

**Deep Learning for Computer Vision**  
**Professor. Vineeth N Balasubramanian**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Hyderabad**  
**Lecture No. 78**  
**Neural Architecture Search**

As the last advanced topic for this week, we will talk about Neural Architecture Search, another contemporary topic that deals with searching for the right Neural Network architecture for a given problem. It is, it should straightaway appeal to you that this is a very important problem considering the various hyper parameters that one has in designing Neural Networks.

(Refer Slide Time: 00:44)

**Neural Architecture Search: Why?**

**Canziani et al (2017)**

**Complex hand-engineered layers from Inception-V4 (Szegedy et al., 2017)**


- Most popular and successful model architectures designed by human experts
- Requires hundreds of hours of arbitrary training and testing and hyperparameter tuning
- However, it doesn't mean we explored the entire network architecture space or that we found an optimal solution
- Can we adopt a systematic and automatic way of learning high-performance model architectures? **Solution:** Neural Architecture Search

Credit: Nikhil Naik, Neural Architecture Search: A Tutorial, 2019  
Vineeth N B (IIT-H) §12.5 Neural Architecture Search 2 / 19


To get a better understanding of why we need this, we have seen over the duration of this course that accuracy and performance has improved with better design of architectures. And this often requires human experts to spend hundreds of hours on arbitrary training and testing and hyper parameter tuning. This, in fact, does not even mean that the entire space of possible architectures has been searched exhaustively, to find the right architecture.

Often, researchers use architectures that were proposed earlier, and adapt a few elements to arrive at an architecture for a given problem. That brings the question, can we adopt the systematic and automatic way of learning high performance model architectures? The solution that we are going to talk about is Neural Architecture Search.

(Refer Slide Time: 01:54)



## Neural Architecture Search (NAS): Brief History




### Early Work

- Neuroevolution: Evolutionary algorithms (e.g., Miller et al., 89; Schaffer et al., 92; Stanley & Miikkulainen, 02; Verbancsics & Harguess, 13)
- Random search (e.g., Pinto et al., 09; Bergstra & Bengio, 12)
- Bayesian optimization for architecture and hyperparameter tuning (e.g., Snoek et al., 12; Bergstra et al., 13; Domhan et al., 15)

### Renewed Interest (2017-)

- Zoph and Le, Neural Network Architecture Search with Reinforcement Learning, ICLR'17.
- Baker et al., Designing Neural Network Architectures using Reinforcement Learning, ICLR'17.





Vineth N B (IIT-H) §12.5 Neural Architecture Search 3 / 19

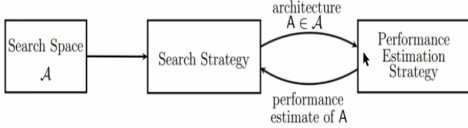
Neural Architecture Search, perhaps not in the same name has been around for some time. In very early phases, one would use Evolutionary algorithms such as, say genetic algorithms, to be able to design neural network architectures. This led to exploration of random search for architectures or even Bayesian optimization for architectures and hyper parameter tuning over the last decade.

Over the last 2 to 3 years, Neural Architecture Search has emerged as an independent important problem. It primarily started with searching for these architectures using principles from reinforcement learning.

(Refer Slide Time: 02:42)

 **NAS: General Problem Setup**






```
graph LR;
  A[Search Space A] --> B[Search Strategy];
  B --> C[Performance Estimation Strategy];
  C --> B;
```

- **Search Space:** Defines which architectures can be represented in principle
- **Search Strategy:** Details how to explore search space (which is often exponentially large or even unbounded)
- **Performance Estimation Strategy:** Estimating an architecture's performance - standard training and validation of architecture on data may be computationally expensive and limits number of architectures that can be explored

Credit: Elksen et al., *Neural Architecture Search: A Survey*, 2018

 Vineeth N B (IIT-H) §12.5 Neural Architecture Search 4 / 19

The general problems set up in NAS or Neural Architecture Search is given by, you have a search space, which defines the space of all possible architectures that you want to look for, for a given problem. You have a search strategy to look for an architecture within the search space, often the search space can be extremely large. And we need a performance estimation strategy to try to estimate an architecture's performance. Using standard training and validation performance could be computationally expensive to be able to try various kinds of architectures.

So, this also needs to be chosen carefully. So, you have a search space, a search strategy, which results in an architecture whose performance is estimated based on the performance estimate, the search strategy looks further for a newer architecture and this loop continues until a desired performance is met on a consistent basis.

(Refer Slide Time: 03:57)



## Neural Architecture Search Space



- Neural networks represent a function that transforms input variables  $x$  to output variables  $\hat{y}$  through a series of operations
- Recall computational graphs (W8P1); each node  $z^{(k)} \in Z$  represents a tensor and is associated with an operation  $o^{(k)} \in O$  on its parent nodes  $I^k$
- Computation at a node  $k$ :

$$z^{(k)} = o^{(k)}(I^{(k)})$$

where operations includes unary operations such as convolutions, pooling, activation functions or  $n$ -ary operations such as concatenation or addition

- **NAS space**: Subspace of this general definition of neural architectures



Vineeth N B (IIT-H)

§12.5 Neural Architecture Search

5 / 19

As we just mentioned, the search space of NAS methods are extremely important. So, how is the search space defined? One can view Neural Networks as a function that transforms input variables to output variables through a series of operations. So, if you look at Neural Networks as computational graphs, recall the lecture we had earlier, each neural network node represents a tensor and this is associated with an operation on its parent nodes  $I$ .

So, you can say that the computation at a node  $k$  is given by  $x^{(k)}$  is the operation at  $k$  of its parent nodes  $I^{(k)}$ . What are the operations? The operations could be convolutions, pooling, activation functions, or even  $n$ -ary operations like concatenation or element wise addition or simple addition, so on and so forth. NAS space is generally a subspace of this general definition of neural architectures. So, what kind of search spaces do NAS methods use?

(Refer Slide Time: 05:16)

The slide is titled "NAS Space: Global vs Cell-based<sup>1</sup>". It is divided into two main sections: "Global Search Space" and "Cell-based Search Space".

- Global Search Space**
  - Large degrees of freedom regarding arrangement of operations
  - Allowed operations examples
    - convolutions, pooling, dense layers (FCs) with activation, global average pooling
  - Constraints examples
    - pooling as first operation; dense layers (FCs) before convolution operations
  - Rigid, impractical to scale and transfer
- Cell-based Search Space**
  - Many effective handcrafted architectures designed with repetitions of fixed structures
  - Network constructed by repeating a **cell** structure, a small directed acyclic graph representing a feature transformation
  - E.g. **NASNet search space**: Learns two types of cells:
    - Normal Cell**: Input and output feature maps have same dimension
    - Reduction Cell**: Output feature map has width and height reduced by half

<sup>1</sup>Zoph et al, Learning Transferable Architectures for Scalable Image Recognition, CVPR 2018  
Vineeth N B (IIT-H) §12.5 Neural Architecture Search 6 / 19

Broadly speaking, one could divide this into two kinds, a Global Space versus a Cell based search space. In a Global Search Space, the method allows any kind of an architecture. So, you have a large degree of freedom in arranging the operations. You have a set of allowed operations, such as convolutions, pooling, dense layers, global average pooling, with different hyper parameter settings like number of filters, filter width, filter height, so on and so forth.

But there are also a few constraints that are specified. For example, you may not want to start the neural network with pooling as the first operation, you may not want to have dense layers, before the first set of convolution operations, so on and so forth. In Global search, the issue is that the search space is a bit rigid, and it could be impractical to scale and transfer considering the extensiveness of this search space.

On the other hand, Cell based Search Space, which was first introduced in this work in CVPR of 2018. They introduced a method called NASNet. The idea here is that a lot of handcrafted architectures are actually designed with repetitions of specific structures. We have seen residual blocks in ResNets. This idea is taken forward here, by defining a cell structure and constructing a network architecture by repeating the cell structure.

What is a cell? You could look at it as a small directed acyclic graph that represents a transformation or a series of transformations. In NASNet, which was the method introduced in

CVPR of 2018, the method learns two kinds of cells, a Normal cell where the dimension of the input and the output of that transformation is retained, example would be Convolution. And then a Reduction cell where the output feature map has width and height reduced for instance, like pooling operations.

(Refer Slide Time: 07:44)

**Global Search Space**

Simplest example: a *chain-structured search space* as shown below

(a) Baker et al. (2017)

$$z^{(k)} = \sigma^{(k)} \left( \left\{ z^{(k-1)} \right\} \right)$$

(b) Zoph and Le (2017)

$$z^{(k)} = \sigma^{(k)} \left( \left\{ z^{(k-1)} \right\} \cup \left\{ z^{(i)} \mid i < k-1 \right\} \right)$$

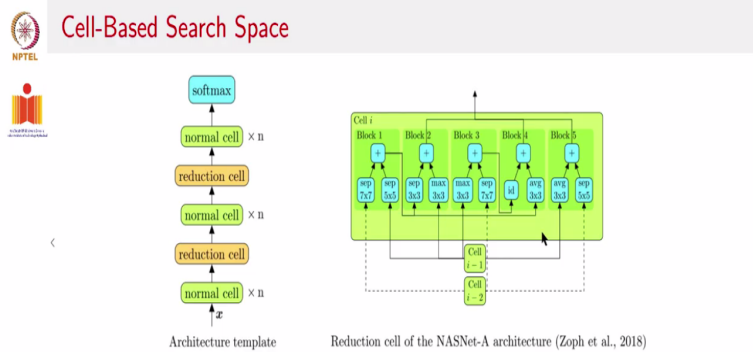
(Skip Connections)

Credit: Wistuba et al, A Survey on Neural Architecture Search, 2019

Vineeth N B (IIT-H)
§12.5 Neural Architecture Search
7 / 19

Here is an example of a Global Search Space where you could have a chain-structured search space as given below, where given an input you have a series of operations and the operations have to be searched for from your search space from your universe of operations. And another variant is where you have a skip connection in your chain-structured search space.

(Refer Slide Time: 08:15)



**NASNet:** While cell topology is maintained across network, its hyperparameters are often varied; merging operation is concatenation

Image Source: Wistuba et al., A Survey on Neural Architecture Search, 2019



Vineeth N. B. (IIT-H)

§12.5 Neural Architecture Search

8 / 19

On the other hand, in a Cell based Search Space, an architecture template is defined. For example, here you see an input, which first goes through a normal cell, then a reduction cell, then a normal cell, reduction cell, and a normal cell and softmax. In very simple terms, you could assume that this normal cell was convolutional plus batch norm and reduction cell could have been pooling in a standard AlexNet context.

But now in NASnet, each of these cells is searched for within within a universe of operations. So, here is the reduction cell of one of the architectures of NASNet called NASNet-A, where you see 5 blocks with different kinds of operations that have been mentioned on each block, which is obtained after the architecture search.

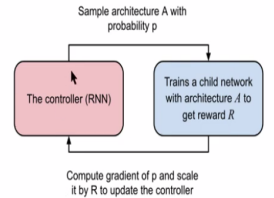
(Refer Slide Time: 09:15)



## NAS with Reinforcement Learning (RL)<sup>2</sup>



- A controller, which proposes child architecture, is implemented as a RNN; outputs a sequence of tokens that configure network architecture
- Controller trained as a RL task using REINFORCE (Monte-Carlo policy gradient)
  - **Action space:** List of tokens  $T (a_{1:T})$  for defining a child network predicted by controller
  - **Reward:** Accuracy of a child network,  $R$ .
  - **Loss:** NAS optimizes controller parameters  $\theta$  with a REINFORCE loss



$$\nabla_{\theta} J(\theta) = \sum_{t=1}^T \mathbb{E}[\nabla_{\theta} \log P(a_t | a_{1:(t-1)}; \theta) R]$$



<sup>2</sup>Zoph and Le, Neural Network Architecture Search with Reinforcement Learning, ICLR 2017.

Vineeth N B (IIT-H)

§12.5 Neural Architecture Search

9 / 19

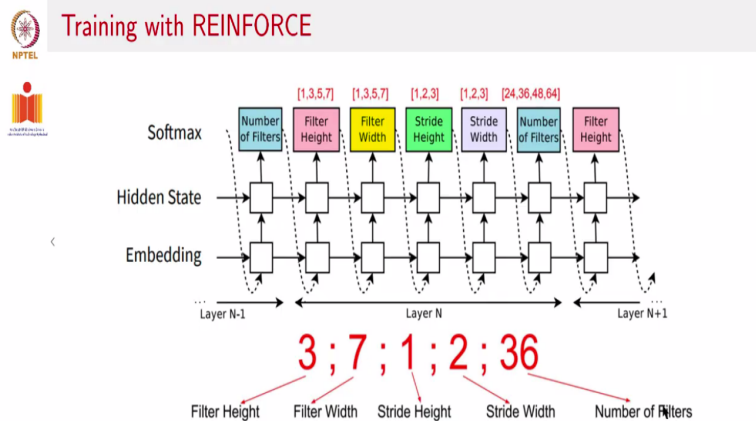
As we just mentioned, some time ago, the space of Neural Architecture Search became popular in 2017 with the introduction of NAS through Reinforcement Learning. While we have not covered reinforcement, reinforcement learning as a topic in this course. We will speak at a very high level to explain how this NAS method works. So, this method proposes the use of a controller, which proposes a child architecture, and the controller is implemented as a Recurrent Neural Network or an RNN which outputs a sequence of tokens that configure the network architecture.

The controller RNN is trained, similar to a reinforcement learning task using a popular algorithm called reinforce, which uses Monte-Carlo policy gradients. The action space in this case is the list of tokens for defining a child network. The reward is the accuracy of the child network. And the loss is the reinforce loss given by this term. Since we have not covered reinforcement learning in this course, we would not go further deeper than this. But if you are interested, you can read this paper for more details.

Here is the schematic you have the controller which samples in architecture with probability  $b, p$ . You train a child network with that architecture to get an accuracy, which is here a reward in the context of reinforcement learning. And then the policy gradient is computed to update the controller RNN.



(Refer Slide Time: 11:09)



Credit: Nikhil Naik, Neural Architecture Search: A Tutorial, 2019



Vineeth N. B. (IIT-H)


§12.5 Neural Architecture Search


10 / 19

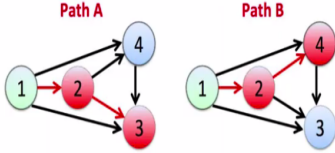
Here is an example of how the controller RNN works, which is trained using reinforce. So, you can see that at each step of the RNN across different layers, multiple hyper parameters, such as number of filters, filter height, filter width, stride height and stride width are predicted for that layer, which is then passed on to the next layer, whose number of filters, filter height and other hyper parameters are predicted.

So, the prediction could be something like 3, 7, 1, 2, 36, where the first parameter says the filter height. The next one gives the filter width, the stride height, the stride width, the number of filters, so on and so forth.

(Refer Slide Time: 11:58)

 **How To Make NAS More Efficient?**






- Models defined by **path A** and **path B** are trained independently
- Instead, can we treat all model trajectories as sub-graphs of a single directed acyclic graph?
- **Efficient NAS (ENAS)**<sup>3</sup>: aggressively shares parameters among child models

Credit: Nikhil Naik, Neural Architecture Search: A Tutorial, 2019

<sup>3</sup>Pham et al., Efficient Neural Architecture Search via Parameters Sharing, ICML 2018


Vineeth N B (IIT-H) §12.5 Neural Architecture Search 11 / 19




One of the limitations of NAS based methods is that when you view your entire search space as a graph, with each path as a certain sequence of operations, which could be an architecture, each of these paths are looked at, and trained independently to be able to find which architecture suits a particular task. So, given the overall search space, the path 123 which could be say a convolution followed by a pooling, and a path 124, which could be convolution followed by a batch norm for instance each of these are trained independently, and their performances are assessed to see which of those operations or sequence of operations suits for a given task.

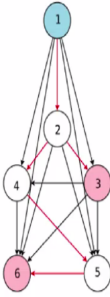
A recent method called Efficient NAS proposed in ICML of 2018 asks the question, can we treat all models trajectories as sub graphs of a single directed acyclic graph? And the answer is, yes. An Efficient NAS shows a way to do this by sharing parent parameters across the child models.

(Refer Slide Time: 13:18)

 **Efficient NAS with Weight Sharing**



- All sampled architecture graphs viewed as *subgraphs* of a larger *supergraph*
- Graph represents entire search space while red arrows define a model in the search space, decided by an RNN controller (trained with REINFORCE)
- Weights of controller and a part of over-parameterized network are alternately updated
- ENAS achieved 2.89% test error on CIFAR-10, took less than 16 hours to search (significantly lesser than other NAS models)



Credit: Pham et al., Efficient Neural Architecture Search via Parameters Sharing. ICML 2018.



Vineeth N B (IIT-H)

§12.5 Neural Architecture Search

12 / 19

So, in this case, if what you see here is a supergraph of all possible sequences of operations, each architecture becomes a subgraph or a trajectory in this graph. So, you could look at these red arrows here as one sequence of operations corresponding to one architecture. And that sequence of operations is decided by an RNN controller, which is trained using reinforcement learning.

How is this trained? The weights of the controller and that the weights corresponding to the chosen paths are trained in an alternate manner and this leads to the final training. And they show that this approach gives about 2.89 percent test error on the CIFAR-10 dataset, which is close to state of the art and took less than 16 hours to search for a right architecture, which is significantly lesser than other NAS based models.

Especially NAS based models, based on reinforcement learning can take many days to search for the right architecture. One other observation to point out here is a lot of the state of the art models today in classification in detection are all based on architectures obtained through Neural Architecture Search.

(Refer Slide Time: 14:54)



## Differentiable Architecture Search: Gradient-based NAS<sup>4</sup>



- Introduced binary variables in  $\{\alpha_{i,j,k}\}$  to make search space continuous. This simplifies definition to:

$$z^{(k)} = \sum_{i \in I^{(k)}} \sum_{j \in O^{(i)}} \alpha_{i,j,k} \cdot o^{(j)}(z^{(i)}) \quad \text{with } \alpha_{i,j,k} \in \{0,1\}$$

- So far, the assumption: every operation is either part of the network or not
- This method relaxes categorical choice of a particular operation as a softmax over all operations

$$\bar{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp(\alpha_{ij}^o)}{\sum_{o' \in O} \exp(\alpha_{ij}^{o'})} o(x)$$

- This reduces search to learning a set of mixing probabilities  $\alpha$




<sup>4</sup>Liu et al, DARTS: Differentiable Architecture Search, ICLR 2019


A more recent development called DARTS, a Differentiable Architecture Search was proposed in ICLR, of 2019. And the key idea here was to define the search space using a set of binary variables on the operations. So, your output of layer  $z^{(k)}$  is now an operation  $O$  multiplied by a binary variable into in terms of the input variables  $z^{(i)}$  from the previous layer. The key insight in this approach is that, until this method in all methods, you could either have an operation or not have an operation.

But this method says, why do not we relax this and make  $\alpha$  something that is learnable. And hence, it relaxes this categorical choice of choosing an operation as a softmax, over all possible operations. So, the operation is not just binary in terms of its existence, it is now a softmax that you have over the space of all possible operations. This allows us to back propagate through the choice of the operation itself.

(Refer Slide Time: 16:21)



### Differentiable Architecture Search: Gradient-based NAS<sup>5</sup>




- Uses an alternating (bilevel) optimization method:
  - learn model parameters  $w$  by minimizing loss on training set
  - learn structural parameters  $\alpha$  by minimizing loss on validation set

$$\min_{\alpha} \mathcal{L}_{\text{validate}}(w^*(\alpha), \alpha)$$
$$\text{s.t. } w^*(\alpha) = \arg \min_w \mathcal{L}_{\text{train}}(w, \alpha)$$

- Final architecture chosen based on:
$$o^{(i,j)} = \arg \max_{o \in O} \alpha_o^{(i,j)}$$
- Elegant solution that makes all parameters differentiable!

<sup>5</sup>Liu et al, DARTS: Differentiable Architecture Search, ICLR 2019



Vineth N B (IIT-H) §12.5 Neural Architecture Search 14 / 19

What does this mean? How do you train? So, in DARTS, an alternating bilevel optimization method is proposed similar to Efficient NAS, where in the first iteration, the model parameters  $w$  of a specific architecture is trained by minimizing loss on a training set. And in the second step, the structural parameters  $\alpha$ , which weight each of your operations are learned using by minimizing another loss on a validation set, which is defined this way, where  $w^*(\alpha)$  is the minimum loss of  $w$  on the training set.

So, you use those weights to be able to get a validation loss, which is then used to minimize the value of  $\alpha$ . So, the final architecture is chosen based on the sequence of operations that maximizes  $\alpha$  or the operation at that step that gives you the maximum  $\alpha$ . This is an elegant solution that makes all parameters differentiable and entire, and thus the entire architecture search as a differentiable procedure.

(Refer Slide Time: 17:42)

### Architecture Transferability of NAS Networks

Model	Params	x+	1/5-Acc (%)
Inception V3	23.8M	5.72B	78.8 / 94.4
Xception	22.8M	8.37B	79.0 / 94.5
Inception ResNet V2	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 (64x4d)	83.6M	31.5B	80.9 / 95.6
PolyNet	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131	79.5M	32.0B	81.5 / 95.8
Squeeze-Excite-Net	145.8M	42.3B	82.7 / 96.2
GeNet-2*	156M	-	72.1 / 90.4
Block-QNN-B (N=3)*	-	-	75.7 / 92.6
Hierarchical (2, 64)*	64M	-	79.7 / 94.8
PNASNet-5 (4, 216)	86.1M	25.0B	82.9 / 96.1
NASNet-A (6, 168)	88.9M	23.8B	82.7 / 96.2
AmoebaNet-B (6, 190)*	84.0M	22.3B	82.3 / 96.1
AmoebaNet-A (6, 190)*	86.7M	23.1B	82.8 / 96.1
AmoebaNet-A (6, 204)*	99.0M	26.2B	82.8 / 96.2
AmoebaNet-C (6, 228)*	155.3M	41.1B	83.1 / 96.3

Dataset	Acc.	Network	Acc.	Best network
Food-101	90.0	Deep layer aggregation [40]	90.1	NASNet-A Large, fine-tuned ✓
CIFAR-10	97.9	AmoebaNet [41]	98.4*	NASNet-A Large, fine-tuned ✓
CIFAR-100	87.8	ShakeDrop [42]	88.2*	NASNet-A Large, fine-tuned ✓
Birdsnap	80.2*	Mask-CNN [43]	78.5	NASNet-A Large, fine-tuned ✓
SUN397	63.2	Places-pretrained VGG [44]	66.5	NASNet-A Large, fine-tuned ✓
Stanford Cars	94.1	Deep layer aggregation [40]	93.0	Inception v4, random init
FGVC Aircraft	92.9*	Deep layer aggregation [40]	89.4	Inception v3, fine-tuned
VOC 2007 Cls.	89.7	VGG [9]	88.4	NASNet-A Large, fine-tuned ✓
DTD	73.5	FC+PV-CNN+D-SIFT [45]	76.7	Inception-ResNet v2, fine-tuned
Oxford-IIIT Pets	93.8	Object-part attention [46]	94.3	NASNet-A Large, fine-tuned ✓
Caltech-101	93.4	Spatial pyramid pooling [47]	95.0	NASNet-A Large, fine-tuned ✓
Oxford 102 Flowers	97.1	Object-part attention [46]	97.7	NASNet-A Large, fine-tuned ✓

CIFAR-10 to ImageNet      Other Datasets and Tasks

NASNet-A: SOTA on 8/13 commonly used classification benchmarks

Credit: Nikhil Naik, Neural Architecture Search A Tutorial, 2019



Vineth N B (IIT-H)

§12.5 Neural Architecture Search

15 / 19

Other considerations that have been looked at in the NAS community is Architecture Transferability. If we learn an architecture, through a NAS method on CIFAR-10, how well does it transfer to ImageNet. And the search is shown that a lot of a contemporary NAS methods deliver fairly well on this count, where an architecture trained on CIFAR-10 also does well reasonably on ImageNet, as well as on other datasets and tasks.

(Refer Slide Time: 18:23)

### Future Directions<sup>6,7,8</sup>

- Search efficiency
- Moving towards less constrained search spaces
- Designing efficient architectures: automated scaling, pruning and quantization
- Joint optimization of all components in deep learning pipeline (data augmentation , architectures, activation functions, training algorithms)

- Designing architectures for multimodal problems, e.g., vision and language

<sup>6</sup>Cubuk et al., AutoAugment: Learning Augmentation Policies from Data, CVPR 2019  
<sup>7</sup>Ramachandran et al., Swish: A Self-Gated Activation Function, NEC Journal 2017  
<sup>8</sup>Bello et al., Neural Optimizer Search with Reinforcement Learning, ICML 2017



Vineth N B (IIT-H)

§12.5 Neural Architecture Search

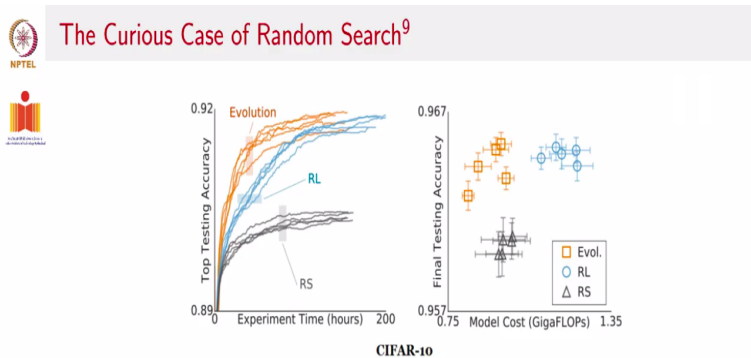
16 / 19

While this discussion on NAS was intended to be brief, keeping in view the scope of this course. Let us discuss a few future directions and open problems in NAS. One of the first issues is search efficiency. As I just mentioned, trying to perform NAS using Reinforcement Learning can create a training overhead of many days on standard GPU architecture. How do you improve the search becomes an important problem. A different perspective is to also move towards less constrained search spaces. One way you could improve your search efficiency is to constrain your search space more.

But now, we also do not want to constrain the search space more to be able to explore better architectures. Designing efficient architectures, such as those we get after pruning or after quantization, or probably finding the lottery ticket using NAS could also be an interesting direction of the space. Another important direction is a joint optimization of all components of a deep learning pipeline, not just architecture. We could also talk about what data augmentation to use, what activation functions to use, what optimizers to use, and so on and so forth.

So, including all of that in the NAS search pipeline would be an interesting future task in the space. Finally, designing architectures for multimodal tasks, such as vision and language tasks, could also be a very important and useful direction.

(Refer Slide Time: 20:22)



Difference in accuracy between best models found by random search, RL, and Evolution is **less than 1%** on CIFAR-10



<sup>9</sup>Real et al, Regularized Evolution for Image Classifier Architecture Search, AAAI 2019


Vineeth N B (IIT-H)

§12.5 Neural Architecture Search


17 / 19

Before we conclude this lecture, there is also an interesting observation by a few researchers about the Curious Case of Random search. In this work, in AAAI 2019, researchers observed that when one randomly searched for a Neural Network Architecture, the difference in accuracy was not too much between using random search, reinforcement learning or evolutionary algorithms. And this leads to an important conversation as to whether we really need Neural Architecture Search.

(Refer Slide Time: 21:02)

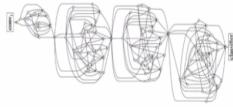


### The Curious Case of Random Search<sup>10,11</sup>



Architecture	Source	Test Error		Params (M)
		Best	Average	
NASNet-A <sup>14</sup>	[52]	N/A	2.65	3.3
AmoebaNet-B <sup>1</sup>	[43]	N/A	2.55 ± 0.05	2.8
ProxylessNAS <sup>1</sup>	[7]	2.08	N/A	5.7
GHN <sup>17</sup>	[50]	N/A	2.84 ± 0.07	5.7
SNAS <sup>1</sup>	[47]	N/A	2.85 ± 0.02	2.8
ENAS <sup>1</sup>	[41]	2.89	N/A	4.6
ENAS	[34]	2.91	N/A	4.2
Random search baseline	[34]	N/A	3.29 ± 0.15	3.2
DARTS (first order)	[34]	N/A	3.00 ± 0.14	3.3
DARTS (second order)	[34]	N/A	2.76 ± 0.09	3.3
DARTS (second order) <sup>3</sup>	Ours	2.62	2.78 ± 0.12	3.3
ASHA baseline	Ours	2.85	3.03 ± 0.13	2.2
Random search WS <sup>5</sup>	Ours	2.71	2.85 ± 0.08	4.3

Li and Talwalker (2019)  
CIFAR-10




network	test size	epochs	top-1 acc.	top-5 acc.	FLOPs (B)	params (M)
NASNet-A [56]	331 <sup>2</sup>	>250	82.7	96.2	23.8	88.9
Amoeba-B [34]	331 <sup>2</sup>	>250	82.3	96.1	22.3	84.0
Amoeba-A [34]	331 <sup>2</sup>	>250	82.8	96.1	23.1	86.7
PNASNet-5 [26]	331 <sup>2</sup>	>250	82.9	96.2	25.0	86.1
<b>RandWire-WS</b>	320 <sup>2</sup>	100	81.6 <sub>±0.13</sub>	95.6 <sub>±0.07</sub>	16.0 <sub>±0.36</sub>	61.5 <sub>±1.32</sub>

Xie et al. (2019)  
ImageNet

Credit: Nikhil Naik, *Neural Architecture Search A Tutorial*, 2019

<sup>10</sup>Li and Talwalker, Random Search and Reproducibility for Neural Architecture Search, UAI 2019

<sup>11</sup>Xie et al., Exploring Randomly Wired Neural Networks for Image Recognition, ICCV 2019

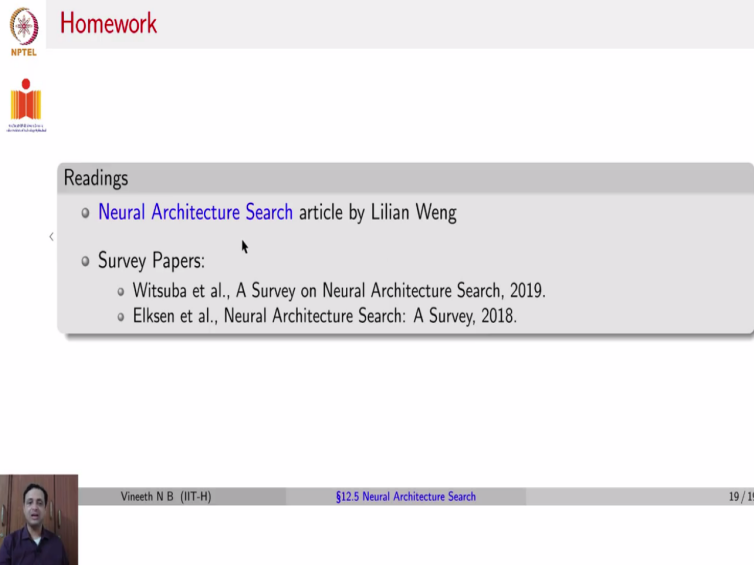


Vineeth N B (IIT-H)
§12.5 Neural Architecture Search
18 / 19

This observation was also seconded by a couple of other papers, which showed that random search and a random wiring of Neural Networks can perform to a reasonable extent in contrast to what was observed earlier in terms of searching Neural Network Architectures. This leaves an interesting an open question for the community.



(Refer Slide Time: 21:31)



The screenshot shows a presentation slide with a grey header containing the NPTEL logo and the word "Homework". Below the header is a small icon of an open book. The main content area is a grey box titled "Readings" containing a bulleted list of references. At the bottom of the slide, there is a small video thumbnail of a man, the name "Vineeth N B (IIT-H)", the slide title "\$12.5 Neural Architecture Search", and the page number "19 / 19".

Homework

Readings

- [Neural Architecture Search](#) article by Lilian Weng
- Survey Papers:
  - Witsuba et al., A Survey on Neural Architecture Search, 2019.
  - Elksen et al., Neural Architecture Search: A Survey, 2018.

Vineeth N B (IIT-H) \$12.5 Neural Architecture Search 19 / 19

The recommended readings for this lecture are once again, a very nice blog article by Lilian Weng on Neural Architecture Search and these two survey papers on Neural Architecture Search if you need more information.