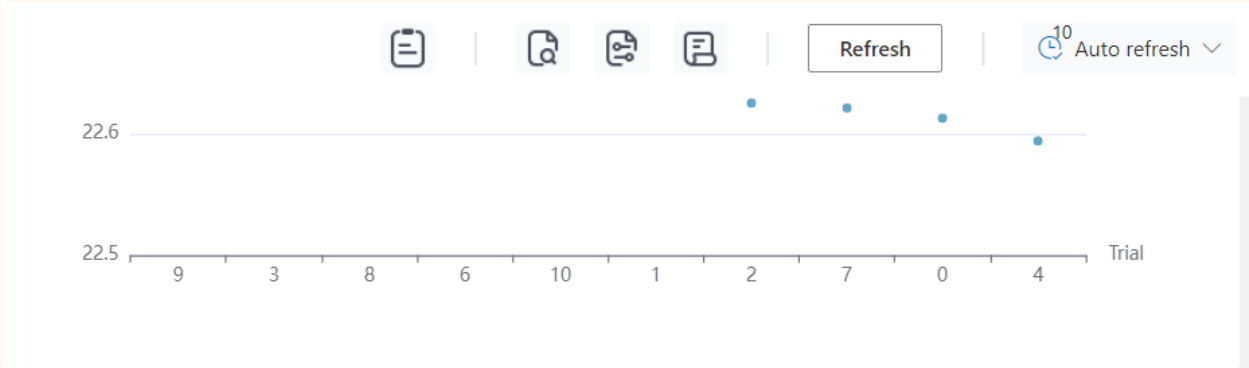
My search space was :

```
{
"batch_size": {"_type": "choice", "_value": [8, 16]},
"pre_train_epoch": {"_type": "choice", "_value": [2000,4000]},
"learning_rate": {"_type": "uniform", "_value": [1e-4, 1e-3]}
}
```

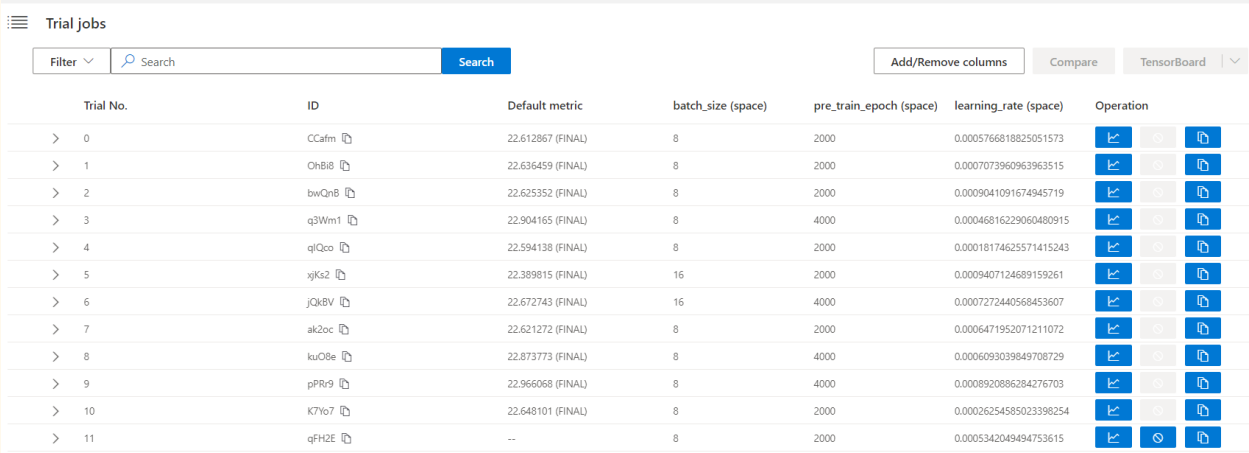
- I used NNI itself for hyper parameter optimization

- My top 5 Configurations were :

S. No	Exp No.	No. of Epochs	Batch Size	Learning Rate	PSNR
1	9- pPRr9	4000	8	0.00089	22.966
2	3- q3Wm1	4000	8	0.00046	22.904
3	8- kuO8e	4000	8	0.00060	22.873
4	6- jQkBV	4000	16	0.00018	22.672
5	10- K7Yo7	2000	8	0.00094	22.648

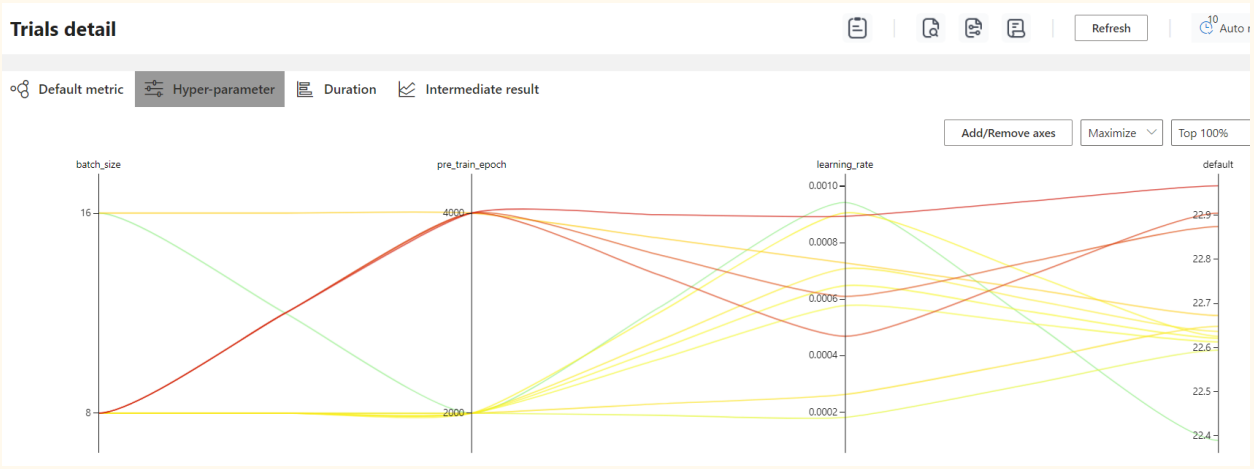


	Trial No.	ID	Duration	Status	Default me
>	9	pPRr9	47m50s	SUCCEEDED	22.96606
>	3	q3Wm1	48m20s	SUCCEEDED	22.90416
>	8	kuO8e	47m9s	SUCCEEDED	22.87377
>	6	jQkBV	44m18s	SUCCEEDED	22.67274
>	10	K7Yo7	23m43s	SUCCEEDED	22.64810

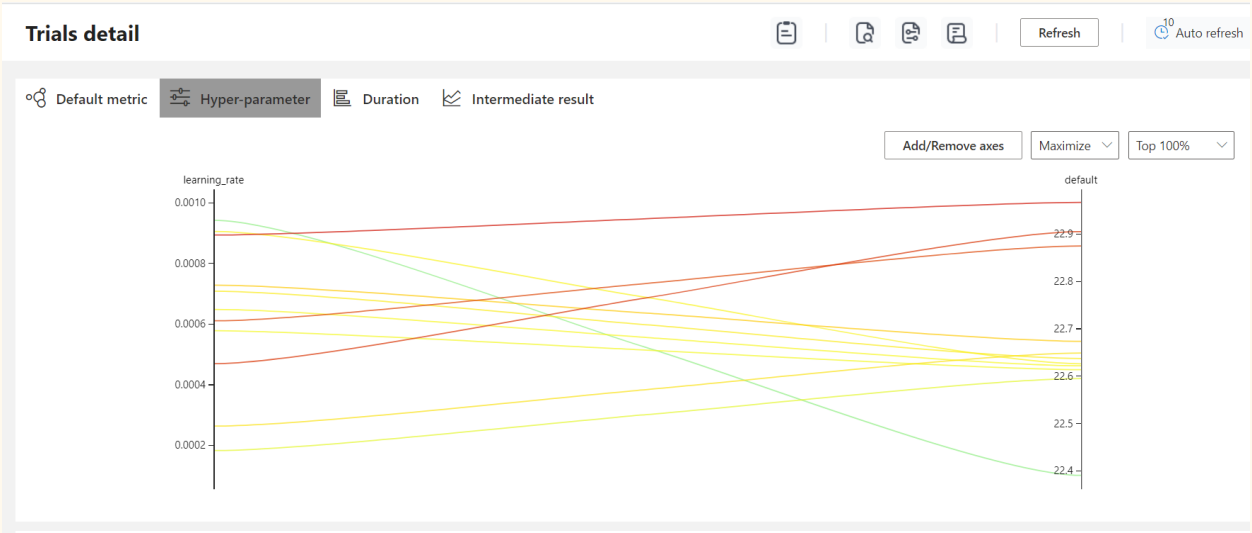


- Log files and screenshots of HPT are available here

From HPT , the following Trends were observed :



- Learning rate peaks while mowing towards to 0.001, but suddenly drops when it is too near to 0.001 (1e-3)

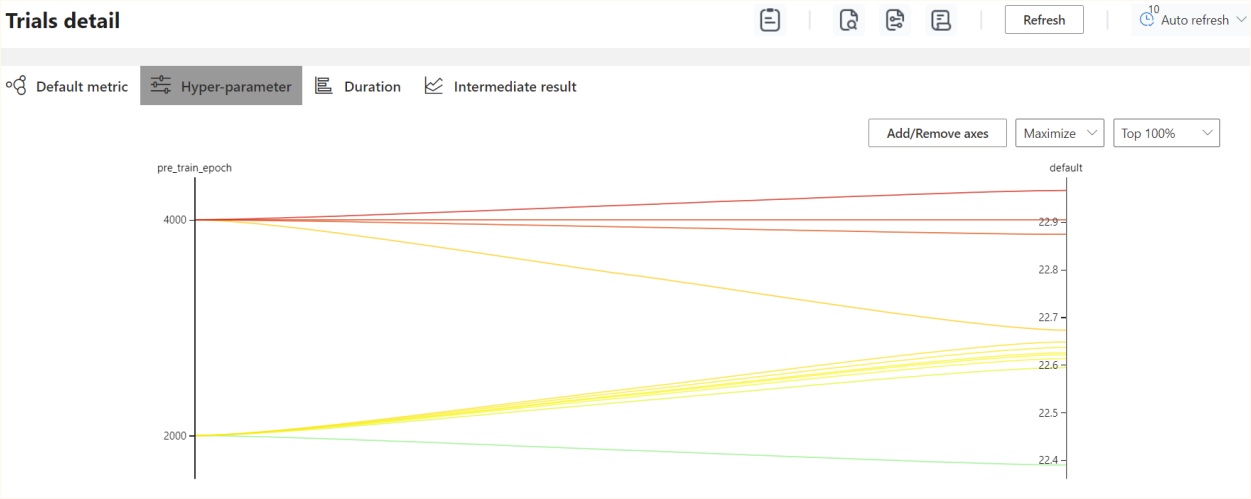


So the desired value should be around 0.008, I proceeded with 1e-4

- For batch size , 8 clearly emerged as far better than 16

Hence I chose 8 for actual training

- More epochs led to more accuracy



So from the result of Hyper Parameter Tuning on a sample dataset using NNI, I decided to do training with a batch size of 8, Learning rate of 1e-4 and 4000 epochs

Training Phase

- Training configuration were set as per the result of NAS and HPT :

```
parser.add_argument("--res_num", type=int, default=8)

parser.add_argument("--batch_size", type=int, default=8)

parser.add_argument("--pre_train_epoch", type=int, default=4000)

parser.add_argument("--mode", type=str, default="train")
```

```
generator = Generator(

        img_feat=3, n_feats=128, kernel_size=3, num_block=args.res_num,
scale=args.scale

    )
```

```
    discriminator = Discriminator(patch_size=args.patch_size * args.scale)
```

```
class Generator(nn.Module):

    def __init__(

        self,

        img_feat=3,

        n_feats=128,

        kernel_size=3,

        num_block=8,
```

```
        act=nn.PReLU(),
        scale=4,
    ):
```

```
class Discriminator(nn.Module):
    def __init__(
        self,
        img_feat=3,
        n_feats=128,
        kernel_size=3,
        act=nn.LeakyReLU(inplace=True),
        num_of_block=4,
        patch_size=96,
    ):
```

- Models were saved after x100 epochs
- Mean Squared Error Loss was used as the loss function

Training Command : !python main.py --LR_path SRGAN_CustomDataset\custom_dataset\train_LR --GT_path SRGAN_CustomDataset\custom_dataset\train_HR

-It took nearly 12 GPU hours for 4000 epochs

Evaluation of the Model

- After the training, I ran inference for validation and testing datasets

Validation

- Validation dataset contained 248 images and it's ground truth
- The saved models were validated based on the PSNR metric
- The top 5 are given below

No. of Epochs	2700	2100	3400	1500	2800
Average PSNR	25.393	25.320	25.291	25.250	25.217

- Surprisingly , a lower number of epochs yielded better results.
- This suggests that the higher performance of 4000 epochs configured trials during the HPT was most likely due to the smaller size of dataset

Testing

- After the validation, model was used for testing and manual inspection
- Few samples are given below

Result Image

Original High Res Image

Original Low Res Image

Low Res Image

SR Image

Results

The summary of the outcomes from different experiment stages are given concisely for a quick overview

NAS

Peak PSNR = **19.739**

Architecture :

- Convolution Window Size : 3
- No. of Generator Blocks : 8
- No. of Discriminator Blocks : 4
- Generator Depth : 128
- Discriminator Depth : 128
- Generator Activation Function : PRelu
- Discriminator Activation Function : LeakyRelu

Hyper Parameter Tuning

Peak PSNR = **22.966**

Hyper Parameters :

- Batch Size = 8
- Learning Rate = 1e-4
- No. of epochs >= 4000

Validation

Peak PSNR : **25.393**

For 2700 epochs

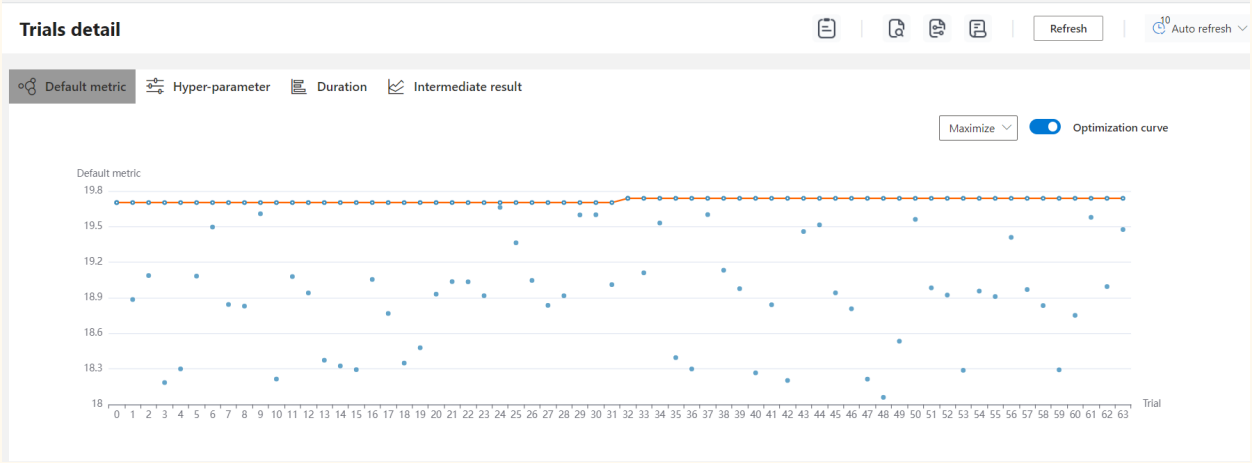
INSIGHTS

- **Choosing for SRGAN** has proven to be a strategic decision. It could address the specific challenges of single image super resolution
- **The results of NAS** were useful in changing a few of my initial preferences for the architecture. The small of the dataset chosen has lowered the PSNR value range and the difference between them.
- The NAS could have been much more efficient and insightful with my initially planned search space :

```
{
  "searchSpace": {
    "num_block": {"_type": "choice", "_value": [8, 16, 24, 32]},
    "num_upsample_blocks": {"_type": "choice", "_value": [1, 2, 3]},
    "generator_activation": {"_type": "choice", "_value": ["relu",
"leaky_relu", "prelu"]},
    "use_skip_connections": {"_type": "choice", "_value": [true, false]},
    "num_discriminator_blocks": {"_type": "choice", "_value": [3, 4, 5]},
    "discriminator_activation": {"_type": "choice", "_value":
["leaky_relu", "sigmoid", "tanh"]},
    "perceptual_loss_coeff": {"_type": "choice", "_value": [0.001, 0.01,
0.1, 1.0]},
    "adversarial_loss_coeff": {"_type": "choice", "_value": [1e-4, 1e-3,
1e-2, 1e-1]},
    "tv_loss_coeff": {"_type": "choice", "_value": [0.0, 1e-4, 1e-3,
1e-2]},
    "conv_kernel_size": {"_type": "choice", "_value": [3, 5, 7]},
    "conv_stride": {"_type": "choice", "_value": [1, 2]},
```

```
    "activation_function": {"_type": "choice", "_value": ["relu",
"leaky_relu", "prelu", "sigmoid", "tanh"]},

    "generator_depth": {"_type": "choice", "_value": [64, 128, 256]},

    "discriminator_depth": {"_type": "choice", "_value": [64, 128, 256]}

  }

}
```

- The search space size was : 14,929,920, i.e. nearly 15 million. Conducting an NAS on this search space with a considerably larger dataset could have resulted in a surprisingly better result. But due to the unavailability of the computational resources and time, I had to sacrifice and choose a smaller search space and sample dataset. With a search space size of 128 and dataset size of 32
- Some of the omissions were strategic, like the discriminator activation function for which LeakyRelu is undisputedly proven to be the best choice and and some were intuitive like the true or false choice for skip connections. And the rest like loss functions and the few choices from the chosen parameters were removed just for the sake of simplicity
- Also the no. of epochs I chose for NAS was very small, i.e. 100 which subsequently caused the small PSNR values.
- Despite all these , I believe the NAS was successful because it could avoid a bunch of bad architectures. The final result may not have yielded the best out of the entire potential search space, but certainly has given something that can produce satisfying results .



- **Similarly for Hyper Parameter Tuning**. the initial search space was larger

```
{
    "learning_rate": {"_type": "choice", "_value": [1e-4, 5e-5, 1e-5]},
    "batch_size": {"_type": "choice", "_value": [8, 16, 32, 64]},
    "weight_decay": {"_type": "choice", "_value": [1e-4, 5e-5, 1e-5]},
    "optimizer": {"_type": "choice", "_value": ["adam", "sgd"]},
    "num_epochs": {"_type": "choice", "_value": [2000, 4000, 8000]}
}
```

- Even though this search space is of a smaller size of 216, it would have taken considerably longer period to complete since the huge number of epochs
- This could have yielded the best result if was tuned for a larger dataset

Finally I had to downgrade to this :

```
{

    "batch_size": {"_type": "choice", "_value": [8, 16]},

    "pre_train_epoch": {"_type": "choice", "_value": [2000,4000]},

    "learning_rate": {"_type": "uniform", "_value": [1e-4, 1e-3]}

}
```

- Had to omit both weight decay and optimizer.
- Instead of giving direct choices for learning rate, I gave a range from 0.0001 to 0.001, which was a strategic decision that saved a lot of resources. The best results were produced around 0.008 which I could not have found if I had gone for choices despite the heavier consumption
- Batch size of 8 gave the best results, which means the choices 32 & 64 may not have given good results. So even if it was by pure chance, that decision helped in faster completion and saving resources
- All of the results with 4000 epochs resulted in larger PSNR values. Which means the number of epochs for best configuration must be greater than 4000. But this intuition was proven to be wrong later when I trained on the complete dataset. There epochs 2700 & 2100 produced the best results which means it was the size of the sample dataset that caused the lower PSNR values
- **In the validation phase**, I observed that a few models gave best results for some images reaching more than 40 PSNR value, but they necessarily did not give the best average value .
- Better generalization is the key for robustness, hence I am going with Model_2700.pt as my best model

Conclusion

In summary, the project successfully applied Neural Architecture Search (NAS), and Hyperparameter Tuning (HPT) to train a deep learning model.

Despite limitations in dataset size and search space, the NAS phase identified a promising architecture with a convolution window size of 3, 8 generator blocks, 4 discriminator blocks, and a depth of 128 for both. HPT refined the model, highlighting trends like an optimal learning rate around 0.0008 and a preference for a batch size of 8.

Finally the best model I could train for the super resolution task was the one with the above configurations and was saved after 2700 epochs.

The project's success lay in achieving peak PSNR values . Despite constraints, the chosen configurations displayed robustness and potential for generalization.

—----------

The entire code , dataset and resultant models are available here

—----------