Appendix: Neural Weight Search for Scalable Task Incremental Learning

Jian Jiang and Oya Celiktutan

Department of Engineering, King's College London, London, UK
{jian.jiang, oya.celiktutan}@kcl.ac.uk

In this appendix we present supplementary materials as follows: Insights into Neural Weight Search are presented in Section 1. Ablation studies are shown in Section 2. Experiments of our method conducted with different hyperparameters are shown in Section 3. Training setup for different network architecture are in Section 4. Implementation details of the baselines used in the main paper are discussed in Section 5. Visualisations of distributions of kernel indices are shown in Section 6.

1. Insights into Neural Weight Search

In transfer learning [10, 13], weights of layers from a pretrained model can be reused for finetuning new models. Considering the training cost, one efficient way is freezing parts of weights and finetuning the rest, e.g., fixing the low-level weights and retraining the high-level weights and vice versa. However, conducting finetuning on all weights can usually increase learning ability. Intuitively, it is inevitable that some weights of good generalization ability are updated during the finetuning and updated weights may be not suitable for finetuning other models.

Given a set of well-generalized weights, is it possible to utilize them to build a model without changing their values? Motivated by this, we propose a new problem setting named Neural Weight Search (NWS). Distinctive from conventional finetuning strategies, **NWS aims to search optimal combinations of frozen weights from a search space with repetition to build different models instead of changing the values of weights**. In this way, the re-utilization of weights is maximized, resulting in a large memory reduction. Nonetheless, we need to take care of the computation cost of the search process which is influenced by two main factors: (1) the size of the search space. (2) the searching strategy. Note that, the computation workload and the running time of a searched model during the inference are the same as those of a vanilla finetuned model. We elaborate on how to design the search space in the rest of this section.

Search Space: We recommend using grouped weights as elements to reduce the size of the search space because treating each weight scalar as an element will induce a search space of large size. In the main paper, we focus on convolution networks so we can group weights in convolution kernels, e.g., for a kernel with size 3×3, 9 weight values are grouped together. In terms of a fully connected (FC) layer, although its weights cannot be grouped into kernels, it is also feasible to divide weights into groups with different strategies. For instance, given an FC layer with input size 4 and output size 6, weights can be divided into 6 groups with each group having 4×1 weight scalars or 4 groups with each group having 2×3 weight scalars. We leave this direction to readers.

2. Ablation Studies

We study the individual impact of two modifications, i.e., pretrained pools, and initialisation of temporary kernels, on the performance of our proposed method using the Split-CIFAR-100 dataset. More explicitly, we conduct the following experiments: 1) We evaluate the importance of the generalization of weights in pools by pretraining the NWS using a subset (100 classes) of ImageNet (NWS-subImg) and the original version (Original) is pretrained using the full set. 2) We use randomly initialized temporary weights for each new task (NWS-random). As shown in Table 1, pretraining NWS with more classes (Original vs. NWS-subImg) on ImageNet benefits incremental learning, implying that more data helps NWS learn more distinctive and generalized weights. NWS-random has the worst average accuracy, demonstrating the importance of the initialisation of temporary kernels.

3. Different Hyperparameters

We evaluate our method from several aspects: *optimizer*, *learning rate (LR)*, *beta* (the coefficient in the Eq. 3 in the main paper), and *random seed*. For stochastic gradient descent (SGD), we used the same setting as that used in our main paper. More specifically, we use 0.9 as momentum,

Table 1: Ablation studies on the Split-CIFAR-100 dataset. NWS-subImg: NWS pretrained using a portion of the ImageNet. NWS-random: NWS with randomly initialised temporary kernels.

Method	Average Accuracy (%)	Per Task (MB)	Assist (MB)	Total (MB)
Original	**73.4**	1.6	1.3	33.9
NWS-subImg	72.5	1.6	1.3	34.1
NWS-random	68.7	**1.5**	1.3	**32.0**

4e-5 as weight decay, and set 'Nesterov' to 'True' for SGD. For Adam optimizer, we use Pytorch default setting to initialise. Milestones are used to divide the learning rate by 10 at certain epochs. We use milestones $\{50, 80\}$ when SGD optimizer is used and $\{50\}$ for Adam optimiser. For example, an initial learning rate 0.1 with milestones $\{50, 80\}$ will become 0.01 after epoch 50 and then become 0.001 after epoch 80. Results are reported in Table 2 and Table 3. We also calculate the mean and standard deviation of average accuracy and total memory. Low standard deviations show our model is stable.

We further investigate the impact of the size of a kernel pool. We pretrain kernel pools with different sizes (128, 256, 512 and 1024) on Sub-ImageNet. We obtain average results on 3 different seeds in Tab. 4. Results show that the size of 512 achieves the best performance.

4. Different Model Architectures

As presented in Section 5.6 in the main paper, we test another 3 different architectures: Resnet-34 [3], MobileNet-V2 [8], VGG-16 [9]. In this section, we show the detailed implementation and training setups.

Model modification. For Finetune-VGG16, We replace the last 3 fully connected (FC) layers with a single FC layer to reduce the training workload. For Finetune-MobileNet-V2, we remove the Dropout layer in the classifier as our results show Dropout can hinder its performance.

Pretraining. The baselines finetune a corresponding pretrained model (pretrained on ImageNet) for each task separately and the pretrained model is taken from Pytorch model zoo. The NWS-Res34 is pretrained for 160 epochs on ImageNet while NWS-VGG16 and NWS-MobileNetV2 are pretrained for 160 epochs on Sub-ImageNet.

Hyperparameters. We use Adam optimizer with an initial learning rate of 0.01, a milestone of $\{50\}$, a batch size of 32 and epochs of 100 for all methods to train new models.

5. Baseline Implementation Details

As mentioned in Section 5.3 in the main paper, following the previous works [5, 12], we use the same set of common hyperparameters for comparing methods and our method. More specifically, there are 5 common hyperparameters: the initial learning rate, batch size, training epochs, attributes of the optimizer and the scheduler. Here we show the model-specific hyperparameters as followed.

- **Finetune** No extra model-specific hyperparameters.
- **KSM** [12]. We adapt from the official repository KSM. We use default values of the learning rate for the mask (2e-4) and initialization of the mask (1e-2). It saves the statistics of batch normalization layers, soft masks for each task, and a single shared backbone for all tasks.
- **PackNet** [5]. We adapt from the repository CPG. The prune ratio is set to the default value of 60%. The model is finetuned from the backbone model for 100 epochs (in default) before pruning is conducted. Then retraining the model for 30 (in default) epochs. It learns a binary kernel-wise mask for each task and updates the pretrained model. After the first task is learned, it frees up the weights at a predefined ratio (60%) and relearns the first task to obtain two groups of weights, namely, 60% learnable weights and 40% fixed weights. It finetunes the released learnable weights for the second task. After the second task is trained, 60% of the learnable weights are freed up, which means only 36% of the total weights remain learnable, and so on. For each task, it saves the statistics of batch normalization layers. Note that all tasks share the same set of kernel masks; so the memory cost of saving kernel masks does not change. The backbone model is updated during the incremental training and only the last updated backbone is saved. The learning of the current model relies on the previous model only.
- **AQD** [1]. We adapt from the official repository AQD. The bitwidths of feature quantization and weight quantization are both 9 bits. It saves a quantised network for each task.

6. Visualization of Distributions of Kernel Indices

To recap, the ResNet-18 has a total of 21 layers (including short-cut layers). Consequently, the layer-wise kernel pool for 8_{th}, 13_{th}, 18_{th} layers are convolution layers with a kernel size of 1×1 for short-cut function. The last layer (the

Table 2: Hyperparameters on split-CIFAR-100. 'LR' refers to the learning rate; 'Beta' refers to the coefficient; 'Seed' refers to the random seed; 'Memory' refers to the total memory cost including saving the kernel pools; 'Mean' ('STD') refers to the mean (standard deviation) of average accuracy or total memory.

Optimizer	LR	Beta	Epoch	Seed	Average Accuracy (%)	Memory
SGD	1e-1	0.5	100	1993	73.4	32.4
SGD	1e-1	0.5	100	1994	73.9	32.7
SGD	1e-1	0.5	100	1995	73.8	32.6
SGD	1e-2	0.1	100	1993	72.9	33.8
SGD	1e-2	0.1	100	1994	72.5	33.9
SGD	1e-2	0.1	100	1995	74.7	33.9
SGD	1e-2	0.5	100	1993	74.1	33.9
SGD	1e-2	0.5	100	1994	72.9	33.9
SGD	1e-2	0.5	100	1995	73.4	33.9
SGD	1e-2	1	100	1993	75.6	33.8
SGD	1e-2	1	100	1994	72.6	33.9
SGD	1e-2	1	100	1995	70.2	34.0
SGD	1e-3	0.5	100	1993	79.8	33.6
SGD	1e-3	0.5	100	1994	79.7	33.6
SGD	1e-3	0.5	100	1995	79.3	33.6
Mean					74.6	33.6
STD					2.85	0.54
Adam	1e-2	0.5	100	1993	71.0	25.4
Adam	1e-2	0.5	100	1994	71.3	25.6
Adam	1e-2	0.5	100	1995	71.9	25.7
Mean					71.6	25.6
STD					0.46	0.15

Table 3: Hyperparameters on CUB-to-Food. 'LR' refers to the learning rate; 'Beta' refers to the coefficient; 'Seed' refers to the random seed; 'Memory' refers to the total memory cost including saving the kernel pools; 'Mean' ('STD') refers to the mean (standard deviation) of average accuracy or total memory.

Optimizer	LR	Beta	Epoch	Seed	Average Accuracy (%)	Memory
SGD	1e-3	0.1	100	1993	81.7	9.9
SGD	1e-3	0.1	100	1994	81.0	9.9
SGD	1e-3	0.1	100	1995	81.4	9.9
Mean					81.4	9.9
STD					0.44	0.0
Adam	1e-3	0.1	100	1993	74.9	7.6
Adam	1e-3	0.1	100	1994	74.8	7.5
Adam	1e-3	0.1	100	1995	75.7	7.5
Mean					75.1	7.5
STD					0.49	0.06

21_{th} layer), the replacement of an FC layer, is implemented as 1×1 convolution layer as well. The layer 1 has a kernel size of 7×7 and that of the remaining ones is 3×3.

We visualise layer-wise distributions of selected kernel indices of 5 trained models for CUB-to-Food , for the 21 layers, from Figure 1 to 5 respectively. In each subplot, the x-axis is the kernel index and the y-axis is the Selection Rate (SR) of a unique kernel. We define SR as h_i^l/d^l, a frequency of a kernel index i is selected. To recall, h_u^l is the selection times of a unique index and d^l is the number of required kernels to build the layer. Note that, SR is in the range of [0, 1] and i is in the range of [0, 511]. We observe that although the distributions of the same layer of the two different tasks are similar, the ordering of indices is

Table 4: Impact of size (the number of kernels) of a pool. Pools are pretrained on Sub-ImageNet. Results are averaged on 3 different random seeds.

Size of a Kernel Pool	Benchmark	Average Accuracy (%)	Memory
128	Split-CIFAR100	68.9	25.6
256	Split-CIFAR100	67.5	28.8
512	Split-CIFAR100	72.5	34.1
1024	Split-CIFAR100	66.2	39.1
128	Cub-to-Sketch	54.9	7.1
256	Cub-to-Sketch	48.7	8.2
512	Cub-to-Sketch	67.3	10.2
1024	Cub-to-Sketch	50.6	12.1

different.

The first two layers among tasks select some kernels with a rate larger than 0.3, resulting in low diversity of kernels while in other layers the selections of kernels are more diverse. Our observations show that, for the first and the second layers of the first task, more than 30% of the selected kernels are the same kernel. We observe similar distributions of the same layer among different tasks. Intuitively, frequently selected kernels are helpful because of their generality but it is hard to infer the actual factor inducing the dominance. To give some insights, low-level kernels capture coarse common local features (e.g., line, curve, and dot), high-level kernels extract fine-grained specialized global features (e.g., ear, eye, and head), and the last layer infers the class based on high-level features. Therefore, we conjecture that low-level kernels are less diverse and high diversity is necessary for high-level kernels to ensure specialisation. Besides, it may be because the former layers have a less number of kernels, while the later layers are over-parameterized. In other words, more kernels in a layer naturally lead to a more diverse selection of kernels from the corresponding kernel pool.

References

[1] Peng Chen, Jing Liu, Bohan Zhuang, Mingkui Tan, and Chunhua Shen. Aqd: Towards accurate quantized object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 104–113, 2021.

[2] Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Transactions on graphics (TOG)*, 31(4):1–10, 2012.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[5] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7765–7773, 2018.

[6] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[7] Babak Saleh and Ahmed Elgammal. Large-scale classification of fine-art paintings: Learning the right metric on the right feature. *arXiv preprint arXiv:1505.00855*, 2015.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[10] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[11] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[12] Li Yang, Zhezhi He, Junshan Zhang, and Deliang Fan. Ksm: Fast multiple task adaption via kernel-wise soft mask learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13845–13853, 2021.

[13] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

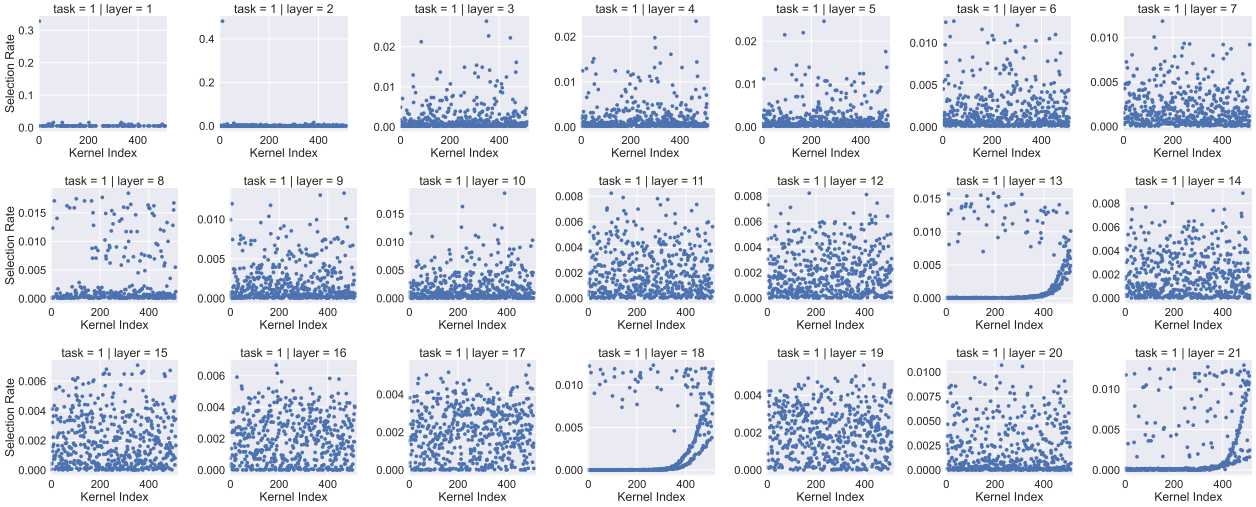

Figure 1: Visualization of the distribution of selected kernel indices of the model trained on CUB [11]

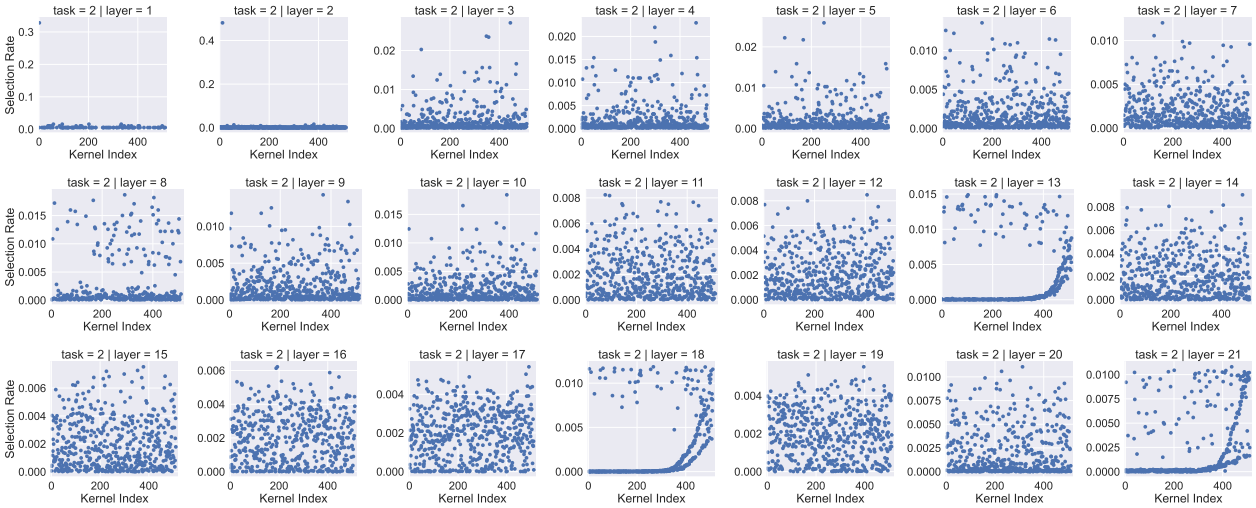

Figure 2: Visualization of the distribution of selected kernel indices of the model trained on Cars [4]

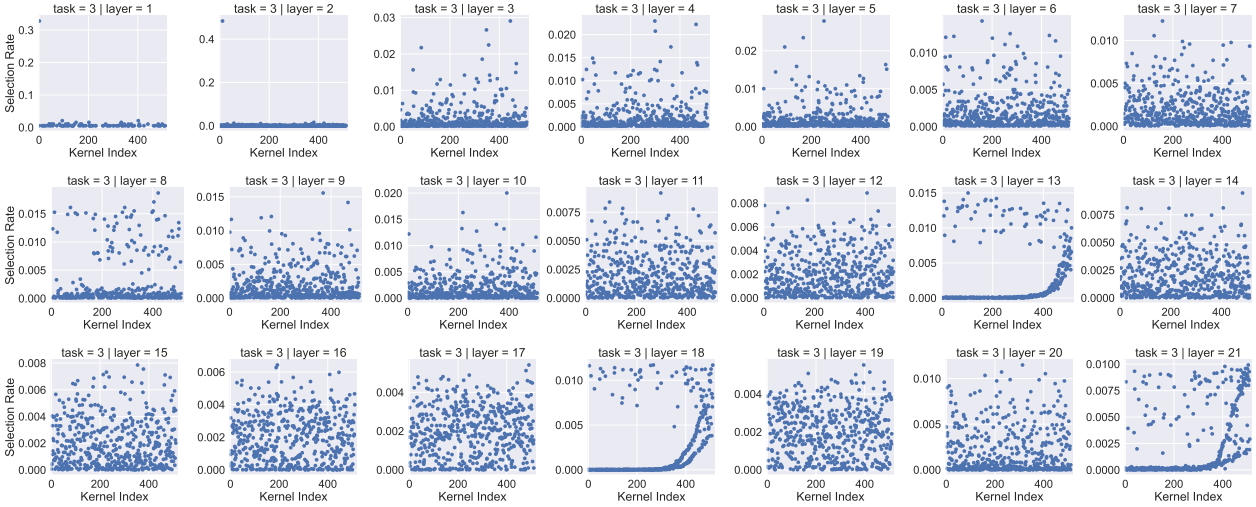

Figure 3: Visualization of the distribution of selected kernel indices of the model trained on Flowers [6]

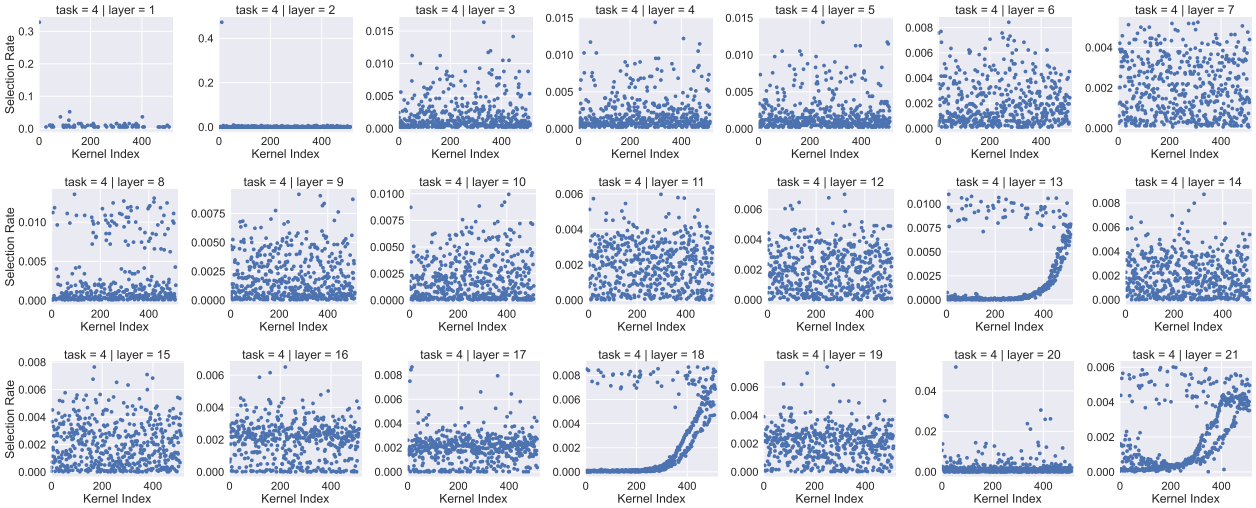

Figure 4: Visualization of the distribution of selected kernel indices of the model trained on WikiArt [7]

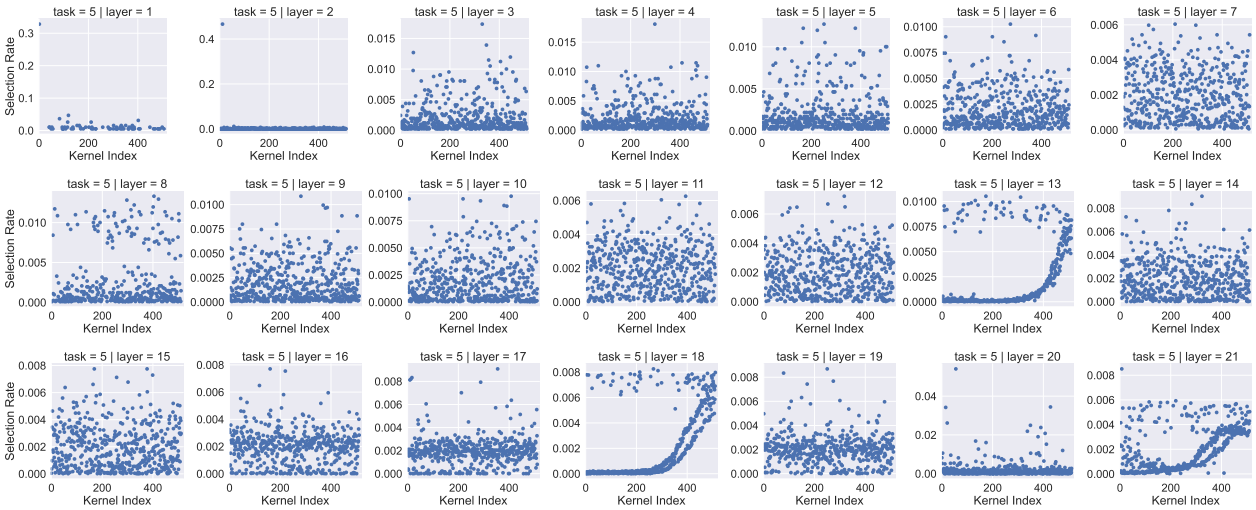

Figure 5: Visualization of the distribution of selected kernel indices of the model trained on Sketches [2]