Scalable Neural Network Architecture Search Applied to Super-Resolution Networks

HONOR THESIS

Presented in Partial Fulfillment of the Requirements for Graduation with Honors Research Distinction in the Degree Bachelor of Science in the Undergraduate College of Engineering at The Ohio State University

By

Lang Xu B.S.

Department of Computer Science and Engineering

The Ohio State University

2022

Abstract

Based on attention mechanisms, the transformer architecture has been widely adopted in machine learning applications, including natural language processing and computer vision. A common strategy for improving transformer models has been to greatly increase the number of parameters. However, training large transformer models requires prohibitively large computation for consumer hardware. High-performance computing clusters equipped with massively-parallel hardware such as graphical processing units (GPU) are the perfect environments to improve model performance by automatically searching possible transformer model architectures, thus removing the requirement of manually tuning the architecture. This project aims to demonstrate the potential of HPC clusters in training huge vision transformer models through Neural Architecture Search (NAS) methods. We chose image super-resolution as our low-level vision task and the Swin Transformer as our vision transformer backbone. This work demonstrates how transformer networks can be improved after conducting NAS on HPC clusters, followed by distributed deep learning training. Our results show that HPC clusters will drastically reduce the searching and training time in NAS while at the same time producing more fine-tuned and accurate super-resolution transformer models.

Acknowledgements

I would like to express my sincerest gratitude to everyone who made this undergraduate thesis a success. I would like to thank Dr. D. K. Panda for offering me the opportunity to conduct academic research under the HiDL project at NOWLAB.

I would like to extend my gratitude to Dr. Christopher Charles Stewart for serving as a committee member for the oral defense in spite of his occupied schedule and my late notice.

The completion of my undergraduate thesis would not have been possible without the guidance and nurturing of Quentin Anthony. I am deeply indebted to him for his unparalleled support both academically and mentally, his profound belief in my abilities and unbelievable patience. I wish him every success in completing his PhD dissertation.

I am also grateful to all current and previous NOWLAB members that for building a valuable and precious environment of academic research and collaboration.

I would like to thank to San Diego Supercomputer Center [1], Lawrence Livermore National Laboratory and Texas Advanced Computing Center for their technical resources and The College of Engineering at The Ohio State University for their financial support.

Finally, I would like to thank my father Bin Xu and my mother Rong Xue for their relentless support and love. Without their sacrifices, I would not have been able to be here and complete my undergraduate study.

Vita

2015......................Shanghai Foreign Language School

2018 to present......B.S. Computer Science and Engineering, The Ohio State University

Publications

Q. Anthony, L. Xu, H. Subramoni and D. K. Panda, ”Scaling Single-Image Super-Resolution Training on Modern HPC Clusters: Early Experiences,” 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 923-932, doi: 10.1109/IPDPSW52791.2021.00143.

Fields of Study

Major Field: Computer Science and Engineering

Table of Contents

List of Tables

List of Figures

Introduction

Deep Learning

Given the emergence of extensive data and parallel computer hardware, deep learning has been driving advances in many machine learning applications, including high-level natural language processing (e.g. natural language generation and understanding), high-level computer vision (e.g. image classification and visual question-answering), and low-level computer vision (e.g. image segmentation, image super-resolution) tasks. Deep Learning relies on a statistical and graphical data structure called a Deep Neural Network (DNN). DNNs serve as a model to find hidden patterns in data and make predictions with great accuracy. A DNN takes a single tensor as input data, propagates it through a series of layers, and produces output predictions. Output predictions can be classification values or real values depending on the task goal. A DNN performs such predictions on many sets of training samples and adjusts the internal parameters to reduce the output error.

A DNN is built upon several layers of ”neurons.” Between a given pair of neurons is a connection with a specific weight value. Each neuron applies a non-linear mapping on the weighted sum of its incoming values. The output is called an **activation value**, and the mapping process is called an **activation function**. The activation value is passed further down the network to the next neuron. For vision applications, the input to the first layer is usually the pixel intensity values of the original image. The second layer fetches input from the first layer's activation value and propagates its outputs to the third layer. This phase of a training iteration is known as a **forward pass**.

After each forward pass, a loss value will be calculated between the DNN's output and the expected value. In order to reduce the loss value, an algorithm called backpropagation will be applied to compute and propagate backwards the gradient of the loss value with respect to each neuron connection's weight in the network. This phase of a training iteration is known as a **backward pass**. The gradient has the information on the direction the weight vectors should proceed to realize the steepest decrease in the loss function. This method of repeatedly evaluating the gradient of the loss function and performing updates on the weight value according to the direction of the gradient is called **gradient descent**. A brief mathematical expression of gradient descent is in the following equation:

$$\vec{w'} = \vec{w} - \alpha \times \nabla E(\vec{w})$$

$\vec{w'}$ is the updated weight vector in the network expected to reduce the loss value. $\nabla E(\vec{w})$ is the gradient of the loss function evaluated on the current weight vector. α is called the learning rate, and indicates how far along a given direction we should step. Choosing the correct learning rate is essential in training a DNN. As expressed in the gradient descent equation, we are making an update in the negative direction of the loss gradient. There are many variations of gradient descent, such as mini-batch gradient descent, where the gradient will be computed over batched of the training data, not the entire training set. This variant performs better on large-scale applications with large amounts of training data.

DNNs have become the basis of many other feed-forward networks, which are experts in handling tasks like computer vision and natural language processing. A **convolutional neural network** (CNN) is a common example.

In Figure 1 is a multi-layer deep fully connected neural network that contains a series of fully connected layers. A fully connected layer establishes connections between every neuron in one layer to another. A fully connected network is easily adoptable to a wide range of tasks since it doesn't stage specific assumptions on the input. However compared to task-specific networks, it has rather weaker performance. Different from a conventional fully connected neural networks, the CNNs assume that inputs are images.

In Figure 2 is a typical cnn architecture. There are several main types of layers used to build CNNs as shown in Figure 2. An input layer will hold the original raw pixel values of the incoming image. A convolution layer will perform a dot product between the neurons' weights and a small region of input data in the image. Dependent on the task type, various functions will be in use. A pooling layer will typically apply a downsampling operation on the incoming volume and reduce its width and height dimensions. Finally, the fully-connected

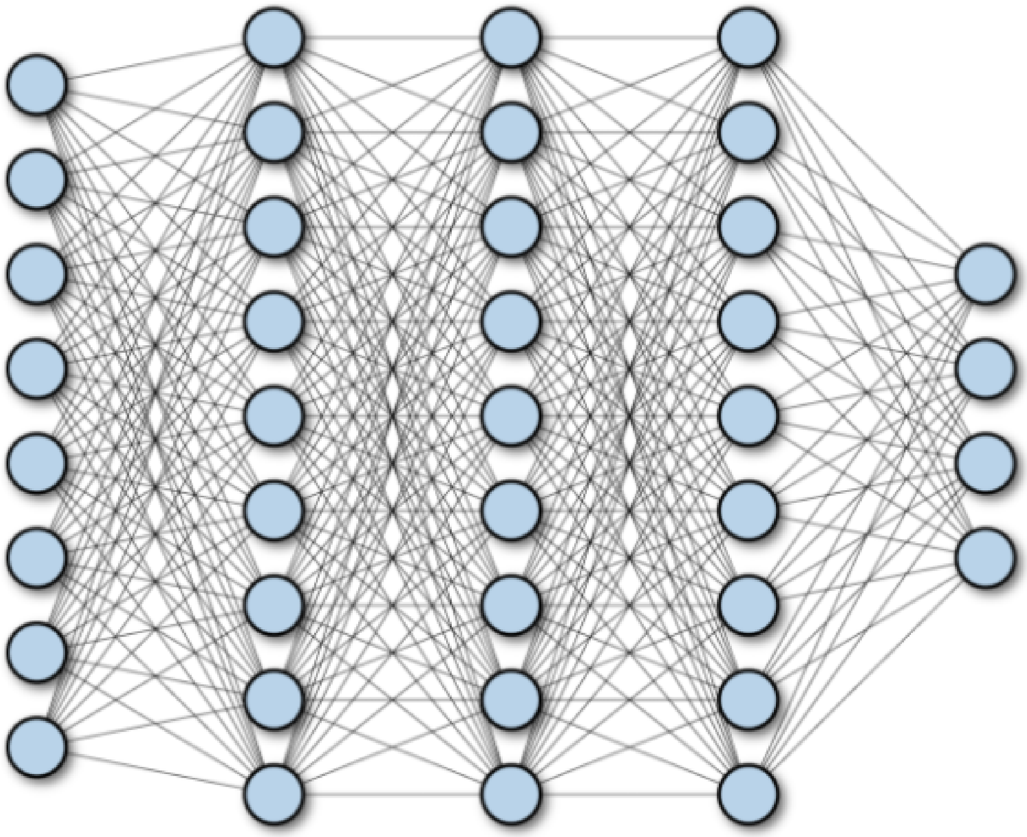


Figure 1: A Multi-layer Deep Fully Connected Network

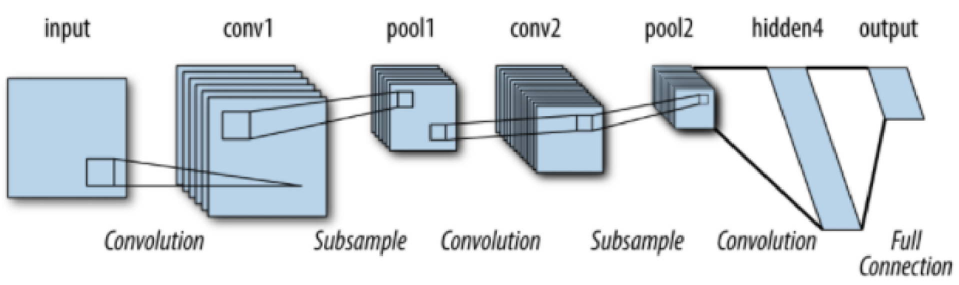


Figure 2: Typical CNN architecture

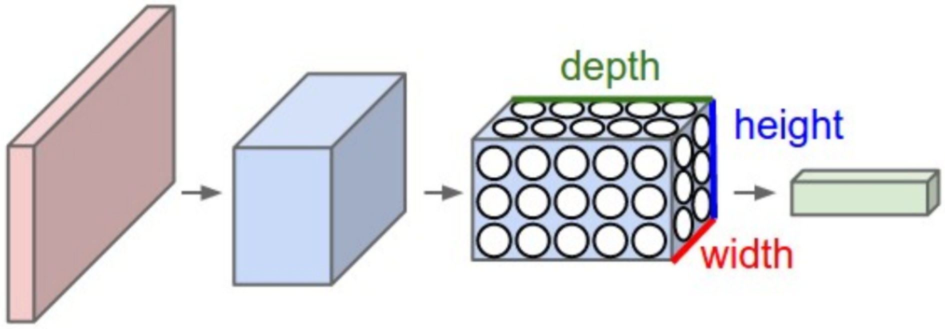


Figure 3: 3D CNN structure

layer at the tail of the network is responsible for translating incoming values to class scores or other notable forms. As the name implies, each neuron in this layer is connected to all the neurons in the previous layer. With the help of these layers and a feed-forward pass, CNN transforms an input image from its raw pixel values in every dimension towards a final score for further applications.

In Figure 3, we can see that a CNN arranges its neurons in a 3D (width, height, depth) fashion in each layer. Every layer of this CNN will transform the input image, a three dimension data type to a 3D matrix of activation values. The red input layer of the CNN holds the input image. Correspondingly, the width, height and depth of that layer will be that of the image. The depth of the layer will be 3, indicating the red, green and blue channels. As shown in the figures above, with the help of convolutional layers, CNNs have less number of layers than fully connected networks. As a result, they require less number of weights during training and thus are highly efficient for image tasks. Also, the internal 3D dimensionality of a convolutional layer caters to input images and excels at extracting out high-quality features from the images [2].

Super Resolution

Image super-resolution (SR) is a long-lasting, low-level vision task that aims to generate high-resolution (HR) images from given low-resolution (LR) images. Image super-resolution has various applications in the real world, including medical imaging [3, 4, 5], media content enhancement, security [6, 7] and so on. Also, it can assist other computer vision tasks in the field [8, 9, 10, 11, 12]. However, given the existence of many possible high-resolution images corresponding to a single low-resolution input image, super-resolution is a demanding problem that is difficult to define. There has been extensive research into solving the problem before deep learning became widespread. Various classical methods have been introduced to model the degradation process from HR images to LR images. Such methods include basic cubic interpolation and filtering [13], image registration [14, 15], gradient-based methods [16], and many more [8, 17], [18, 19, 20, 21].

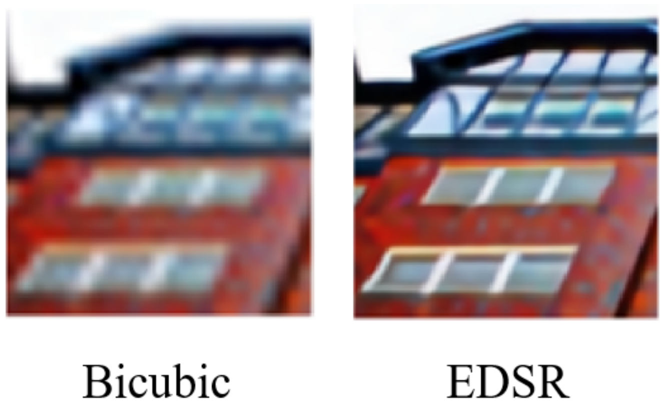


Figure 4: Traditional bicubic super-resolution vs Deep Leaning based super-resolution using EDSR [22]

With the help of deep learning techniques and a growing scale of datasets, DNN-based super-resolution methods have gained significant adoption and produce state-of-the-art performance on many benchmarks. From the early CNN-based methods, [23] to recent transformer-based methods [24, 25], techniques usually differ in the following areas: novel network architectures, exploration of different loss functions, and learning strategies. To illustrate the effect of upscaling an image using deep learning, Figure 4 provides a comparison between the effect of traditional bicubic interpolation(left) and modern DNN-based method(right). The image quality is highly improved with deep learning. Popular datasets for SR include DIV2K [26], Set5 [27] and Set14 [28]. Unlike other vision tasks, image quality assessment(IQA) is the loss function for image SR. The Peak Signal-to-Noise Ratio (PSNR), together with

Structural Similarity Index (SSIM) is a popular combination in addressing both subjective methods based on humans' perception and objective computational methods.

Transformer Models

The transformer architecture was introduced to handle natural-language processing tasks, specifically sequence transduction. It is based on the attention mechanism [29], which is able to model dependencies between different language tokens regardless of their distances. This mechanism dispenses the need for recurrent models and achieves a higher degree of parallelization and high translation quality. Orthogonal to recurrent neural networks, a transformer doesn't handle data in order but instead provides information for any position in the input sequence. The original transformer architecture is depicted in Figure 5.

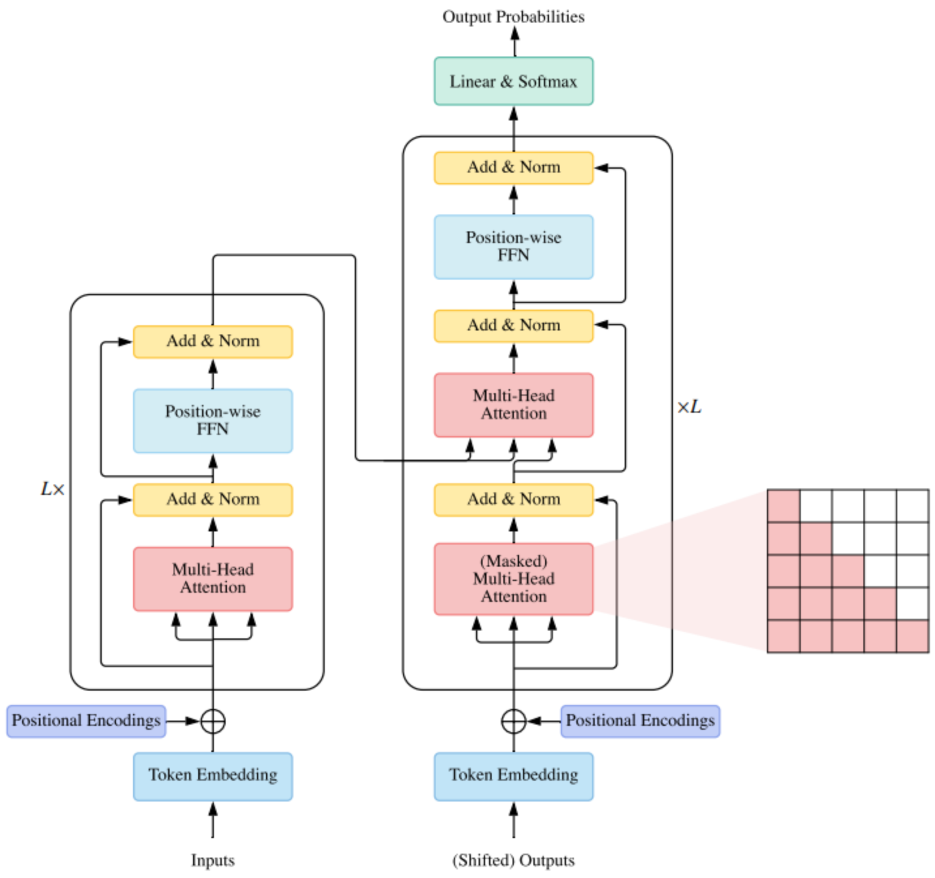


Figure 5: Overview of original transformer architecture [29]

The original transformer is a sequence-to-sequence model and includes an encoder-decoder design. Each encoder block is composed of a multi-head self-attention module (MHSA) and a position-wise feed-forward network (FFN). There are also residual connections [30] around each module, followed by layer normalization [31]. The decoder module is similar to the encoder but is different in that an additional multi-head attention is computed over the output of the encoder module. Also, An additional masking is applied to the first MHSA module so that previous positions are prevented to attend subsequent positions. This masking together with a one position offset on the output embedding ensure that information on current position i is only dependent on positions less than i. When a sentence is passed into a transformer, the attention layer will calculate the attention weights between every language token at the same time. The attention module will embed every token with the context of the token itself as well as a weighted matrix of other tokens in the sentence. Each token in the matrix is assigned an attention weight. The attention module is inspired by how the human brain assigns uneven attention across input data. The calculation of the attention matrix can be expressed as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The inputs are queries (Q), keys (K) and values (V). The dot products of the query with all the keys are computed before dividing by $\sqrt{d_k}$. Then, a softmax function is applied to obtain each value's weights.

Inspired by the success of applying Transformers to NLP, many researchers have become curious about the transformer architecture's potential in handling computer vision tasks. Many works have emerged trying to combine convolution architectures with the attention mechanism. Typical approaches use a method of dividing up images into patches before feeding them as image tokens into the Transformer [32]. As we can see in Figure 6, the image patches are handled similarly to the tokens (words) in NLP. After the input image is splitted into fixed sized patches, the patches then go through a process of linear projection with position embeddings and are further fed into a regular Transformer Encoder. However, while images illustrate information at various scales, a sufficient vision transformer should not be limited to fixed-scale tokens. The standard vision transformer conducts global self-attention, which computes the relationships between an image patch and all other patches. Such computation exhibits quadratic computational complexity with respect to the number of patches, making it unsuitable for dense image prediction and high-resolution image tasks (e.g. image segmentation and image super-resolution).

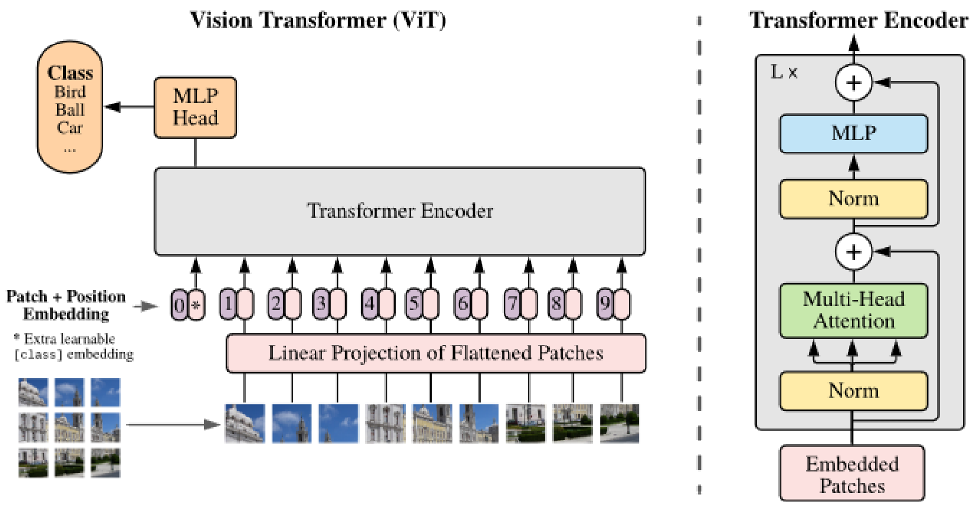


Figure 6: Overview of vision transformer (ViT) [32]

The Swin Transformer [33] recently emerged as a solution to the previous concerns. It features a hierarchical representation that can extract information from small-sized patches and leverage them for bigger patches. In Figure 7, we can see that compared to ViT that produces feature maps of only one low resolution, Swin Transformer gradually merges patches as it dives into deeper layers. With this strategy, Swin Transformer is able to leverage and integrate different features from different scales, making it a general-purpose backbone for various vision tasks. [33]. In addition, Swin Transformer designed a local non-overlapping window structure that includes a fixed size of image patches and conducted local attention computation within windows. Different than the global attention scheme in ViT, this approach reduced the computation complexity from quadratic to linear. In order to ensure context exchange between windows, Swin Transformer also implements cross-window connections by shifting windows in successive transformer blocks as illustrated in Figure 8. In general, as depicted in Figure 9, a Swin Transformer architecture added additional patch merging stages which conducts self-attention at its own scale. Each stage has several successive blocks of Swin Transformer encoder-decoder structure that includes multi-head attention modules with shifted window approach. Such designs make Swin Transformer extremely efficient and powerful in achieving new state-of-the-art models in vision tasks.[33].By addressing long-range dependency modeling and fewer computational resources, Swin Transformers have been applied to image super-resolution. The recent SwinIR [25] model consisting of several Swin Transformer layers has shown better Peak Signal-to-Noise Ratio (PSNR) and fewer parameters. We choose the Swin Transformer as a general backbone for vision transformers in this project.

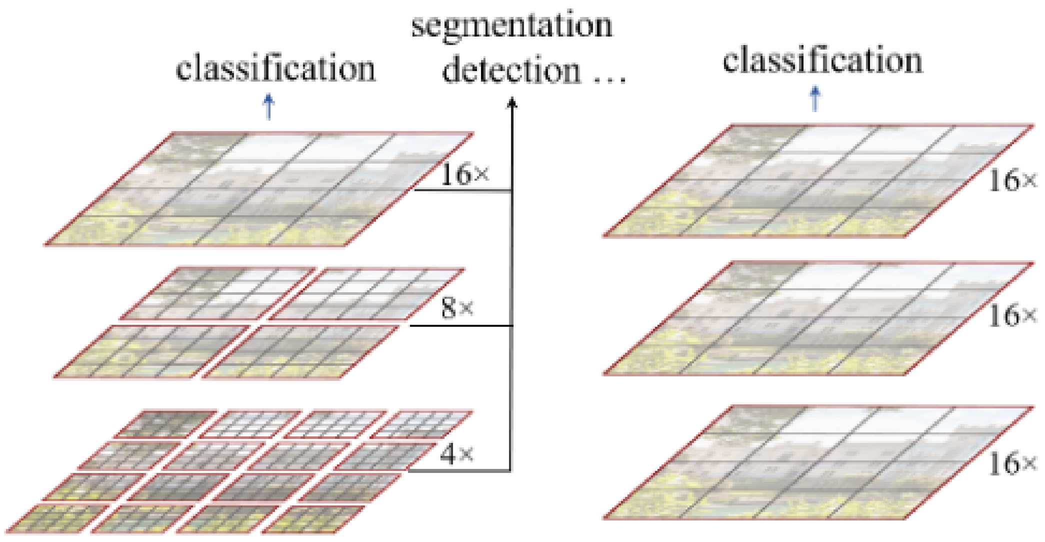


Figure 7: (left)Swin Transformer [33] vs (right)Vision Transformer [32] in feature representation

Neural Architecture Search

Tuning a vision Transformer network to reach its best performance is always challenging. Many hyperparameters are vital in this process, including the depth, embedding dimension, and the number of heads. It is difficult to find a good combination of them. Most previous

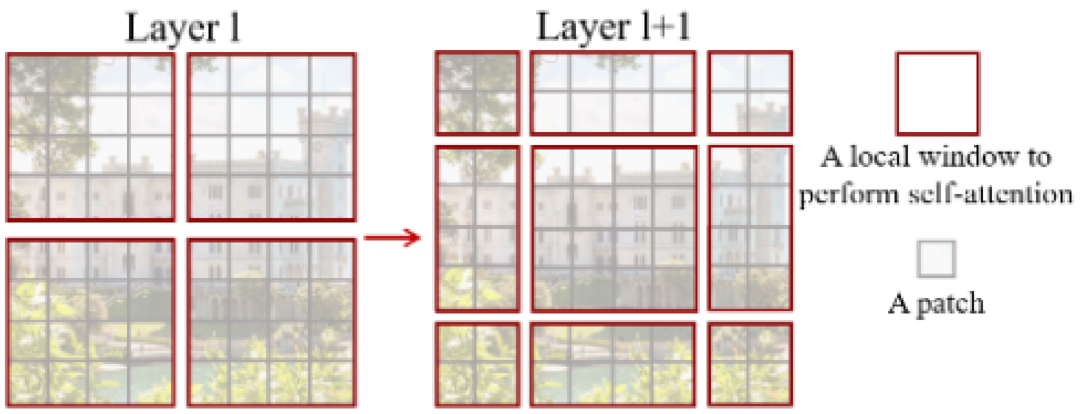


Figure 8: shifted window approach [33]

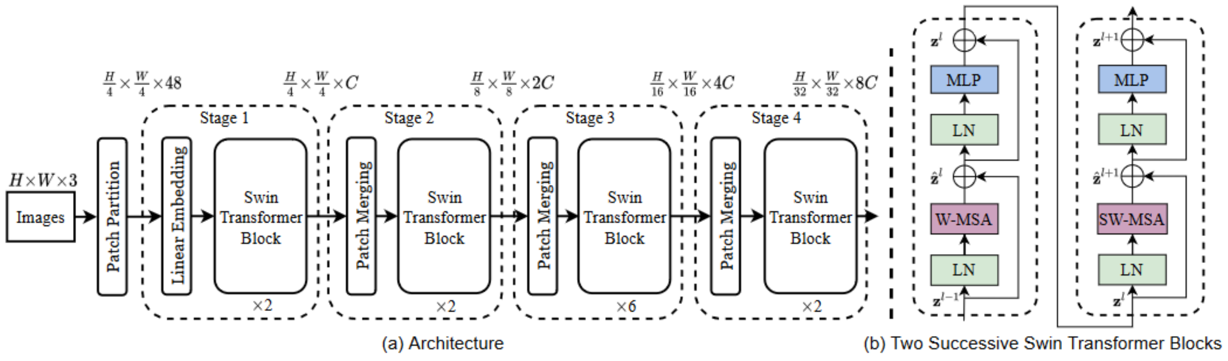
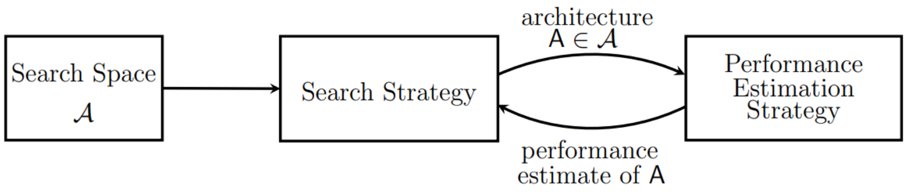


Figure 9: Swin Transformer architecture [33]

works focus on manually crafting the best vision Transformer, while only a few elaborated on automating transformer design using neural architecture search (NAS). NAS is the process of automating architecture engineering. Specifically, it is a strategy for automating machine learning, a subfield in AutoML [34, 35]. NAS is characterized by three dimensions: **search space**, **search strategy**, and **performance estimation strategy** [36]. Relationships between those three fields are illustrated in Figure 10



Search Space $\mathcal{A}$
Search Strategy
architecture $\mathsf{A} \in \mathcal{A}$
Performance Estimation Strategy
performance estimate of A

Figure 10: Abstract illustration of NAS methods

In short, a search strategy will select a candidate architecture from a predefined search space. The candidate will be evaluated under the performance estimation strategy. The final estimate will be returned to the search strategy for the following search. The search space is the definition of the candidates than can be selected. The size of the search space depends on the preset knowledge of the typical properties of possible architectures. The search strategy draws the details of how to explore in a search space. The performance estimation strategy fulfills the objective of NAS: to find architectures that entail the best performance on unknown data [36]. It is the process of evaluating an architecture's potential performance. A basic strategy is training the candidate networks and comparing their metrics. In this project, we integrated ideas from a one-shot vision Transformer search framework called AutoFormer, and a hierarchical search strategy proposed by GliT [37]. The AutoFormer adopted a search strategy where different transformer blocks share weights for their parts in each layer [38]. The so-called weight entanglement enabled many subnets in the supernet to be well-trained. It further allows more architectures to reach different resource constraints while also displaying similar accuracy.

The larger a search space, the more architectures that need to be tested, trained and evaluated. Although there have been much effort in reducing search space complexity [36], NAS methods are generally very computationally expensive. High-performance clusters can provide such a demanding environment. With the help of thousands of nodes, each with several advanced Graphical Processing Units (GPU), the search for candidate networks and the ensuing full training session can be accelerated. The results of this project are validated and obtained from several large-scale national high-performance clusters.

Related Work

Image Super-Resolution

Generating a high-resolution image from a low-resolution ground-truth is a long-standing research challenge in the computer and information sciences. The first attempts at super-resolution algorithms involved transforming (wavelet- or fourier-based) the LR image to the frequency domain, then estimating the HR image in the frequency domain, before finally transforming the HR image back to the spatial domain [39, 40, 15]. While the frequency domain is attractive due to its theoretical simplicity (the LR and HR images are directly related in the frequency domain) and computational efficiency (efficient fourier transform algorithms have received significant attention due to their ubiquity in scientific applications), such methods are restricted to simple models of motion blur corrections and are highly sensitive. All subsequent super-resolution techniques, including this work, are focused on the spatial domain. Early algorithmic attempts include clever interpolation [17], projections [14, 18], and regularizations [19, 20, 21].

Once deep learning was demonstrated with CNN architectures [41] on the imagenet dataset [2], early super-resolution CNNs began to emerge [23, 42]. The main intuition for CNN-based SR is that if CNN architectures are provided enough LR-HR image pairs, they will learn to reconstruct common image substructures. Therefore, most early CNN SR solutions first upscale the LR image via interpolation, then "fill in the gaps" using convolutional layers. Subsequent works introduced architectural improvements such as residual blocks [30] and residual scaling [22]. We note that CNN-based image super-resolution techniques can be easily scaled to HPC systems using data parallelism [43].

Vision Transformers for Super-Resolution

Attention-based methods such as transformers have recently been applied to image SR [24, 25]. The first transformer for super resolution was TTSR [24], which uses HR reference images of similar subjects to transfer high-resolution textures to more efficiently "fill in the gaps" of upscaled images. SwinIR [25] is a more recent SR transformer that both reduces the computational complexity (by replacing TTSR's self-attention with the window-attention mechanism introduced in [33]) and removes the training restriction of HR reference images.

Neural Architecture Search

Neural Architecture Search (NAS) is a DL architecture-selection paradigm that iteratively modifies a base architecture within a space of architecture settings (known as the **search space**) using a **search strategy**. While NAS has been successfully applied to CNN architectures [44], the search space of transformer models is prohibitively large, which requires restrictive search spaces or aggressive search strategies. Some recent works have addressed these issues for vision transformers [45, 37, 38]. While such methods are effective at finding improved architectures, they inefficiently parallelize to HPC systems.

Method

Early Investigations: TTSR

In our early investigation on applying NAS to transformer architecture. We mainly introduced and tested a NAS architecture combining the differentiable multi-scale search space [46] and the TTSR network [24].

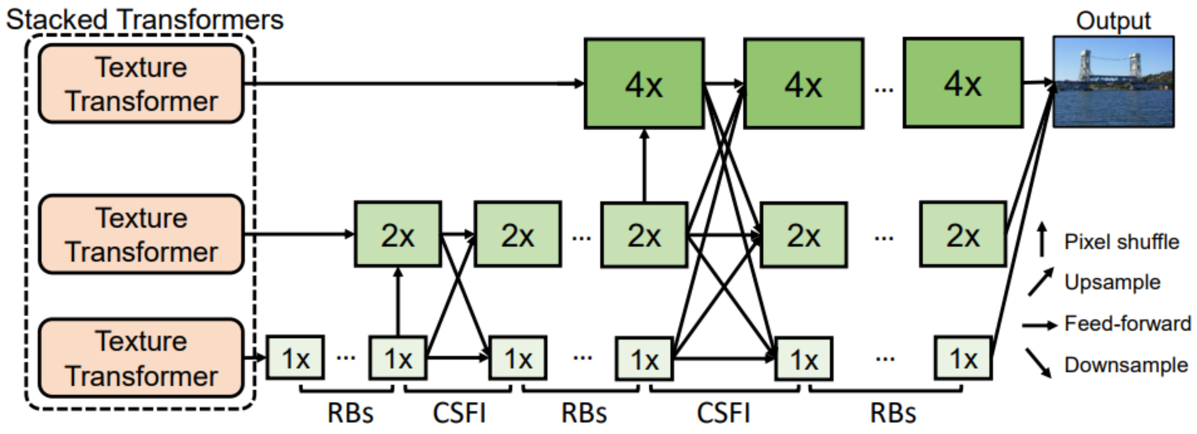


Figure 11: TTSR Architecture (Courtesy [24])

Constructing differentiable search space

In order to introduce NAS into the TTSR architecture, we need to first define a search space. We achieved this by replacing the Cross-Scale Feature Integration (CSFI) Module with the Mix Module. The CSFI Module is inspired by [47, 48], it is designed to exchange texture information between features of different scales. This module is inserted after the extracted feature from LR images is up-sampled to the next scale. For each participating scale, the features will receive information from other scales by either up-sampling or down-sampling. Such process is always followed by a concatenation operation and a convolution layer. The module is proved to achieve a more powerful feature representation, which is fused from synthesized features of different scales. To visualize our changes, please refer to the original TTSR network structure in Figure 11, and the altered version in Figure 12. In Figure 12, we redesigned the CSFI stage in the original architecture, making it a mix module. For

CSFI module between scale 1× and 2×, we replace it with Mix Module 2×, indicating the exchange between 2 scales. For CSFI module between scale 1×, 2× and 4×, we replace it with Mix Module 3×, indicating the exchange between 3 scales.

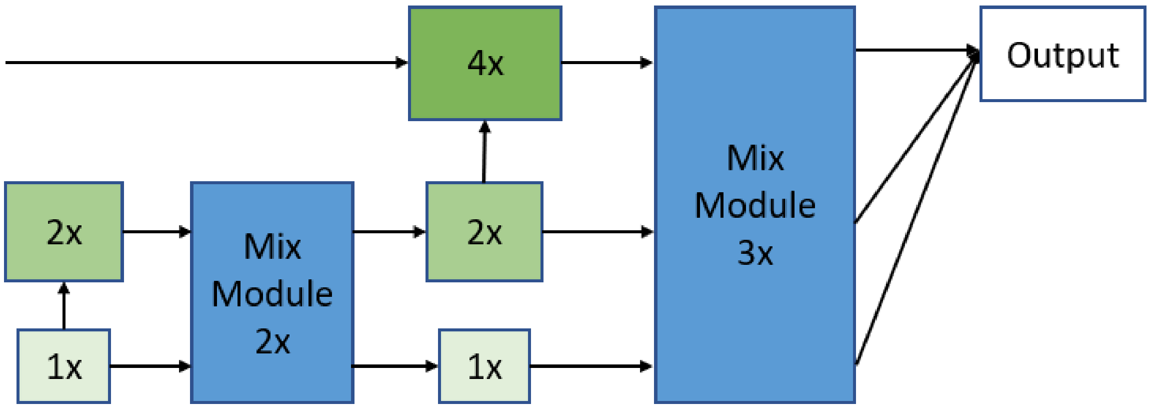


Figure 12: MixModule Architecture

A search space is the space in which possible architectures may reside and be found. Inspired by CLEARER [46], we introduced the notion of differentiable search space into TTSR through Mix Modules. Within each Mix Module, there are N columns. Each column is a soft attention module or a CSFI module for the current scales. Let x_n be the input to the n-th column of a mix module and let f_n^{SA}, f_n^{CSFI} be the soft attention module and the CSFI module at the n-th column. We will have:

$$x_{n+1} = w_n^{SA} \times f_n^{SA}(x_n) + w_n^{CSFI} \times f_n^{CSFI}(x_n)$$

where $w_n^{SA} = 0$, $w_n^{CSFI} = 1$ or $w_n^{SA} = 1$, $w_n^{CSFI} = 0$. To efficiently search for a desirable architecture in this discrete search space, we apply a continuous relaxation of w_n^{SA}, w_n^{CSFI} as indicated in CLEARER [46]. The above constraint can be re-written as $w_n^{SA}, w_n^{CSFI} \in (0, 1)$ and $w_n^{SA} + w_n^{CSFI} = 1$. We used the softmax to achieve the relaxation [49]. A detailed process is illustrated in Figure 13: (a) The modules in a Mix Moduleare initially unset. (b) We fill the mix module with a combination of TTSR modules. (c) We apply iterative optimization using continuous relaxation on the network weights and architecture loss. (d) We found the final architecture constructed from the learned probabilities.

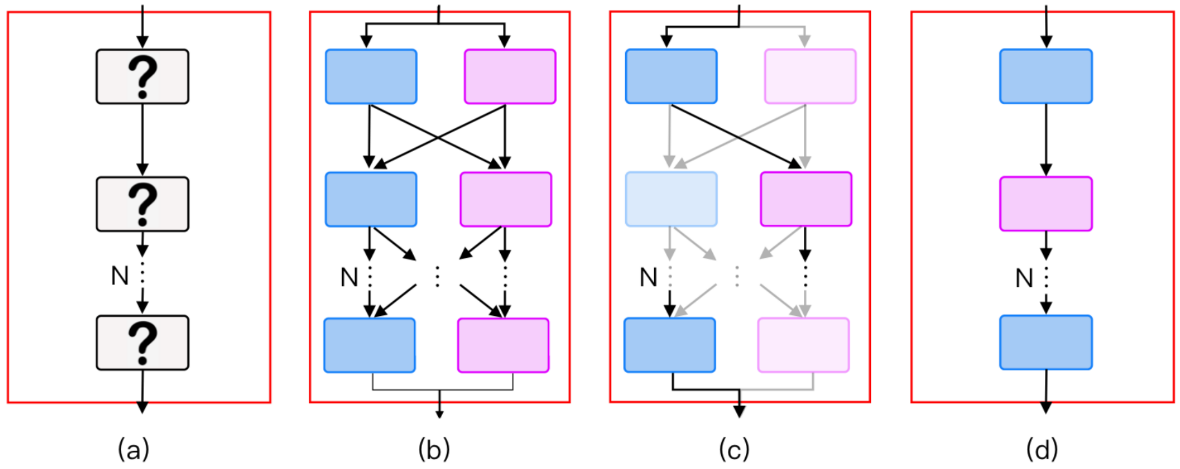


Figure 13: CLEARER Architecture (Courtesy [46])

Loss Function

In addition to the loss functions introduced in TTSR, we also integrated the architecture loss in CLEARER as indicated below:

$$L_{Arch} = (\frac{-1}{N}) \sum_{w \in (w^{SA}, w^{CSFI})} (w \log w + (1-w)\log(1-w))$$

We include the architecture loss [46] to make the SA modules and CSFI modules within a Mix Module distinguishable. This regularization approach will enforce w to approach either 0 or 1, keeping them away from lingering around 0.5.

Our final loss evaluation equation added in the Reconstruction Loss, the Adversarial Loss, the Perceptual Loss and the architecture Loss and is expressed as below:

$$L_{overall} = \lambda_{rec} L_{rec} + \lambda_{adv} L_{adv} + \lambda_{per} L_{per} + \lambda_{arch} L_{arch}$$

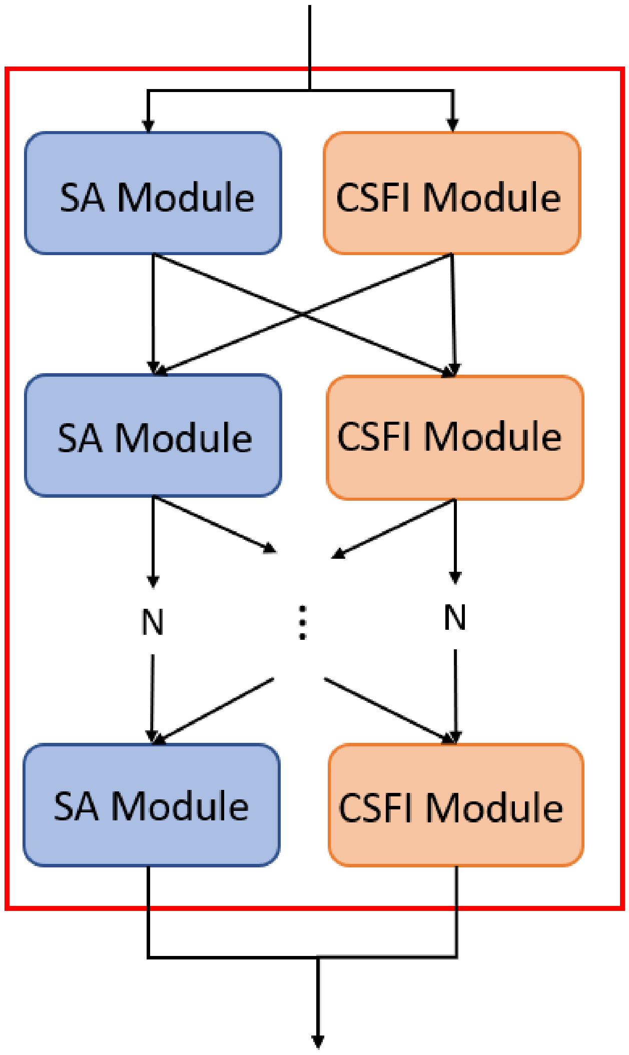


Figure 14: TTSR-NAS Architecture

SwinIR NAS

First we must introduce a NAS scheme to the SwinIR transformer architecture. We created a novel NAS process and applied it to SwinIR. Specifically, we define the following NAS solution:

- Search Space: The set of possible architecture changes we wish to explore within (e.g. number of attention heads, number and size of MLPs, etc).
- Search Strategy: An algorithm to traverse the search space by iteratively modifying, training, and evaluating a set of possible architectures.

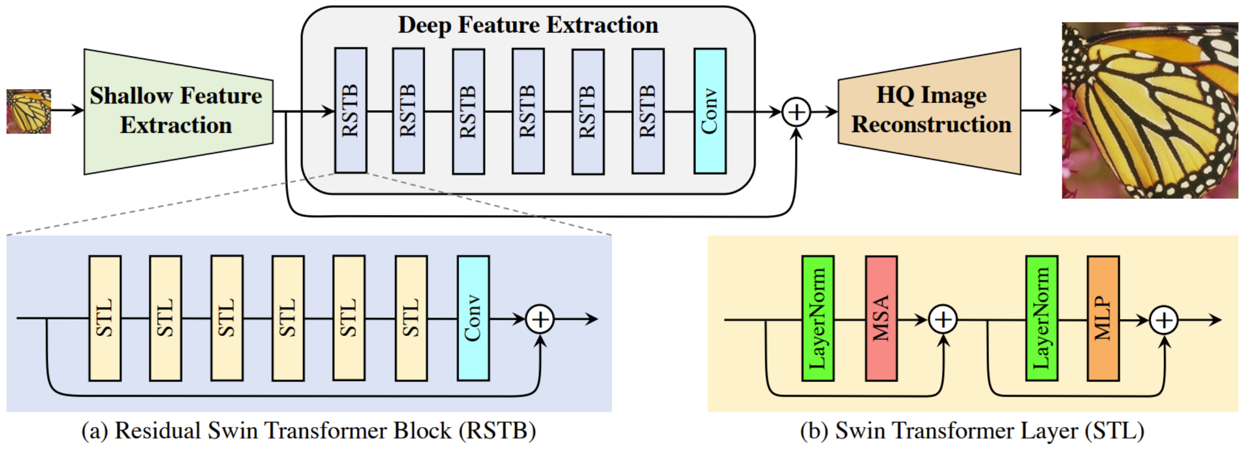


Figure 15: SwinIR Architecture (Courtesy [25])

Search Space

The search space of transformer models is notoriously broad, and transformer models are notoriously expensive to train. Therefore, a well-designed search space is vital to avoid wasting computation and converge on a final architecture quickly. We have adopted the hierarchical strategy in [37] to first train a *supernet* over a coarse-grained search space consisting of possible high-level module distributions. Then, many *subnets* are trained over many fine-grained search spaces within the individual modules. Referencing the architecture of SwinIR depicted in Figure 15, the coarse-grained search space contains the shallow and deep feature extraction modules (e.g. number of RSTBs and the number of STLs, etc), while the fine-grained search space contains each module's architectures (e.g. attention heads for each RSTB, MLP dimension of each STL, etc).

Search Strategy

We wish to use a search algorithm that avoids proposing architectures that are exceedingly costly or overlap heavily with prior architectures. Further, we wish to find a strategy that will converge to a final architecture rapidly while traversing enough of the search space to avoid disregarding good architectures. Evolutionary algorithms (EAs) [36] have demonstrated these qualities on many past NAS studies [50, 51, 52, 53], therefore we employ EAs for both the coarse-grained and fine-grained search spaces.

DistNAS

We designed and implemented a scalable distributed NAS framework based on DeepSpeed and applicable to SwinIR and other transformer architectures. We call this NAS framework DistNAS. There are several key features of DistNAS that are depicted in Figure 16, which are:

- User Configurations: These configurations contain information on the NAS settings (block architecture, #subnets, #subnet architectures, etc), model training settings (#epochs, batch size, learning rate, etc), and distributed training settings (i.e. parallelism and sharding settings for DeepSpeed).
- Controller: The controller manages training by launching and monitoring DeepSpeed training jobs within training nodes
- DeepSpeed Layer: A lightweight interface built atop DeepSpeed to help the Controller launch and manage training jobs.

Each component is described in detail in the following sections:

User Configurations

We seek to maximize user control over the NAS process while minimizing their required knowledge of the underlying framework. Therefore, we adopt the configuration approach and allow users to tune as many clearly-documented knobs as possible without modifying several complex training files to meet their specific NAS training job.

Listing 1 contains a snippet of an example configuration file.

```
# DistNAS User Config

{
   # Distributed Training Settings
   "pipe-parallel-size": 1,
   "model-parallel-size": 1,
   "data-parallel-size": 2
   # Choose the number of subnets to parallelize
   "num-parallel-subnets": 4
   "node-ids": [0,1]  //Which nodes to use for training
   "gpu-ids": [0,1,2,3]
   # Map each subnet to a pair of "node:[list of GPU IDs]
   "subnet-mappings": {0:[0,1], 0:[2,3], 1:[0,1], 1:[2,3]}

   # NAS Settings
   "num-subnets": 1000,
   "strategy": "ea",
   ...

   # Model Settings
   "model": "swinir",
   "scale": 2,
   "n_channels": 3,
   ...

   # Training Settings
   "train_micro_batch_size_per_gpu": 4,
   "num_epochs": 1000,
   "log-interval": 100,
   "checkpointing": True
      "optimizer": {
     "type": "Adam",
     "params": {
       "lr": 0.0002,
     ...
     }
   },
   "fp16": true,
   "zero_optimization": {
    "stage": 1,
  ...
  },
   ...
}
```

Listing 1: Example of user configuration in DistNAS

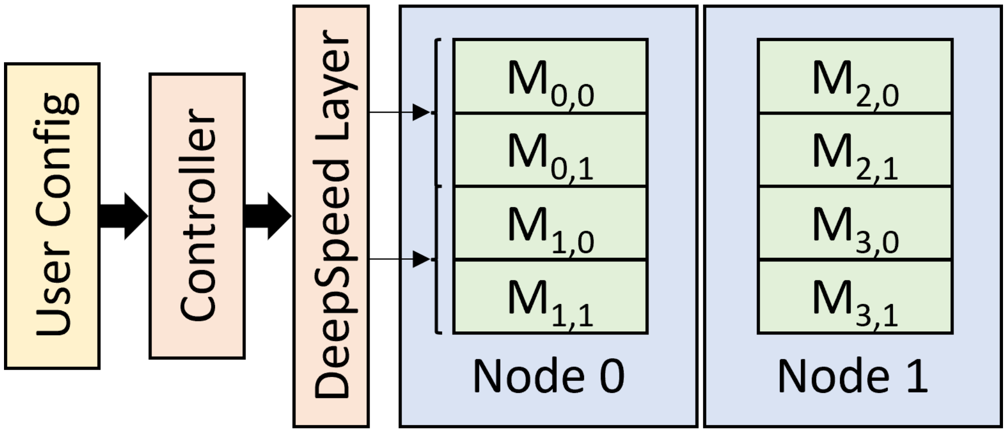
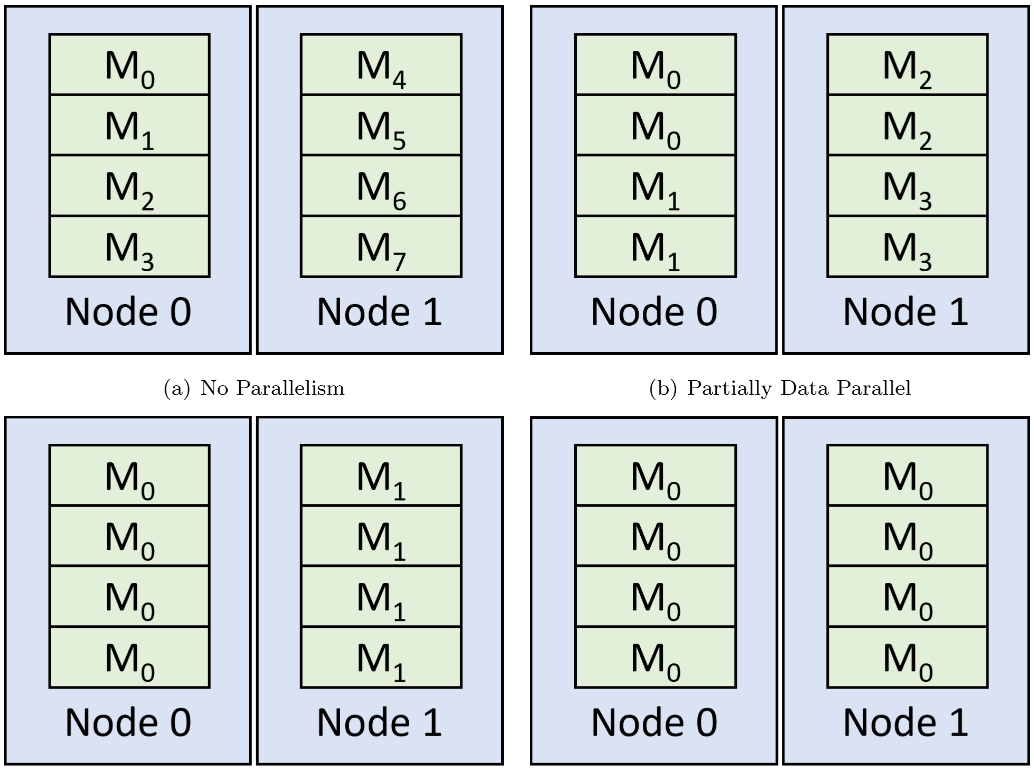


Figure 16: DistNAS Architecture

Controller

Standard NAS implementations train each subnet architecture via data-parallelism (e.g. Figure 17(d), and therefore inherit the massive communication overhead from data parallelism. Instead, DistNAS supports training multiple subnet architectures at once by specifying the desired mapping within the user configuration. Consider the difference between Figures 17(a) and 17(d). Training a single subnet at a time via data-parallelism (i.e. Figure 17(d)) will suffer from communication overhead, but if the model fits in memory we can assign a single subnet to each GPU **and completely remove the need for communication operations among GPUs.** The controller launches a distinct training job on each GPU, and monitors their completion. Note that distributed overhead is small but not zero because each NAS training iteration is synchronous (i.e. all subnets must complete training and evaluating this iteration's subnets before proceeding). Once all subnets within a NAS iteration have been trained and evaluated, the new subnet training jobs are assigned across GPUs. Note that this scheme requires the number of subnets to equal the number of total GPUs. In order to support any number of subnets, any intermediate parallelism scheme (e.g. data-parallelism of a single subnet within each node as in Figure 17(c)) is available by specifying the desired mapping in a user configuration file. Finally, DistNAS inherits DeepSpeed's ability to split large models that cannot fit inside a single GPU's memory via three-dimensional parallelism (e.g. Figure 16).


M_0 M_1 M_2 M_3 Node 0
M_4 M_5 M_6 M_7 Node 1
M_0 M_0 M_1 M_1 Node 0
M_2 M_2 M_3 M_3 Node 1
M_0 M_0 M_0 M_0 Node 0
M_1 M_1 M_1 M_1 Node 1
M_0 M_0 M_0 M_0 Node 0
M_0 M_0 M_0 M_0 Node 1

(a) No Parallelism

(b) Partially Data Parallel

(c) Node-boundary Data Parallelism

(d) Fully Data Parallel

Figure 17: Comparison of parallelization strategies in DistNAS. Each box within a node is a GPU, and M_n refers to training model subnet n on a given GPU

DeepSpeed Layer

The controller launches training jobs on target GPUs via DeepSpeed. In order for the controller to log and synchronize training jobs, a lightweight API between the controller and DeepSpeed was developed. The full API is depicted in Listing 2

```
# Maintains overall training information such as the best subnets, iteration number, etc
class DistNAS_Status
# Stores the parsed configuration file
class DistNAS_Args config

# Returns subnet training status (e.g. "training", "complete", "failed")
status(int gpu_id)
# Launches a DeepSpeed training job
launch(List<int> gpu_ids, DistNAS_Args config)
# Evaluates trained subnets and updates status
evaluate(List<int> gpu_ids, DistNAS_Status status)
# Updates subnet architectures and updates status
update_parameters(DistNAS_Args config, DistNAS_Status status)
# Synchronizes each subnet step
synchronize(List<int> gpu_ids)
# Displays status to user
log(List<int> gpu_ids, DistNAS_Status status)
```

Listing 2: DeepSpeed Layer API

Experiments

Early Investigations: TTSR

We began our evaluation with the original TTSR network architecture using 1 GPU for 30 epochs. The results of the training sessions are reflected in loss curves depicted in Figure 18. We noticed that, according to reconstruction loss, perceptual loss, and adversarial loss of the original model, we are not able to produce the same convergence. In addition, we also evaluated on the architecture discovered by the TTSR-NAS scheme. The results are also attached in Figure 19. Unfortunately, we are still not able to produce any convergence with the architecture found by NAS.

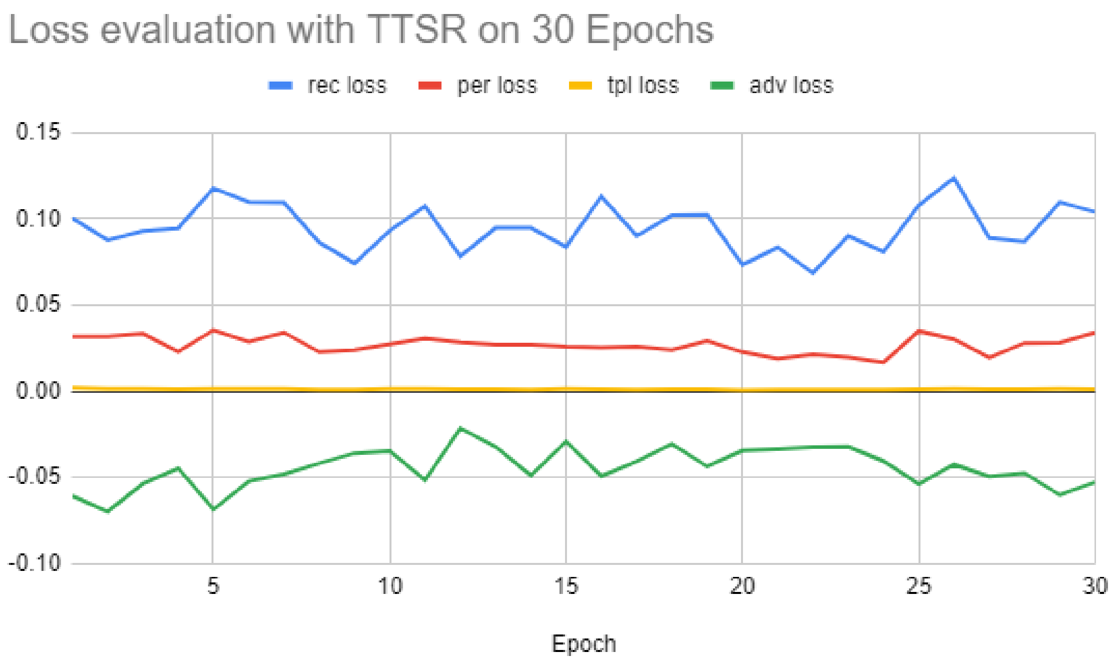


Figure 18: Loss evalutaion with TTSR on 30 Epochs

Given these convergence results coupled with the release of SwinIR [25], we pivoted our investigations to focus on the more flexible Swin transformer architecture.

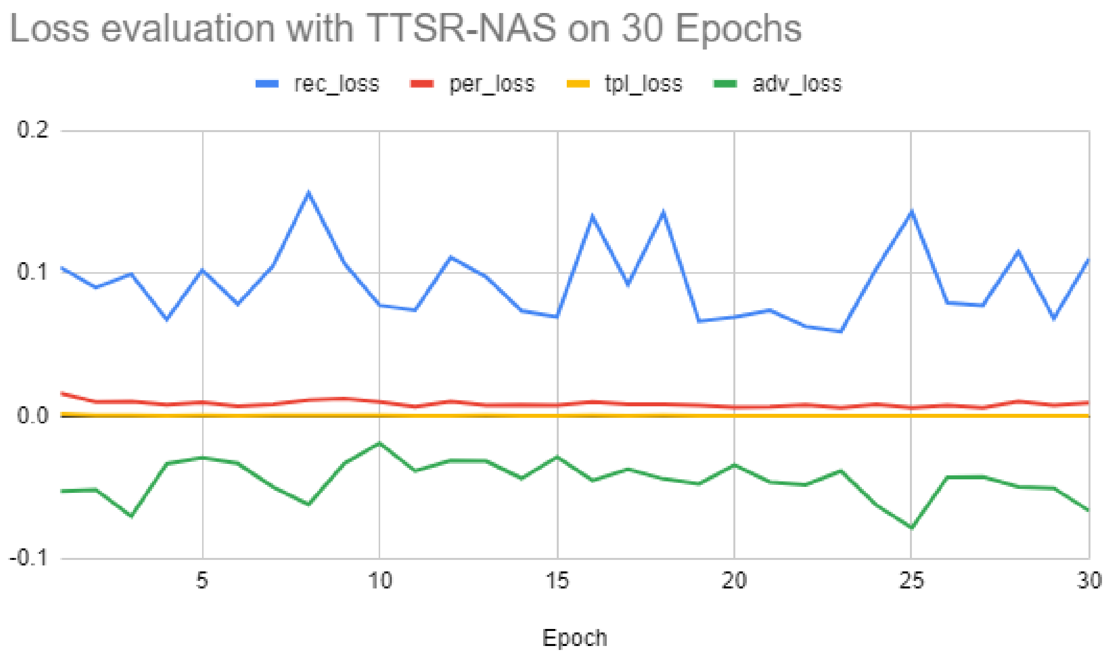


Figure 19: Loss evalutaion with TTSR-NAS on 30 Epochs

DistNAS

We first evaluated our distributed NAS framework (DistNAS) on a variety of system scales within the Lassen supercomputer (Figure 20). The results of training each of the parallelism strategies in Figure 17 with DistNAS on 4-32 V100 GPUs on Lassen are depicted below in Figures 21 and 22. Note that No Parallelism (**NP**) refers to the parallelism strategy in Figure 17(a), Half-Node Data-Parallel (**half-node DP**) refers to Figure 17(b), Node Data-Parallel (**Node DP**) refers to Figure 17(c), and Fully Data-Parallel (**Fully DP**) refers to Figure 17(d).

The key takeaway of Figures 21 and 22 is that each parallelism strategy has a progressively less demanding communication overhead. Consider the case at 8 GPUs. **Node DP** removes the inter-node communication present in **Fully DP**, **half-node DP** removes the X-Bus communication present in **Node DP** [1], and **NP** removes the comunication overhead entirely. Note that the NAS parallelization strategy is dependent on the number of concurrent subnets being trained at a given time. The exact number of concurrent subnets trained for each

[1]Remember that pairs of GPUs are connected via NVLINK on Lassen. See Figure 20

Table 1: Number of subnets trained concurrently for each example parallelism scheme in DistNAS

Parallelism Scheme	**Number of Concurrent Subnets**
Fully DP	1
Node DP	$\frac{\text{Number of GPUs}}{\text{(GPUs/Node)}}$
Half-node DP	$2 \times \frac{\text{Number of GPUs}}{\text{(GPUs/Node)}}$
NP	Number of GPUs

DistNAS example parallelism scheme is depicted below in Table 1. While other parallelism schemes such as mixed (e.g. **Node DP** on node 1, **NP** on node 2) are supported in DistNAS, we didn't evaluate such schemes in this work.

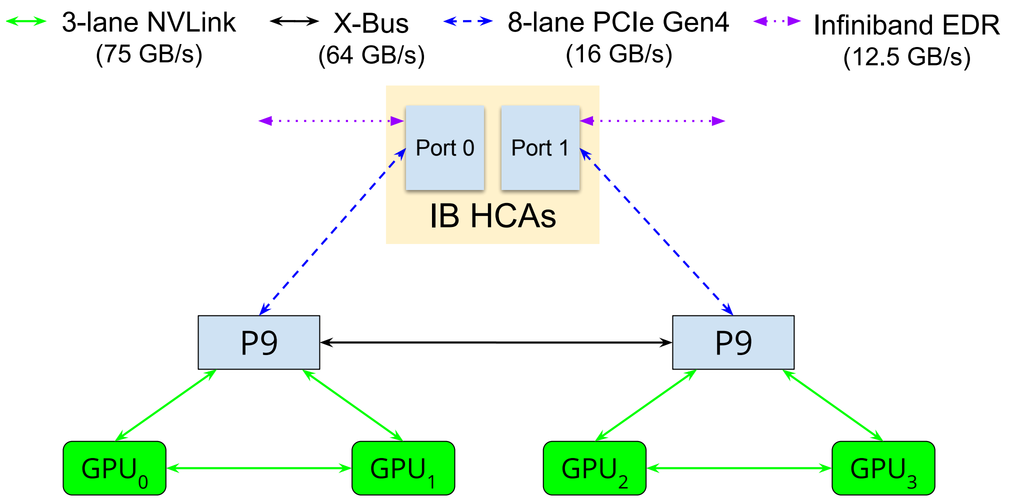


Figure 20: Node topology of the Lassen supercomputer at LLNL

SwinIR NAS

We trained our NAS SwinIR (which we denote as SwiNASIR) using the same hyperparameters as the original SwinIR paper:

We trained on the DIV2K dataset [26] and evaluated on the Set5 [27], Set14 [28], BSD100 [54], Urban100 [55], and Manga109 [56] test datasets. High-quality and low-quality image pairs are generated with a bicubic interpolation kernel. We trained for 500K iterations and set the batch size to 32. The learning rate is initialized to 2e-4 and halved at iterations: [250K,400K,450K,475K]. We used the Adam [57] optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

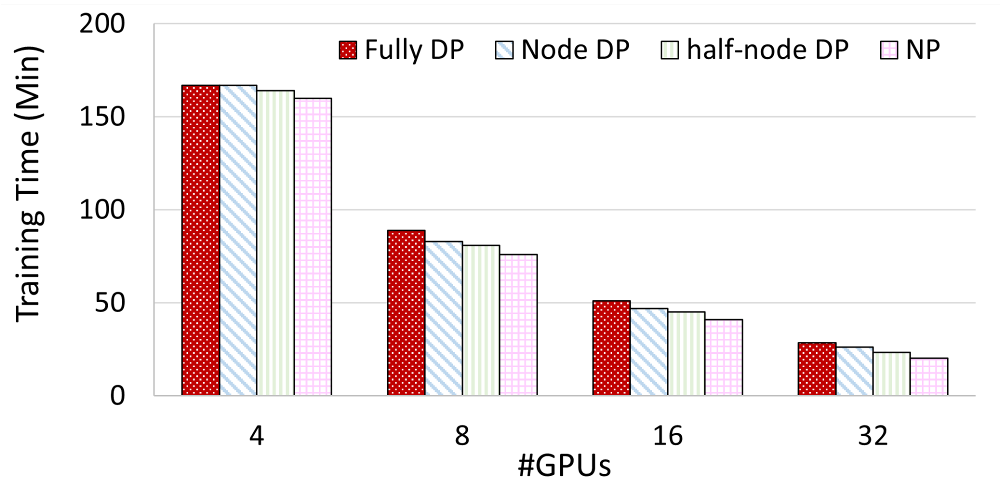


Figure 21: Time to train subnets in DistNAS on a variety of parallelism strategies and system scales

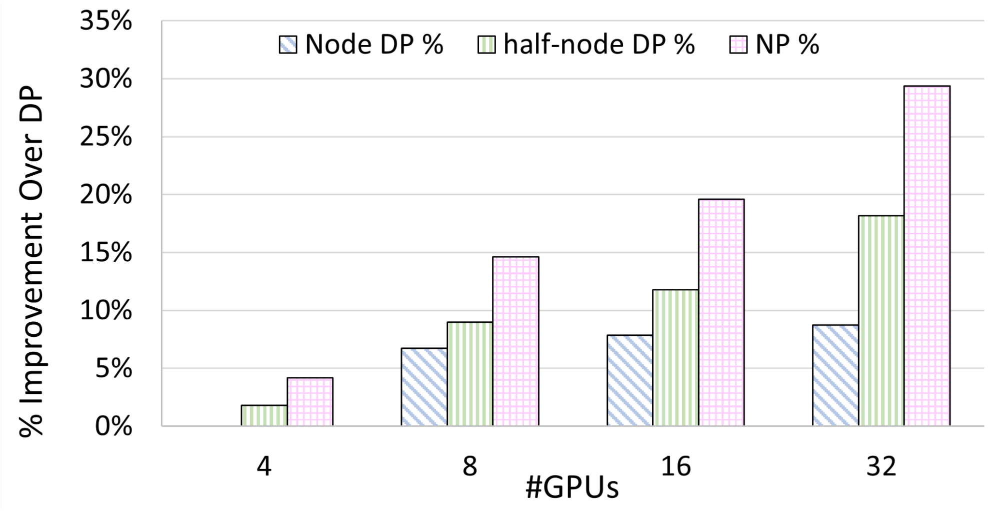


Figure 22: Percent improvement in parallelism strategies in relation to pure data parallelism

Table 2: Quantitative comparison of evaluations (average PSNR/SSIM) with standard SwinIR for classical image SR on benchmark datasets. Best and second best performance are colored red and blue, respectively.

Model	Scale	Training Dataset	Set5		Set14		BSD100		Urban100		Manga109	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
SwinIR	x2	DIV2K	38.35	0.962	34.14	0.9227	32.44	0.903	33.4	0.9393	39.6	0.9792
SwinIR+	x2	DIV2K	38.38	0.9621	34.24	0.9233	32.47	0.9032	33.51	0.9401	39.7	0.9794
SwiNASIR (Ours)	x2	DIV2K	38.42	0.9623	34.27	0.9236	32.48	0.9034	33.56	0.9406	39.7	0.9793
SwiNASIR+ (Ours)	x2	DIV2K	38.45	0.9625	34.32	0.9241	32.52	0.9035	33.65	0.9412	39.9	0.9795
SwinIR	x3	DIV2K	34.89	0.9318	30.77	0.8503	29.37	0.8124	29.29	0.8744	34.74	0.9518
SwinIR+	x3	DIV2K	34.95	0.9322	30.83	0.8511	29.41	0.813	29.42	0.8761	34.92	0.9526
SwiNASIR (Ours)	x3	DIV2K	35.09	0.9324	31.01	0.854	29.51	0.8153	29.65	0.879	35.03	0.9531
SwiNASIR+ (Ours)	x3	DIV2K	35.16	0.9328	31.07	0.8545	29.56	0.8159	29.75	0.8803	35.19	0.954
SwinIR	x4	DIV2K	32.72	0.9021	28.94	0.7914	27.83	0.7459	27.07	0.8164	31.67	0.9226
SwinIR+	x4	DIV2K	32.81	0.9029	29.02	0.7928	27.87	0.7466	27.21	0.8187	31.88	0.9423
SwiNASIR (Ours)	x4	DIV2K	32.9	0.9041	29.07	0.7945	27.9	0.7487	27.39	0.8231	31.99	0.9258
SwiNASIR+ (Ours)	x4	DIV2K	32.95	0.9046	29.717	0.796	27.93	0.7492	27.45	0.825	32.17	0.9269

Conclusion

In this work, we propose SwiNASIR, a NAS scheme built on top of SwinIR transformer architecture. Inspired by GLiT and AutoFormer, we constructed a hierarchical search space. Firstly, a supernet was constructed and trained on a coarse-grained level where high-level module distributions reside. Furthermore, a number of subnets will be built and evaluated on a fine-grained level where detail parameters vary within individual modules. We adopted an evolutionary algorithm for both levels to search and evaluate candidate network architectures.

Given the high computational complexity and demanding hardware requirements of NAS methods, efficient NAS training is a vitally important to advance the field. In this work, we also designed and implemented DistNAS, a scalable distributed NAS training framework that supports DeepSpeed. The framework grants users with fine-grained configuration control over the parameters of NAS processes and manipulates training sessions using DeepSpeed. The training sessions are parallelized over a number of GPUs, enabling the simultaneous training process of many subnets.

We applied the DistNAS training strategy on SwinIR NAS and tested on several datasets (DIV2K, Set5, Set14, BSD100, Urban100, Manga109). With drastically reduced training time consumption and elevated hardware utilization, we are able to produce results better than state-of-the art standards. In the future, we will further extend our NAS framework towards other parallelism schemes and vision transformer tasks.

References

[1] Shawn Strande et al. “Expanse: Computing without Boundaries: Architecture, Deployment, and Early Operations Experiences of a Supercomputer Designed for the Rapid Evolution in Science and Engineering”. In: *Practice and Experience in Advanced Research Computing.* New York, NY, USA: Association for Computing Machinery, 2021. ISBN: 9781450382922. URL: `https://doi.org/10.1145/3437359.3465588`.

[2] Jia Deng et al. “ImageNet: A large-scale hierarchical image database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition.* 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[3] Hayit Greenspan. “Super-Resolution in Medical Imaging”. In: *Comput. J.* 52 (Jan. 2009), pp. 43–63. DOI: `10.1093/comjnl/bxm075`.

[4] Jithin Saji Isaac and R. K. Kulkarni. “Super resolution techniques for medical image processing”. In: *2015 International Conference on Technologies for Sustainable Development (ICTSD)* (2015), pp. 1–6.

[5] Yawen Huang, Ling Shao, and Alejandro F. Frangi. *Simultaneous Super-Resolution and Cross-Modality Synthesis of 3D Medical Images using Weakly-Supervised Joint Convolutional Sparse Coding.* 2017. DOI: `10.48550/ARXIV.1705.02596`. URL: `https://arxiv.org/abs/1705.02596`.

[6] Pejman Rasti et al. “Convolutional Neural Network Super Resolution for Face Recognition in Surveillance Monitoring”. In: *AMDO.* 2016.

[7] Liangpei Zhang et al. “A Super-Resolution Reconstruction Algorithm for Surveillance Images”. In: *Signal Process.* 90.3 (Mar. 2010), pp. 848–859. ISSN: 0165-1684. DOI: `10.1016/j.sigpro.2009.09.002`. URL: `https://doi.org/10.1016/j.sigpro.2009.09.002`.

[8] Z. Wang, J. Chen, and S. H. Hoi. “Deep Learning for Image Super-Resolution: A Survey”. In: *IEEE Transactions on Pattern Analysis & Machine Intelligence* 43.10 (Oct. 2021), pp. 3365–3387. ISSN: 1939-3539. DOI: `10.1109/TPAMI.2020.2982166`.

[9] Dengxin Dai et al. “How Useful Is Image Super-resolution to Other Vision Tasks?” In: (Sept. 2015).

[10] Muhammad Haris, Greg Shakhnarovich, and Norimichi Ukita. *Task-Driven Super Resolution: Object Detection in Low-resolution Images.* 2018. DOI: `10.48550/ARXIV.1803.11316`. URL: `https://arxiv.org/abs/1803.11316`.

[11] Mehdi S. M. Sajjadi, Bernhard Schölkopf, and Michael Hirsch. *EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis.* 2016. DOI: `10.48550/ARXIV.1612.07919`. URL: `https://arxiv.org/abs/1612.07919`.

[12] Yongqiang Zhang et al. "Multi-Task Generative Adversarial Network for Detecting Small Objects in the Wild". In: *Int. J. Comput. Vision* 128.6 (June 2020), pp. 1810–1828. ISSN: 0920-5691. DOI: 10.1007/s11263-020-01301-6. URL: https://doi.org/10.1007/s11263-020-01301-6.

[13] R. Keys. "Cubic convolution interpolation for digital image processing". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 29.6 (1981), pp. 1153–1160. DOI: 10.1109/TASSP.1981.1163711.

[14] Michal Irani and Shmuel Peleg. "Improving resolution by image registration". In: *CVGIP: Graphical models and image processing* 53.3 (1991), pp. 231–239.

[15] R Tsai. "Multiframe image restoration and registration". In: *Advance Computer Visual and Image Processing* 1 (1984), pp. 317–339.

[16] Jian Sun, Zongben Xu, and Harry Shum. "Image super-resolution using gradient profile prior". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition* (2008), pp. 1–8.

[17] Nhat Nguyen, Peyman Milanfar, and Gene Golub. "A computationally efficient superresolution image reconstruction algorithm". In: *IEEE transactions on image processing* 10.4 (2001), pp. 573–583.

[18] A Murat Tekalp, Mehmet K Ozkan, and M Ibrahim Sezan. "High-resolution image reconstruction from lower-resolution image sequences and space-varying image restoration". In: *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 3. IEEE. 1992, pp. 169–172.

[19] Michael K Ng and Andy M Yip. "A fast MAP algorithm for high-resolution image reconstruction with multisensors". In: *Multidimensional Systems and Signal Processing* 12.2 (2001), pp. 143–164.

[20] Sina Farsiu et al. "Robust shift and add approach to superresolution". In: *Applications of Digital Image Processing XXVI*. Vol. 5203. SPIE. 2003, pp. 121–130.

[21] Ce Liu and Deqing Sun. "On Bayesian adaptive video super resolution". In: *IEEE transactions on pattern analysis and machine intelligence* 36.2 (2013), pp. 346–360.

[22] Bee Lim et al. "Enhanced Deep Residual Networks for Single Image Super-Resolution". In: *CoRR* abs/1707.02921 (2017). arXiv: 1707.02921. URL: http://arxiv.org/abs/1707.02921.

[23] Chao Dong et al. "Image Super-Resolution Using Deep Convolutional Networks". In: *CoRR* abs/1501.00092 (2015). arXiv: 1501.00092. URL: http://arxiv.org/abs/1501.00092.

[24] Fuzhi Yang et al. "Learning Texture Transformer Network for Image Super-Resolution". In: *CVPR*. June 2020.

[25] Jingyun Liang et al. "Swinir: Image restoration using swin transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1833–1844.

[26] Eirikur Agustsson and Radu Timofte. "NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, pp. 1122–1131. DOI: `10.1109/CVPRW.2017.150`.

[27] Marco Bevilacqua et al. "Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding". In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2012, pp. 135.1–135.10. ISBN: 1-901725-46-4. DOI: `http://dx.doi.org/10.5244/C.26.135`.

[28] Roman Zeyde, Michael Elad, and Matan Protter. "On Single Image Scale-Up Using Sparse-Representations". In: vol. 6920. June 2010, pp. 711–730. ISBN: 978-3-642-27412-1. DOI: `10.1007/978-3-642-27413-8_47`.

[29] Ashish Vaswani et al. "Attention Is All You Need". In: (June 2017).

[30] Yulun Zhang et al. "Residual dense network for image super-resolution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2472–2481.

[31] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. DOI: `10.48550/ARXIV.1607.06450`. URL: `https://arxiv.org/abs/1607.06450`.

[32] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: `10.48550/ARXIV.2010.11929`. URL: `https://arxiv.org/abs/2010.11929`.

[33] Ze Liu et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *CoRR* abs/2103.14030 (2021). arXiv: `2103.14030`. URL: `https://arxiv.org/abs/2103.14030`.

[34] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[35] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search". In: ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Springer, 2019. Chap. 3, pp. 69–86.

[36] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. "Neural Architecture Search: A Survey". In: (2018). DOI: `10.48550/ARXIV.1808.05377`. URL: `https://arxiv.org/abs/1808.05377`.

[37] Boyu Chen et al. "GLiT: Neural Architecture Search for Global and Local Image Transformer". In: *CoRR* abs/2107.02960 (2021). arXiv: `2107.02960`. URL: `https://arxiv.org/abs/2107.02960`.

[38] Minghao Chen et al. "Autoformer: Searching transformers for visual recognition". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 12270–12280.

[39] RW Gerchberg. “Super-resolution through error energy reduction”. In: *Optica Acta: International Journal of Optics* 21.9 (1974), pp. 709–720.

[40] P De Santis and F Gori. “On an iterative method for super-resolution”. In: *Optica Acta: International Journal of Optics* 22.8 (1975), pp. 691–695.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS’12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105.

[42] Chao Dong, Chen Change Loy, and Xiaoou Tang. *Accelerating the Super-Resolution Convolutional Neural Network*. 2016. DOI: 10.48550/ARXIV.1608.00367. URL: https://arxiv.org/abs/1608.00367.

[43] Quentin Anthony et al. “Scaling Single-Image Super-Resolution Training on Modern HPC Clusters: Early Experiences”. In: *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 2021, pp. 923–932. DOI: 10.1109/IPDPSW52791.2021.00143.

[44] Esteban Real et al. “Regularized Evolution for Image Classifier Architecture Search”. In: *CoRR* abs/1802.01548 (2018). arXiv: 1802.01548. URL: http://arxiv.org/abs/1802.01548.

[45] Xiu Su et al. “Vision Transformer Architecture Search”. In: *CoRR* abs/2106.13700 (2021). arXiv: 2106.13700. URL: https://arxiv.org/abs/2106.13700.

[46] Yuanbiao Gou et al. “CLEARER: Multi-Scale Neural Architecture Search for Image Restoration”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 17129–17140. URL: https://proceedings.neurips.cc/paper/2020/file/c6e81542b125c36346d9167691b8bd09-Paper.pdf.

[47] Ke Sun et al. *High-Resolution Representations for Labeling Pixels and Regions*. 2019. DOI: 10.48550/ARXIV.1904.04514. URL: https://arxiv.org/abs/1904.04514.

[48] Yanhong Zeng et al. *Learning Pyramid-Context Encoder Network for High-Quality Image Inpainting*. 2019. DOI: 10.48550/ARXIV.1904.07475. URL: https://arxiv.org/abs/1904.07475.

[49] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search*. 2018. DOI: 10.48550/ARXIV.1806.09055. URL: https://arxiv.org/abs/1806.09055.

[50] Peter J. Angeline, Gregory M. Saunders, and Jordan B. Pollack. “An evolutionary algorithm that constructs recurrent neural networks”. In: *IEEE TRANSACTIONS ON NEURAL NETWORKS* 5 (1994), pp. 54–65.

[51] Kenneth O. Stanley and Risto Miikkulainen. "Evolving Neural Networks through Augmenting Topologies". In: *Evolutionary Computation* 10.2 (2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[52] Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. "A Hypercube-Based Encoding for Evolving Large-Scale Neural Networks". In: *Artificial Life* 15.2 (2009), pp. 185–212. DOI: 10.1162/artl.2009.15.2.15202.

[53] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An Empirical Exploration of Recurrent Network Architectures". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2342–2350. URL: https://proceedings.mlr.press/v37/jozefowicz15.html.

[54] D. Martin et al. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics". In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 2. 2001, 416–423 vol.2. DOI: 10.1109/ICCV.2001.937655.

[55] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. "Single image super-resolution from transformed self-exemplars". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 5197–5206. DOI: 10.1109/CVPR.2015.7299156.

[56] Azuma Fujimoto et al. "Manga109 Dataset and Creation of Metadata". In: *Proceedings of the 1st International Workshop on CoMics ANalysis, Processing and Understanding*. MANPU '16. Cancun, Mexico: Association for Computing Machinery, 2016. ISBN: 9781450347846. DOI: 10.1145/3011549.3011551. URL: https://doi.org/10.1145/3011549.3011551.

[57] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.